IBM Session Manager for z/OS

Panels, Scripts and Variables

Version 2 Release 2

SC34-7068-01

IBM Session Manager for z/OS

Panels, Scripts and Variables

Version 2 Release 2

SC34-7068-01

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 295.

This edition applies to Version 2 Release 2 of IBM Session Manager for z/OS, program number 5697-N61, and to all subsequent versions, releases, and modifications until otherwise indicated in new editions.

Contents

About this manual

This is the *Panels, Scripts, and Variables* manual for IBM® Session Manager for z/OS®. It describes the definition and usage of the PANEL and SCRIPT statements, in addition to the Panel and Script Language (TPSL) and Session Manager variables.

It is intended to be read by a Technical Programmer who wishes to understand the supplied panels and scripts so that they can create their own. You should read this manual if you are responsible for installing, configuring, or supporting the product.

Note New users should review the *Installation and Customization* manual, which should give a good under-standing of the product's concepts and enable Technical Programmers to tailor Session Manager to Installation requirements.

Note Any references in this manual to "Session Manager version 1.3.15" and to "1.3 Functional Enhancement PTF 3" are synonymous.

Online and Batch Administration

Online Administration

Instead of supplying product configuration statements directly, Online Administration (hereafter "OLA") enables administrators and end-users of Session Manager to tailor the product using a series of menus, lists and attribute display panels.

Batch Administration

If many changes are required to a large number of configuration definitions, this capability enables administrators and end-users of Session Manager to tailor the product using a batch job.

Note To use Online and/or Batch Administration, you must first upgrade to Session Manager 1.1.10 or higher and then, if your existing configuration is Classic (that is, all configuration definitions are stored in members of PDS(s) allocated to the DDNAME of CONFIG), you must run the OLA Enabler to implement the new format configuration. For details, see 'OLA Enabler' in the *Online and Batch Administration* manual.

External Security Managers

External Security Managers (hereafter "ESMs"), such as RACF®, can be used with Session Manager to authenticate users, set their authorization level and OLA security class, and determine which applications a user can access.

For details, see the ‘Defining security and implementing dynamic menus’ chapter in the *Installation and Customization* manual.

Session Manager documentation

The following documentation accompanies Session Manager:

Manual	Purpose
Installation and Customization	Goes through the steps required to install the Session Manager software, and provides general information on the methods and options available to configure and operate your system.
User and Administrator	Describes in detail the features and facilities provided by Session Manager.
Online and Batch Administration	Explains the set-up and configuration of OLA, how to use the interface, and how to utilize both OLA and Batch Administration to modify the Session Manager configuration.
Technical Reference	Provides a detailed reference for Session Manager commands and configuration statements, along with problem diagnosis assistance.
Quick Reference	Provides a quick way to find the correct syntax for commands, configuration statements, and variables, without detailed explanations.
Panels, Scripts and Variables	Gives a detailed technical account of defining panels, using scripts and variables, and the product's Panel and Script Language (TPSL).
Messages and Codes	Contains explanations of all messages issued by Session Manager, and the actions that should be taken.

Additionally, the *Program Directory* contains information for systems programmers about the program material and procedures for installing IBM Session Manager under z/OS.

Conventions

The following typographic conventions are used:

`boldface`	Indicates a command or keyword that you should type, exactly as shown. When mixed case is used, the element in upper case represents the shortest acceptable form. For example, `MSGsuffix` can be abbreviated as far as `MSG`.
`italics`	Indicates a variable for which you should substitute an appropriate value.
`monotype`	Indicates literal input and output.
Ctrl+D	Indicates two or more keys pressed simultaneously.
`[ ]`	Brackets surround an optional value.
`|`	Vertical bars separate alternative values from which you must make a selection.
. . .	Ellipsis indicates that the preceding element may be repeated.
@	Some commands or key sequences make use of the 0x7C (i.e. x'7C') character. When using the English language code page, this character is displayed as the @ sign, but may be displayed as a different character in some other code pages. In this document, the 0x7C character is always presented as the @ sign. You should enter the appropriate 0x7C character symbol for the code page you are using.

Summary of new features

For a summary of changes made to the product in its most recent releases, please refer to the *Installation and Customization* manual.

CHAPTER 1

Scripts: definition and usage

This chapter provides information on the Session Manager Script facility.

Panels and scripts are used extensively in the Session Manager system. Scripts consist of a set of parameters, which may almost be considered to be 'verbs'. Scripts perform certain tasks, for example, a script invoked from a Menu screen can perform a logon to a CICS® system.

Scripts usually fall into one of the following categories:

- Session scripts
- Exit scripts
- Command scripts
- Window scripts
- Application Builder scripts

Authorized scripts

For Session Manager 1.1.05 and higher, suitably authorized scripts are able to issue product commands with a Session Manager user authority (AUTH) of 9, the highest user authority. Authorized scripts are described in more detail later in this chapter.

Overview of scripts

Session scripts

Session scripts relate to sessions invoked from a Menu screen and are further sub-divided into six categories:

Script	Description
INITSCRIPTs	invoked before the application is initiated.
ENVIRONSCRIPTs	invoked when the application has been contacted.
STARTSCRIPTs	invoked after the Environscript has run.
AUTOSCRIPTs	invoked by a user entering a key sequence.
ENDSCRIPTs	invoked to terminate the application session.
TERMSCRIPTs	invoked after application session has terminated.

Application Builder functions

Any of the above can also carry out Application Builder functions by use of certain Application Builder verbs. These verbs enable a script to selectively combine data output from one or more sessions and present it on a single panel. See 'Application builder verbs' on page 113 for further details of Application Builder script verbs.

Exit scripts

Exit scripts can be defined to run in place of many of the exit points of the Session Manager User exit. See page 33 for further details of exit scripts.

Command scripts

Command scripts can be defined to run in place of Session Manager defined commands. They can also be used to specify user-defined script commands. See 'Command scripts' on page 45 further details.

Window scripts

A Window script can be invoked from the Menu screen, but it does not relate to any particular session. A Window script displays one or more windows which can be used to run sessions. Multiple sessions can then be displayed simultaneously on one physical screen.

Understanding execution of a script

The TTPSL command can be used for tracing the logic flow through panels and scripts. It is useful in understanding what is happening in an existing script, and for debugging a new script. The command and its parameters are described fully on page 252.

Session Manager variables

Session Manager variables are also extensively used in panels and scripts. There are numerous different types of supplied variables (see 'Session Manager variables' on page 197), and an Installation can also define their own variables.

Authorized scripts

For Session Manager 1.1.05 and higher, suitably authorized (system) scripts are able to issue product commands with a Session Manager user authority (AUTH) of 9, the highest user authority.

For a particular script to acquire 'authorized' status, it must have been loaded from an ASCRIPT Session Manager configuration PDS(E). (For more information on Session Manager configuration DDNAMEs, please refer to the *Online and Batch Administration* manual.)

CAUTION If an authorized script is subsequently made into a command script then it could be used to issue authorized commands. Therefore, ensure that the ability to create commands is available only to authorized users.

The one function that an authorized script can perform that an unauthorized script cannot is to invoke the script verb ISZCMDA to issue a Session Manager command. ISZCMDA, which was introduced in Session Manager 1.1.05, is the same as the script verb ISZCMD (see the *Panels, Scripts and Variables* manual), except that the command is issued with an AUTH of 9 rather than the user's AUTH value.

Notes

1. An authorized script will become unauthorized during a particular invocation if it is called by, or calls, an unauthorized script. In other words, an unauthorized script does not acquire 'authorized' status on invoking an authorized script.
2. If user variable t_deauthmsg is set to 'Y', and an unauthorized script calls an authorized script, message 4007 is issued identifying the unauthorized script that has de-authorized the previously authorized script(s).
3. If session variable s_script_cmds is set to 'Y' then ISZCMD/A will support command scripts, as well as native Session Manager commands. It will do this by converting the ISZCMD/A into the equivalent of a TPSL call. Similarly, if s_script_cmds is set to 'Y' then SAUTOSEQ scripts can run with session scripts. It does this by temporarily converting a WAITTERM, WAITDATA or WAITAPPL into a CALL for the SAUTOSEQ script.

Defining session scripts and automatic input streams

Each session script defines either one, or a series of input data streams to be passed to an application. The name of each script is used on the INITSCRIPT, ENVIRONSCRIPT, STARTSCRIPT, AUTOSCRIPT, ENDSCRIPT, or TERMSCRIPT parameters so that the input is passed at the appropriate time. The script comprises control information, such as SBA and cursor information, and the actual text to be passed to the application. Scripts are able to search for specific information in the data stream output by the application, and TPSL statements may then alter the script processing according to what is found. This is described in a later section entitled 'Defining conditional scripts' on page 25. Scripts may also be nested up to any level using a standard CALL statement.

A single script can comprise a series of input data streams, passed sequentially to the application. An input data stream in a script is recognized by the occurrence of an INPUT parameter of the SCRIPT statement. The script should wait for a response from the application before passing the next input data stream to it.

Such input sequences may be considered to operate in exactly the same way as if the user were actually sitting at the terminal, keying the input and pressing the Enter or function keys at the appropriate points. Each INPUT parameter, apart from the last one in a script, should be followed by a WAITAPPL parameter to cause the script to wait for a response.

For example, a SCRIPT statement might define a CICS logon sequence as follows:

```
SCRIPT CICSLOGN
       Waitappl
       Input  cursor 1 5 key enter
          sba 1 1 text 'CSSN'
       Waitappl
       Input cursor 6 20 key Enter
          sba 4 12 text 'JOHN'
          sba 6 16 text 'PXYZ'
```

which is equivalent to the user keying the CICS signon transaction code and pressing the Enter key, then keying the operator name and password followed again by the Enter key.

Although this is only a single script, it actually consists of two input streams. The script ensures that it has a response from CICS before passing the first input stream, and waits again before passing the second input stream. It is not necessary for the script to wait after the second input stream since control is passed to the application, and the last screen is displayed at the terminal. There are methods to prevent the script waiting an inordinate amount of time. These are described later in this section. There is no limit to the number of inputs that may be defined on a single script.

By default, except for the response from the last input of a script, output screens are not shown at the terminal. However, a WAITAPPL DISPLAY parameter causes Session Manager to display the intermediate output screens produced by the application. This option would not normally be required, and it would be disconcerting for a user to see a series of output screens flash quickly by as Session Manager passes a series of input data streams to an application.
WAITAPPL DISPLAY OFF causes Session Manager to suppress the output screen.

When there are multiple input fields on a physical screen, they are passed to the application in ascending order from the top left hand corner down to the bottom right hand corner. In a script, each input field is coded as SBA and TEXT subparameters and it should be ensured that the SBAs are also in ascending order, unless it is known that the receiving application will not be confused by out of sequence data. Each SBA subparameter should precede the TEXT subparameter for which it is intended. If no SBA precedes a TEXT subparameter, an SBA of 1,1 is automatically generated.

If key values of CLEAR, PA1, PA2 or PA3 are specified on the INPUT parameter, all CURSOR, SBA, and TEXT subparameters are ignored, rather than Session Manager initiation or update processing of the script definition being terminated with invalid statements.

At any time, the script may issue the AUDITMSG parameter, which sends message 226 to the Audit file. This can be useful when testing a script to plot its progress. It may also be useful when a script has detected a problem. The AUDITMSG parameter may be coded to issue the required message, For example:

```
AUDITMSG ‘Script CICSLOGN has started’
```

Not only can a script issue a message to the Audit file or console, but it is possible to issue a message to the terminal. This is done using the USERMSG parameter, for example:

```
SCRIPT CICSLOGN
       Usermsg text 'CICS logon script now running. Please wait'
       Waitappl
       Input key clear
       Waitappl
       Input text 'CSSN' key enter
```

In addition, the script may wait for a response from the terminal using the WAITTERM parameter. This might only be that the user is required to press the Enter key in order for the script to proceed, but it ensures that the message remains on the screen, and is read by the user, before being displaced by another display. If any input is entered in response to the USERMSG, the input is placed in the Session Manager variable, t_command, which can then be processed by the script.

The USERMSG and WAITTERM combination may be extended by using a panel definition, and any values entered at the terminal are assigned to the panel variables. Thus, a conversation may be created between the script and the terminal user.

Note Care should be taken with USERMSG, since if it is used with application input or output, the terminal mode may be altered and data stream contents may be misinterpreted by the application. In addition, the application screen is not saved or restored when a USERMSG is issued.

Since a number of scripts may have some common processing, Session Manager provides the ability to call a script from within a higher level script. This effectively reduces the number of parameters required in higher level scripts, since a complete new script need not be coded when only a minor difference in processing is required.

At any point in time, a script may cause the application session to be terminated unconditionally, for example, when the application is not responding. The ENDSESS parameter provides this capability.

More complex scripts can be coded where conditional statements may be used to check values or response times, and the flow of the script may be altered accordingly. Conditional scripts are described on page 25.

Potential problems to consider

When designing scripts, careful consideration should be given to the way in which the application will react to each separate input sequence being passed to it. Once an input stream has been passed to an application, the script should wait for the application output in response to the input. Usually this technique functions perfectly, but users should note that there are situations which can cause a synchronization problem.

For example, suppose the application performs more than one output in response to an input stream. If the script does not wait for each application output, the next input stream could be sent to the application at the wrong time. A WAITAPPL parameter should be coded in the script for each application output. Alternatively, WAITDATA, possibly with the OUTSCAN verb, may be used to overcome this type of problem. See 'Defining conditional scripts' on page 25.

Also, an application may issue a write prior to accepting the first INPUT. When this happens, it is possible for the input from Session Manager, and the write from the application to 'cross', and as a result the input is lost. In this situation, a WAITAPPL parameter must be coded in the script *before* the first INPUT parameter.

An exception to this is when the application issues a Write Structured Field (WSF) transmission for a Read Partition Query (RPQ) in order to determine the characteristics of the terminal. By default, Session Manager does not pass the query to the controller, but simulates the response to the query back to the application. This means that a WAITAPPL parameter should not be coded for an RPQ and an INPUT parameter should not be coded for the reply.

If, however, it is necessary for the script to know when the RPQ is issued, the RPQ simulation can be switched off by placing 'SIMRPQ NO' at the start of the script. Note that the Query List is not supported by 'SIMRPQ YES' and users should specify the `RPQLScriptauto` parameter instead (see page 108).

The script must then ensure that the query is passed to the terminal, and that the response is returned directly to the application. This is achieved by coding WAITAPPL DISPLAY to ensure that the query reaches the terminal, followed by WAITTERM PASS to intercept the reply, followed by INPUT PASS to ensure the response is passed to the application. The PASS option is required to prevent Session Manager altering the format of the reply. For example:

```
SCRIPT RPQ
    Simrpq no
    Waitappl display       /* send the RPQ to the terminal */
    Waitterm pass          /* wait for terminal to respond */
    Input pass             /* pass directly to the application */
```

Again, WAITDATA and OUTSCAN can be used as an alternative.

Since for some reason the application may not be responding, it is advisable to place an APPLTIME parameter at the start of a script to prevent the script waiting for an unlimited time. A timeout occurs when the interval specified on this parameter expires. The NORESP condition may be tested to detect this.

Usually the script processing is dependent upon finding an application output stream containing a specific string of characters, for example, the signon to an application may cause several output streams to be issued, one of which contains the character string 'ENTER NAME'. Any string can be searched for using the OUTSCAN parameter in the script. The OUTSCAN parameter searches the data stream. This means that care should be taken when a string appears to include spaces on the screen, since they may be field attribute characters in the data stream and a match will not be found.

A convenient way to use OUTSCAN processing is in a conditional script and 'Defining conditional scripts' on page 25 gives further information.

It is advisable to trace the sequence of application, or session, inputs and outputs because applications may not respond predictably or consistently, and to test scripts with their target applications before committing them to the production system. Tracing is discussed in the next section.

Tracing for script processing

Tracing of the sequence of events may be started using one of the TRACE commands:

```
TRACE LU luname ON DATA

TRACE USER userid ON DATA

TRACE ON DATA                          (issued at the terminal)
```

Once all the steps required for the script are complete, the trace should be switched off, using the TRACE OFF DATA command. The SPIN TRACE command may be issued to print the trace output, without switching the trace off.

The trace should contain four types of entry:

- Terminal output (Session Manager to terminal)
- Terminal input (terminal to Session Manager)
- Session output (application to Session Manager)
- Session input (Session Manager to application)

The following are examples of the four formats of the entries:

```
ISZ0387I TERMINAL INPUT - USER=MGC - time 12:54:06 ..
           00000007/X'00000007'            <--- length of data
           ... data ...
ISZ0387I SESSION INPUT - USER=MGC - time 12:54:06 ..
ISZ0525I Application S09TSO (1) data MGC
           00000007/X'00000007'            <--- length of data
           ... data ...
ISZ0387I SESSION OUTPUT - USER=MGC - time 12:54:06 ..
ISZ0525I Application S09TSO (1) data MGC
 00000000  24800000 00000000 00000000      <--- internal flags
           00000058/X'0000003A'            <--- length of data
           ... data ...
ISZ0387I TERMINAL OUTPUT - USER=MGC - time 12:54:06 ..
 00000000  34800000 00000000 00000000      <--- internal flags
           00000058/X'0000003A'            <--- length of data
           ... data ...
```

For every session output with data, code one WAITAPPL or WAITDATA statement.

For every session input, code one INPUT statement. Generally, code the INPUT from the data portion as follows:

```
Byte 1    : Aid            : KEY parameter
Bytes 2,3 : cursor address : CURSOR parameter row,col
Byte 4    : X'11'          : SBA parameter
Bytes 5,6 : buffer address : SBA row,col
Byte 7    : data           : TEXT parameter
  .... more SBAs ....
```

For an explanation of the data streams, see the IBM manual *3270 Data Stream Programmers Reference.*

A different form of tracing can be done with the TTPSL command. The TTPSL trace records show the script logic flow and calls. Details of this command are on page 252.

Defining conditional scripts

Session Manager provides a special language, the Panel and Script Language (TPSL) which enables more complex scripts to be coded.Conditional processing and assignment of values to variables is provided. The format of TPSL statements is given later in this manual. This section describes how the facility might be used.

For example, suppose a particular user is likely to be logged onto three applications in the course of a day, each one requiring a different sequence of inputs at session start. This situation can be approached in two different ways. The first and most direct approach is to define three scripts, one for each of the input streams. Then, the relevant script name is specified on the STARTSCRIPT parameter of the appropriate sessions in the user's profile.

A second method is to specify a single script defining the inputs for each of the three applications. These input streams can then be made conditional upon the application by using the TPSL SELECT statement:

```
SCRIPT ...
       Select
          When s_appl = appl1
             Waitappl
             Input text ...
          When s_appl = appl2
             Waitappl
             Input text ...
          When s_appl = appl3
             Waitappl
             Input text ...
       End
```

Note that the special name 's_appl' is predefined within Session Manager.The complete list of special variable names is given in 'Session Manager variables' on page 197.

Conditions may also be compounded, for example:

```
SCRIPT ...
       If s_appl = 'CICSTEST' and t_auth = 7 then
          Waitappl
          Input text ...
```

Full IF processing may be used. That is nested IFs, AND, OR, NOT, and parentheses are all valid. The valid comparators are:

Operator	Description
EQ	Equal
NE	Not equal
GT	Greater than
LT	Less than
GE	Greater than or equal
LE	Less than or equal

Alternatively, symbolic comparators may be used. A full list is given in 'The Panel and Script Language (TPSL)' on page 145.

Conditional text input

If a script has the requirement to send fields, some of which are fixed and some dependent on certain conditions, this can be done in two ways. The IF statement could be placed before the INPUT parameter as in the example below:

```
If t_user = 'Arthur'
   Input key enter sba 1 2 text 'LOG'
                   sba 2 2 text 'DALEY'
                   sba 3 2 text 'BEGIN'
Else
   Input key enter sba 1 2 text 'LOG'
                   sba 2 2 text 'ANYONE'
                   sba 3 2 text 'BEGIN'
End
```

but a more elegant solution would be to place the conditional statements around the part of the text that is conditional, and use the INPUT KEY NO parameter.

```
Input key no    sba 1 2 text 'LOG'
If t_user = 'Arthur'
   Input key no sba 2 2 text 'DALEY'
Else
   Input key no sba 2 2 text 'ANYONE'
End
Input key enter sba 3 2 text 'BEGIN'
```

The text data is not sent to the application until INPUT KEY ENTER. KEY NO causes the text to be stored without being sent.

Scripts for searching data streams

TPSL statements may be used to specify loops which can be especially useful when searching for a particular character string in an application output data stream. The following example may be used when logging on to TSO.

```
Domax 5
Let found = false
Do while not found
    Waitappl
    .
    .
    .
    Outscan 'LOG ON'
End
```

The DO loop in the above example is limited to a maximum of five iterations. 'Found' is a special condition used in scripts when scanning for a particular character string, in this instance 'LOG ON'.

The WHILE condition is evaluated before entering the loop, whereas an UNTIL condition is tested at the end of the loop, thus ensuring at least one iteration of the loop. If DO WHILE NOT FOUND is coded, then 'Let found = false' should be coded before the loop, as in the previous example. The DO loop may alternatively have been coded using an UNTIL FOUND condition, for example:

```
SCRIPT TSO
    Appltime 30
    Domax 5
    Do until found
        Waitappl
        If noresp then
            Auditmsg 'Timeout on script TSO - session ended'
            Usermsg text 'TSO is not responding - session ended'
            Waitterm
            Endsess
            Return
       End
       Outscan 'LOG ON'
    End
    Input key clear
```

The maximum length of time that the script may wait for output from the application is limited to 30 seconds in this example. If no response is obtained from TSO in 30 seconds, a message is sent to the Audit file and another to the terminal. NORESP is a special condition used in script processing when the script is caused to wait for application output and is set when the interval has expired. The WAITTERM parameter ensures that the message remains displayed on the terminal until the Enter key is pressed. After this the session is terminated unconditionally by the ENDSESS parameter and the script is terminated. If the RETURN parameter were omitted, assuming TSO still does not respond, the loop continues until the loop limit is reached, in this instance, five. Any input from the terminal or output from the application is ignored, due to the preceding ENDSESS parameter.

In the following example, the KEEP parameter is specified on the second WAITAPPL to prevent the buffer from the first WAITAPPL being released. The OUTSCAN searches the data stream buffers. These are eventually sent to the terminal in the order they were sent by the application. The first WAITAPPL does not need to specify KEEP as the buffers are only released on subsequent WAITAPPLs. However, if the application sends an erase-type write buffer, any queued buffers are released, even when KEEP is specified.

```
SCRIPT KEEPBUFF
   Waitappl           /* Application sends an EWRT buffer
                      /* containing ‘TEST1’
                      /* containing 'TEST1' */
   Input 'LOGOFF'
   Do while not found
      Waitappl keep   /* Previous buffer is retained; application
                      /* sends WRT 'TEST2'
     Outscan 'TEST1'  /* outscan data found in first buffer */
   End
   Dispappl           /* Both buffers are sent to the terminal in
                      /* the order they were sent by the
                      /* application */
```

The preceding examples have used the WAITAPPL parameter to wait until a data stream from the application enables terminal input, normally using the script INPUT parameter.

Application output processing using the WAITDATA parameter differs from that for WAITAPPL. WAITDATA monitors the inputs from the terminal as well as outputs from the application.

The next example shows a sample script to monitor terminal input and application output using WAITDATA.

```
SCRIPT MONITOR
   Appltime 5
      Do forever
         Waitdata  pass
         Select
            When t_waitdata = 'T' then
                  /* ....                              */
                  /* input scanning code using INSCAN  */
                  /* ....                              */
               Input pass
            When t_waitdata = 'A' then
               If s_a  > 40 and s_a ¬= 65 then
                  If noresp then
                     Auditmsg 'time out'
                     Leave
                  Else
                        /* ....                 */
                        /* output scanning code */
                        /*  using OUTSCAN       */
                        /* ....                 */
                     Dispappl
```

```
            End
         Else
            Auditmsg 'session has ended'
            Leave
         End
      Else
         Auditmsg 'neither TERMINAL or APPL'
         Leave
      End
   End
```

The t_waitdata variable is used to indicate the type of input or output being received. If the variable t_waitdata is set to 'T', it indicates that the data is terminal input. The PASS operand causes this input data to be passed to the application. If the variable t_waitdata is set to 'A', it indicates application output. If the application session terminates during WAITDATA processing, then the script continues to run and WAITDATA will complete with t_waitdata set to 'A'. It is advisable therefore to test the status of the application session before continuing with any application output processing. The following has been coded for this purpose:

```
If s_a > 40 And s_a ¬= 65 Then
```

If the statement is true, then the application is still available. The output data stream can be sent to the terminal using the DISPAPPL parameter, as WAITDATA KEEP is the default.

Note If the WAITDATA statement occurs in a DO loop which is in the same script, then the DOMAX check is bypassed for that DO loop.

From these simple examples, it can be seen that scripts ranging greatly in power can be created to cater for all situations which might arise in an Installation. Further examples are given in the later section 'Conditional script examples' on page 31.

Input variable substitution

Variables in an input stream can be substituted by their appropriate values and thereby customize the input stream, for example, for a session or a specific user. User definable variables as well as Session Manager supplied variables can be specified. A full list of the supplied variables is given in the 'Session Manager variables' on page 197.

By preceding and following a variable name with an ampersand (&), the actual value of the variable will be substituted in the input stream.

For example, suppose an input stream is being set up to perform the CICS signon. The Installation has configured the system so that the user id and the password in CICS are the same as those for Session Manager. If the input stream is set up as follows:

```
SCRIPT CICSLOGN
       Waitappl
       Input sba 1 1 cursor 1 5 key enter
          text 'CSSN'
       Waitappl
       Input cursor 6 20 key enter
          sba 4 12 text '&t_user&' sba 6 16 text '&t_pass&'
```

the actual userid and password are substituted in the fields &t_user& and &t_pass&, prior to the input stream being passed to the application.

The following example generates a data stream which invokes a transaction 'TRN1' and supplies data which is obtained from a user defined variable, uc_tran. The user variable may have been assigned using a LET statement in a SCRIPT or PANEL, or taken from its input field on a panel, or assigned by the Session Manager User exit.

```
       Input sba 1 1 cursor 1 5 key enter
          text 'TRN1'
       Waitappl
       Input cursor 6 20 key enter
          sba 1 1 text '&uc_tran&'
```

Conditional script examples

This section contains two complete examples of more complex scripts. The first may be used to logon to TSO. The limit set for the script to wait for a response from the application is 60 seconds. The script scans the application's output data stream for the string 'LOGON'. If not found then the USERMSG parameter issues a message to the terminal. The user must press the Enter key for the script to continue. The session is not terminated and therefore the user may attempt the logon manually.

If the logon is successful, the script simulates the Enter key being pressed. It then searches for three asterisks and again simulates the Enter key being pressed when found. The script searches for the ISPF menu and issues the command '2.1' when the menu is found. Once the script has detected a response using the WAITAPPL parameter, the screen is split using PF2.

```
SCRIPT TSOLOGON
   Appltime 60
   Domax 5
   Usermsg text 'You are being logged on to TSO. Please wait'
   Do until found
      Waitappl
      If noresp then
         Usermsg text 'LOGON screen not found.
                       Press enter to continue'
         Waitterm
         Return
      End
      Outscan 'LOGON'
   End
   Input key enter
   Do until found
      Waitappl
      If noresp then
         Usermsg text "***'s not found. Press enter to continue"
         Waitterm
         Return
      End
      Outscan '***'
   End
   Input key enter
   Do until found
      Waitappl
      If noresp then
         Usermsg text 'ISPF menu not received.
                    Press enter to continue'
         Waitterm
         Return
      End
      Outscan 'ISPF'
   End
   Input key enter sba 2 15 text 's.log'
   Waitappl
   Input key pf2 cursor 1 15
```

The second example performs a logon to a virtual machine with userid and password matching those of the Session Manager userid and password. The first DO loop searches the data streams from the application for the character string 'VIRTUAL'. If found, the script simulates the Clear key being pressed to clear the VM logo. If there is no application output in response to this, the script outputs a message to the terminal, and when the Enter key is pressed, the session is terminated unconditionally. If there is a response, then the script sends the LOGON to VM.

```
SCRIPT VM
   Appltime 30
   Domax 5
   Do until found
      Waitappl
      If noresp then
         Auditmsg 'Script VM waitappl timeout - session ended 1'
         Usermsg text 'Application is not responding - try later'
         Waitterm
         Endsess
         Return
      End
      Outscan 'VIRTUAL'
   End
   Input key clear
   Waitappl
   If noresp then
      Auditmsg 'Script VM waitappl timeout - session ended 2'
      Usermsg text 'Application is not responding - try later!'
      Waitterm
      Endsess
      Return
   End
   Input text 'LOGON  &t_user& &t_pass&'
```

Exit scripts

Scripts can be defined to run for an exit point, instead of the User exit coded in Assembler/COBOL.

For most exit scripts, this is achieved by specifying the exit point name and the option 'S' on the OPTION configuration statement. A script of the form EXIT*xx* must be defined for the relevant exit point, where 'xx' is the exit point. For example, if E33 S is specified on the OPTION statement, a script EXIT33 will be called. A special return code, 52, enables the User exit to be called *after* the script call, if desired.

Notes

1. Exit points E01, E09, E25, E35, and E71 must be coded as Assembler/COBOL exits: they cannot be specified as scripts.
2. The OPTION parameters only take effect on the initial load, not by using update. To update an Exit script, see 'Using the Update Facility' in the *Technical Reference*.
3. Full details of the exit point parameters of the OPTION statement are in the *Technical Reference*.

For the input (E25) and output (E35) 3270 datastream exits, load module names are identified on the SYSTEM statement by the INPUTEXIT and OUTPUTEXIT parameters respectively. An additional, related, SYSTEM statement parameter, EXITWALEN, specifies the size of the work area made available to the exits. For details on these parameters, see the *Technical Reference*.

Valid SCRIPT parameters

Exit scripts can use the following SCRIPT parameters:

- All TPSL verbs
- AUDITMSG
- CALLEXIT
- ISZCMD
- CALL

If the CALLEXIT parameter is specified, then exit point E09 is called.

Return codes

The script *must* set the variable t_rc to the appropriate exit return code. Valid return codes are the same as those for the User exit. In addition to these, return code 52 is available for Exit scripts. If a return code 52 is set, the User exit is invoked immediately *after* the Exit script terminates, with the same parameters as were passed to the script.

Use of ISZCMD

The Exit script is invoked with a user authority of 9. This provides access to the full set of available commands, even if the exit is invoked for a user who does not have the required authority.

The Exit script can invoke the following commands using ISZCMD:

BLOCK, BRECEIVE, BROADCAST, CLOSEDOWN, DELETE, DISCONNECT, DUMP, END, FLASH, FORCE, HALTSCRIPT, HARDCOPY, HCOPTION, LOCK, LOGOFF, MSG, PASSFREE, PCTRANSFER, QQUIT, RECORD, RESET, SPIN, STARTLINK, STOP, STOPACB, STOPLINK, TERMINATE, TRACE, TTPSL, UPDATE.

If session variable s_script_cmds is set to 'Y' then ISZCMD/A will support command scripts, as well as native Session Manager commands. It will do this by converting the ISZCMD/A into the equivalent of a TPSL call. Similarly, if s_script_cmds is set to 'Y' then SAUTOSEQ scripts can run with session scripts. It does this by temporarily converting a WAITTERM, WAITDATA or WAITAPPL into a CALL for the SAUTOSEQ script.

Use of user defined variables

All 'gc_' and 'gn_' variables are available in all Exit scripts.

In exit scripts E11, E21, E22, E29, E31, E33, E39, and E79, the user associated 't_', 'uc_', and 'un_' variables are available and related to the user or terminal which is the subject of the Exit script. Local user defined variables 'lc_' and 'ln_' are available, but are defined and available for the length of the function only. For example:

- E11, E21, E22 local variables are shared by the Signon panel.
- E31, E33, E39, and E79 local variables are available for the duration of the session, i.e., from initiation to termination.

's_' variables are available without a subscript in the session related Exit scripts (E31, E33, E39, and E79) where they refer to the session that is the subject of the exit. These session variables can be referenced from the user related exits (E11, E21, E22, and E29) by using a subscript. The session exits can also refer to a user's other session variables using a subscript. The system related Exit scripts (E05, E06, E08, and E99) cannot access any session variables.

The system related Exit scripts (E05, E06, E08, and E99) are associated with a system user, i.e, t_user = ISZSMGR, t_term = SYSTEM. Only global variables are available. The system user is not shown in Query displays and cannot be sent a message or broadcast.

Accessing the exit parameters

An Exit script can access the exit parameters by using the exit parameter variables. These are fully described in 'Session Manager variables' on page 197.

Notes

1. To access the date and time, use the t_date (or t_date_l) and t_time variables.
2. The second user exit parameter, the Variable Access Routine, is not required in exit scripts.
3. The third user exit parameter, the message parameter, is not required in exit scripts. To send a message the following is required:
   - To send a message to the Audit file, use the AUDITMSG SCRIPT parameter.

- To send a message to a user's screen, assign the required message to the t_message variable using a LET TPSL verb.

Window scripts

Overview

The Windows feature of Session Manager is implemented by special scripts containing window-related parameters. These parameters are usually known as *window verbs* and they control the way windows are used and manipulated.

The Windows feature can be invoked from the Menu display by the WINDOWS command. There are two sets of window scripts supplied on the distribution tape. One of them invokes CUA® type windowing where each window is like a tile, the other invokes PC-type windowing where windows can be pulled down, overlayed, and re-sized at will.

The window scripts are quite extensive, so they are not fully listed in this manual. All the necessary members i.e., Scripts and other panels are copied by member ISZSWIN.

The Help panels for SAAWINDO are in member ISZHSWIN.

The Help panels for XWINDO are in member ISZHXWIN.

Member ISZPWIN contains all the other panel definitions. The following table gives the relationship between source members and panel definitions.

Panel name	OPEN by SAAWINDO member	OPEN by XWINDO member	Function	To invoke	Where it appears on screen
SOPENLST	ISZSWOPE	ISZSXOPE	List applications (menu)	OPEN (PF17)	cursor posn or Row
SBMSGS	ISZSSWIN	ISZSXWIN	Broadcasts/msgs	DMSGS	Under DMSGS
SLISTW	ISZSSWCA	ISZSXWCA	List windows	LISTW	Under LISTW
SWHOAMI	ISZSSWCA	ISZSXWCA	User information (toggle)	DETAIL	Row 2
SSCRLOP	ISZSWSCR	ISZSWSCR	Scroll a window	OFFSET	Under OFFSET
SSCRLIND	ISZSSWIN	ISZSXWIN	Scrolling information	N/A	Top right of active window
SACTBAR	ISZSSWIN		Action bar	N/A	Row 1
SPFNEW	ISZSSWCA		PFkeys (toggle)	DETAIL	Last row
SETUP	ISZSSWCA	ISZSXWCA	Invoke Setup to set/load/change window config	SETUP	Under SETUP
XACTBAR		ISZSXWIN	Action bar	N/A	Row 1
XPFNEW		ISZSXWCA	PFkeys (toggle)	DETAIL	Last row
UNHIDE		ISZSXHID	Enter password to unhide if window already hidden	HIDE	Under HIDE

Diagram for CUA-type window script

To assist in understanding how the various window scripts interact, the following diagram shows the connections between members and scripts. The diagram is for CUA-type windows.

In the diagram, the member name appears in the top line of each box. The scripts defined within the box appear in uppercase. Calls by the script are listed below in lowercase.

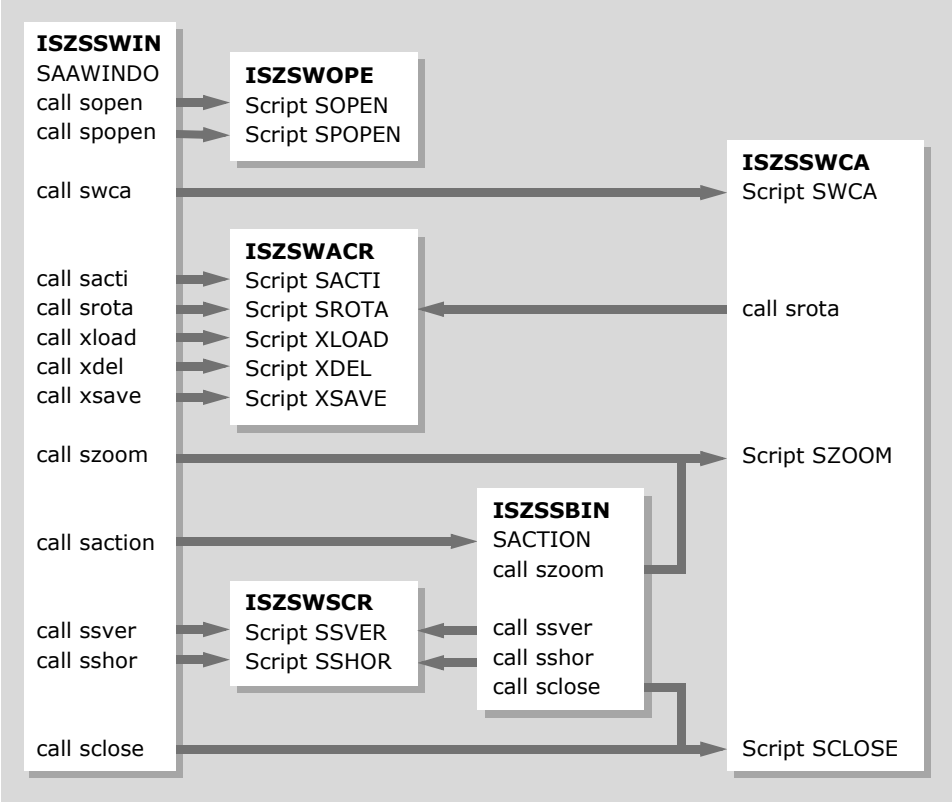
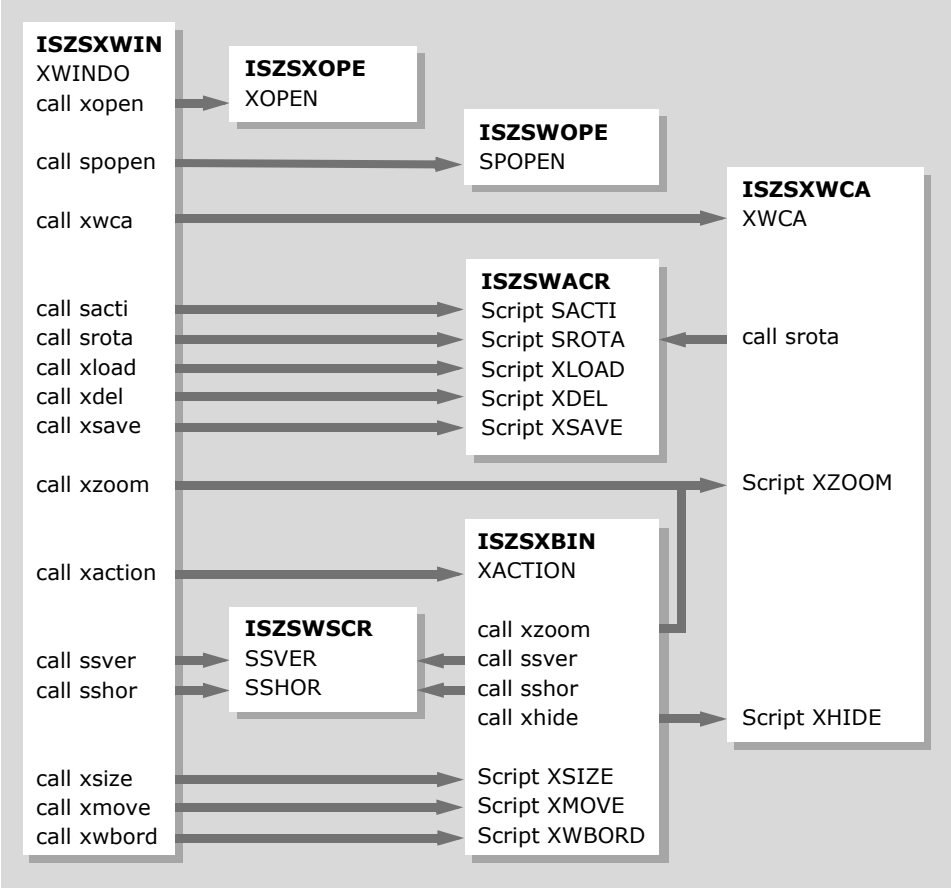


Diagram for PC-type window script

The following diagram shows the connections between members and scripts for PC-type windows.

In the diagram, the member name appears in the top line of the box. The scripts defined within the box appear in uppercase. Calls by the script are listed below in lowercase.



Windows checksums

The Windows scripts supplied on the product tape contain the statements:

ISZCMD WINCH1	in the SAAWINDO script
ISZCMD WINCH2	in the XWINDO script

These cause Session Manager to keep an internal checksum for the script. This enables your local support personnel to decide whether a script has been modified from the supplied form. *These statements should not be deleted.*

Windows script auditing

Member ISZSWIN contains the script 'wrcmsg' which can be used to track what the window script is doing. In any place in a script where you want a message to appear, displaying variables etc., code the following;

```
If t_rc <> 0 or gcwm <> ' '
    Let ucwm = 'your variables' Call wrcmsg
End
```

A message is displayed if a window verb has returned a non-zero return code (t_rc), or if the variable GCWM contains blank.

To set the variable GCWM, use the FLASH command as follows:

```
FLASH 'ON' variable GCWM
```

To reset the variable GCWM and stop the script writing messages, use the FLASH command as follows:

```
FLASH ' ' variable GCWM
```

A number of these calls are already contained in the supplied members.

Window script variables

The following list provides an explanation of the local and user variables that are used in the supplied Windows scripts.

Variable	Description
uc_wscript	Window script name - SAAWINDO or XWINDO.
ucsamount	Border input area for the active window.
ucbord	Top line of border for the active window - window name and error information.
uc_no	Hide function - flag to indicate user pressed PF3 from panel.
uclasttime	Time when last broadcast/message was received.
uc_swit	Last command that caused a switch, DLOG, QUERY, etc.
uc_bmsg	Text of a broadcast/message received. (subscriptable)
uc_bmt	Details of a broadcast/message received. (subscriptable)
ucprof	Override profile, used in SETUP panel and ISZSWSET.
ucuser	Override user, used in SETUP panel and ISZSWSET.
uc_setup	Identifies type of call, used in SETUP panel and ISZSWSET.
uc_winload	Details of the window to load, returned by the exit after a load.
unmaxwid	Max Window id created by this script (prevent loops to 255).
unoldwin	Window id of active window when scroll offset window requested.
un_bno	Number of broadcasts received in windows.

Variable	Description
unvnum	Vertical scroll amount requested.
unhnum	Horizontal scroll amount requested.
unnum	Session detail number to open a window requested or indicator from UNHIDE panel.
un_term_depth	Terminal depth.
un_term_width	Terminal width.
unorg	Origin of window for session to be opened.
unxsiz	Depth of window for session to be opened.
unysiz	Width of window for session to be opened.
lcopen	Window escape to open a new window.
lcclose	Window escape to close the active window.
lcaction	Window escape to perform various actions from border input.
lczoom	Window escape to zoom the active window.
lcactiva	Window escape to activate a different window.
lcrotate	Window escape to rotate to the next window in sequence.
lcwbord	Window escape to switch borders on and off.
lcmove	Window escape to move the active window.
lcresize	Window escape to resize the active window.
lccmd	Internal command - displayed in top border of active window. (subscriptable)
lcwm	Message used for auditing.
lc1	Temporary character variable.
lnactcol	Column where the top line starts.
lnintrup	Window id of interrupted zoomed window due to a broadcast.
lnhelp	Window id of help window.
lndetail	To indicate if the help lines are displayed or not; 2=no, 3=yes.
ln_hidew	Window id of hidden window.
lc_pass	Password field in the unhide window.
lcvnum	Vertical scroll request (+/-) in scroll offset window.
lchnum	Horizontal scroll request (+/-) in scroll offset window.
lcbord	Used in panel SSCRLIND to generate scrolling indicators.

Window script messages

Message	Text and description
MSG1	`Scroll amount of 0 specified` A scrolling action has been requested either using border input or the OFFSET window action bar. The scroll amount specified was zero; no scroll action is performed.
MSG2	`Scroll accepted` A requested scrolling action has completed successfully.
MSG3	`Scroll amount exceeds max` The scroll value entered exceeds the maximum required to display the last row or column of the data in the window. The scroll value actioned is the maximum value required to display this last row or column.
MSG4	`No action taken` An escape sequence has been entered to process input in the border but no input could be found. No action is taken. Ensure there is valid input in the border and reissue the escape sequence.
MSG5	`Invalid action for` *`window-name`* An escape sequence has been entered to process input in the border of the window identified in the message, but the input found is not valid. Correct the input and reissue the escape sequence.
MSG6	`Help already in another window` A help request has been made but a help window already exists. The help request is not actioned; only one help window may be open at any one time.
MSG7	`Zoom failed - incompatible session` A Zoom request has been issued, but the terminal model screen size is smaller than the screen size in use for the session. For example, if the session under windows is running as a model 5 and the actual terminal screen size is a model 2, the session cannot be zoomed.
MSG8	`Zoom invalid - window hidden` A Zoom request has been issued for a window, but the window is hidden. Zoom requests are not valid on a hidden window. This message is issued by the XWINDO script only.
MSG9	`Cursor position incorrect for open` An attempt has been made to open another window but the cursor was not on the border of the active window. Position the cursor on the border and reissue the request. This message is issued by the SAAWINDO script only.
MSG10	`Minimum width must be >= 6 for open` An attempt has been made to open a window with a width of less than 6 columns. The minimum requirement to open a window is 6 columns. This message is issued by the XWINDO script only.

Message	Text and description
MSG11	`Minimum depth must be >= 3 for open` An attempt has been made to open a window with a depth of less than 3 rows. The minimum requirement to open a window is 3 rows. This message is issued by the XWINDO script only.
MSG12	`Move right was too far - max set` In the XWINDO script, if a move request is made, the window size remains the same. The move request that has been issued would prevent all columns of the window being displayed. The move request that is actioned is the maximum value required to ensure the data displayed in the window remains the same.
MSG13	`Move down and right was too far - max set` In the XWINDO script, if a move request is made, the window size remains the same. The move request that has been issued would prevent all rows and columns of the window being displayed. The move request that is actioned is the maximum value required to ensure the data displayed in the window remains the same.
MSG14	`Move down was too far - max set` In the XWINDO script, if a move request is made the window size remains the same. The move request that has been issued would prevent all rows of the window being displayed. The move request that is actioned is the maximum value required to ensure the data displayed in the window remains the same.
MSG16	`Only 10 windows can be saved` There are currently more than 10 application or openlist windows on the screen. Close windows until only 10 are open, then save the configuration again using the SETUP key.
MSG17	`Save failed - no record for profile` *`profile`* `model` *`model`* An UPDATE request was entered by a user, but no record exists for that user with that profile and model. An ADD request is required. If model is 'P', the record that is searched for would contain PF Keys only.
MSG18	`Delete failed - no record for profile` *`profile`* `model` *`model`* A DEL or DELP request was entered by a user, but no record exists for that user with that profile and model. An ADD request is required. If model is 'P', the record that is searched for would contain PF Keys only.
MSG19	`Error reading VSAM file - FDBK=`*`code(dec)`* An error was returned by VSAM while accessing the file.
MSG20	`Error opening VSAM file - ERROR=`*`code(dec)`* An error was returned by VSAM while opening or closing the file.

Message	Text and description
MSG21	`Empty file - ADD first` A load/update request was received, but the file is empty. ADD an entry before proceeding
MSG22	`Load failed - Profile` *`profile`* `Model` *`model`* A load request was received but no record exists for the key above. Additionally, no default could be found for the key with user SYSADMIN, profile *`profile`* and model *`model`* or key with user SYSADMIN, profile *`userprofile`* and model *`usermodel`*.
MSG23	`Load failed - No default for` *`model`* A load default request was received but no record existed for the key with user SYSADMIN, profile *`profile`* model *`model`* or key with user SYSADMIN, profile *`userprofile`* model *`model`*.
MSG24	`Windows exit not loaded` A request was received which required access to the VSAM file, but could not be actioned as the Windows exit ISZE09WN is not loaded (as ISZEXT09). Add EXIT ISZEXT09 to the OPTION statement, or issue the UPDATE EXIT ISZEXT09 command.
MSG25	`Save failed - no record for user` *`userid`* `profile` *`profile`* `model` *`model`* An UPDATE request was entered but no record exists for the specified user with that profile and model. An ADD request is required. If model is 'P', the record that is searched for would contain PF Keys only.
MSG26	`Delete failed - no record for user` *`userid`* `profile` *`profile`* `model` *`model`* A DEL or DELP request was entered but no record exists for the specified user with that profile and model. If model is 'P', the record that is searched for would contain PF Keys only.

Command scripts

Command scripts enable a script to be run in place of the corresponding Session Manager defined command or to support a user-defined command. To run a script in place of a Session Manager command, specify the CMDSCRIPT parameter on the COMMAND statement:

```
COMMAND cmdname CMDSCRIPT Yes | No | ON | OFF [SNAME name].
```

If a script is specified but no name is provided, Session Manager attempts to run a script with the same name as the command. If a *name* is provided, it is this script that is run in place of the command. For example:

- `COMMAND logoff CMDSCRIPT Yes`
  means that a script called 'logoff' is run instead of the command being actioned.
- `COMMAND logoff CMDSCRIPT Yes SNAME fred`
  means that the script 'fred' is run instead of the command being actioned.

If CMDSCRIPT YES is specified and the script has not been defined during configuration load or update, an error message is issued and the command is actioned in the normal way. If CMDSCRIPT NO is specified and the command had previously had CMDSCRIPT YES specified, the script is disabled and the command is actioned in the normal way. The AUTH parameter of the COMMAND statement may be specified, but any KEYWORD parameters are ignored if CMDSCRIPT YES is specified.

Any command name can be used as a script; it is not restricted to Session Manager defined commands. Using this facility a user can effectively define extra user-defined commands. For example,

```
COMMAND bill CMDSCRIPT Yes
```

runs a script called 'bill'.

The entered command's parameters can be accessed using the t_scparms variable. See 'Command script examples' on page 47 later in this section.

Valid CMDSCRIPT parameters

The following SCRIPT parameters may be used in Command scripts; all others are ignored.

- All TPSL verbs
- AUDITMSG
- CALLEXIT
- CALL
- ISZCMD

Usage notes

1. The command name, *cmd-name* can be up to 16 characters in length. However, a script name can only be 8 characters in length. Thus, if a command name, for example, BROADCAST, is specified with CMDSCRIPT YES, but no SNAME operand, a script called 'broadcas' would be searched for in the configuration file.

2. Command abbreviations, or synonyms, can be specified for user-defined commands. To do this define the required abbreviations on separate COMMAND statements. For example,

   ```
   COMMAND operator CMDSCRIPT Yes SNAME bill
   COMMAND oper CMDSCRIPT Yes SNAME bill
   ```

3. If CMDSCRIPT YES is defined for a command and the script contains a ISZCMD for a command which has CMDSCRIPT specified, the script is not run when the ISZCMD is actioned. This situation is detected by Session Manager and the command itself is always invoked, thus preventing a loop. For example,

   ```
   COMMAND logoff CMDSCRIPT YES
   SCRIPT LOGOFF
   ...
   ...
   ISZCMD logoff                /* executed using           */
                                /* Session Manager logic    */
   ...
   ```

   If a command script attempts to execute a ISZCMD for a user-defined command. the script associated with the command will *not* be run; use CALL instead.

4. In Session Manager 1.3.05 and above, if the command definition doesn't specify the full command name it will not be regarded as a replacement for the command and therefore as defining a new command.

   For example:

   ```
   COMMAND h auth 1 cmdscript y sname newcmdh
   ```

   defines a command named 'h' but since it doesn't define the full name of the HELP command it will not replace it. So if the user enters 'h' in the command line the new command will be invoked. If the user enters 'he', or 'hel', or 'help' then the HELP command will be invoked. Prior to this new command definition the user could enter 'h' and invoke the HELP command.

5. If a CALL is specified which is recursive, that is, the script calls itself, message 580 is issued to the terminal (in the case of a command script) and message 228 is output to the Audit file. However, if the variable t_recur_script is set to 'Y' then recurrence is allowed, up to a maximum number of times governed by the DOMAX value. If message 580 is issued under any other circumstances, please contact your local support representative.

Command script examples

Example 1

```
COMMAND broadcast CMDSCRIPT Yes    /* Session Manager command  */
SCRIPT broadcas                    /* > 8 chars so 1st 8 used  */
*                                  /* t_scparms contains the   */
*                                  /* complete command as typed*/
Let un_words = words t_scparms     /* How many words in command*/
Let lc_text = word t_scparms 2     /* 2nd word = message text  */
Let lc_dest = word t_scparms 3     /* 3rd word = message dest  */
Callexit rsn 'BRO'                 /* Call User exit E09 to    */
*                                  /* validate command e.g.,   */
*                                  /* check RACF authority     */
If t_rc = 0                        /* If exit said ok          */
 If t_termid = m'S06*'             /* If term name starts S06  */
       And lc_dest = 'ALL'         /* and it a BRO ALL         */
       ISZCMD 'BRO &lc_text& lu S06*'/* make it a BRO to terms */
*                                  /*   starting S06 only      */
   Else                            /*  Else                    */
       ISZCMD '&t_scparms&'        /*   Send the bro as entered*/
   End                             /*  End                     */
End                                /* End                      */
```

Example 2

```
COMMAND language AUTH 7 CMDSCRIPT Yes /* User command LANGUAGE */
*
SCRIPT language
Auditmsg 'command language executing'
If t_lang <> 'FR'                  /* If current language is   */
*                                  /* not French               */
   Let t_lang = 'FR'               /*   switch to French       */
Else                               /* Else                     */
   Let t_lang = 'DE'               /*   switch to German       */
End                                /* End                      */
```

Example 3

```
COMMAND operator CMDSCRIPT on SNAME bill /* User command OPERATOR
COMMAND oper CMDSCRIPT on SNAME bill   /* Command abbreviation */
*
SCRIPT bill                          /*OPERATOR used script BILL*/
Auditmsg 'command operator executing'
Let lnwords = words t_scparms       /* how many words entered */
Let sub1 = 2                        /* Start at the 2nd word  */
Let lnwords = lnwords - 1           /* go to the last word    */
Let lc_text = ' '
Do for lnwords
   Let lc_bit  = word t_scparms sub1 /* Get next word          */
```

```
   Let lc_text = lc_text ' ' lc_bit  /* add it onto the message*/
   Let sub1 = sub1 + 1
End
Auditmsg  lc_text                    /* Send the message       */

* So if a user enters the command Operator and the text
* 'Please mount tape on 581' at the Menu, the command script
* extracts the data and message 226 will appear on the log (if
* LOG Yes is specified - the default) with the following message
* text 'operator Please mount tape on 581
```

CHAPTER 2

Panels: definition and usage

This chapter provides information on the Session Manager Panel facility.

Such displays as the Signon screen and the Menu screen are panels and are defined by the PANEL statement in the configuration file. This means that all panels can be tailored.

Each panel can also be given logic capabilities by using the Panel and Script Language (TPSL). This is also described in this manual on page 145.

Overview

It is likely that each Installation using Session Manager has unique requirements for the layout of the various panels, especially the Menu panel. Session Manager enables each required panel format to be defined using the PANEL statement. The statement syntax and definition is given later in this manual. This chapter of the manual is intended to give some additional usage information which may be of assistance to all users taking advantage of the Session Manager panel definition facility.

There are two methods by which a panel may be defined, either field by field with column numbers and attribute values being supplied for each field, or by using the TEXTSTART and TEXTEND parameters to delimit the logical records which are used to build the panel. In this latter method, each record corresponds to a line on the screen and field attributes are denoted by special characters with predefined meanings. It is acceptable to use a combination of both methods to define any panel. There is no limit to the number of lines defined for each panel, since Session Manager provides scrolling commands to move the display backwards and forwards.

If using the TEXTSTART and TEXTEND method, each field must be preceded by a control character, called an attribute character, which determines the field type, its colour and intensity. Three intensities are available, normal, high, or 'dark' intensity for an undisplayed field. Defining a panel by this means restricts the width to 80 characters. To define a panel of up to 132 characters, the field by field method must be employed.

Each panel may use TPSL, the Panel and Script Language to control the output screen. Session Manager provides a vast array of variables and TPSL may be used to inspect and update some of the variables. This means that panel displays may be varied according to certain conditions. For example, the status of sessions may alter the panel layout, or the preferred language of a user may be used to control the language in which the panel contents are displayed. Furthermore, command input may be intercepted and converted prior to being passed to Session Manager. This enables users to enter commands in their own language, or to use a preferred form of a command.

The main differences between the two methods of panel definition are:

- With the TEXTSTART/TEXTEND method, the format of the output panel can be easily recognized from the configuration records, whereas with the field by field method this is not the case.
- The TEXTSTART/TEXTEND method is restricted to 80 bytes in width, whereas the field by field method allows up to 132 bytes to be defined for each line.
- TPSL statements can only control a line, or number of lines, if using the TEXTSTART/TEXTEND method. With the field by field method, TPSL statements can control the position of fields, in addition to suppressing fields and inspecting and assigning values.
- Due to the need for field attribute characters to be predefined when using the TEXTSTART/TEXTEND method, errors may easily occur by assigning the wrong attribute characters to fields. With the field by field method, each field definition can be given a meaningful attribute name, although an attribute character optionally may be assigned.

- Depending on the panel layout, it may be more concise to use field by field definitions, since they can be generated within a loop.

Each panel that is defined must be given a panel name. This panel name can then be used on the appropriate Session Manager parameter, for example the SIGNONPANEL, DEFMENU and MENU parameters. Certain panels with specific names are supplied with the product. These are used by Session Manager as defaults if the Installation does not supply their own panel name. The supplied panels should always be available to Session Manager. The names are: SESS1, SESS2, BCAST, DATA, IDLEW, GFSSTATS, GFSUSE, GFSSTOR. These panel definitions may be modified by the Installation to meet their requirements, but the names should not be changed. The panel names used for the Signon screen and the Menu panel can be changed by the Installation on the SYSTEM, PROFILE, USER, and TERMINAL statements. The supplied panels are SIGNON and MENU. If not overridden, these should also be available to Session Manager.

- In addition to the panels outlined above, several of the Session Manager features and facilities also require set panels to be available. These are:
- The CUT and PASTE panels are required for the Cut-Paste feature.
- The DEMO panel is required for the Demonstration and View feature.
- The SPYTELL panel is required for the Spy facility.
- The PULL panel is required for the Push-Pull facility.
- The REPLAY panel is required for the Record-Replay facility.

A more detailed explanation of different field types and the way in which attributes are used appears in later sections.

Panel sections

Session Manager panel definitions consist of the following optional sections:

- Define
- Header
- Content
- Trailer
- Process

The Define section, if specified, must appear first. It is used to define options which apply to the whole panel, such as the width of the panel (80 or 132), whether normal or alternate screen size is to be used, the name of the field that the cursor is to be positioned under, and the maximum number of times that any TPSL loop defined for the panel may be executed.

Each panel display may be optionally divided into a maximum of three sections. These are:

- The Header section is a fixed portion at the top of the display.
- The Trailer section is a fixed portion at the bottom of the display.
- The Content section appears between the Header and Trailer sections. It may have a greater number of lines than the lines available between the last line of the header, and the first line of the trailer. The Content section can be moved backward and forward using the Session Manager scrolling commands.

As stated above, the Header and Trailer sections are fixed in position. It is generally advisable to define at least one of these for each panel, to ensure that the command area is defined in a fixed portion. The cursor should also be positioned initially on a field in a fixed portion, since if it is defined in the Content section, it may not appear on the display where intended when the number of content lines is greater than the lines available.

It is not necessary to define a Content section; a panel definition may consist of just a Header and/or a Trailer section. For example, a Help panel definition may consist solely of a Header section.

The Process section of a panel definition contains TPSL statements which are executed when input is received. This is used to check which attention key or PF key was pressed, or to inspect the command that was entered. This can then be used to validate or control the command that is passed to Session Manager. For example:

```
PROCESS
    If t_command = 'grab' or 'acq' or 'flip'
        Let t_command = 'transfer'
End
```

Thus, synonyms for the TRANSFER command may be entered on this panel. This can prove particularly useful in Installations where users prefer to enter commands in their own language.

Panel sub-definitions

The Header, Trailer, Content, and Process sections of a panel can specify a CALL parameter. This enables a sub-definition to be included and allows sub-definitions to be shared by several panels. The sub-definitions are defined by the following statements:

- PHEADER
- PCONTENT
- PTRAILER
- PPROCESS

A sub-definition can only be included in a corresponding definition, for example, a PHEADER can only be called from a HEADER or another PHEADER. Refer to page 55 for details on common panel definitions.

Basic field types

There are three basic field types, input, output, and literal. These are all defined by field attribute names, and are described in the following sections.

Input fields

Input fields are variables into which the user can enter data. These fields are recognized by having been assigned an attribute of 'IN'. Examples of input fields are the signon id and password. These would normally be entered by a user at the Signon screen. Input fields should be unprotected and are left justified for both numeric and character variables, with any truncation occurring on the right.

Output fields

Output fields are variables which are intended for output, and to which Session Manager ascribes values. These fields may be either Session Manager supplied variables or may be user-definable variables. Session Manager writes the contents of the variable referred to by the name whenever it detects such a field in a screen definition. These fields are recognized by having been assigned an attribute of 'OUT'. Output fields should be protected and numeric fields are right justified; truncation occurs on the left.

For example, to build a useful Menu screen it is necessary to display – for each application that may be selected – a brief description of the application, any data that is passed to the application, and the escape command sequences.

There are several variables which may be used to build a screen layout. These include the date, the time, the signon userid and the luname of the terminal. Session Manager has many such fields which are fully documented in 'Session Manager variables' on page 197.

Literals

Literals are simply character strings which Session Manager displays on the output screen exactly as they are defined on the input records or field definitions. These would therefore be used mostly for headings and for Help screens, and should be protected fields, so that they may not be altered on the screen display. These fields are recognized by having been assigned an attribute of 'LIT', or they may have been allowed to default to LIT, i.e., neither 'OUT' nor 'IN' are specified for the field.

Common panel definitions

Common panel definitions can be defined by using the PHEADER, PCONTENT, PPROCESS and PTRAILER statements.

The statements are defined in exactly the same way as the normal panel definition statements, the only difference being that these definitions can be used by other panels.

The definitions are used when the CALL statement processes the panel. The panel definition that is called is known as a *sub-panel definition*. The following rules apply when using sub-panel definitions:

- Any ATTR symbol used in the sub-panel must be defined in the same sub-panel definition.
- The LINES parameter on the HEADER and TRAILER *cannot* be specified on the PHEADER or PTRAILER statements.
- If the cursor is to be positioned in any of the fields specified, in PHEADER, PCONTENT, or PTRAILER, then the CURSOR parameter must be defined in the sub-panel definition. The cursor can be positioned as normal in a sub-panel definition, by specifying the ICURSOR parameter.
- Any modifications to a sub-panel definition are available immediately. PANEL statement updates are only available when the panel is next displayed.

For example, if a menu panel is displayed and the menu definition is then updated, the original menu definition remains in effect until the menu function is complete; that is, a session is selected and becomes the visible session. This is not true of any called sub-panel definitions. When these are updated, the new versions are used immediately.

Common panel example

The example below shows the use of common panel sub-definitions. The sub-panel definitions in this example define common header, content, trailer, and process sections.

```
PANEL fred
  Header lines 10
    field ...
    call comhead     /* display sub-panel definition
    field ...
  Content
    field ...
    call comcont     /* display sub-panel content
    field ...
  Trailer lines 10
    field ...
    call comtrail    /* display common trailer definition
    field ...
  Process
    If t_aid = pf3
      call compf3    /* action common panel process
    End

PHEADER comhead      /* a common panel header
  field ...
  Textstart
    ...
  Textend
  field ...

PCONTENT comcont     /* common panel content
  field ...
  field ...

PTRAILER comtrail    /* common panel trailer
  field ...
  Textstart
    ...
  Textend
  field ...

PPROCESS compf3      /* common processing
  Let t_command = 'logoff'
```

Field attributes

Attributes for each field may be defined in several ways, although the particular method chosen is partly dependent on the method used for the field definition. This is described more fully in 'Sample field definitions' on page 60.

When the TEXTSTART/TEXTEND method is used to define a screen layout, field attribute names must be associated with a set of special attribute characters. One of these characters must be placed before each field in the panel layout in order to associate those attributes with the field. Each attribute character must be followed by a field or a space; two or more immediately adjacent attribute characters will cause unpredictable results.

When the field by field method is used, attributes may be associated with each field by following the field name with a list of attributes. Alternatively, the REFATTR parameter may be specified following the field name to refer to a field attribute character, which in turn can be a variable assigned by the TPSL LET statement or an attribute definition.

Acceptable field attribute names are:

For protection and content

```
[ASKIP | SKIP]
[DETect | NODETect]
[UNProtected | PROTected]
[NUMeric | ALPHanumeric]
[NODisp | NONdisp]
```

For colour and effects

```
[RED | YELlow | WHIte | GREen | BLUe | PINk | TURquoise]
[HIgh | NORMal]
[REVerse | BLInking | UNDErscore | USCOre]
```

Special attributes

```
[ETMODE | DBCS]
[ICursor]
[BLANK | NOBLANK]
[IN | OUT | LITeral]
```

Attribute characters

Attribute characters may be defined as a system-wide default, by specifying them on the ATTR parameter of the SYSTEM statement. They may also be specified for individual screens by using the same parameter on the PANEL statement. The table below gives the default settings for the attribute characters if they are specified on neither of these statements:

Character	Default setting
#	UNPROT,NORM,IN
%	UNPROT,HIGH,IN
"	UNPROT,NOND,IN
?	PROT,NORM,OUT
&	PROT,HIGH,OUT
<	PROT,NORM,LIT
@	PROT,HIGH,LIT

(Note that for output and literal fields there is no default attribute character for nondisplay). Therefore, to define a heading on a screen, the following could be coded within TEXTSTART/TEXTEND:

```
@S E S S I O N  M A N A G E R
```

which would cause the heading '`S E S S I O N  M A N A G E R`' to appear on the terminal in high intensity because the '@' is the attribute for high intensity literal display. An example of normal intensity output fields would be:

```
?t_date   ?t_time
```

These two fields are Session Manager special variables which cause the current date and time to be displayed at the terminal. The final example is of an 'Enter Command' line for a Menu screen. This is a combination of a literal and an input area into which the command may be keyed:

```
<Enter command@===>%t_command     <
```

In this case, the text 'Enter command' is defined as a normal intensity literal. The '`===>`' is a high intensity literal, and the input command area is unprotected and also high intensity. Therefore, the text of any command keyed into this field would be displayed in high intensity. Note that the minimum length of the input area is determined by the length of the special variable name – t_command in this instance. In this particular example, the input area has been extended by appending several spaces to the variable name and placing a normal literal field attribute (which is by definition protected) where the input field actually stops.

Since it is the attribute character which appears in the panel definition prior to each field that determines the field type, it is important to select the correct attribute when defining fields. For example, if the (normally) output field 't_date' is preceded by an input attribute character such as '#'', Session Manager would display the current date on the screen, but it would be unprotected and therefore the user could overkey it.

Suppose a user has a Menu screen with the default Escape Forward, Escape Backward and Escape command sequences displayed using the provided t_esc, t_bwd, t_fwd special variables. By preceding these fields with an input attribute, the user may then overkey the escape command sequences and thus set them to any value required. If, for example, the user is aware that none of the applications use the PF24 key, then the Escape command could be set to PF24 on the display simply by overkeying the default value of '@@'. The escape command line on the display might be defined like this:

```
<ESCAPE = #t_esc   <  FORWARD = #t_fwd   <  BACKWARD = #t_bwd   <
```

Additional spaces have been appended to the 'ESCAPE' and 'ESCAPE FORWARD' fields to allow the full eight-character escape commands to be entered.

Attributes definitions and attribute references

With the field-by-field method of defining panel layouts, attributes may be specified after each field name. For example:

```
field t_message(75) prot,hi
```

provides a field for a message to be displayed. The message will be in high intensity, protected, and will be truncated to 75 characters.

Alternatively, each field may be associated with a predefined set of attributes by using the REFATTR parameter to relate to a special character, for example:

```
ATTR '!' PROT,HI
Field t_message(75) REFATTR '!'
```

would have the same effect as in the first example.

Another example of the use of REFATTR is:

```
FIELD    'S E S S I O N  M A N A G E R' REFATTR '@'
```

causes the heading 'S E S S I O N M A N A G E R' to appear on the terminal in high intensity because the '@' is the default special character attribute for high intensity literal display.

Note If REFATTR is used, the special character must be enclosed with single delimiters. The REFATTR parameter can also reference an attribute held in a variable. The variable is assigned using the TPSL LET statement. See 'The LET statement' on page 157 for further details.

Sample field definitions

As mentioned previously, fields may be defined using the TEXTSTART/TEXTEND method, or by the field by field method. For example, assuming a set of field attribute definitions as follows:

```
ATTR '!' IN,UNPROT,NORM
ATTR '"' IN,UNPROT,HIGH
ATTR '£' IN,UNPROT,NONDISP
ATTR ':' OUT,PROT,NORM
ATTR '%' OUT,UNPROT,HIGH
ATTR '(' PROT,NORM
ATTR ')' PROT,HIGH
```

The header section for a menu panel may be defined using the TEXTSTART/TEXTEND method as follows:

```
-----------------------------------------------------------------------------

HEADER lines 5
TEXTSTART
:t_pgmname           )        WIDGETS INC       :t_date     :t_time
(USER=%t_user     (LU=%t_termid    (
%gcflash

(KEY   IND DESCRIPTION    *-*  MENU  *-*      APPLID  AVAIL
(---   --- -----------                        ------  -----
TEXTEND

-----------------------------------------------------------------------------
```

The definition of the same header section using the field by field method might look like this:

```
HEADER lines 5
         Field t_pgmname                        out,prot,norm
  Col 31 Field 'WIDGETS INC'                        prot,high
  Col 50 Field t_date                           out,prot,norm
  Col 62 Field t_time                           out,prot,norm
  Col  1 Field 'USER='                              prot,norm
         Field t_user(8)                            refattr '%'
  Col 19 Field 'LU='                                refattr '('
  Col 23 Field t_termid(8)                          refattr '%'
  Col  1 Field gcflash                          out,prot,high
  Col  1 Field 'KEY   IND DESCRIPTION    *-*  MENU  *-*' refattr
'('
  Col 46 Field 'APPLID  AVAIL '                     refattr '('
  Col  1 Field '---   --- -----------'              refattr '('
    Col 46 Field '------  ----- '                     refattr
'('
```

DBCS support for field definition

If your terminal supports the double byte character set, fields may be defined to accept these character sets using the ETMODE and DBCS attribute settings.

Setting	Description
DBCS	An entire field can be set to accept double byte character mode using the DBCS attribute setting. This sets the attribute 43 with F8 added to enable Kanji.
ETMODE	This attribute can be used to specify that shift in (SI)/shift out (SO) orders can appear within the field. SO=x'0e' and SI=x'0f' are set to go in and out of double byte character mode. If ETMODE is specified on an input field then the attribute FE is added to the attributes with 01 specified, which means SO/SI creation is enabled.

Panel variables

Session Manager supplies a great number of variables which may be used when building panels. In addition to the Session Manager supplied variables, users may define their own variables. There is no limit to the number of user definable variables. They must conform to certain naming conventions and must have a value assigned in order to appear on a screen. Values may be assigned using the TPSL LET statement, by the FLASH command, by using the User exit, or by input from a panel. A full description of each Session Manager supplied variable is given in 'Session Manager variables' on page 197.

Some of these variables are internally generated by Session Manager, and some are taken from configuration statement parameters. If not defined in the configuration, the default value is used. Some of the variables may be modified.

Certain variables are associated with a session, which is the case for several variables appearing in Menu screens. These variables are prefixed by 's_' For these, the character string, e.g., 's_appl' may be followed by a one to three digit number which denotes the session subscript number to which it relates. So, 's_appl.12' is the application defined as the twelfth session, which need not necessarily be associated with session detail number 12, or PF key 12. 9999 fields of this type may appear in a single panel definition, since up to 9999 sessions may be defined for a Menu screen.

The length of each Session Manager variable is given in the appropriate description in 'Session Manager variables' on page 197. The lengths of user variables are as defined (maximum of 256 characters for character variables and a maximum of 9 for numeric variables).

For input fields, the length is determined by the position of the next field attribute character. For this reason, it is important to use the COL parameter in field by field panel definitions. For TEXTSTART/TEXTEND definitions, the input field is often terminated with a 'protect' (that is, output or literal) attribute character. For example,

```
#T_COMMAND              <
```

where the literal attribute character '<' acts as the input area terminator. Any attribute will terminate an input area.

The majority of these variables are intended for building a Menu screen. The t_npass and t_nprof variables are the only two specifically intended for the Signon screen only. Some variables, such as the date and time, may be used in any Session Manager panel. Where any restriction on the use of a variable exists, it is indicated in the description for that variable. Those variables to which '.*nnn*' is appended are session, or window, associated and '*nnn*' can therefore be set from '1' to '999'.

Initializing substitutable variables in scrollable panels

Execution of the Contents, Header, and Trailer sections stops when that section of the screen has been filled, so the usual way to initialize variables in the Contents, Header, or Trailer section of a potentially long panel is to construct a DO loop. An example might be:

```
Content
Let sub1 = 0
Do while sub1 < smax
   Let sub1 = sub1 + 1
   Let uc_logd.sub1 = s_logd.sub1(3,8)
   Nline
   Field uc_logd.sub1
End
```

This loop equates the user variable to the logon data value at position 3 for a length of 8. The loop is halted either when smax is reached, or when there are no more lines to fill on the screen.

This should not cause a problem unless the same variables are used in the Process section. This is because the Process section is executed in its entirety each time an input key is pressed. So, if a Process section contained the following statements:

```
Process
Let sub1 = 0
Do while sub1 < smax
   Let sub1 = sub1 + 1
   Let s_logd.sub1 = 'l' &uc_logd.sub1&
End
```

The session lines that do not appear on the first page of the content do not have their value of uc_logd defined, so their s_logd value is set to 'l' in the process section. uc_logd therefore appears as blank when the 'FWD' command is issued in the content section.

The solution is to create another variable to keep count of the number of lines created. The Content section would be:

```
Content
Let sub1 = 0
Do while sub1 < smax
   Let sub1 = sub1 + 1
   Let un_lines = sub1
   Let uc_logd.sub1 = s_logd.sub1(3,8)
   Nline
   Field uc_logd.sub1
End
```

and the Process section would be:

```
Process
Let sub1 = 0
Do while sub1 < un_lines
   Let sub1 = sub1 + 1
   Let s_logd.sub1 = 'l' &uc_logd.sub1&
End
```

Using TPSL with panels

TPSL statements may be used in the Header, Trailer, Content, and Process sections of any panel definition. In fact, the Process section consists entirely of TPSL statements which are executed when an attention key or a PF key is pressed. The TPSL statements are: IF, SELECT, DO, LET, ITERATE, LEAVE, and RETURN. The format of each statement is given in 'The Panel and Script Language (TPSL)' on page 145.

In the following example of a Header section, the IF TPSL statement is used to check if there is any error message. If there is not, then the system flash is displayed on the third line (lca), otherwise the error message is displayed in this area.

```
Header lines 6
If t_message = ' ' then let lca = gcflash
else let lca = t_message
End
Textstart
<t_pgmname    @Hardcopy Option Panel%t_panel    &t_date   &t_time
<
&lca
<
<Hardcopy Options               Format    Route
<
Textend
```

Note In this document, the 0x7C (i.e. x'7C') character is always presented as the @ sign. It may be displayed as a different character in some non-English code pages. You should enter the appropriate 0x7C character symbol for the code page you are using.

TPSL statements are especially useful in the CONTENT section when the same fields need to be defined on several lines, for example, on a Menu panel. The following example shows how TPSL statements can be used to build such a panel. The 's_' variables are Session Manager supplied session variables.

```
Let sub1 = 0
* smax is the number of sessions defined for the user
 Do while sub1 < smax
   Let sub1 = sub1 + 1
*
* s_a is the numeric status of each session
* s_n is the session detail number
* s_desc is the description of each session
* The SELECT statement sets different field attributes
* for the description field, depending on the status of
* each session.
*
   Select
     When s_a.sub1 <= 20 then     /* unavailable or blocked */
                           Field     s_n.sub1(3)  prot
                           Field  s_desc.sub1(20) unprot,in
     When s_a.sub1 <  20 then     /* command session        */
                           Field     s_n.sub1(3)  prot
```

```
                            Field  s_desc.sub1(20) prot
     Else                          /* Active                 */
                            Field     s_n.sub1(3)  prot,high
                            Field  s_desc.sub1(20) prot,high
   End
*
* s_aa is the status description for each session
*
   col 27 Field    s_aa.sub1(3)          prot turq
If s_logm.sub1 ¬= ' '
   Nline 2
End
```

The contents portion of the resulting Menu screen might look like the following:

```
   1 Help !              CMD

   3 Display storage     CMD

   4 IVY in prodgen      -A-

   5 VSWITCH             (X)

   6 CICS in PRODGEN     (X)

   9 Free for use        (X)
```

In the Process section, TPSL statements are especially useful for modifying the command that is typed in the command area before it is passed to Session Manager. This enables a user to have various alternatives to the valid Session Manager commands. Another use is to assign commands to PF keys. An example of both these uses is shown next. The example is taken from the Process section of a Help screen main menu, where items are selectable by number.

```
PROCESS
  Select
    When t_command = '1' then Let t_command = 'HELP helpcmd'
    When t_command = '2' then Let t_command = 'HELP sign'
    When t_command = '3' then Let t_command = 'HELP menu'
    When t_command = '4' then Let t_command = 'HELP escape'
    When t_command = '5' then Let t_command = 'HELP command'
    When t_aid = pf2   then Let t_command = 'B 99'
    When t_aid = pf3   then Let t_command = 'QUIT'
    When t_aid = pf4   then Let t_command = 'RETURN'
    When t_aid = pf7   then Let t_command = 'BWD'
    When t_aid = pf8   then Let t_command = 'FWD'
  End
```

Cursor positioning and testing

Session Manager provides several operands and variables to effect control over cursor positioning and testing within panels. This section outlines these various methods:

Cursor positioning

The CURSOR subparameter can be used to specify the variable at which the cursor is to be positioned. Using this method, all occurrences of the field will have a cursor generated, but the cursor appears on the last occurrence on the physical screen.

Field attributes may be defined as cursor control, i.e., the ICURSOR attribute, for any field, but neither the CURSOR sub-parameter or the field attribute takes priority – *it is always the last field with a cursor definition that has the cursor positioned on it.* This applies to subscripted fields as well as multiple definitions of the same field.

The cursor is typically defined to appear in the command area, and this is usually specified in the TRAILER section of the panel. If no cursor position is defined, or for any reason the selected field does not appear on the screen, the cursor is positioned at the top left hand corner (1,1) of the screen.

The t_cursor_name variable can be assigned the name of the field in which the cursor is to be positioned in the HEADER, CONTENT, or TRAILER sections of the panel. For example:

```
Let t_cursor_name = 't_command'
```

Positioning within a field

The position of the cursor within a field can be controlled by the FIELD ICDISP subparameter. It positions the cursor at a specified offset from the start of the field. The offset can be specified as either a literal or as a variable containing the offset to be used.

A value of 1 positions the cursor at the first position of the field. A value of 2 positions the cursor at the second position, and so on. If zero (0) is specified, the default, the cursor is not positioned in the field. The following example illustrates how you could specify the offset from the start of the field t_command at which the cursor should be positioned:

```
If ln_fred = 0 Then
   Let ln_fred = 1
End
Field 'cursor offset ===>'          prot hi
Field ln_fred(4)                    in
Field 'command ===>'                prot hi
Field t_command                     in unprot rev
   ICDISP ln_fred
```

There is no check to ensure that the offset is within the field, so if incorrectly specified, the variable could position the cursor anywhere on the screen. If, however, the variable attempts to position the cursor beyond the screen, then the cursor is set to the bottom right hand corner of the screen.

Cursor testing

There are various variables that can be tested to determine where on the screen the cursor is positioned. These variables can be tested in the PROCESS section.

Variable	Returns
t_cursor_name	The complete name of the field where the cursor was when the panel was displayed. For example, if the cursor is in field uc_field.2, then t_cursor_name = uc_field.2. If the cursor is in field t_command, then t_cursor_name = t_command.
t_csr_row	The row number of the current cursor position.
t_csr_col	The column number of the current cursor position.
t_csr_ofs	The screen offset of the current cursor position.
t_cursfld	The name of the field, without any subscript. For example, if the cursor is in field uc_field.2, then t_cursfld = uc_field.2. If the cursor is in field t_command, then t_cursfld = t_command.
t_curssub	The subscript value of the field name where the cursor is on input. For example, if the cursor is in field uc_field.2, then t_curssub = 2. If the cursor is in field t_command, then t_curssub = 0.

Sample panel layouts

This section contains two sample panel layouts. There are several samples provided in source form on the product tape which may be used as a basis for creating Installation panel designs. The following sample is a simple Session Manager Signon panel.

```
PANEL SIGNON
Define mode norm  width 80  domax 20
*Default ATTR characters are to be used

HEADER lines 24

Textstart
<-------------------------------------------------------------
&t_panel              <IBM Session Manager


<Date &t_date  <
<Time &t_time  <

&t_message
            <Enter   Userid      |===>#t_user  <
                   < Password    |===>"t_pass  <
                   < New Password|===>"t_npass <
                   < New Profile |===>%t_nprof <

  LAST COMMAND@===>&lc_last_cmd
<ENTER COMMAND@===>%t_command
<      PANEL  ==>%t_panel            < PA1 to repeat last command
<
<pf3 - Exit    pf8 - Override    pf10 - Override Force
<-------------------------------------------------------------

Textend

PROCESS

Select
  When t_aid = pf3
    Let t_command = 'QQUIT'
  When t_aid = pf8
    Let t_command = 'TRANSFER'
  When t_aid = pf10
    Let t_command = 'TRANSFER OVERRIDE'
End

If t_command ne ' ' then let lc_last_cmd = t_command
End
```

The next sample is a rather more complex signon panel making full use of different methods of field definition, and of TPSL statements to provide logic for the panel.

```
PANEL SIGNON
Define mode alt   width 80  domax 20
HEADER lines 7
Let t_command = ' '
Attr '%'  unprot,hi,in
Attr '"'  unprot,nodisp,in
Attr '?'  prot,norm,out
Attr '&'  prot,hi,out
Attr '<'  prot,norm
Attr '!'  prot,hi
Attr '#'  unprot,hi,in,ic
Textstart
<-------------------------------------------------------------
&t_panel              <IBM Session Manager
&t_termid
<                            Signon panel
<Date &t_date  <
<Time &t_time  &gcflash
<
Textend
CONTENT
Let sub1 = 0
 Do while sub1 < 9
   Let sub1 = sub1 + 1
   If s_cmd.sub1 = ' ' then
     If s_a.sub1 > 0 then
       col 20 Field  s_desc.sub1(20)                    prot,norm
       If s_a.sub1 = 20 then
           col 42 Field 'Blocked'                       prot,high
       Else
           If s_a.sub1 >= 30 then
                col 42 Field 'Available'                prot,high
           Else
                col 42 Field 'Unavailable'              prot,high
           End
       End
       If s_flash.sub1  = ' ' then
                col 20  Field  s_flash.sub1(58)         prot,high
       End
     Nline
     End
   End
 End
TRAILER lines 10
Textstart
            <Enter   Userid      !===>#t_user  <
                   < Password    !===>"t_pass  <
                   < New Password!===>"t_npass <
                   < New Profile !===>%t_nprof <
<ENTER COMMAND@===>%t_command
< Last command was:&lc_last_cmd
< Last message was:&t_message
<
```

```
<pf1 - HELP pf3 - LOGOFF pf4 - LOGOFF LOGON pf8 - TRANSFER pf9 -
T'FER OVERRIDE
<-------------------------------------------------------------
Textend
PROCESS
Let lc_last_cmd = t_command
Select
  When t_aid = pf1
    Let t_command = 'help'
  When t_aid = pf3
    Let t_command = 'logoff exit'
  When t_aid = pf4
    Let t_command = 'logoff logon'
  When t_aid = pf8
    Let t_command = 'transfer'
  When t_aid = pf9
    Let t_command = 'transfer override'
  End
```

CHAPTER 3

PANEL and SCRIPT statement definition

PANEL and SCRIPT statements differ from the other configuration control statements in that they impart a great deal of flexibility to the product due to their logic capabilities. For this reason they are documented in this manual; descriptions of all other control statements are contained in the *Technical Reference.*

PANEL and SCRIPT statement prototypes

This section shows the PANEL and SCRIPT statements with their parameters. It is intended as a quick reference for those who are already familiar with panel and script definitions. Any parameters surrounded by '[' and ']' are optional.

Most parameters can be abbreviated. The mandatory part of each is shown in upper case. The substitutable subparameters are shown in *italics* when the manual is printed on a laser-printer using the correct fonts. When printed on a line-printer, italics appear as bold lower case.

PANEL and SCRIPT statements may contain logic statements for loops and conditional processing. These logic statements are referred to as TPSL, the Panel and Script Language, and the prototypes for these are shown separately in 'TPSL syntax' on page 78.

PANEL

```
[PANEL panel-name                                    (Full definition: page 82)
   [BRANCH label name ]
      [LAB label name ]
   [DEFINE
      [LANGuage languageid]
      [WIDTH 80 | 132]
      [MODE NORMal | ALTernate]
      [DOMAX domax-limit]
      [CURSOR field-name]
      attr-definitions [...]
```

*attr-definition*s (one or more, as desired):

```
      [ATTR attr-symbol-n OFF | field-attribute-parameters]
```

field-attribute-parameters:

Protection, content:

```
      [ASKIP | SKIP]
      [DETect | NODETect]
      [UNProtected | PROTected]
      [NUMeric | ALPHanumeric]
      [NODisp | NONdisp]
```

Colour, effects:

```
      [RED | YELlow | WHIte | GREEN | BLUe | PINk | TURquoise]
      [HIgh | NORMal]
      [REVerse | BLInking | UNDErscore | USCOre]
```

Special attributes:

```
      [ETMODE | DBCS]
      [ICursor]
      [BLANK | NOBLANK]
      [IN | OUT | LITeral]
   [HEADER                                           (Full definition: page 87)
      [LINES line-number attr-definitions] ]
```

attribute-definitions (one or more, as required).
Note that these may not be defined within TPSL structures:

[ATTR *attribute-character* OFF | *field-attribute-parameters*]

For *field-attributes*, see PANEL DEFINE.

The following PANEL operations may be embedded within TPSL structures and interspersed with TPSL LETs and RETURNs.

```
[CALL panel-sub-definition]
[COLumn col]
[FIELD variable[(nn)] | literal
   [field-attributes |
   REFattr attribute-character] |
   character-variable]
[FORMATMSG (nnnn 'SEND'|'SENDNO' 'MSGID'|'ID'|'NOID' P1 P2...P9)]
```

For *field-attributes*, see PANEL DEFINE.

```
   [ICDISP cursor-displacement]
[NLIne lines]
[TEXTSTART panel-image-definition TEXTEND]
[TRACEON]
[TRACEOFF]
```

[CONTENT *parameters*]

Full definition on page 87. The *parameters* are the same as the HEADER sub-parameters (except LINES).

[TRAILER [LINES *line-number*] *parameters*]

Full definition on page 87. The *parameters* are the same as the HEADER sub-parameters.

[PROCESS [CALL *process-sub-definition*] [TRACEON] [TRACEOFF]]]

The rest is defined using TPSL structures and operations. For full definition, see page 90.

[PHEADER *sub-header-name*]

Full definition on page 91. The PHEADER statement is used to define a common panel HEADER which can be CALLed from another HEADER section. The PHEADER definition is specified in the same way as the HEADER parameter, but with these exceptions:

- The LINES parameter on the HEADER parameter *cannot* be specified on the PHEADER definition.
- The LANGUAGE parameter on the PHEADER definition *cannot* be specified on the HEADER parameter.

[PCONTENT *sub-content-name*]

Full definition on page 92. The PCONTENT statement is used to define a common panel CONTENT which can be CALLed from another CONTENT section. The PCONTENT definition is specified in the same way as the CONTENT parameter, but with this exception:

- The LANGUAGE parameter on the PCONTENT definition *cannot* be specified on the CONTENT parameter.

```
[PTRAILER sub-trailer-name ]
```

Full definition on page 93. The PTRAILER statement is used to define a common panel TRAILER which can be CALLed from another TRAILER section. The PTRAILER definition is specified in the same way as the TRAILER parameter, but with these exceptions:

- The LINES parameter on the TRAILER parameter *cannot* be specified on the PTRAILER definition.
- The LANGUAGE parameter on the PTRAILER definition *cannot* be specified on the TRAILER parameter.

```
[PPROCESS sub-process-name ]
```

Full definition on page 94. The PPROCESS statement is used to define a common panel PROCESS which can be CALLed from another PROCESS section. The PPROCESS definition is specified in the same way as the PROCESS parameter, but with this exception:

- The LANGUAGE parameter on the PPROCESS definition *cannot* be specified on the PROCESS parameter.

SCRIPT

```
[SCRIPT script-name                                  (Full definition on page 95)
   [APPLTIME application-timeout-limit]
   [DOMAX domax-limit]
   [RPQLSCRIPTAUTO [YES | NO]]
   [SIMRPQ [YES | NO]]
```

The following SCRIPT operations may be embedded within TPSL structures, and interspersed with TPSL LETs and RETURNs.

```
[AUDITMSG message-text ]
[BRANCH label name ]
   [LAB label name ]
[CALL script-name ]
[CALLEXIT [DATA  user-data ] [RSN exit-reason-code ]]
[DISPAPPL [FLUSH] ]
[ENDSESS [SESSION session-id ]]
[EXTRACT DATA variable-name
   [FROM row | variable-name [col | variable-name ]]
   [TO col | variable-name ]]
[INPUT [PASS | [CURSOR   row col ]
   [KEY  key | NO]
   [SBA [row | variable-name col | variable-name ] |
        [NO | OFF]]
   [TEXT text ] ]]
[INSCAN inscan-string
   [FROM row | row-variable
      [col | col-variable ]
      [TO row | row-variable [col | col-variable ] ] |
       AT row | row-variable
      [col | col-variable ] ]
[OUTSCAN outscan-string
```

```
   [FROM row | row-variable [col | col-variable
      [TO row | row-variable [col | col-variable ] ] |
       AT row | row-variable
      [col | col-variable ] ]
[PAUSE seconds | variable-name ]
[TDEQ enqueue-value ]
[TENQ enqueue-value ]
[TRACEON]
[TRACEOFF]
[ISZCMD command] (any valid Session Manager command)
[ISZCMDA command] (any valid Session Manager command)
[USERMSG PANEL usermsg-panel-name | TEXT usermsg-text ]
[VCALL name | 'script-name' ]
[WAITAPPL [DISPLAY [Yes | No | ON | OFF] ]
   [TIMEOUT timeout-value ]
   [KEEP] ]
[WAITDATA [PASS] [TIMEOUT timeout-value ] [KEEP] ]
[WAITTERM [PASS] ]
```

The following SCRIPT parameters are applicable to Application Builder scripts only:

```
[HALTSCRIPT session-id ]
[READSESS user-variable-name
   [FROM row | row-variable [col | col-variable ] ]
   [TO col | col-variable ]  ]
[RUNSCRIPT session-id script-name ]
[SENDDATA variable-name | literal ...
   SESSION session-id | RPARTNER]
[STARTSESS session-id ]
[XYZSCREEN user-variable-name]
   [FROM row | row-variable [col | col-variable ] ]
   [TO col | col-variable ]  ]
```

The following SCRIPT parameters are applicable to Windows scripts only:

TPSL syntax

The Panel and Script Language consists of seven logic statements. DO, IF, and SELECT statements may be used to build TPSL structures, and LET, ITERATE, LEAVE, and RETURN statements are used for TPSL operations. TPSL structures and operations may be used in combination with specialized PANEL and SCRIPT keywords, within the PANEL and SCRIPT statements, to control logic and to alter or define data for output on screens, or input for Session Manager and applications. See the full definition on page 145. (A subset of these logic statements can also be used in the configuration file to control definition of the Session Manager system – see the *Technical Reference* for details.)

Testing: simple condition

```
IF [NOT] conditional-expression
   [AND | OR [NOT] conditional-expression]
[THEN]
   one or more operations
[ ELSE one or more operations]
END
```

Testing: multiple choice

```
SELECT | CASE
   WHEN [NOT] conditional-expression-1
      [AND | OR   [NOT] conditional-expression-2
      [THEN]
         one or more operations
   WHEN [NOT] conditional-expression-3
      [AND | OR   [NOT] conditional-expression-4
      [THEN]
         one or more operations
         ...
   WHEN [NOT] conditional-expression-n
      [AND | OR   [NOT] conditional-expression-n
         [THEN]
         one or more operations
   [ELSE
   one or more operations]
END
```

Looping

```
DO
   WHILE conditional-expression    |
      UNTIL conditional-expression |
      FOREVER                      |
      FOR nnn | variable-name]
   one or more operations
END
```

For a full description of *conditional-expression* as used in the previous three statements, refer to 'The IF statement' on page 147.

Altering the flow within a DO loop

```
ITERATE
```

Assignment

```
LET variable-name | &variable-name = variable              |
                                     literal               |
                                     string                |
                                     arithmetic expression |
                                     function
```

where *function* can be one of:

```
ABBREV (string input length)
ABS (number)
ADDOUT (string row col attr)
CENTRE (string length pad)
CHANGESTR (string1 string2 new)
COMPARE (string1 string2 pad)
COPIES (string n)
COUNTSTR (string1 string2)
DELSTR (string n length)
DELWORD (string n length)
D2X (variable)
ENTDATA (fieldnum extdata row-col-prefix)
ENTRY_IO ('operation' ddname other_parms)
EXTRAUTH (command_variable keyword_variable)
FORMATMSG (nnnn 'SEND'|'SENDNO' 'MSGID'|'ID'|'NOID' P1 P2...P9)
INSERT (new string n length pad)
KEYPOS (string words n)
LASTPOS (string1 string2 n)
LEFT (string length pad)
LENGTH (string)
MAX (n1 ... nx)
MIN (n1 ... nx)
OVERLAY (new string n length pad)
PASSIN (session)
PASSOUT (session)
POS (string1 string2 n)
REVERSE (string)
RIGHT (string length pad)
SIGN (number)
SPACE (string n pad)
STRIP (string ‘B’ | ‘L’ | ‘T’ pad)
SUBSTR (string n length pad)
SUBWORD (string n length)
TMSG (variable)
UPPER (string)
VERIFY (string chars ‘N’ ‘M’ n)
WORD (variable)
WORDINDEX (string n)
WORDLENGTH (string n)
WORDPOS (string1 string2 n)
WORDS (variable)
X2D (variable)
```

Exiting: immediately from a DO loop

LEAVE

Exiting: early termination of logic processing

RETURN

PANEL statement

The PANEL statement enables Installation-specific Signon, Menu, and Data Display screens to be defined. Default screen layouts are supplied with the product and may be used as they are, or may be altered to suit Installation standards, if required, using the information in this section.

The full function of each terminal may be used; it is not necessary to specify special panels for each terminal type. Multi-page panels may be defined. There is no limit on the number of lines that may be defined, since paging commands are available for the pageable section of each panel. Most panels are defined with fixed header and trailer portions, and a pageable part between.

A panel definition may be divided into five sections, each section associated with one of the PANEL statement parameters. The five sections are:

- The DEFINE parameter
- The HEADER parameter
- The CONTENT parameter
- The TRAILER parameter
- The PROCESS parameter

Each of the five optional parameters may be specified only once in each panel definition, but may be supplied in any order. However, *they are always processed in the sequence given above.* This is significant for field attribute definitions, since the DEFINE, HEADER, CONTENT, and TRAILER sections may all define field attributes. Any definition remains in effect until the end of the panel definition, or until overridden, but in the sequence shown above.

For example, a field attribute definition for a particular symbol specified in the HEADER section may be altered in the TRAILER section for the remainder of the panel, even if the TRAILER section is specified before the HEADER section. The maximum number of field attribute definitions that may be in use simultaneously is 50.

The Panel and Script Language (TPSL) may be used in a panel definition to give the panel logic processing ability. 'The Panel and Script Language (TPSL)' on page 145 gives a description of the TPSL statements. None of the parameters given above may appear in a TPSL logic structure, although TPSL statements may be used in all but the DEFINE parameter.

Session Manager provides two methods for panel definition, either by specifying each field position using a column number, or by the 'text' method where each field is positioned appropriately in the logical record so that each line looks like its output line, i.e., WYSIWYG. It is also acceptable to use a combination of both methods. Field and text definitions may be used in header, trailer, and content definitions.

The Session Manager variables which may appear in panel definitions are described in 'Session Manager variables' on page 197.

A detailed description of how the panel defining feature of Session Manager works can be found in 'Panels: definition and usage' on page 49.

The PANEL statement is defined as follows:

PANEL *panel-name*

Defines the name by which this panel is known. A minimum of one alphanumeric character and up to a maximum of eight may be specified.

BRANCH *label name*
[LAB *label name*]

The BRANCH statement in TPSL allows for switching the processing in a script/panel to another location rather than the next sequential statement. It makes migration from other products more straightforward. The label name (maximum of 8 characters) is the name specified on the LAB statement.

Example:

```
SELECT
   WHEN t_user= 'USER1'
          THEN
         BRANCH USER1
   WHEN t_user='USER2'
          THEN
         BRANCH USER2
   ELSE
          BRANCH USERX
   END
LAB EXIT
RETURN

********************************

* Routines for different users  *

********************************

LAB USER1
...Processing particular to USER1...
   BRANCH EXIT
LAB USER2
...Processing particular to USER2...
  BRANCH EXIT
LAB USERX
...Processing for other users...
  BRANCH EXIT
```

The BRANCH and LAB combination can be used in scripts and panels. The only restriction is that branching into a DO loop from outside it will be signalled as an error at configuration processing time. Branching out of a DO loop will be allowed.

DEFINE [LANGuage *languageid*]
[WIDTH 80 | 132]
[MODE NORMal | ALTernate]
[DOMAX *domax-limit*]
[CURSOR *field-name*]
[ATTR *attr-character* OFF | *field-attribute-params*]

The DEFINE parameter and its associated subparameters enable general characteristics for the panel to be specified. If omitted, then certain defaults are assumed. If specified, then the DEFINE parameter must appear before any other PANEL statement parameter.

`LANGuage` *`languageid`*

Language Packs (LPs) which are supplied with Session Manager contain all the Panels, Scripts, Text and Messages that have been translated. For more information on LPs and national language support in Session Manager, see the *Installation and Customization* manual.

The LANGUAGE parameter specifies the language in which screen text for the panel is displayed at a user's terminal. *`languageid`* must be two characters consisting only of letters of the alphabet or numerals.

Any `PROFILE`, `USER` or `TERMINAL`, which has a matching language id specified, will receive the screen text for the panel defined by the `LANGUAGE` set. If the user has not specified a `LANGUAGE` value, or has specified an invalid value for *`languageid`* (that is, there is no corresponding LP) then the default LP (English) is used.

`WIDTH 80 | 132`

Specifies the screen width requirement for this panel definition. A panel defined with a width of 80 appears centralized on a screen displaying 132 columns. If the width is defined as 132, the panel is truncated on the right if it appears on a screen displaying only 80 columns.

This subparameter also provides some checking on the length of the panel header, trailer, and content definitions. The default is 80 when this subparameter is omitted.

Note that a panel definition of 132 columns may not be defined using the textstart/textend method to create the screen layout. Each field position must be specified.

`MODE NORMal | ALTernate`

Specifies whether the panel definition is to use the larger alternate screen size where possible, or the normal 24 by 80 size. The default is NORMAL.

`DOMAX` *`domax-limit`*

Specifies the maximum number of times any DO loop in this PANEL definition may be executed. *`domax-limit`* may be any value in the range 1-9999. The default is 100.

`CURSOR` *`field-name`*

Specifies the name of the variable at which the cursor is to be positioned. All occurrences of the field will have a cursor generated, but the cursor appears on the last occurrence on the physical screen.

Field attributes may be defined as cursor control for any field, but neither the cursor parameter nor the field attribute takes priority – it is always the last field with a cursor definition that will have the cursor positioned on it. This applies to subscripted fields as well as multiple definitions of the same field.

Typically the cursor is defined to appear in the command area, and this is usually specified in the trailer section. If no cursor position is defined, or for any reason the selected field does not appear on the screen, the cursor is positioned at the top left hand corner (1,1) of the screen. It is therefore not advisable to position the cursor at a variable in the CONTENT section, since the variable may be lost when paging forward or backward.

`ATTR` *`attr-character`* `OFF | ` *`field-attribute-parameters`*

The ATTR subparameter is used to define field attributes to be associated with a special symbol. When specified in the DEFINE parameter, it applies throughout the panel definition. The HEADER, TRAILER, and CONTENT parameters also provide an ATTR subparameter which may be used to define additional symbols, or override current settings of existing symbols. The specified value should be enclosed in delimiters. For example:

```
ATTR '?' PROT,NORM,OUT
ATTR !%! UNPROT,HI,IN
```

Valid delimiters are:

`'this is delimited'`	– quote marks
`"this is delimited"`	– double quotes
`(this is delimited)`	– parentheses
`!this is delimited!`	– exclamation marks
`?this is delimited?`	– question marks

Symbols may also be defined in the SYSTEM statement to apply globally throughout all panel definitions, although any symbol may be overridden.

OFF may be used to clear any current setting for a symbol. The default is OFF when no field attributes parameters are supplied.

Acceptable field attribute parameters are as follows:

For protection and content:

```
[ASKIP]
[DETect | NODETect]
[UNPRotected | PROTected]
[NUMeric | ALPHanumeric]
[NONdisp]
```

For colour and effects:

```
[RED | YELlow | WHIte | GREen | BLUe | PINk | TURquoise]
[REVerse | BLInking | UNDErscore]
[HIgh | NORMal]
```

Special attributes:

```
[ETMODE | DBCS]
[ICursor]
[BLANK | NOBLANK]
[IN | OUT | LITeral]
```

Synonyms may be used. Valid synonyms are:

- SKIP for ASKIP
- NODisp for NONdisp

- USCOre for UNDErscore

The following is a brief explanation of each set of attributes:

The 'protection and contents' attributes are all attribute type C0.

The 'colour and effects' attributes are attribute type 42.

The 'special attributes' each have the meaning given below.

Special attributes

`ETMODE | DBCS`

Used for enabling a field to accept double byte character sets.

ETMODE is used to enable shift in (SI)/shift out (SO) orders within a field and sets SO = x'0E' and SI = x'0F' to go in and out of double byte character mode. If specified as an input field, the attribute FE is added to the attributes with 01 specified (meaning SO/SI creation enabled).

DBCS means the whole field is in double byte character mode. The attribute 43 with F8 specified is added to the attributes to enable Kanji.

`ICursor`

This sets a 3270 insert cursor buffer control code x'13'

`BLANK | NOBLANK`

BLANK specifies that any zeros, used as padding in a field, are to be converted to blanks.

NOBLANK, the default, performs no conversion; zeros are displayed as zeros.

`IN | OUT | LITeral`

IN specifies that the field is an input field into which data can be entered. A field with an attribute of IN has a default attribute setting of UNPROT.

OUT specifies that the field is an output field which is used to display variables on the panel. A field with an attribute of OUT has a default attribute setting of UNPROT.

LITERAL specifies that the field is for displaying character strings. A field with an attribute of LITERAL has a default attribute setting of PROT.

Note that certain combinations of attributes may be invalid; the *IBM 3270 Display System Reference* should be consulted for confirmation of valid combinations. Note also that a field defined without a PROT attribute is unprotected even when it is assigned an OUT attribute. Any data keyed into such a field is ignored.

Session Manager supplies default attribute definitions for specific symbols. These are:

Symbol	Definition
#	UNPROT,NORM,IN
%	UNPROT,HIGH,IN
"	UNPROT,NOND,IN
?	PROT,NORM,OUT

Symbol	Definition
&	PROT,HIGH,OUT
<	PROT,NORM,LIT
@	PROT,HIGH,LIT

When ATTR is omitted for a field, the default attributes are:

- UNPROTected
- ALPHanumeric
- NODETect
- NORMal
- SKIP
- LITeral
- Default colour – usually green

```
HEADER [LINES line-number]
TRAILER [LINES line-number]
CONTENT
```

The HEADER, TRAILER, and CONTENT parameters denote the start of each section defining the header lines which are to appear at the top of a screen, the trailer lines which are to appear at the bottom of the screen, or the middle part of the screen, i.e., the screen content.

The header and trailer lines are 'fixed' in that they do not change when the paging keys, or commands, are used to page forward on a multiple page panel definition. It is advisable to define a message area and a command area in the fixed portion. These may be either in the HEADER or TRAILER section.

Optionally, the number of header lines and trailer lines may be specified. *line-number* may be any value in the range 1-48; if omitted the default is 3. The total number of lines specified in the header and trailer sections, when combined, should not be greater than the screen size.

If the number of lines defined is greater than the number specified on the LINES subparameter, the bottom excess lines in the header or trailer section do not appear on the screen display.

If the number specified on the LINES subparameter is greater than the number of lines defined, then blank lines appear at the bottom of the header section on the screen display. Similarly for the trailer section, blank lines are inserted at the bottom of the trailer.

The LINES subparameter is not valid for the CONTENT section. This section may be any number of lines, since it is pageable.

Text is specified in the form of variables or literals, and these may be on a field by field basis, or can be supplied in 'WYSIWYG' format using TEXTSTART and TEXTEND parameters. These are described later.

Processing logic may be used in all three sections and is provided by the Panel and Script Language (TPSL). This is described separately on page 145.

```
ATTR attr-character OFF | field-attribute-parameters
```

Specifies the field attributes that are to be ascribed to special symbols, or overrides existing settings. Field attribute definition is described previously under the DEFINE parameter.

Note that the definition remains in effect until the end of the panel definition, or until overridden.

Note also that attribute definitions may not be defined in TPSL structures.

The PANEL operations described below may be interspersed with TPSL statements.

`CALL panel-sub-definition`

Invokes other panel definitions. The *panel-sub-definition* is a one to eight alphanumeric character name which specifies the name of a corresponding definition created using the PHEADER, PCONTENT, or PTRAILER statements.

`COLumn col`

Specifies the position of the next field's attribute character. If omitted, the next available column after the current field is used.

col may be any value in the range 1-80, or 1-132 for wide screens. If the number specified is less than the current position, a new line is automatically generated.

```
FIELD  variable[(nn)] | literal
[field-attributes |
REFattr attr-character | variable-name]
[ICDISP cursor-displacement]
```

Defines a single field which may be a variable or a literal. If it is a variable, it may be a Session Manager supplied variable, an existing User Defined variable, or a new User Defined variable.

It is advisable to supply a two-byte length in parentheses after the variable name, particularly for User Defined variables, and even for some of the Session Manager supplied variables, such as t_message, which may not have a fixed length. If length is omitted, then the length of the field on the screen display will vary depending on the data assigned to it. This gives the panel display an inconsistent layout, particularly when the COLUMN parameter has not been used to fix fields in certain positions.

If a literal is specified, it must be enclosed in delimiters. The field length is implied by the length of the literal.

Optionally field attributes may be assigned to the field. A list of field attributes may be specified. Alternatively, the REFATTR subparameter can be used to refer to a previously defined attribute symbol.

REFATTR can specify a user-defined character variable name that contains an ATTR symbol. When the ATTR symbol is assigned to a character variable it must be prefixed with the letter **A**. The attribute can then be controlled from TPSL code, as the example below shows:

```
If ln_rcode > 4
   Let lcattr = A'&'  /* '&' attr value
Field lcmsg REFATTR lcattr
```

Valid attributes are the same as those described for the ATTR subparameter of the DEFINE parameter.

If no attributes are supplied, fields are displayed in normal intensity, and are assumed to be unprotected, although anything overtyped is ignored.

The ICDISP keyword optionally positions the cursor at an offset from the start of the field. *cursor-displacement* can be a literal or a variable containing the offset to be used.

A value of 1 positions the cursor at the first position of the field. A value of 2 positions the cursor at the second position of the field, and so on. If zero (0) is specified, the cursor is not positioned in the field. This is the default.

The following example illustrates how you can specify at what offset from the start of t_command the cursor should be positioned:

```
If ln_fred = 0 then
  Let ln_fred = 1
End
Field 'cursor offset ===>' prot hi
Field ln_fred(4) in
Field 'command ===>' prot hi
Field t_command in unprot rev
ICDISP ln_fred
```

Note There is no check to ensure that the offset is within the field. So, for example, if the first field in a panel definition contained the ICDISP parameter, then the variable could position the cursor anywhere on the screen. If, however, the variable attempts to position the cursor beyond the screen, then the cursor is set to the bottom right-hand corner of the screen.

NLIne [*lines*]

Causes a new line to be generated on the display. *lines* may be a literal or a variable containing the value to be used. *nn* may be any value in the range 1-10; if omitted, the default is 1. The column position is set to 1.

TEXTSTART

Specifies the start of a panel section to be defined by specifying each logical record such that it looks like a line on the screen, i.e., WYSIWYG.

The subparameter must appear on a separate record. It implies that the subsequent 80 byte records define the layout of the display. These records should therefore contain a combination of literals, variables, and associated attribute values, which Session Manager can then use to build the output screen image. The screen contents are defined by all records encountered up to the TEXTEND subparameter. The area being defined starts in column 1 of the current, or next, empty line.

Note that literals do not need to be enclosed in delimiters.

TEXTEND

This subparameter should be placed at the end of all the records which constitute a screen image text definition. It should appear on a separate record.

TRACEON

Indicates the point at which tracing commences. Tracing is only used when the TTPSL command is issued. Refer to 'System Operator Commands' in the *Technical Reference.*

`TRACEOFF`

Indicates the point at which tracing terminates. Tracing is only used when the TTPSL command is issued. Refer to 'System Operator Commands' in the *Technical Reference.*

`PROCESS`

The processing section is comprised entirely of TPSL statements, which are executed when input is received. For more information about TPSL statements, see 'The Panel and Script Language (TPSL)' on page 145.

`CALL` *`process-sub-definition`*

Invokes other process definitions. The *`process-sub-definition`* is a one to eight alphanumeric character name which specifies the name of a corresponding process definition, created using the PPROCESS statement.

`TRACEON`

Indicates the point at which tracing will commence. Tracing is only used when the TTPSL command is issued. Refer to 'System Operator Commands' in the *Technical Reference.*

`TRACEOFF`

Indicates the point at which tracing will terminate. Tracing is only used when the TTPSL command is issued. Refer to 'System Operator Commands' in the *Technical Reference.*

PHEADER statement

The PHEADER statement defines a common panel header section which can be incorporated in other panel headers by specifying the CALL subparameter.

Syntax

PHEADER *panel-sub-definition*

Description

Defines the name by which this sub-definition is known. A minimum of one alphanumeric character and up to a maximum of eight may be specified. This name can be specified on a CALL subparameter of a HEADER definition to include this sub-definition.

The PHEADER definition is specified in the same way as the HEADER parameter, but with these exceptions:

- The LINES parameter on the HEADER parameter *cannot* be specified on the PHEADER definition.
- You can specify a LANGUAGE parameter on the PHEADER definition (see below) but not on the HEADER parameter.

LANGUAGE parameter of PHEADER statement

Language Packs (LPs) which are supplied with Session Manager contain all the Panels, Scripts, Text and Messages that have been translated. For more information on LPs and national language support in Session Manager, see the *Installation and Customization* manual.

The optional LANGUAGE parameter of the PHEADER statement specifies the language in which screen text for the panel header is displayed at a user's terminal.

Syntax

[LANGuage *languageid*]

Parameter

languageid

Must be two characters consisting only of letters of the alphabet or numerals.

Usage notes

Any PROFILE, USER or TERMINAL, which has a matching language id specified, will receive the screen text for the panel header defined by the LANGUAGE set. If a LANGUAGE value is not specified, or an invalid value for *languageid* is specified (that is, there is no corresponding LP) then the default LP (English) is used.

The LANGUAGE parameter should be placed on the line following the PHEADER statement name, although intervening comments are allowed, for example:

```
PHEADER xxxxxxxx
* This is a French definition
LANG FR
...
...
```

PCONTENT statement

The PCONTENT statement defines a common panel content section which can be incorporated in other panel contents by specifying the CALL subparameter.

Syntax

```
PCONTENT panel-sub-definition
```

Description

Defines the name by which this sub-definition is known. A minimum of one alphanumeric character and up to a maximum of eight may be specified. This name can be specified on a CALL subparameter of a CONTENT definition to include this sub-definition.

The PCONTENT definition is specified in the same way as the CONTENT parameter, but with this exception:

- You can specify a LANGUAGE parameter on the PCONTENT definition (see below) but not on the CONTENT parameter.

LANGUAGE parameter of PCONTENT statement

Language Packs (LPs) which are supplied with Session Manager contain all the Panels, Scripts, Text and Messages that have been translated. For more information on LPs and national language support in Session Manager, see the *Installation and Customization* manual.

The optional LANGUAGE parameter of the PCONTENT statement specifies the language in which screen text for the panel content is displayed at a user's terminal.

Syntax

```
[LANGuage languageid]
```

Parameter

languageid

Must be two characters consisting only of letters of the alphabet or numerals.

Usage notes

Any PROFILE, USER or TERMINAL, which has a matching language id specified, will receive the screen text for the panel content defined by the LANGUAGE set. If a LANGUAGE value is not specified, or an invalid value for *languageid* is specified (that is, there is no corresponding LP) then the default LP (English) is used.

The LANGUAGE parameter should be placed on the line following the PCONTENT statement name, although intervening comments are allowed, for example:

```
PCONTENT xxxxxxxx
* This is a French definition
LANG FR
...
...
```

PTRAILER statement

The PTRAILER statement defines a common panel trailer section which can be incorporated in other panel trailers by specifying the CALL subparameter.

Syntax

```
PTRAILER panel-sub-definition
```

Description

Defines the name by which this sub-definition is known. A minimum of one alphanumeric character and up to a maximum of eight may be specified. This name can be specified on a CALL subparameter of a TRAILER definition to include this sub-definition.

The PTRAILER definition is specified in the same way as the TRAILER parameter, but with these exceptions:

- The LINES parameter on the TRAILER parameter *cannot* be specified on the PTRAILER definition.
- You can specify a LANGUAGE parameter on the PTRAILER definition (see below) but not on the TRAILER parameter.

LANGUAGE parameter of PTRAILER statement

Language Packs (LPs) which are supplied with Session Manager contain all the Panels, Scripts, Text and Messages that have been translated. For more information on LPs and national language support in Session Manager, see the *Installation and Customization* manual

The optional LANGUAGE parameter of the PTRAILER statement specifies the language in which screen text for the panel content is displayed at a user's terminal.

Syntax

```
[LANGuage languageid]
```

Parameter

languageid

Must be two characters consisting only of letters of the alphabet or numerals.

Usage notes

Any PROFILE, USER or TERMINAL, which has a matching language id specified, will receive the screen text for the panel trailer defined by the LANGUAGE set. If a LANGUAGE value is not specified, or an invalid value for *languageid* is specified (that is, there is no corresponding LP) then the default LP (English) is used.

The LANGUAGE parameter should be placed on the line following the PTRAILER statement name, although intervening comments are allowed, for example:

```
PTRAILER xxxxxxxx
* This is a French definition
LANG FR
...
...
```

PPROCESS statement

The PPROCESS statement defines a common panel process section which can be incorporated in other panel processes by specifying the CALL subparameter.

Syntax

PPROCESS *process-sub-definition*

Description

Defines the name by which this sub-definition is known. A minimum of one alphanumeric character and up to a maximum of eight may be specified. This name can be specified on a CALL subparameter of a PROCESS definition to include this sub-definition.

The PPROCESS definition is specified in the same way as the PROCESS parameter, but with this exception:

- You can specify a LANGUAGE parameter on the PPROCESS definition (see below) but not on the PROCESS parameter.

LANGUAGE parameter of PPROCESS statement

Language Packs (LPs) which are supplied with Session Manager contain all the Panels, Scripts, Text and Messages that have been translated. For more information on LPs and national language support in Session Manager, see the *Installation and Customization* manual.

The optional LANGUAGE parameter of the PPROCESS statement specifies the language in which screen text for the panel content is displayed at a user's terminal.

Syntax

[LANGuage *languageid*]

Parameter

languageid

Must be two characters consisting only of letters of the alphabet or numerals.

Usage notes

Any PROFILE, USER or TERMINAL, which has a matching language id specified, will receive the screen text for the panel process defined by the LANGUAGE set. If a LANGUAGE value is not specified, or an invalid value for *languageid* is specified (that is, there is no corresponding LP) then the default LP (English) is used.

The LANGUAGE parameter should be placed on the line following the PPROCESS statement name, although intervening comments are allowed, for example:

```
PPROCESS xxxxxxxx
* This is a French definition
LANG FR
...
...
```

SCRIPT statement

Overview

Each SCRIPT statement defines a set of executable parameters which can be thought of as a small program. Because there are numerous parameters, scripts are very flexible and can perform many different functions. However, scripts generally fall into one of these categories:

- Session scripts
- Exit scripts
- Command scripts
- Window scripts
- Application Builder scripts

As the name suggests, a session script relates to a specific application session, or to a session parameter, and can only be invoked when an INITSCRIPT, ENVIRONSCRIPT, STARTSCRIPT, AUTOSCRIPT, ENDSCRIPT, or TERMSCRIPT parameter is defined in the Configuration file.

Exit scripts can be used to run in place of, or in addition to, many of the exit points of the Session Manager Assembler/COBOL User exit.

Command scripts can be used to define user-defined commands and can also be run instead of Session Manager logic to action a command.

Window scripts relate to the Windows feature and are not session-specific. They are invoked from a Menu by the WINDOWS command and enable a single terminal screen to be used to view different sessions simultaneously.

Application Builder scripts enable output to be gathered from two or more existing applications and presented together on the screen. The information gathered may be presented in any desired layout on the screen by means of a panel definition. In addition, any chosen output from a session can be used as input to another session.

Session scripts

A script can comprise one or more input data streams. When a script consists of multiple input streams, the first is passed to the application when the logon to it completes, and subsequent input streams are passed to the application without interruption unless parameters are coded in the script which cause it to wait.

All the session related scripts can be specified at SYSTEM, PROFILE, USER, TERMINAL, LU, and APPL levels.

INITSCRIPTs

An ‘initialization’ script may be processed before session start and therefore does not communicate with an application. This type of script may be used to alter session parameters. All script parameters used to communicate with an application (for example, INPUT, WAITAPPL, and so on) are ignored by Session Manager.

For Session Manager 1.1.10 and higher, the INITSC command enables a suitably authorized user to disable or (re-)enable the running of the session ‘initialization’ script. For details, see the *Technical Reference*.

ENVIRONSCRIPTs

This enables all processing common to a specific operating environment to be held in just one script. For example, an Environscript can deal with messages and required input when a TSO environment is initiating.

STARTSCRIPTs and Autoscripts

Scripts may be input at session start (Startscript), or by the user issuing the special command sequence defined by the AUTOSEQ or the SAUTOSEQ parameters of the SYSTEM, USER, TERMINAL, LU, or PROFILE statement (Autoscript).

For Session Manager 1.1.10 and higher, the STARTSC command enables a suitably authorized user to disable or (re-)enable the running of the session start script. For details, see the *Technical Reference.*

ENDSCRIPTs

This type of script is run when the user issues the END command for an active session. It is also run when the LOGOFF command is entered, if LOGOFF END has been defined for the user and is in effect for the active session.

TERMSCRIPTs

A script may be defined to be processed when a session has terminated and therefore does not communicate with an application. This type of script is generally used to reset session options. All script parameters used to communicate with an application, e.g., INPUT, WAITAPPL, are ignored by Session Manager.

TPSL can solve the problems of automatic input where the responses from the target application are inconsistent or unpredictable. Further explanation and some examples have been included in 'Defining conditional scripts' on page 25.

Exit scripts

An Exit script can be defined to run in place of, or in addition to, most Assembler/COBOL exit points of the Session Manager User exit.

For most exit scripts, this is achieved by specifying the exit point name and the option 'S' on the OPTION statement. A script of the form EXIT*xx* must be defined for the relevant exit, where '*xx*' is the exit point. For example, if E33 S is specified on the OPTION statement, a script EXIT33 will be called.

For the input (E25) and output (E35) 3270 datastream exits, load module names are identified on the SYSTEM statement by the INPUTEXIT and OUTPUTEXIT parameters respectively. An additional, related, SYSTEM statement parameter, EXITWALEN, specifies the size of the work area made available to the exits. For details on these parameters, see the *Technical Reference.*

A description of Exit scripts, together with the script verbs/parameters and variables that can be used, is given on page 33.

Command scripts

A Command script can be run in place of the corresponding Session Manager defined command. This facility is enabled by the CMDSCRIPT parameter of the COMMAND statement.

A description of Command scripts, together with the script verbs/parameters that can be used, is given on page 45.

Window scripts

A Window script can be invoked from a Menu screen by the WINDOW command. Depending on the contents of the script, the terminal screen is then divided into areas, and each area would usually contain its own command field so that each area can perform its own function. Typically, several sessions would be active and viewable simultaneously.

A Window script comprises mainly 'window verbs' and TPSL statements. Window verbs/parameters are described separately on page 119.

Application builder scripts

An Application Builder script can be defined to selectively combine output from multiple sessions and present it on a single panel. Additionally, output from one or more sessions may be input to any other sessions. This is achieved by running scripts on each of the sessions concerned. As with normal session scripts, each can provide input to its application, await output and scan the output for various content. However, Application Builder verbs enable each script to start a session; terminate any session; run a script on a specified session; halt a script in progress on a session; copy any part of an application screen to a variable; send data to, and receive data from, a script running on another session.

Details of Application Builder verbs/parameters are detailed on page 113.

Session script parameters

The SCRIPT statement is defined as follows:

`APPLTIME` *`application-timeout-value`*

Specifies the maximum number of seconds that the script is to wait for output from the application. It applies to all WAITAPPL parameters that do not have an associated TIMEOUT subparameter. *`application-timeout-value`* may be any value in the range 1-999 seconds. If no value is specified, no time limit is set.

An application timeout is detected if a WAITAPPL or WAITDATA parameter is actioned, and the application does not reply.

If a timeout occurs, control is returned to the parameter following the WAITAPPL or WAITDATA parameter. The special variable NORESP is set on and can therefore be tested using TPSL statements.

`AUDITMSG` *`message-text`*

Specifies the text to be output as message 226 and may be different for each AUDITMSG parameter. Since message 226 is a null message, the full text should be supplied. This may be a variable or a literal. A literal may contain embedded spaces and must be enclosed in delimiters. The maximum length is 256.

The choice of delimiters enables quotes to be specified unambiguously within text data. Valid delimiters are:

- ‘this is delimited’
- “this is delimited”
- (this is delimited)
- !this is delimited!
- ?this is delimited?

When several variables and literals are specified, each is sent as a separate message. If a single message is to contain both variable and literal parts, variable substitution can be used within the literal. For example:

`AUDITMSG ‘Userid: &`*`t_userid`*`&’`

The message is sent to the console or Audit file depending on the options set by the MESSAGE statement, or by the MSGSUFFIX parameter of the SYSTEM statement.

The variable t_result contains an error 4 if the message text is null.

Return code	Description
0	Message sent

`BRANCH` *`label name`*
`[LAB` *`label name`*`]`

The BRANCH statement in TPSL allows for switching the processing in a script/panel to another location rather than the next sequential statement. It makes migration from other products more straightforward. The label name (maximum of 8 characters) is the name specified on the LAB statement.

Example:

```
SELECT
   WHEN t_user= 'USER1'
          THEN
         BRANCH USER1
   WHEN t_user='USER2'
          THEN
         BRANCH USER2
   ELSE
          BRANCH USERX
   END
LAB EXIT
RETURN
```

```
*********************************
* Routines for different users  *
*********************************
LAB USER1
...Processing particular to USER1...
   BRANCH EXIT
LAB USER2
...Processing particular to USER2...
  BRANCH EXIT
LAB USERX
...Processing for other users...
  BRANCH EXIT
```

The BRANCH and LAB combination can be used in scripts and panels. The only restriction is that branching into a DO loop from outside it will be signalled as an error at configuration processing time. Branching out of a DO loop will be allowed.

`CALL` *`script-name`*

Scripts may be nested up to any level, and the CALL parameter enables another script to be executed at a lower level. The processing for lower level scripts is the same as for the top level script.

The CALL parameter is treated as a TPSL instruction, not a script verb, and therefore does not set t_rc or t_result. Note, however, that they may be set by verbs or LET statements, in the called script.

Care should be taken that a script CALL is not recursive, i.e., it does not call itself, as this will cause a loop.

`CALLEXIT [DATA` *`user-data`*`] [RSN` *`exit-reason-code`*`]`

CALLEXIT invokes the E09 exit point.

DATA enables up to 255 bytes of user data to be passed to the exit. It can be specified as either a character-type variable which contains the value to be passed, or as a character literal value. If a character literal value is specified, it can contain embedded character or numeric variables. See page 197 for a description of variable substitution.

RSN specifies a user-defined exit reason code to be passed to the exit. Its maximum length is eight bytes and it may be specified as either a character literal value, or as a character-type variable which contains the value to be passed.

The return code from the CALLEXIT parameter can be found in the variable t_rc. The possible values are:

Return code	Description
0	Operation successful
28	Contact your local support representative.
40	No user exit loaded, or E09 exit point disabled.

The variable t_result contains the value set by the exit when t_rc is zero. For a return code other than zero, its value remains unchanged.

DISPAPPL [FLUSH]

Displays at the terminal a data stream or queue of data streams from the application, received by using either a WAITAPPL or WAITDATA parameter.

If WAITAPPL DISPLAY is in effect, a subsequent DISPAPPL will send that data stream a second time. WAITAPPL DISPLAY NO can be specified to prevent this.

The queue is retained until a DISPAPPL FLUSH is received, or the script terminates. DISPAPPL can therefore be specified multiple times to send the same queue of data streams.

To create a queue of data streams, the KEEP operand must be in effect for the second and subsequent WAITAPPLs or WAITDATAs, which will cause the previous data streams to be held.

The optional FLUSH operand releases any buffers that are queued, without displaying at the terminal.

Return code	Description
0	Data displayed
4	No data to display

DOMAX *domax-limit*

Specifies the maximum number of times that any single DO loop may be executed for each invocation of each level of script. This is to prevent any script from unintentionally looping forever. *domax-limit* may be any value in the range 1-9999 and this overrides any value that may have been specified on the SYSTEM statement. Only one DOMAX parameter may be specified in a level of script.

A DOMAX value specified at a high level has no effect on values specified for lower levels of script. Each value pertains to that level of script only.

If a WAITEVENT windows verb, or a WAITDATA verb, is executed in a DO UNTIL, DO FOREVER, or DO WHILE loop, the iteration is not counted towards the DOMAX count for that DO loop. However, all other DO loops are treated as normal. This means that in a window script, WAITEVENT should be specified in the never-ending high-level DO loop, not in an inner DO loop. If it is specified in an inner DO loop, the never-ending DO loop will exit with a DOMAX exceeded error if the number of iterations exceeds the specified or defaulted DOMAX value.

ENDSESS

Causes the current application session to be terminated unconditionally.

ENDSESS is usually placed at the end of a script. If, however, it is placed before the end of a script, the script process continues, but all parameters to communicate with the application are ignored, e.g., INPUT, WAITAPPL.

If the t_message variable has been assigned a value, when the ENDSESS verb is processed, the t_message value is displayed as the 'session complete' message on the Menu display. The session ended message sent to the Audit file is either 730 or 731, and contains the value of t_message.

Return code	Description
0	Session termination has started
8	Session could not be ended either because: ▪ it has not been defined ▪ it is a Command session ▪ it is not active.

Note This parameter can also be used in Application Builder scripts, where it can take an optional SESSION operand. See the section starting on page 113 for further details.

```
ERASE [Eof | All]
```

Default: Eof

Clears field or fields on the screen. The `Eof` parameter will clear the field in which the cursor is positioned, up to the end of the field. The `All` parameter will clear all unprotected input fields on the screen.

This script verb, together with the `HOME` and `TAB` verbs, remove the need to specify the SBA parameter on `INPUT` statements.

```
EXTRACT DATA variable-name
[FROM row | variable-name] [col | variable-name]
[TO col | variable-name]]
```

Enables data to be extracted from the data stream received from any application. The extracted data is stored in the specified variable, *variable-name*. A variable name must be supplied. The maximum amount of data that can be extracted using a single EXTRACT DATA verb is a screen row.

Location details of the data to be extracted are given in the following sub-parameters:

```
FROM row | variable-name [col | variable-name]
```

`FROM` specifies the screen row number from which the extract is to start, and optionally the column number within the row. The screen row number can be specified as a literal, *row*, or can be contained in a variable. If omitted, the default is screen row 1. The first screen column number of the extract can be specified as a literal, *col*, or can be contained in a variable. If omitted, the default is column 1.

```
TO col | variable-name
```

TO specifies the last screen column number of the extract. It can be specified as a literal, *col*, or can be contained in a variable. If omitted, the default is the last column on the screen row.

There may be occasions when the EXTRACT DATA verb is unable to extract all the data at the specified screen location. These are:

- from data streams which contain non-ascending 3270 orders
- from screens that are built up by several separate data streams.

In these circumstances, it may be necessary to make repeated extract attempts.

Note Most requirements can be met using the EXTRACT DATA verb. However, the above problems can be avoided by using the Application Builder verb READSESS, which uses MISER's screen image mapping. EXTRACT DATA can be used in preference where the overhead of MISER's screen image mapping is not justified.

Example:

```
SCRIPT extract
  Do forever
     Waitdata pass timeout 10
     Select                        /* on waitdata type */
     When t_waitdata = 'A' then    /* application input */
     If s_a > 40 and s_a = 65 then
        If NORESP then            /* session has timed out */
           Leave
        Else                       /* session data found */
           EXTRACT DATA lc_xn_id FROM 2 4 TO 12
                     /* get transaction ID from data stream */
                     /* starts at row 2 column 4 */
                     /* ends at row 2 column 12 */
            Select   /* transaction ID */
               When lc_xn_id = 'TESTXN' then
                  ...
               When lc_xn_id = 'PRODXN' then
                  ...
               Else
                  ...
            End
            DISPAPPL
         End
      Else                /* session has gone */
         Leave
      End
      ...
   End /* select on waitdata type */
End          /* do forever */
```

Return code	Description
0	Extract was successful
4	No application data has been received by the script
12	Incorrect FROM/TO specification.

HOME

Places the cursor at the beginning of the first (top left) input field.

This script verb, together with the ERASE and TAB verbs, remove the need to specify the SBA parameter on INPUT statements.

```
INPUT [PASS] | [[CURSOR row col]
   [KEY key | NO]
   [SBA [row | variable-name  col | variable-name] | [NO | OFF]]
   [TEXT text]]
```

The INPUT parameter and its various subparameters generate the actual 3270 data streams required for automatic input. As many INPUT parameters may be submitted as there are separate input streams in this script.

As many SBA and TEXT subparameters can be specified as are needed for a single input.

The various subparameters have the following meanings:

PASS

Causes a buffer, which has been received from a terminal and previously stored as a result of a WAITTERM PASS script parameter, to be passed to the application. The buffer is passed in the same format that it was received.

PASS may not be used with any other INPUT subparameters.

CURSOR *row col*

Causes Session Manager to simulate the cursor being at a particular location on the screen from which the input was supposed to have come. The two fields denoted by *row* and *col*, are the row and column specification which Session Manager converts to the necessary 3270 data stream address according to the terminal model number derived from the session logmode. If omitted, the default is row 1 column 1, which is the top left hand corner of the screen.

KEY *key* | NO

Simulates a specific key being pressed, or event happening, to cause the input. Valid specifications are ENTER, PF1 to PF24, PA1 to PA3, CLEAR, PEN (for the Selector Light Pen), MAGNET (for the Magnetic Slot Reader or Magnetic Hand Scanner), or OID (for the Operator Identification Card Reader). The default value is ENTER.

If KEY NO is specified the data is not sent to the application immediately, but is saved until the next 'INPUT key' parameter. This enables conditional statements to be placed within the INPUT stream specification. For example:

```
Input key no sba 1 2 text 'LOG'
If t_user = 'Arthur'
   Input key no sba 2 2 text 'DALEY'
Else
   Input key no sba 2 2 text 'ANYONE'
End
Input key enter sba 3 2 text 'BEGIN'
```

The resultant data stream when t_user is 'Arthur' is:

```
'LOG DALEY BEGIN'
```

Else it is:

```
'LOG ANYONE BEGIN'
```

SBA [*row* | *variable-name col* | *variable-name*] | [NO | OFF]

Simulates the SBA order which generally precedes input data passed to the application. It identifies the specific location on the screen from which the data is supposed to have come.

The two numbers denoted by *row* and *col* are the row and column specification. These may be specified by either numeric variables, or by decimal values. If *row* and *col* are omitted, the default is row 1 column 1, which is the top left hand corner of the screen.

When SBA is specified, it must appear immediately prior to the input TEXT subparameter to which it relates. The SBA and TEXT pair of subparameters can be repeated as often as necessary to simulate multiple input fields. Consecutive SBA orders can simulate erased fields.

Alternatively, specify SBA NO, or OFF, to generate no SBA field prior to the input TEXT. This facility is provided for those Installations which have applications that expect to read unformatted screens.

TEXT *text*

Specifies the text that is to be passed as input. It may be a literal, or a variable name. The variable will be substituted with its current value. The literal may contain a number of variable substitutions.

All variables may be used for substitution in the input stream. These are described in 'Session Manager variables' on page 197.

If the text is a literal it is assumed to end at the first space. If the literal includes embedded spaces, it should be enclosed in delimiters. Valid delimiters are: single quotes ('), double quotes ("), exclamation marks (!), question marks (?), or parentheses.

The text may be continued across as many input records as required, up to a maximum length of 80 bytes.

Return code	Description
0	Input generated successfully
4	Bad SBA specified for the terminal being used or for the terminal mode in use
8	Session is not active

INSCAN *inscan-string*
[FROM *location* [TO *location*] | AT *location*]

where:

location = *row* | *row-variable* [*col* | *col-variable*]

Scans the input data stream for a particular string. The data must have been captured by either a WAITDATA PASS or a WAITTERM PASS and is available for scanning until the next INPUT parameter.

The string may be defined as a literal or a variable name. If a variable, the current value is substituted. The maximum acceptable length is 256. The string is case-sensitive.

The search is performed on the data as it appears in the data stream, not as it was entered at the terminal. The scan assumes data streams are in the correct sequence, so if for any reason the data streams are out of sequence the results may be unpredictable.

The start position for the scan may be specified using the FROM operand to specify row and column. The default is 1.

The end position for the scan may be defined with the TO subparameter. Row and column are specified, the default is the maximum number of rows and columns for the screen.

A specific location for the scan may be defined by the AT parameter with row and column specified.

The entire data stream is searched when no FROM, TO, or AT subparameters are specified.

When the scan is successful, a condition, FOUND, is set which may be tested for using TPSL. An example of this is:

```
Waitterm pass
   /* Wait for terminal input */
   Inscan 'FRED'
   /* Look for specified text */
   If found
      /* Test for text being found */
         Do found processing
```

The variables s_inscan_col and s_inscan_row are set to the column and row number of the beginning of the matching text found in the data stream.

Return code	Description
0	Scan successful
4	No WAITTERM PASS outstanding

ISZCMD *literal | variable-name*

Issues a Session Manager command from a script. The command, which is issued with the user's AUTH value, can be specified as a literal in delimiters, or as a variable which contains the command to be issued. If the command specified is a Command script, the script is not run; instead, the corresponding Session Manager defined command is used. For example, if the COMMAND statement has specified:

```
COMMAND LOGOFF CMDSCRIPT YES
```

issuing ISZCMD LOGOFF does not cause the script 'logoff' to run, instead the LOGOFF command is executed by Session Manager directly.

If the command specified is a user-defined Command script, for example:

```
COMMAND BILL CMDSCRIPT YES
```

then the command is not recognized by Session Manager and the script will not be executed.

ISZCMD can also process session selection ids, as well as transids, enabling a session script to alter the visible session. For example:

Session Selection commands:

```
SESSION 20 start20 ...
```

Script contains:

```
ISZCMD 'start20'
```

Session numbers:

```
SESSION 20 ...
```

Script contains:

```
ISZCMD '20'
```

Key selection -

```
KEY PF4 ...
```

Script contains:

```
ISZCMD 'pf4'
```

See page 197 for a description of variable substitution.

Return code	Description
0	Command issued
4	Command rejected

The read/write `s_cmdret` variable determines whether, when a screen is closed that was opened as the result of a command issued using ISZCMD/A, the user returns to the session that invoked the command (if set to 'Y') or to the menu screen (if set to 'N'). The default is for the user to be returned to the menu screen.

If the user escapes from the ISZCMD/A-initiated screen, or the invoking session terminates before the user closes the screen, then the user will be returned to the menu screen regardless of the setting of this variable.

It is recommended that a `LET` is placed immediately before the ISZCMD/A statement if `s_cmdret` is to be set to 'Y', and that a `LET` placed immediately after the statement to reset `s_cmdret` to 'N'.

If session variable s_script_cmds is set to 'Y' then ISZCMD/A will support command scripts, as well as native Session Manager commands. It will do this by converting the ISZCMD/A into the equivalent of a TPSL call. Similarly, if s_script_cmds is set to 'Y' then `SAUTOSEQ` scripts can run with session scripts. It does this by temporarily converting a WAITTERM, WAITDATA or WAITAPPL into a `CALL` for the `SAUTOSEQ` script.

The script designer should be aware that the command/SAUTOSEQ script will use the same set of local variables as the invoking script. Similarly, it will use the same subscript variables (sub1, - sub9) as the invoking script. Consequently, the designer should ensure there is no conflict between these scripts. At a minimum it is advisable to save any subscripts used by the invoking script. It is for this reason the default value for s_script_cmds is 'N'.

ISZCMDA *literal* | *variable-name*

Same as ISZCMD (see above) except the command is issued with an AUTH of 9 rather than the user's AUTH value.

This command can only be invoked by authorized scripts, see 'Authorized scripts' on page 17. If invoked by a non-authorized script then ISZCMDA will fail with a return code of 4.

If session variable s_script_cmds is set to 'Y' then ISZCMD/A will support command scripts, as well as native Session Manager commands. It will do this by converting the ISZCMD/A into the equivalent of a TPSL call.

The script designer should be aware that the command/SAUTOSEQ script will use the same set of local variables as the invoking script. Similarly, it will use the same subscript variables (sub1, - sub9) as the invoking script. Consequently, the designer should ensure there is no conflict between these scripts; probably at a minimum it is advisable to save any subscripts used by the invoking script. It is for this reason the default value for s_script_cmds is

OUTSCAN *outscan-string*
[FROM *location* [TO *location*] | AT *location*]

where:

location = row | row-variable [col | col-variable]

Searches an application's output data stream for a particular string, regardless of whether the output was displayed at the terminal. The search is performed on the data as it appears in the data stream, not as it appears on the terminal when displayed. The string is case-sensitive.

The rules for specifying the OUTSCAN subparameters are the same as those for INSCAN.

When the scan is successful, a condition, FOUND, is set which may be tested for using TPSL. An example of this is:

```
Waitappl
   /* Wait for application output */
   Outscan 'ENTER USERID'
      /* Look for specified text */
   If found
      /* Test for text being found */
      Do found processing
```

The variables s_outscan_col and s_outscan_row are set to the column and row number of the beginning of the matching text found in the data stream.

Return code	Description
0	Scan successful

Return code	Description
4	No data to scan
8	Session not active

`PAUSE seconds | variable-name`

Specifies the number of seconds that the script is to pause, before executing the next verb. The number of seconds can be expressed as a literal, *seconds*, or as a variable. *seconds* can be any value in the range 1-32000 seconds. *variable-name*, contains the pause value in seconds. Indirect variable names may be used and non-numeric values are treated as a pause of zero seconds.

The parameter can be used in session and window scripts only, i.e., it is ignored in exit and command scripts.

Return code	Description
0	Script paused
4	Value specified was not numeric

`RPQLScriptauto [YES | NO]`

Default: No

This parameter defines what action the system is to take if a Read Partition Query List is detected from the application. If RPQLS Y is specified and a Read Partition Query List is detected from the application, the Query List dialogue will be handled by the system before returning to the script. In other words, all that is required is a `WAITAPPL` to intercept the RPQ List and everything else will be handled automatically.

If RPQLS N is specified (or defaulted), the script must handle the conversation using the method described for `SIMRPQ N` (see below).

`SCRIPT script-name`

Defines the name by which this script is known. A minimum of one alphanumeric character and up to a maximum of eight may be specified.

`SIMRPQ [YES | NO]`

Specifies the action Session Manager should take when a Read Partition Query (RPQ) is detected in the data stream from an application while this script is running.

If YES is specified or allowed to default, the script processor intercepts any Read Partition Queries and returns an automatic reply to the application. The Query List is not supported by SIMRPQ YES and users will need to use the `RPQLScriptauto` parameter instead.

When NO is specified, the script must cater for Read Partition Queries. The recommended method is:

```
WAITAPPL DISPLAY
WAITTERM PASS
INPUT PASS
```

These three parameters are described elsewhere in this chapter. The WAITAPPL DISPLAY causes a Read Partition Query to be sent to the terminal. The reply is intercepted by the WAITTERM PASS and the buffer containing the reply is sent to the application by the INPUT PASS.

The following SCRIPT operations, listed in alphabetical order, may be interspersed with TPSL statements.

Each parameter sets a return code at the time of execution, indicating the success or failure of the operation; this is held in t_rc. In some instances a result, held in t_result is also produced. These are detailed at the end of each parameter description.

`TAB [Forward | Backward]`

Default: Forward

This parameter will move the cursor to the next/previous unprotected field. The parameters Forward and Backward may be abbreviated to a single letter (F and B).

This script verb, together with the ERASE and HOME verbs, remove the need to specify the SBA parameter on INPUT statements.

`TDEQ` *`enqueue-value`*

Causes the session to dequeue the enqueue which was created by a corresponding TENQ *enqueue-value* parameter.

Return code	Description
0	Dequeue OK
4	No current enqueue
8	No dequeue value
12	No previous TENQ

`TENQ` *`enqueue-value`*

Causes the session to enqueue on the *enqueue-value*. This value can be a literal or a variable containing the value. The TENQ and TDEQ parameter pair can be used to enclose script statements that need to be protected from concurrent execution, for example where a global variable is updated.

Return code	Description
0	Enqueue OK
8	No enqueue value

`TRACEOFF`

Indicates the point at which tracing is to terminate. Tracing is only used when the TTPSL command is issued, with the TRACEVERB operand in effect. Refer to the 'System Operator Commands' section in the *Technical Reference*.

No return codes are set by this parameter.

`TRACEON`

Indicates the start point from which tracing commences. Tracing is only used when the TTPSL command is issued. Refer to the 'System Operator Commands' section in the *Technical Reference.*

No return codes are set for this parameter

`USERMSG PANEL usermsg-panel-name | TEXT usermsg-text`

Formats a data stream and sends it to a terminal. It is created as a 3270 ERASE WRITE data stream.

PANEL specifies the name of a panel that is processed to create the data stream.

TEXT defines the text to be included in the data stream. It may be a variable name or a literal. The literal may contain a number of variable substitutions.

Either TEXT or PANEL must be specified, there is no default.

The return code set is always zero.

`VCALL variable-name | 'script-name'`

VCALL is functionally the same as the CALL parameter except that VCALL is conditional and the script name is supplied either in a variable, or specified in quotes if the actual script name is supplied. If the script exists, then on return from the script t_rc will be set to 0. However, if the script does not exist then t_rc will be set to 4. For example:

```
let lc_name = 'newscript'
vcall lc_name
if t_rc = 4
   auditmsg '&lc_name& not found'
end
```

or

```
vcall 'newscript'
if t_rc = 4
   auditmsg 'newscript not found'
end
```

`WAITAPPL [DISPLAY  [Yes | No | ON | OFF]] [TIMEOUT nnn] [KEEP]`

Causes the script to stop and wait for output from the application. This is useful when processing is dependent on the contents of the output. The OUTSCAN parameter may be used to search for a particular string in the output. The EXTRACT DATA verb can also be used.

DISPLAY specifies what Session Manager does with the output screen. DISPLAY, DISPLAY YES, and DISPLAY ON cause it to be displayed at the terminal, DISPLAY NO, and DISPLAY OFF, cause the output to be suppressed. The output can be displayed later in the script by using the DISPAPPL verb.

When WAITAPPL is specified without any operands, the last WAITAPPL specified defaults to DISPLAY YES. All previous WAITAPPLs default to DISPLAY NO.

The APPLTIME parameter, described previously, may be used to set a time limit during which the script will wait for a response. If this limit is reached, control is passed to the parameter following the WAITAPPL and the special variable NORESP is set on. This variable may then be tested using TPSL statements. NORESP is automatically set OFF prior to a WAITAPPL and therefore does not need to be reset.

TIMEOUT overrides the value of APPLTIME for this WAITAPPL parameter. The TIMEOUT value represents the number of seconds. If TIMEOUT is not specified, then the value on the APPLTIME parameter is used.

The KEEP operand retains any buffers queued as a result of a previous WAITAPPL. The OUTSCAN parameter can look for a particular string in these retained buffers and they may be sent to a terminal by a DISPAPPL. The default is to release any previous buffers. Because it is previous buffers that are released, the first WAITAPPL need not specify KEEP.

Return code	Description
0	Successful completion
8	Session not active

Provided the s_script_cmds variable is set to 'Y' (the default is 'N') before a WAITTERM, WAITAPPL or WAITDATA, then SAUTOSEQ scripts can run with the session scripts. It does this by temporarily converting a WAITTERM, WAITDATA or WAITAPPL into a CALL for the SAUTOSEQ script. The SAUTOSEQ will run before the PROCESS section of any displayed panel. If s_script_cmds is set to 'N' (the default), then SAUTOSEQ scripts do not run when a session script is active.

The script designer should be aware that the SAUTOSEQ script will use the same set of local variables as the invoking script. Similarly, it will use the same subscript variables (sub1, - sub9) as the invoking script. Consequently, the designer should ensure there is no conflict between these scripts. At a minimum it is advisable to save any subscripts used by the invoking script. It is for this reason that the default value for s_script_cmds is 'N'.

Note If s_script_cmds is set to 'Y' then ISZCMD or ISZCMDA will support command scripts as well as native Session Manager commands.

```
WAITDATA [PASS] [TIMEOUT nnn] [KEEP]
```

Monitors a session's terminal inputs, application outputs, and inputs from a SENDDATA verb (when the Application Builder feature is enabled). It returns control to the next script parameter when either input is received from the terminal or a SENDDATA verb, or if the application sends an output stream.

The session variable t_waitdata is set to 'A' to indicate application output, 'T' to indicate terminal input, or 'S' to indicate SENDDATA received.

The PASS subparameter only applies to terminal input. See the description of the WAITTERM parameter for further details of usage.

When t_waitdata is 'A', the application output data stream is stored and passed to the next DISPAPPL parameter so that it can be displayed at the terminal. OUTSCAN can be used to scan the output data stream before it is sent to the terminal.

TIMEOUT overrides the value of APPLTIME for this WAITDATA parameter. The TIMEOUT value represents the number of seconds. If TIMEOUT is not specified, the value on the APPLTIME parameter is used. When an application timeout is detected, the NORESP condition is set on and t_waitdata is set to 'A'.

The timer is set on only after an INPUT parameter has sent data to the application. It is reset when the first application data is received in response to the INPUT data. Data received from the application, but not in response to INPUT data, is not timed.

The KEEP operand retains any buffers queued as a result of a previous WAITDATA. The OUTSCAN parameter can look for a particular string in these retained buffers and they may be sent to a terminal by a DISPAPPL. The default is to release any previous buffers. Because it is previous buffers that are released, the first WAITDATA need not specify KEEP.

For details of actions performed when t_waitdata is set to 'S', see the following section, 'Application builder verbs' on page 113.

When WAITDATA is specified in a DO WHILE, DO UNTIL, or DO FOREVER loop, the Do iteration does not count towards the DOMAX count for that loop. Other loops are treated normally, so it is possible that if WAITDATA is coded in an inner loop, the DOMAX count may be exceeded. WAITDATA should be specified in a outer never-ending loop.

Refer to page 28 for an example of WAITDATA in a script.

The return code t_rc is always set to zero.

Provided the s_script_cmds variable is set to 'Y' (the default is 'N') before a WAITTERM, WAITAPPL or WAITDATA, then SAUTOSEQ scripts can run with the session scripts. It does this by temporarily converting a WAITTERM, WAITDATA or WAITAPPL into a CALL for the SAUTOSEQ script. The SAUTOSEQ will run before the PROCESS section of any displayed panel. If s_script_cmds is set to 'N' (the default), then SAUTOSEQ scripts do not run when a session script is active.

The script designer should be aware that the SAUTOSEQ script will use the same set of local variables as the invoking script. Similarly, it will use the same subscript variables (sub1, - sub9) as the invoking script. Consequently, the designer should ensure there is no conflict between these scripts. At a minimum it is advisable to save any subscripts used by the invoking script. It is for this reason that the default value for s_script_cmds is 'N'.

Note If s_script_cmds is set to 'Y' then ISZCMD or ISZCMDA will support command scripts as well as native Session Manager commands.

WAITTERM [PASS]

Causes the script to wait for input from the terminal before continuing.

PASS should be coded when input is expected from an application screen. In other cases a response is expected from a previous USERMSG parameter and PASS should **not** be specified.

When the response is from a USERMSG PANEL, the panel definition is used to map input from the terminal into any panel variables. When the response is from a USERMSG TEXT, the first field in the input stream is placed in the Session Manager supplied variable, t_command.

When the response is from an application screen, the buffer from the terminal is stored and passed to the next INPUT parameter that specifies PASS as a subparameter. This method enables input from a terminal to be passed unchanged to a script and from the script to an application. The INSCAN verb can be used to scan the entered data. This is a useful technique when an application issues a command such as a Read Buffer. The data length can be found in the variable t_inplen.

The return code t_rc is always set to zero.

Provided the s_script_cmds variable is set to 'Y' (the default is 'N') before a WAITTERM, WAITAPPL or WAITDATA, then SAUTOSEQ scripts can run with the session scripts. It does this by temporarily converting a WAITTERM, WAITDATA or WAITAPPL into a CALL for the SAUTOSEQ script. The SAUTOSEQ will run before the PROCESS section of any displayed panel. If s_script_cmds is set to 'N' (the default), then SAUTOSEQ scripts do not run when a session script is active.

The script designer should be aware that the SAUTOSEQ script will use the same set of local variables as the invoking script. Similarly, it will use the same subscript variables (sub1, - sub9) as the invoking script. Consequently, the designer should ensure there is no conflict between these scripts. At a minimum it is advisable to save any subscripts used by the invoking script. It is for this reason that the default value for s_script_cmds is 'N'.

Note If s_script_cmds is set to 'Y' then ISZCMD or ISZCMDA will support command scripts as well as native Session Manager commands.

Application builder verbs

This section describes all the Application Builder script parameters (otherwise known as verbs). These verbs can be used with other ordinary session script parameters to define scripts that enable you to build a new application. A general description of the Application Builder feature can be found in the *Installation and Customization* manual.

Parameter summary

The following is a summary of the parameters that enable you to build a new application, by combining two or more of your existing applications:

Parameter	Description
ENDSESS	End any active session. (The SESSION operand is only available when used in Application Builder scripts. The ENDSESS parameter with no operands may be used in ordinary session scripts).
HALTSCRIPT	Halt the script currently running on a session.
READSESS	Transfer all, or part, of the current session image into a variable.
RUNSCRIPT	Cause a script to be run on a started session.
SENDDATA	Send data to a script running on another session.
SENDVAL	Synonym of SENDDATA.

Parameter	Description
STARTSESS	Start another session (as configured on a USER, PROFILE, or TERMINAL statement).
WAITDATA	Wait for data to arrive from a script running on another session. (The use of WAITDATA to monitor the arrival of data sent by using the SENDDATA verb is only available in Application Builder scripts. The WAITDATA parameter to monitor application and terminal inputs and outputs may be used in ordinary session scripts).
XYZSCREEN	Transfer all or part of the current session image into a variable.

Each parameter sets a return code at the time of execution, indicating the success or failure of the operation; this is held in t_rc. In some instances a result, held in t_result is also produced. These are detailed at the end of each parameter description.

Note The Application Builder feature must be enabled on your system before these verbs can be used. If the feature is not enabled, then message 420 '`APPLBLD feature not supported - related parameters ignored`' is issued and t_rc is set to -8.

Parameter details

`ENDSESS [SESSION session-id]`

If the optional SESSION operand is specified, it causes the named application session, *session-id*, to terminate unconditionally. *session-id* may be specified as a literal or a variable. The value must be *one* of the following as coded in an existing application session definition:

- the number following a SESSION keyword, e.g., 25
- the key name following a KEY keyword, e.g., PF4
- any of the multiple selection command sequences on a session, e.g., CICS.

It is usually placed at the end of a script. If, however, it is placed before the end of a script, the script process continues, but any further parameters that communicate with the application are ignored, e.g., INPUT, or WAITAPPL.

If the t_message variable has been assigned a value, when the ENDSESS verb is processed, the t_message value is displayed as the 'session complete' message on the Menu display. The session ended message sent to the Audit file is either 730 or 731, and contains the value of t_message.

The SESSION operand is only available in Application Builder scripts.

Return code	Description
-8	Application Builder feature not enabled
0	Session termination has begun
8	Session could not be ended either because: ▪ it has not been defined ▪ it is a Command session ▪ it is not active

`HALTSCRIPT session-id`

HALTSCRIPT causes the script executing on the specified session, *session-id*, to be terminated. If the script that is executing is an Endscript, the Haltscript causes immediate session termination. For other types of script, the session remains active.

session-id may be specified as a literal or a variable. The value must be *one* of the following as coded in an existing application session definition:

- the number following a SESSION keyword, e.g., 25;
- the key name following a KEY keyword, e.g., PF4;
- any of the multiple selection command sequences on a session, e.g., CICS.

Return code	Description
-8	Application Builder feature not enabled
0	Script termination has begun
4	Script has not terminated either because: ▪ session has not been defined ▪ it is a Command session ▪ session is not currently running a script

```
READSESS variable-name
   [FROM row | row-variable [col | col-variable] ]
   [TO col | col-variable]
```

READSESS extracts from the session image and stores in a user-defined character variable.

If *row* is not specified, row 1 on the screen is assumed. If a FROM *col* is not specified, the first column of the row is assumed. If a TO *col* is not specified, then the last column of the row is assumed. Any attributes or graphic escape characters are converted to blanks.

The READSESS parameter requires MISER's screen image buffering. If a script contains a READSESS parameter, then Session Manager starts MISER for that session when the script starts. However, the READSESS parameter does not require the full MISER feature to function. If image buffering was started by a READSESS parameter it will be stopped when the script containing this first READSESS parameter completes and the next data stream is sent to the terminal. It is advisable therefore to put the first READSESS parameter in a top level script, to prevent the overhead of repeatedly starting and stopping the image buffering.

Return code	Description
-8	Application Builder feature not enabled
0	Session image read

Return code	Description
4	Either all or part of the screen line specified by the FROM and TO operands lies outside of the current screen size. This may be because the script was intended to be run on a session started as a different terminal model, or because the script expects the terminal to be in Alternate mode, whereas the application has placed it in Normal mode. Or, the FROM and TO values have been specified the wrong way round; the FROM value exceeds the TO value.
8	There was no data available to satisfy the READSESS, i.e., there has been no screen output since MISER was started for the session.

`RUNSCRIPT` *`session-id script-name`*

RUNSCRIPT causes the named script, *`script-name`*, to be run on the started session, specified as *`session-id`*. The script runs as if it had been invoked as an Autoscript on the session.

`session-id` may be specified as a literal or a variable. The value must be **one** of the following as coded in an existing application session definition:

- the number following a SESSION keyword, e.g., 25
- the key name following a KEY keyword, e.g., PF4
- any of the multiple selection command sequences on a session, e.g., CICS.

Return code	Description
-8	Application Builder feature not enabled
0	Script has been initialized on the specified session
4	The script could not be run either because: ▪ another script is already running on the specified session ▪ session has not been defined ▪ it is a Command session ▪ session is not active

`SENDDATA [`*`variable-name | literal`*` ] ...`
`  SESSION `*`session-id`*` | RPARTNER`

SENDDATA causes data to be passed to a script running on another session. The values in the named *`variable-name`*s and/or literals are transmitted to the specified session, *`session-id`*. The values will be set in the target session as t_sendval.1-*n* variables and the session will resume processing after its current Waitdata (or next Waitdata, if not currently waiting).

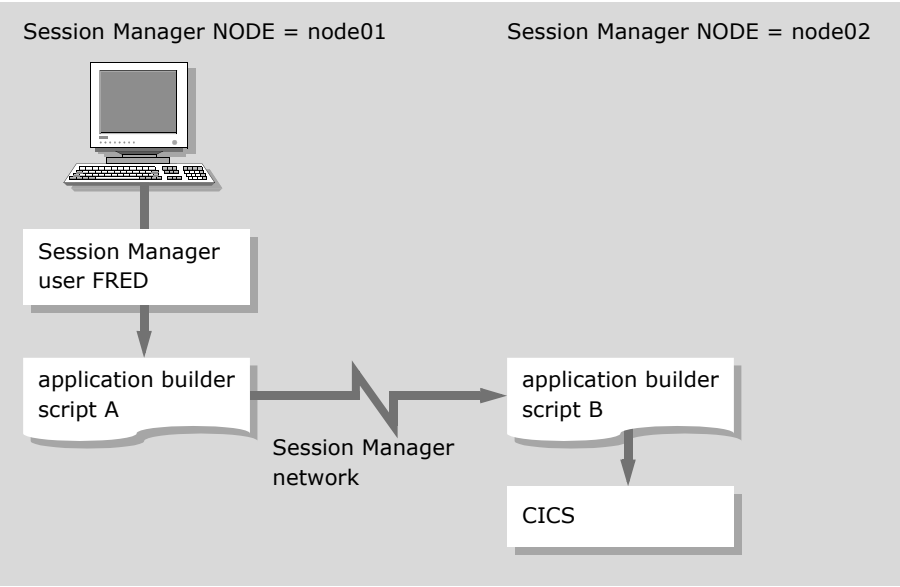
Up to ten values can be sent, *each* one must be either a variable or a literal.

`session-id` may be specified as a literal or a variable. The value must be **one** of the following as coded in an existing application session definition:

- the number following a SESSION keyword, e.g., 25
- the key name following a KEY keyword, e.g., PF4
- any of the multiple selection-ids on a session, e.g., CICS.

The RPARTNER operand can be used when running remote sessions across the Session Manager network. It enables communication between an Application Builder script running on the local node and an Application Builder script running on the remote node (its partner).

For example, in the following diagram, the user FRED has signed on to a terminal at NODE01 and started a remote session in NODE02. Script A has been specified as the local Startscript, by defining it in the definitions. Script B has been specified as a remote Startscript, by defining it in the appropriate APPL definition. By specifying the RPARTNER operand on the SENDDATA verb, in place of the SESSION operand, Script A can send data through the network to Script B. In exactly the same manner, Script B can send data through the network to Script A.



Return code	Description
-8	Application Builder feature not enabled
0	Values sent
4	Data could not be sent to the specified session either because: ▪ session has not been defined ▪ it is a Command session ▪ it is not active ▪ RPARTNER was specified, but the session has no partner

Note A return code of zero indicates that the data was sent to the session; it does *not* mean that a Waitdata has been satisfied (e.g., the receiving script running on the specified session might be in a Waitterm at that point in time). If confirmation of receipt is required, the scripts must be coded in such a way as to agree a confirmation protocol.

`STARTSESS session-id`

STARTSESS causes the specified session, *session-id*, to be started. Any associated Initscript and/or Startscript is run as if the session had been started from the Menu screen.

session-id may be specified as a literal or a variable. The value must be **one** of the following, as coded in an existing application session definition:

- the number following a SESSION keyword, e.g., 25
- the key name following a KEY keyword, e.g., PF4
- any of the multiple selection-ids on a session, e.g., CICS.

Return code	Description
-8	Application Builder feature not enabled
0	Session has started
4	Session could not be started either because: ▪ it has not been defined ▪ it is a Command session ▪ it is already active

`WAITDATA [PASS] [TIMEOUT nnn] KEEP`

WAITDATA monitors a session's terminal inputs, application outputs, and inputs from a SENDDATA verb. It returns control to the next script parameter when either input is received from the terminal, from a SENDDATA, or the application sends an output stream.

The session variable t_waitdata is set to 'A' to indicate application output, 'T' to indicate terminal input, or 'S' to indicate SENDDATA.

If a SENDDATA verb is directed at this session, t_waitdata is set to 'S'. In this case, t_sendvals is set to the number of values sent and t_sendval.1 to t_sendval.*n* contain the individual values. See the description of the SENDDATA verb earlier in this section.

In all other respects the WAITDATA verb is actioned as described on page 111.

The return code t_rc is always set to 0.

```
XYZSCREEN variable-name
   [FROM row | row-variable [col | col-variable] ]
   [TO col | col-variable]
```

XYZSCREEN extracts all or part of the current session image and stores it in a user-defined character variable.

If *row* is not specified, row 1 on the screen is assumed. If a FROM *col* is not specified, the first column of the row is assumed. If a TO *col* is not specified, then the last column of the row is assumed. Any attributes or graphic escape characters are converted to blanks.

XYZSCREEN performs the same function as READSESS, but does not require MISER screen image buffering.

Return code	Description
-8	Application Builder feature not enabled
0	Session image read
4	Either all or part of the screen line specified by the FROM and TO operands lies outside of the current screen size. This may be because the script was intended to be run on a session started as a different terminal model, or because the script expects the terminal to be in Alternate mode, whereas the application has placed it in Normal mode. Or, the FROM and TO values have been specified the wrong way round; the FROM value exceeds the TO value.

Window verbs

This section describes all the window-related script parameters, otherwise known as window verbs. Sample Window scripts are distributed in source form on the product tape, and a general description of the Session Manager Windows feature can be found in the *User and Administrator* manual.

The verbs are described in alphabetical order, with specific usage notes and return codes following each description.

When specifying a window verb, any parameter that is specified as a character literal, such as a window name, should be enclosed in delimiters, otherwise the literal is interpreted as a variable. As an example:

```
WACTIVATE WINDOW 'WIN1'
```

should be specified.

Valid delimiters are:

- 'this is delimited'
- "this is delimited"
- (this is delimited)
- !this is delimited!
- ?this is delimited?

WACTIVATE: Make a window active

Causes an inactive window to become active.

```
WACTIVATE WINDOW window-identifier | window-name
```

Parameters

`WINDOW window-identifier | window-name`

Specifies either the identifier, as assigned by the WOPEN operation, or the name of the window or window control area, to be made active.

Either may be specified as a literal value, or as a variable containing the identifier or name of the window, or window control area.

This parameter must be specified.

WACTIVATE usage notes

The WACTIVATE parameter has no effect if the window identifier or name refers to the currently active window.

The WACTIVATE operation does not cause the window to become fully visible. To make a window fully visible the WPOP parameter must be used; refer to page 136.

WACTIVATE return codes

The success or failure of the WACTIVATE operation is indicated by a return code in the variable t_rc. A list of possible values is given below:

Return code	Description
-4	Window environment not active.
0	Successful completion.
4	WINDOW parameter invalid. One of the following errors was encountered: ▪ Window identifier not in range 1-255. ▪ Window name contains all blanks. ▪ No window with the specified identifier or name exists.
44	Operation invalid on a window control area.

WAITEVENT: Wait for window event

Causes script processing to be suspended until one of a list of nominated events occurs. One of the parameters must be specified.

```
WAITEVENT ESCAPE | MSG | SESSEND | SWITCH
```

Parameters

`ESCAPE`

Specifies that the script is to resume processing when one of the windowing escape sequences, defined in the subscripted variable t_w_esc, is entered at the terminal, or input is received from the terminal with the cursor located in a window control area.

`SWITCH`

Specifies that the script is to resume processing when a windowmsg panel changes to a Spy, View, Query, Dlog, or Qtask display.

`MSG`

Specifies that the script is to resume processing when a message arrives, due to a MSG or BROADCAST command having been issued.

SESSEND

Specifies that the script is to resume processing when a windowed session terminates.

WAITEVENT usage notes

After completion of the WAITEVENT parameter, the variable t_w_event may be interrogated to determine which event caused script processing to resume.

For the ESCAPE event, the escape sequence, if any, is placed in the variable t_w_input.

For the MSG event, the text of the message or broadcast is placed in the variable t_w_msg.

For the SESSEND and SWITCH events, the window identifier associated with the session which has terminated or switched is placed in the variable t_result.

When WAITEVENT is specified in a DO WHILE, DO UNTIL, or DO FOREVER loop, the do iteration does not count towards the DOMAX count for that loop. Other loops are treated normally, so it is possible that if WAITEVENT is coded in an inner loop, the DOMAX count may be exceeded. WAITEVENT should be specified in an outer never-ending loop.

WAITEVENT return codes

The success or failure of the WAITEVENT operation is indicated by a return code in the variable t_rc. A list of possible values is given below:

Return code	Description
-4	Window environment not active.
0	Successful completion.

WALTER: Modify a window

Alters a window or window control area according to the specified characteristics.

```
WALTER [WINDOW window-identifier | window-name]
[ORIGIN ROW row COLumn col | OFFSET offset]
[ENDPOS ROW row COLumn col | OFFSET offset]
[SIZE DEPTH depth WIDTH width]
[NAME window-name]
[BORDER border-set-name | NONE]
[OWNER window-identifier | window-name | NONE]
[REMATTr [Yes | No | ON | OFF]]
[   SESSION session-number | session-pfkey
 [OUTPUT [Yes | No | ON | OFF]]
or  WINPANEL panel-name
or  WINTEXT message-text  ]
```

Parameters

WINDOW *window-identifier* | *window-name*

Specifies either the identifier, as assigned by the WOPEN operation, or the name of the window or window control area which is to be altered.

The parameter value may be specified as either a literal value, or as a variable containing the identifier or name of the window or window control area to be altered.

If the WINDOW parameter is omitted, the currently active window is altered.

For details of the remaining parameters, refer to the WOPEN parameter on page 132.

WALTER usage notes

Only the specified characteristics are altered, all other characteristics remain unchanged.

The size of a window or window control area must be defined such that the amount of session output, or window message, visible inside the window is at least one row by one column. When defining the size of a window, allowance should be made for the following:

- The depth of the top border (1 byte).
- The depth of the bottom border (1 byte).
- The attribute byte for the left vertical border.
- The width of the left border (1 byte).
- The attribute byte for the window contents.
- The attribute byte for the right vertical border.
- The width of the right border (1 byte).

If a window control area is being altered, the SESSION and OUTPUT parameters should not be specified.

The SESSION, WINPANEL, and WINTEXT parameters are mutually exclusive.

The ENDPOS and SIZE parameters are mutually exclusive.

WALTER return codes

The success or failure of the WALTER operation is indicated by a return code in the variable t_rc. A list of possible values is given below:

Return code	Description
-4	Window environment not active.
0	Successful completion.

Return code	Description
4	WINDOW parameter invalid. One of the following errors was encountered: ▪ Window identifier not in range 1-255. ▪ Window name contains all blanks. ▪ No window with the specified identifier or name exists. ▪ The WINDOW parameter was omitted, indicating that the operation should be performed on the active window, but no window is active.
8	ORIGIN parameter invalid. The ROW, COLUMN, or OFFSET value is out of range for this terminal.
12	ENDPOS parameter invalid. The ROW, COLUMN, or OFFSET value is out of range for this terminal.
16	SIZE parameter invalid. The WIDTH or DEPTH value is out of range for this terminal.
20	NAME parameter invalid. One of the following errors was encountered: ▪ Name contains all blanks. ▪ Name already in use in another window or window control area.
24	BORDER parameter invalid. One of the following errors was encountered: ▪ Border set name contains all blanks. ▪ No border set with the specified name exists.
28	OWNER parameter invalid. One of the following errors was encountered: ▪ Window identifier not in range 1-255. ▪ Window name contains all blanks. ▪ No window with the specified identifier or name exists. ▪ The specified window is already an owner.
32	SESSION parameter invalid. One of the following errors was encountered: ▪ Session number not in range 1-9999. ▪ Session number is not defined on the user's menu. ▪ Session number refers to a command session.
36	WINPANEL parameter invalid. The specified panel does not exist.
44	Operation invalid on a window control area.

Return code	Description
48	Positioning or size error. One of the following errors was encountered: ▪ Part of the window falls outside the bounds of the current terminal, due to an invalid combination of ORIGIN, ENDPOS, and SIZE values. ▪ The window size is insufficient to display at least one row and one column of session output or window message.

WBACKGROUND: Set screen background

Defines the characteristics of the screen area not occupied by any window or window control area.

```
WBACKGROUND CHAR background-character]
   [attr-list | REFATTR attr-character | variable-name]
```

Parameters

`CHAR background-character`

Specifies a single character which is to fill any position on the screen not occupied by a window or window control area. It may be specified as either a character literal value, or as a character-type variable containing a character.

`attr-list | REFATTR attr-character | variable-name`

Specifies the field attributes of the background area. Refer to 'Field Attributes' earlier in this manual.

WBACKGROUND usage notes

The default background character is a blank. The default attributes are protected, normal intensity, and no extended highlighting.

WBACKGROUND return codes

The success or failure of the WBACKGROUND operation is indicated by a return code in the variable t_rc. A list of possible values is given below:

Return code	Description
-4	Window environment not active.
0	Successful completion.

WBORDER: Define a window border set

Defines the characteristics of a border set.

A border set consists of the characters used to make up the vertical borders, horizontal borders, corners, and their respective field attributes. Text may also be included, for example, an action bar which is to appear in horizontal borders.

```
WBORDER NAME border-set-name
   [TOP [CHAR border-character]
   [attr-list | REFATTR [attr-character | variable name]]
   [TEXT
      [COLumn col | variable-name]
      [FIELD variable[(nnn)] | literal
         [attr-list | REFATTR [attr-character | variable-name]]]]
      [NONE]]
   [BOTTOM (same as TOP parameter) ]
   [LEFT (same as TOP parameter, except TEXT not allowed) ]
   [RIGHT (same as TOP parameter, except TEXT not allowed) ]
   [DEFAULT (same as TOP parameter, except TEXT not allowed) ]
   [CORNER CHAR corner-character]
   [BOX]
```

Parameters

`NAME` *`border-set-name`*

Specifies a name by which the border set can be referred to in subsequent window operations. The border set name can be 1 to 8 characters in length, and may be specified as either a character literal value, or as a character-type variable containing the border set name.

`TOP`

Specifies the characteristics of the top border:

`CHAR` *`border-character`*

Specifies the character to be used to build the top border, and may be specified as either a character literal value, or as a character-type variable containing the character.

`attr-list` | `REFATTR` [*`attr-character`* | *`variable-name`*]

Specifies the field attributes of the top border. Refer to ‘Field Attributes’, earlier in this manual.

`TEXT`

Indicates that text is to appear in the top border:

`COLumn` *`col`* | *`variable-name`*

Specifies the position of the attribute byte in the next field, relative to the first column of the border. This can either be a numeric variable, or a decimal. If omitted, the next available column after the current field is used. A value from 1 to 132 can be specified.

`FIELD` *`variable`* [(*`nnn`*)] | *`literal`* [*`attr-list`* | `REFATTR` *`attr-character`* | *`variable-name`*]]]

Specifies a single field which is to be embedded in the top border, and optionally defines the field attributes to be used. If the field is specified as a variable, it is advisable to follow it by a length. This ensures that the field always has the same length even though its value may vary in length. Otherwise, subsequent fields in the border will move according to the length of the value. Refer to 'Field Attributes' earlier in this manual.

`NONE`

Indicates that a top border is not required. When specified, it must be the only parameter.

`BOTTOM`

Specifies the characteristics of the bottom border. The syntax of the BOTTOM parameter is identical to that of the TOP parameter.

`LEFT`

Specifies the characteristics of the left border. The syntax of the LEFT parameter is similar to that of the TOP parameter, the only difference being that the TEXT sub-parameter may not be specified.

`RIGHT`

Specifies the characteristics of the right border. The syntax of the RIGHT parameter is identical to that of the LEFT parameter.

`DEFAULT`

Specifies defaults for the current border set definition. This may then be overridden for individual components of the border set as required.

The syntax of the DEFAULT parameter is identical to that of the LEFT parameter.

If the DEFAULT parameter is not specified, standard defaults are used for any part of the border set which is not explicitly defined. The default CHAR value is a minus sign ('-') for horizontal borders, and a vertical bar ('|') for vertical borders. The default field attributes are protected, normal intensity, and default colour.

`CORNER CHAR` *`corner-character`*

Specifies the character to be used where horizontal and vertical borders meet. It can be a character literal or variable. The default, if the CORNER parameter is omitted, is a plus sign ('+').

`BOX`

For terminals which are capable of showing a continuous line, this parameter causes a continuous line border to be displayed. For other terminals, any other definitions for the border characters in the WBORDER statement will be used.

For example, with the following definition:

```
Wborder name 'DIM'
Top Text Field t_user Field in 'IVY.1'   unprot white
        char '-' prot blue
bottom char '-' prot blue
left   char ':' prot blue
right  char ':' prot blue
corner char '+'
box
```

a continuous line box border will be displayed in blue on a terminal with appropriate capability, but on a simpler terminal the border characters will be '-' for top and bottom, and ':' for left and right, with '+' at the corners.

WBORDER usage notes

The NAME parameter must be specified.

If a horizontal border is defined with text, this text is overlayed onto the standard border, thus any portions of the border without text contain the standard border character. The first available position for text is the character following the corner character.

A corner character is inserted at any points where a horizontal and vertical border meet. The field attributes used are the same as those of the adjoining horizontal border.

If using REFATTR, the attribute symbol must be defined on the SYSTEM statement. An alternative attribute symbol can be used by assigning the symbol to a variable using the LET TPSL statement. For example:

```
Let uc_attr = A'%'
```

ATTR parameters are not valid in a script.

WBORDER return codes

The success or failure of the WBORDER operation is indicated by a return code in the variable t_rc. A list of possible values are given below:

Return code	Description
-4	Window environment not active.
0	Successful completion.
20	NAME parameter invalid. One of the following errors was encountered: ▪ Name contains all blanks. ▪ Name already in use in another border set. ▪ 'BOX' was specified as the border set name.

WCLOSE: Close a window

Closes a window or window control area.

```
WCLOSE [WINDOW window-identifier | window-name]
```

Parameters

`WINDOW window-identifier | window-name`

Specifies either the identifier, as assigned by the WOPEN operation, or the name of the window or window control area to be closed.

The parameter value may be specified as either a literal value, or as a variable containing the identifier or name of the window or window control.

If the WINDOW parameter is omitted, the currently active window is closed.

WCLOSE usage notes

If the WCLOSE operation causes the currently active window to be closed, the most recently accessed window is automatically made active, as though a WACTIVATE operation had been performed.

After the last remaining window has been closed, a value of 4 is placed in variable t_result.

WCLOSE return codes

The success or failure of the WCLOSE operation is indicated by a return code in the variable t_rc. A list of possible values is given below:

Return code	Description
-4	Window environment not active.
0	Successful completion.
4	WINDOW parameter invalid. One of the following errors was encountered: ▪ Window identifier not in range 1-255. ▪ Window name contains all blanks. ▪ No window with the specified identifier or name exists. ▪ The WINDOW parameter was omitted, indicating that the operation should be performed on the active window, but no window is active.

In addition, when a return code of 0 is set, the variable t_result has the following values set:

Return code	Description
0	Successful completion.
4	There are no remaining windows.

WHIDE: Hide a window

Hides a window, or window control area, so that it becomes invisible.

```
WHIDE [WINDOW window-identifier | window-name]
```

Parameters

`WINDOW window-identifier | window-name`

Specifies either the identifier, as assigned by the WOPEN operation, or the name of the window or window control area which is to be hidden. It may be specified as either a literal value, or as a variable containing the identifier or name of the window or window control area.

If the WINDOW parameter is omitted, the currently active window is hidden.

WHIDE usage notes

If the currently active window is hidden, it remains active after the WHIDE operation.

If the specified window is already hidden, WHIDE has no effect.

If the specified window is zoomed, it must be returned to its normal size, by using the WNORM parameter, before it can be hidden.

WHIDE return codes

The success or failure of the WHIDE operation is indicated by a return code in the variable t_rc. Possible values are:

Return code	Description
-4	Window environment not active.
0	Successful completion.
4	WINDOW parameter invalid. Error could be: ▪ Window identifier not in range 1-255. ▪ Window name contains all blanks. ▪ No window with specified identifier or name exists. ▪ The WINDOW parameter was omitted, indicating that the operation should be performed on the active window, but no window is active.
60	Operation invalid on a zoomed window.

WIDENTIFY: Identifier of a window

Enables the identifier, as assigned by the WOPEN operation, of a window to be determined from the cursor position, a screen location, or a window or name.

```
WIDENTIFY [CURSOR]
   [LOCATION ROW row COLumn col | OFFSET offset]
   [WINDOW window-identifier | window-name]
```

Parameters

`CURSOR`

Indicates that the identifier of the window at the cursor position is required. This is the default when no parameter is specified.

`LOCATION ROW row COLumn col | OFFSET offset`

Indicates that the identifier of the window at the specified screen location is required.

The screen location may be specified in either row and column format, or as a screen offset. The subparameters may be specified as either numeric literal values, or as numeric-type variables containing the values to be used.

Note If WIDENTIFY is performed to a position where a window overlaps, the results are unpredictable.

`WINDOW window-identifier | window-name`

Specifies either the identifier, as assigned by the WOPEN operation, or the name of the window whose identifier is required. It may be specified as either a literal value, or as a variable containing the identifier or name of the window. The WINDOW parameter specified with a window identifier can be used to determine whether a particular window is open.

WIDENTIFY return codes

The success or failure of the WIDENTIFY operation is indicated by a return code in the variable t_rc. A list of possible values is given below:

Return code	Description
-4	Window environment not active.
0	Successful completion.
4	WINDOW parameter invalid. One of the following errors was encountered: ▪ Window identifier not in range 1-255. ▪ Window name contains all blanks. ▪ No window with the specified identifier or name exists.
8	LOCATION parameter invalid. The ROW, COLUMN, or OFFSET value is out of range for the current terminal.

In addition, when a return code of 0 is set, variable t_result is set as follows:

Value	Description
0	No window or window control area at specified screen location CURSOR and LOCATION only.
Other	The identifier of the window or window control area.

WNORM: Normalize a window

Causes a zoomed or hidden window or window control area to be normalized. That is, returned to its previously defined size, as specified on a WOPEN or WALTER parameter and made fully visible.

```
WNORM [WINDOW window-identifier | window-name]
```

Parameters

`WINDOW window-identifier | window-name`

Specifies either the identifier, as assigned by the WOPEN operation, or the name of the window or window control area which is to be normalized. It may be specified as either a literal value, or as a variable containing the identifier or name of the window or window control area.

If the WINDOW parameter is omitted, the currently active window is normalized.

WNORM usage notes

The WNORM operation does not make the normalized window active, refer to the WACTIVATE verb earlier.

If the specified window is not zoomed or hidden, the WNORM parameter has no effect.

WNORM return codes

The success or failure of the WNORM operation is indicated by a return code in the variable t_rc. A list of possible values is given below:

Return code	Description
-4	Window environment not active.
0	Successful completion.
4	WINDOW parameter invalid. One of the following errors was encountered: ▪ Window identifier not in range 1-255. ▪ Window name contains all blanks. ▪ No window with the specified identifier or name exists. ▪ The WINDOW parameter was omitted, indicating that the operation should be performed on the active window, but no window is active.

WOPEN: Open a window

Creates a new window or window control area with the specified characteristics. Once the window is created it is the active window.

Syntax

```
WOPEN [ORIGIN ROW row COLumn col | OFFSET offset]
   [ENDPOS ROW row COLumn col | OFFSET offset]
   [SIZE DEPTH depth WIDTH width]
   [NAME window-name]
   [BORDER border-set-name | NONE]
   [OWNER window-identifier | window-name | NONE]
   [REMATTr [Yes | No | ON | OFF]]
   [SESSION session-number | session-pfkey
   [OUTPUT [Yes | No | ON | OFF]]
   or  WINPANEL panel-name
   or  WINTEXT message-text  ]
   [WCA [Yes | No | ON | OFF]]
```

Parameters

`ORIGIN ROW row COLumn col | OFFSET offset`

Specifies the screen location of the top left-hand corner of the window or window control area including its border.

It may be specified in either row and column format, or as a screen offset. The subparameters may be specified as either numeric literal values, or as numeric-type variables containing the values to be used.

A default of row 1, column 1 (offset 0) is used when the ORIGIN parameter is omitted.

`ENDPOS ROW row COLumn col | OFFSET offset`

Specifies the screen location of the bottom right-hand corner of the window or window control area including its border.

It may be specified as either a row or column, or as a screen offset. The subparameters may be specified as either numeric literal values, or as numeric-type variables containing the values to be used.

`SIZE DEPTH depth WIDTH width`

Specifies the depth as the number of rows and width as the number of columns of the window or window control area including its border.

The subparameters may be specified as either numeric literal values, or as numeric-type variables containing the values to be used.

`NAME window-name`

Specifies a name by which the window or window control area can be referred to in subsequent window operations.

The window name may be specified as either a character literal value, or as a character-type variable containing the window name.

If the NAME parameter is omitted, the window or window control area is unnamed but may be referred to by its window identifier, which is assigned automatically by the WOPEN operation, and returned in t_result.

`BORDER border-set-name | NONE`

Specifies the name of the border set to be used with the window or window control area.

The name of the border set may be specified as either a character literal value, or as a character-type variable containing the border set name.

If the BORDER parameter is specified with the NONE sub-parameter, or is omitted, no border set is used.

For details regarding definition of border sets refer to the WBORDER parameter on page 124.

`OWNER window-identifier | window-name | NONE`

Specifies the identifier or name of another window which is to own the new window or window control area.

The window identifier may be specified as either a numeric literal value, or as a numeric-type variable containing the window identifier. The window name may be specified as either a character literal value, or as a character-type variable containing the window name.

If the OWNER parameter is specified with the NONE sub-parameter, or is omitted altogether, the new window, or window control area, is not owned.

The only implication of a window being owned is when the owner is closed. At closure, the owned window is also closed. However, an owned window can be opened and closed without affecting the owner.

`REMATTr [Yes | No | ON | OFF]`

When Yes or ON is specified, any extended attributes and set attribute orders for the session in the window are removed when the window becomes inactive.

When No or OFF is specified, any extended attributes and set attribute orders are retained. This is useful for applications which write set attributes and reverse video blanks; for example to draw graphs on colour screens. The entire graph would be lost if REMATTR NO or OFF is not specified.

The default, REMATTR YES or ON, is to remove the extended attributes and set attribute orders.

```
SESSION session-number | session-pfkey
   [OUTPUT [Yes | No | ON | OFF]]
```

Specifies the number of the session or the PF key associated with the session, which is to be started in the window upon completion of the WOPEN operation.

The session number may be specified as either a numeric or character literal value, or as a numeric or character-type variable containing the session number. The session PF key may be specified as either a character literal value, or as a character-type variable containing the session PF key.

OUTPUT indicates whether or not session output should be reflected to the window when it is inactive.

The YES and ON subparameters indicate that session output should be written to the window when it is inactive. The NO and OFF subparameters indicate that session output should not be written to the window when it is inactive.

The default when no sub-parameter is specified is YES. The default when the OUTPUT parameter is omitted is NO.

`WINPANEL panel-name | WINTEXT message-text`

The *panel-name* is the name of a panel definition which is to be processed to create the message.

The *message-text* specifies the text of a message, and may be specified as either a character literal value, or as a character-type variable containing the message text.

There is no default; either WINPANEL or WINTEXT must be specified, if SESSION is omitted.

`WCA [Yes | No | ON | OFF]`

Indicates whether a window control area is to be created. A window control area is a special type of window which remains active at all times. It may only contain a panel or text, so can be used for such things as general action bars and status information, but cannot contain a session.

The YES and ON subparameters indicate that a window control area is to be created. The NO and OFF subparameters indicate that a window is to be created.

The default when no sub-parameter is specified is YES. The default when the WCA parameter itself is omitted is NO.

WOPEN usage notes

The ENDPOS and SIZE parameters are mutually exclusive.

The size of a window or window control area should be defined such that the amount of session output, or window message visible inside the window, is at least one row by one column. When defining the size of a window, allowances should be made for the following:

- The depth of the top border (1 byte).
- The depth of the bottom border (1 byte).
- The attribute byte for the left vertical border.
- The width of the left border (1 byte).
- The attribute byte for the window contents.
- The attribute byte for the right vertical border.
- The width of the right border (1 byte).

The default window size, if both the ENDPOS and SIZE parameters are omitted, is calculated as though an ENDPOS parameter referring to the last screen position were coded. For example, on a model 2 terminal this would be row 24, column 80 (offset 1919).

If a window control area is being created using the WCA parameter, the SESSION and OUTPUT parameters may not be specified.

Either a SESSION, WINPANEL, or WINTEXT parameter must be specified, but note that they are mutually exclusive.

When either a window (WCA NO) or window control area (WCA YES) is created using the WOPEN parameter, in both instances the window identifier is placed in the t_result variable. When a window (WCA NO) is started using the WOPEN parameter, the window identifier is also placed in the t_w_id variable. This variable is reserved for the identifier of the active window (WCA NO).

WOPEN return codes

The success or failure of the WOPEN operation is indicated by a return code in the variable t_rc. A list of possible values is given below:

Return code	Description
-4	Window environment not active.
0	Successful completion.
8	ORIGIN parameter invalid. The ROW, COLUMN, or OFFSET value is out of range for this terminal.
12	ENDPOS parameter invalid. The ROW, COLUMN, or OFFSET value is out of range for this terminal.
16	SIZE parameter invalid. The WIDTH or DEPTH value is out of range for this terminal.
20	NAME parameter invalid. One of the following errors was encountered: ▪ Name contains all blanks. ▪ Name already in use in another window or window control area.
24	BORDER parameter invalid. One of the following errors was encountered: ▪ Border set name contains all blanks. ▪ No border set with the specified name exists.
28	OWNER parameter invalid. One of the following errors was encountered: ▪ Window identifier not in range 1-255. ▪ Window name contains all blanks. ▪ No window with the specified identifier or name exists. ▪ The specified window is already an owner.
32	SESSION parameter invalid. One of the following errors was encountered: ▪ Session number not in range 1-9999. ▪ Session number is not defined on the user's menu. ▪ Session number refers to a command session. ▪ Session cannot be started.

Return code	Description
36	WINPANEL parameter invalid. The specified panel does not exist.
48	Positioning or size error. One of the following errors was encountered: ▪ Part of the window falls outside the bounds of the current terminal, due to an invalid combination of ORIGIN, ENDPOS, and SIZE values. ▪ The window size is insufficient to display at least one row and one column of session output.
56	Maximum number of windows of 255 are already open.

WPOP: Make a window fully visible

Causes a window or window control area, which may be partly or fully overlaid by other windows, to be made fully visible.

```
WPOP [WINDOW window-identifier | window-name]
```

Parameters

`WINDOW window-identifier | window-name`

Specifies either the identifier, as assigned by the WOPEN operation, or the name of the window or window control area to be made visible. It may be specified as either a literal value, or as a variable containing the identifier or name of the window or window control area.

If the WINDOW parameter is omitted, the currently active window is made visible.

WPOP usage notes

If the window identifier or name refers to a window or window control area which is already fully visible, the WPOP parameter has no effect.

The WPOP operation does not cause the window to become active. For activation, refer to the WACTIVATE parameter.

A zoomed or hidden window cannot be made fully visible using WPOP. The window should be returned to its 'normal' size, by using the WNORM parameter, which also has the affect of making the window fully visible.

WPOP return codes

The success or failure of the WPOP operation is indicated by a return code in the variable t_rc. A list of possible values is given below:

Return code	Description
-4	Window environment not active.
0	Successful completion.

Return code	Description
4	WINDOW parameter invalid. One of the following errors was encountered: ▪ Window identifier not in range 1-255. ▪ Window name contains all blanks. ▪ No window with the specified identifier or name exists. ▪ The WINDOW parameter was omitted, indicating that the operation should be performed on the active window, but no window is active.
60	Operation invalid on a zoomed or hidden window.

WSCROLL: Scroll data in a window

Causes the contents of a window or window control area to be scrolled vertically, horizontally, or both vertically and horizontally.

```
WSCROLL [WINDOW window-identifier | window-name]
[VERTICAL vertical_scroll_value]
[HORIZONTAL horizontal_scroll_value]
```

Parameters

`WINDOW window-identifier | window-name`

Specifies either the identifier, as assigned by the WOPEN operation, or the name of the window or window control area to be scrolled. It may be specified as either a literal value, or as a variable containing the identifier or name of the window or window control area.

If the WINDOW parameter is omitted, the currently active window is scrolled.

`VERTICAL vertical-scroll-value`

Indicates that a vertical scroll either up or down, is to be performed, and specifies the number of rows to be scrolled. A positive value indicates scroll down, and a negative value indicates scroll up.

The scroll amount may be specified as a numeric literal value, or as a numeric-type variable containing the scroll amount.

`HORIZONTAL horizontal-scroll-value`

Indicates that a horizontal scroll either to the left or right, is to be performed, and specifies the number of columns to be scrolled. A positive value indicates scroll to the right, and a negative value indicates scroll to the left.

The scroll amount may be specified as either a numeric literal value, or as a numeric-type variable containing the scroll amount.

WSCROLL usage notes

Either the VERTICAL or the HORIZONTAL parameter or both must be specified.

A positive VERTICAL scroll shows lines following the bottom of the current display, whereas a negative VERTICAL scroll shows lines preceding the top of the current display.

A positive HORIZONTAL scroll shows columns to the right of the current display, whereas a negative HORIZONTAL scroll shows columns to the left of the current display.

If the scroll amount exceeds the number of rows or columns remaining in the specified direction, the scroll operation proceeds as far as possible, and a value of 4 is placed in variable t_result.

WSCROLL return codes

The success or failure of the WSCROLL operation is indicated by a return code in the variable t_rc. A list of possible values is given below:

Return code	Description
-4	Window environment not active.
0	Successful completion.
4	WINDOW parameter invalid. One of the following errors was encountered: ▪ Window identifier not in range 1-255. ▪ Window name contains all blanks. ▪ No window with the specified identifier or name exists. ▪ The WINDOW parameter was omitted, indicating that the operation should be performed on the active window, but no window is active.

In addition, when a return code of 0 is set, variable t_result is set as follows:

Return code	Description
0	Successful completion.
4	The scroll amount exceeded the number of rows or columns remaining in the specified direction.

WZOOM: Zoom a window

Causes a window or window control area to be zoomed, so that it appears as though it were a non-windowed display.

```
WZOOM [WINDOW window-identifier | window-name]
```

Parameters

`WINDOW window-identifier | window-name`

Specifies either the identifier, as assigned by the WOPEN operation, or the name of the window or window control area which is to be zoomed. It may be specified as either a literal value, or as a variable containing the identifier or name of the window or window control area.

If the WINDOW parameter is omitted, the currently active window is zoomed.

WZOOM usage notes

The WZOOM operation does not cause the zoomed window to become active. Refer to WACTIVATE for activation.

If the specified window is already zoomed, the WZOOM parameter has no effect.

If the specified window is hidden, it must be returned to its normal size, by using the WNORM parameter, before it can be zoomed.

A window may only be zoomed if the session running in it will fit on the screen. For example, a window running a model 2 session on a model 4 terminal can be zoomed, but a window running model 4 session on a model 2 terminal cannot.

WZOOM return codes

The success or failure of the WZOOM operation is indicated by a return code in the variable t_rc. A list of possible values is given below:

Return code	Description
-4	Window environment not active.
0	Successful completion.
4	WINDOW parameter invalid. One of the following errors was encountered: ▪ Window identifier not in range 1-255. ▪ Window name contains all blanks. ▪ No window with the specified identifier or name exists. ▪ The WINDOW parameter was omitted, indicating that the operation should be performed on the active window, but no window is active.
52	Window cannot be zoomed due to incompatible session and terminal types.
60	Operation invalid on a hidden window.

Script return code summary

The following table is a summary of the return codes that can be set by script verbs at execution time. All return codes are held in the variable t_rc. The variable t_result is also set in certain instances.

Verb	t_rc	t_result	Explanation
Any verb	-4	0	Out of context (e.g., a Windows verb in a Startscript).
AUDITMSG	0	0	Successful completion.
	0	4	Text is null.
CALL	-	-	This is treated as a TPSL instruction, not a Script verb and therefore does not set t_rc or t_result.
CALLEXIT	0	x	Successful completion.
	28	x	Report to your local support representative.
	40	-	The E09 user exit is not loaded or the exit point is disabled.
DISPAPPL	0	0	Successful completion.
	4	0	No data to display.
ENDSESS	-8	0	Application Builder feature not enabled (if applicable).
	0	0	Successful completion.
	8	0	Session not active; not defined; or not an application session.
EXTRACT DATA	0	0	Successful completion.
	4	0	No application data received by the script.
	12	0	Incorrect FROM/TO specified.
HALTSCRIPT	-8	0	Application Builder feature not enabled.
	0	0	Successful completion.
	4	0	Session not active; not running a script; or not an application session.
INPUT	0	0	Successful completion.
	4	0	Invalid SBA for Device/Mode.
	8	0	Session not active.
INSCAN	0	0	Successful completion.
	4	0	No WAITTERM PASS outstanding.
OUTSCAN	0	0	Successful completion.
	4	0	No data to scan.
	8	0	Session not active.

Verb	t_rc	t_result	Explanation
PAUSE	0	0	Successful completion.
	4	0	Duration specified not numeric.
READSESS	-8	0	Application Builder feature not enabled.
	0	0	Successful completion.
	4	0	Row or column specified is off the screen.
	8	0	MISER not active.
RUNSCRIPT	-8	0	Application Builder feature not enabled.
	0	0	Successful completion.
	4	0	Session not active; already running a script; or not an application session.
SENDDATA/ SENDVAL	-8	0	Application Builder feature not enabled.
	0	0	Successful completion.
	4	0	Session not active: not an application session: or session has no partner (if RPARTNER is specified).
STARTSESS	-8	0	Application Builder feature not enabled.
	0	0	Successful completion.
	4	0	Session already active, or not an application session.
TDEQ	0	0	Successful completion.
	4	0	No current enqueue.
	8	0	No dequeue value.
	12	0	No previous TENQ.
TENQ	0	0	Successful completion.
	8	0	No enqueue value.
ISZCMD	0	0	Successful completion.
	4	0	Command rejected.
ISZCMDA	0	0	Successful completion.
	4	0	Command rejected.
USERMSG	0	0	Successful completion.
VCALL	0	0	Successful completion.
	4	0	Script does not exist.

Verb	t_rc	t_result	Explanation
WACTIVATE	0	0	Successful completion.
	4	0	WINDOW parameter invalid.
	44	0	Operation invalid on a window control area.
WAITAPPL	0	0	Successful completion.
	8	0	Session not active.
WAITDATA	0	0	Successful completion.
WAITEVENT	0	0	Successful completion.
WAITTERM	0	0	Successful completion.
WALTER	0	0	Successful completion.
	4	0	WINDOW parameter invalid.
	8	0	ORIGIN parameter invalid.
	12	0	ENDPOS parameter invalid.
	16	0	SIZE parameter invalid.
	20	0	NAME parameter invalid.
	24	0	BORDER parameter invalid.
	28	0	OWNER parameter invalid.
	32	0	SESSION parameter invalid.
	36	0	WINPANEL parameter invalid.
	44	0	Operation invalid on a Window Control Area.
	48	0	Positioning or size error.
WBACKGROUND	0	0	Successful completion.
WBORDER	0	0	Successful completion.
	20	0	NAME parameter invalid.
WCLOSE	0	0	Successful completion.
	0	4	No remaining windows to close.
	4	0	WINDOW parameter invalid.
WHIDE	0	0	Successful completion.
	4	0	WINDOW parameter invalid.
	60	0	Operation invalid on a zoomed window.

Verb	t_rc	t_result	Explanation
WIDENTIFY	0	0	No window or window control area at specified screen location CURSOR and LOCATION only.
	0	n	n specifies the identifier of the window or window control area.
	4	0	WINDOW parameter invalid.
	8	0	LOCATION parameter invalid.
WNORM	0	0	Successful completion.
	4	0	WINDOW parameter invalid.
WOPEN	0	0	Successful completion.
	8	0	ORIGIN parameter invalid.
	12	0	ENDPOS parameter invalid.
	16	0	SIZE parameter invalid.
	20	0	NAME parameter invalid.
	24	0	BORDER parameter invalid.
	28	00	OWNER parameter invalid.
	32	0	SESSION parameter invalid.
	36	0	WINPANEL parameter invalid.
	48	0	Positioning or size error.
	56	0	Maximum number of windows of 255 are already open.
WPOP	0	0	Successful completion.
	4	0	WINDOW parameter invalid.
	60	0	Operation invalid on a zoomed or hidden window.
WSCROLL	0	0	Successful completion.
	0	4	The scroll amount exceeds the number of rows and columns remaining in the specified direction.
	4	0	WINDOW parameter invalid.
WZOOM	0	0	Successful completion.
	4	0	WINDOW parameter invalid.
	52	0	Window cannot be zoomed due to incompatible session and terminal types.
	60	0	Operation invalid on a hidden window.

Verb	t_rc	t_result	Explanation
XYZSCREEN	-8	0	Application Builder feature not enabled.
	0	0	Successful completion.
	4	0	Row or column specified is off the screen.

CHAPTER 4

The Panel and Script Language (TPSL)

As its name suggests, the Panel and Script Language may be used in panel and script definitions. In fact, the processing section of a panel definition is made up entirely of TPSL statements. The statements enable conditional execution of panel and script commands, and allow the assignment of values to variables.

Overview

The following statements are provided:

Statement	Description
IF	Enables testing of single, compound, and multiple conditions.
SELECT	Enables multiple conditions to be evaluated and actioned.
CASE	This is a synonym of the SELECT statement.
DO	Denotes the start of a loop.
LET	Assigns a value to a variable.
ITERATE	Alters the flow within a repetitive DO loop.
LEAVE	Causes immediate exit from a repetitive DO loop.
RETURN	Terminates current panel or script processing.

Statements are not case dependent. For example, IF, if, and If all invoke the same statement.

Panels and Scripts can be traced by using the TTPSL command (see page 252).

Examples of the use of TPSL statements can be found earlier in this manual. The following sections describe the syntax of each statement.

The IF statement

The IF statement is used to define and control an IF structure. It enables a statement or group of statements to be conditionally tested for single, compound and multiple conditions and executed.

```
IF [NOT] conditional-expression
[AND | OR [NOT] conditional-expression]
[THEN]
   one or more operations
[ ELSE one or more operations]
END
```

conditional-expression has two forms:

subject comparator object

and

subject [*comparator*] *special-test*

subject and *object* are any value, variable or arithmetic expression.

comparator is either a symbol or mnemonic from the table below:

Symbol	Mnemonic	Meaning
=	EQ	Equal
>	GT	Greater Than
<	LT	Less than
¬=	NE	Not equal
<>	NE	Not equal
><	NE	Not equal
¬>	LE	Less than or equal
<=	LE	Less than or equal
=<	LE	Less than or equal
¬<	GE	Greater than or equal
>=	GE	Greater than or equal
=>	GE	Greater than or equal
	LIKE	Matches the specified mask

Notes

1. Mnemonic comparators must have a blank either side, but blanks are optional with symbol comparators.
2. The LIKE operator checks whether a string, or variable containing a string, matches a mask, or variable containing a mask. Wildcards '*' (any character or set of characters) and '+' (any single character) can be used within the mask.

special-test is one of the following:

ZEROS	test a field for 0.
SPACES	test a field for spaces.

When these special tests are used no *comparator* is required:

CURSOR	test a field for the cursor.
NUMERIC	test a field for values 0-9.
ALPHABETIC	test a character field for values A-Z and spaces.
MASK	test a field for Bits

For both NUMERIC and ALPHABETIC tests, spaces are allowed. In fact a field containing only spaces will pass both tests. Similarly, a null field will pass both tests. To eliminate blank/erased fields from, say, NUMERIC tests, spaces should be tested for explicitly before the NUMERIC test.

There are also two special conditions that can be tested for which can be applied. These are:

FOUND	Indicates a string has been found after an OUTSCAN or INSCAN.
NORESP	Indicates there has been no response from an application within a specified time.

For further explanation of their use, refer to the OUTSCAN, INSCAN, WAITAPPL, and WAITDATA parameters of the SCRIPT statement.

Special test examples

The NUMERIC, ALPHABETIC, CURSOR and MASK tests *do not* require a preceding comparator; an error condition is raised if one is supplied. For example:

```
If t_command NUMERIC
```

is valid, while

```
If t_command EQ ALPHABETIC
```

is not. However, the ZEROS and SPACES tests *do* require a preceding comparator. For example:

```
If t_command = SPACES
```

The CURSOR test can be coded in the Process section of a panel definition to determine if the cursor was in a field when the Enter key was pressed. The only valid fields for which this test can be applied are unprotected input fields.

The following example shows how cursor selection could be defined. The following code could be included in the Menu panel CONTENT section.

```
Let sub1 = 0
Do while sub1 < smax
Let sub1 = sub1 + 1
Field uc_c.sub1(1)       in,unprot,yellow      /*setup input area
Field s_desc.sub1(10)    prot,blue             /*display options
End
```

The following code would be included in the PROCESS section.

```
If t_aid = enter
And t_command = spaces
    Let sub1 = 0
    Let lnext = 0
    Do while (sub1 < smax) and (lnext = 0)
       Let sub1 = sub1 + 1
       If uc_c.sub1 CURSOR                  /*Is the cursor here?
          Let t_command = s_n.sub1
          Let lnext = 1
       End
    End
End
```

The user would now be able to select applications by positioning the cursor adjacent to the required entry in the menu panel and pressing Enter.

The MASK test can be used for bit testing. The format is:

IF *item_name* MASKxx [ON | OFF | MIXED] THEN ...

item_name

The variable to be tested. The variable must be a single byte.

MASKxx

The MASK keyword is suffixed with a valid hexadecimal mask in the range 00 to FF.

ON

All masked bits are on. This is the default.

OFF

All masked bits are off.

MIXED

Some of the masked bits are on and some are off.

For example, in a script you can code:

```
If uc_var MASK10 ON then              /* test for B'00010000' */
          Auditmsg 'fred'
End
If uc_var MASK90 ON then              /* test for B'10010000' */
          Auditmsg 'sid'
End
If uc_var MASK90 MIXED then           /* test for B'10010000' */
          Auditmsg 'bill'
       Else
            Auditmsg 'neither'
End
```

Thus, if uc_var = x'3F', which is Binary B'00111111' then MASK10 is true as the required bit is on.

MASK90 ON is false as only one of the two required bits are on. MASK90 MIXED is true as one of the required bits is on, so Auditmsg 'fred' is output and then Auditmsg 'bill'.

Generic comparisons are made by using the mask facility for any number of bytes. This is denoted by the letter M preceding a literal in quotes, which enables the generic comparison, as shown below:

M'*generic-value*'

generic-value

incorporates one of the following special characters:

- Asterisk (*) to indicate any number of any character.
- Plus (+) to indicate one of any character.

For example:

```
If t_termid = M'A*'
```

This condition would be true for all t_termid fields starting with the letter A.

```
If t_termid = M'A+T*'
```

This condition would be true for all t_termid fields with A in the first position and T in the third.

Further notes on using the IF statement

Evaluation of the expression proceeds from left to right. Expressions within parentheses are evaluated first.

Full IF capability is provided, that is nested IFs may be specified and IF statements containing AND, OR, and NOT are valid. Parentheses may be used to group conditions together.

Implied objects and relational operators may be used:

```
IF A = B AND A = C AND A NE D
```

is the same as:

```
IF A = B AND C AND NE D
```

Associated keywords

Keyword	Description
ELSE	This is used to identify the actions to be executed if the *conditional-expression* is false. If ELSE is omitted, control passes to the statement immediately following an END. An IF need not have an ELSE, it may simply have an END. This is particularly useful in single condition nested IF statements.
END	This is the final keyword of the IF statement and must be specified.
NOT	This is used with parentheses to reverse the entire condition from the left-hand parenthesis up to the matching parenthesis. While a NOT is in effect to reverse a condition, another NOT may not be specified.
THEN	This is used to indicate the end of a conditional expression. THEN may be omitted. The actions immediately following the *conditional-expression* are executed when the condition is true.

A more complex example of the IF statement:

```
IF conditional-expression-1 AND conditional-expression-2 [THEN]
   1 or more operations
ELSE
   1 or more operations
END
```

The SELECT statement

The SELECT statement enables multiple conditions to be evaluated and different operations to be performed depending on the results of the evaluation. The CASE statement is a synonym of the SELECT statement.

The format of the SELECT statement is:

```
SELECT | CASE
        WHEN [NOT] conditional-expression-1
            [AND | OR   [NOT] conditional-expression-2
                [THEN]
                one or more operations
        WHEN [NOT] conditional-expression-3
            [AND | OR   [NOT] conditional-expression-4
                [THEN]
                one or more operations
          ...
        WHEN [NOT] conditional-expression-n
            [AND | OR   [NOT] conditional-expression-n
                [THEN]
                one or more operations
        [ ELSE
                one or more operations]
    END
```

When a conditional expression is found to be true, the statements immediately following it are executed. In any SELECT statement, only the set of statements associated with the first condition found to be true are executed. Any number of WHEN keywords may be specified. If none of the conditional expressions are found to be true, the statements following the ELSE keyword are executed provided it has been coded, otherwise control passes to the statement following the END keyword.

Any valid statements may appear as part of a SELECT structure. *conditional-expressions* are the same as those entered for an IF statement.

The DO statement

The DO statement enables a loop structure to be specified to allow iterative processing of a group of operations. The format of the statement is:

```
DO
   WHILE conditional-expression |
   UNTIL conditional-expression |
   FOREVER                      |
   FOR nnn | variable-name
   one or more operations
END
```

When DO is specified, all statements following it are executed as part of the loop until an associated END keyword is encountered. This includes nested DO statements. The DO statement provides four ways of controlling the number of times that the set of operations is executed. These are:

`DO WHILE conditional-expression`

The loop is executed while the specified condition is true. It is a conditional loop.

`DO FOREVER`

The loop is executed until LEAVE is executed by the logic or the DOMAX limit is exceeded. This is an unconditional loop.

`DO UNTIL conditional-expression`

The loop is executed until the specified condition is true. This is a conditional loop.

`DO FOR nnn | variable-name`

where *nnn* is an integer literal in the range 1-999, and *variable-name* is a variable containing an integer. Either of these may be used to cause the loop to be executed a specific number of times. This is an unconditional loop.

Where a loop is to be executed a set number of times, this method is more efficient than a conditional loop.

Conditional expressions are specified in the same way as for the IF statement. They have two forms:

subject comparator object

and

subject [*comparator*] *special test*

See the section on the IF Statement on page 147 for further details.

Condition testing for DO WHILE loops is performed before the loop is executed. For DO UNTIL loops, the condition is tested at the end of the loop to ensure that the loop is executed at least once. This is particularly useful in scripts where an OUTSCAN parameter has been coded in a DO UNTIL FOUND loop; there is no need to specify LET FOUND=FALSE before the start of the loop. For both DO WHILE and DO UNTIL loops, the condition is evaluated on each iteration.

The DOMAX parameter provides an inbuilt facility to prevent excessive looping. This automatically interrupts the DO processing after a predefined number of times and terminates execution of the panel or script. An error message is also issued.

The DOMAX predefined limit is 100, unless overridden. The DOMAX subparameter of the PANEL DEFINE parameter may be used to specify a limit for panels. The DOMAX parameter of the SCRIPT statement overrides the default limit for scripts. A global limit can be set using the DOMAX parameter of the SYSTEM statement.

It is possible to interrupt loop processing by two other methods:

- By using the LEAVE statement within the DO loop. LEAVE can be used to terminate a loop if a set condition is proved true.

or

- By using the ITERATE statement to alter the flow of a repetitive DO loop. This causes the rest of the instructions in a loop to be bypassed and the end condition is tested, or control passes to the first statement of the loop if the loop is unconditional.

An example using both of these statements is shown below:

To create a DO loop that is executed 50 times, to terminate the loop at the end of 50 iterations, enter:

```
Let un_count = 1
Do forever
...
    If un_count = 50
       Leave
    Else
       Let un_count = un_count+1
       Iterate
    End
...
End
```

In window scripts when WAITEVENT is specified, or in session scripts when WAITDATA is specified, in a DO WHILE, DO FOREVER, or DO UNTIL loop, the DO iteration does not count towards the DOMAX count for that loop. Other loops are treated normally, so it is possible that if WAITEVENT or WAITDATA is coded in an inner loop, the DOMAX count could be exceeded. WAITEVENT or WAITDATA should be specified in an outer never-ending loop.

Note When a DO end-condition is met, control is passed to the END keyword.

The *1 or more operations* can be any instruction, command, or other TPSL instructions such as IF, SELECT and the DO instruction itself.

Simple DO examples

```
Do until found
    Waitappl
    Outscan 'LOGON'
End
```

This is the same as:

```
Let found = false
Do while not found
    Waitappl
    Outscan 'LOGON'
End
```

In these examples, the loop is repeated until 'LOGON' is output by the application as part of a data stream.

```
Let uncount = 5
Do for uncount
   Let uc_var = uc_var'AB'
End
```

In the above example, `uc_var` would contain 'ABABABABAB' (provided it was null to begin with).

```
Do until found
    Waitappl
    Outscan 'Good Morning'
    If noresp
     Leave
    End
    Input key clear
End
```

The controlled repetitive loop specifies a control variable which is assigned an initial value before the first execution of the instruction list. The variable is then stepped before the second and subsequent times that the instruction list is executed. The instruction list is executed repeatedly while the *`conditional-expression`* is not met.

For example:

```
Let sub1 = 1
Do while sub1 <= smax
  Col 1
  Field s_desc.sub1(20)            in,unprot,norm
  Let sub1 = sub1 + 1
End
```

This is less efficient than the following example, as the condition is evaluated each time the loop is executed. The following:

```
Let sub1 = 1
Do for smax
  Col 1
  Field s_applid.sub1(8)           in,unprot,norm
  Let sub1 = sub1 + 1
End
```

is more efficient, as there is no condition to be evaluated, regardless of the number of iterations.

The LET statement

The facility to assign values to variables is provided by the LET statement. The format is:

```
LET variable-name | &variable-name = variable |
                                      literal|
                                      string |
                                      arithmetic expression |
                                      function
```

where *function* can be one of the following:

```
ABBREV (string input length)
ABS (number)
ADDOUT (string row col attr)
CENTRE (string length pad)
CHANGESTR (string1 string2 new)
COMPARE (string1 string2 pad)
COPIES (string n)
COUNTSTR (string1 string2)
DELSTR (string n length)
DELWORD (string n length)
D2X (variable)
ENTDATA (fieldnum extdata row-col-prefix)
ENTRY_IO ('operation' ddname other_parms)
EXTRAUTH (command_variable keyword_variable)
FORMATMSG (nnnn 'SEND'|'SENDNO' 'MSGID'|'ID'|'NOID' P1 P2...P9)
INQUIRE ( )
INSERT (new string n length pad)
KEYPOS (string words n)
LASTPOS (string1 string2 n)
LEFT (string length pad)
LENGTH (string)
MAX (n1 ... nx)
MIN (n1 ... nx)
OVERLAY (new string n length pad)
PASSIN (session)
PASSOUT (session)
POS (string1 string2 n)
REVERSE (string)
RIGHT (string length pad)
SIGN (number)
SPACE (string n pad)
SPLXLOCUSER ('U' | 'T', object, stem <,'ALL'>)
SPLXLOG (stem)
SPLXNODES (stem)
STRIP (string 'B' | 'L' | 'T' char)
SUBSTR (string n length pad)
SUBWORD (string n length)
TMSG (variable)
UPPER (string)
VERIFY (string chars 'N' | 'M' n)
WORD (variable)
```

```
WORDINDEX (string n)
WORDLENGTH (string n)
WORDPOS (string1 string2 n)
WORDS (variable)
X2D (variable)
```

The *variable-name* may be a modifiable Session Manager-supplied variable, or a user-defined variable. If it is a user-defined variable, the first assignment of a value to the variable defines the variable. Refer to 'User definable variables' on page 249 for a description. The chapter 'Session Manager variables' on page 197 contains a fuller description of each variable type.

Indirect variable substitution

The *variable-name* to which a value is to be assigned can itself reference another variable. This form of indirect variable substitution requires an ampersand (&) to be inserted before the variable-name. For example:

```
If ...
   Let uc_var = 't_cut'
Else
   Let uc_var = 't_paste'
End
Let &uc_var = 'Fred'
```

In the example, the variable *uc_var* itself either contains the name t_cut, or the name t_paste. If, say, after the first LET assignment completes, it contains the name t_cut, then the variable t_cut will ultimately be assigned the value of 'Fred'. The *uc_var* variable remains unchanged; it still contains the value t_cut.

Assigning a single variable or literal

```
LET variable-name = variable | literal
```

The new variable and the value assigned to it need not be the same data type. This therefore allows conversion between character and numeric, and numeric and character data types.

Different portions of the source variable may be assigned to a new variable using the length and displacement option, described on page 160.

Assigning a string of values

```
LET variable-name = string
```

If a string of variables or literals is to be assigned to a variable, the receiving variable must be a character type variable. The string may be a combination of character variables, numeric variables, and literals. No concatenating symbols are required.

Length and displacement modifiers may be specified on the string variables. However, if no modifiers are specified, the length of the string is calculated as the sum of the lengths of each field after the trailing spaces, or leading zeros, have been removed. For example:

- If the character variable, t_user, contained 'TXS' and the numeric variable, s_n, contained the value 5, then the statement:

  ```
  let uc_var1 = t_user s_n
  ```

  would cause uc_var1 to be assigned the value of 'TXS5' and an implicit length of four.

  However, the statement:

  ```
  let uc_var1 = t_user(8) s_n(3)
  ```

  would cause uc_var1 to be assigned a value 'TXS 5' and an implicit length of eleven.

A variable can be embedded in a literal assignment by preceding the variable name with an ampersand (&). For example:

```
let uc_lit1 = (This is &t_user&'s Special Menu)
```

so for TXS, this becomes 'This is TXS's Special Menu'. If the ampersands are omitted, no substitution takes place and the result is 'This is t_user's Special Menu'. Note the use of parentheses as string delimiters to enable a quotation mark to be included in the string.

Assigning a hexadecimal string

A hex string can be assigned to a character variable if required using:

```
X'hex-string'.
```

The hex-string must be enclosed in quotes.

For example:

```
Let uc_var = X'3F'
```

Assigning an attribute character

A user defined character variable can contain an ATTR symbol. When the ATTR symbol is assigned to a character variable it must be prefixed with the letter A. The attribute can then be controlled from TPSL code.

For example:

```
If ln_rcode > 4
   Let lcattr = A'&'      /* Highlight attribute
 Else
   Let lcattr = A'%'      /* Normal attribute
End
```

Note This only applies to PANEL definition, it should *not* be used in scripts.

Assigning an arithmetic expression

```
LET variable-name = arithmetic expression
```

Only numeric variables and integers may be used in *arithmetic expressions*. The receiving variable may, however, be numeric or character. It is also valid to assign a single character variable to a numeric variable, provided the value is numeric, otherwise a value of 0 is assigned.

A simple expression consists of three parts:

variable-1 operator variable-2

operators are:

- * multiply
- / divide
- + add
- - subtract

Multiple simple expressions and parentheses may be used to construct more complex expressions. In all expressions, normal algebraic priority is observed. Therefore, expressions in parentheses are evaluated first, from innermost to outermost parentheses. Multiplication and division operations are processed before addition and subtraction.

Any division operation which does not result in an integer is truncated after the decimal point to give an integer result. Intermediate results will overflow if they are larger than 15 digits. Any division by zero causes the result to be zero.

Assigning one portion of a variable to another

Length and displacement modifiers may be used with character and numeric variables to assign only part of a variable or literal to a receiving variable. Both the length and the displacement arguments may be specified as variables. The modifiers are specified in parentheses after the variable name, as follows:

variable([*displacement*,]*length*)

Displacement is optional, and if omitted is assumed to be 1, that is the first position in the variable. The maximum value that either modifier may have is 256.

For example, if the user defined variable, uc_number, contains the value '2345', then the following:

```
Let uc_lit = uc_number(2,2)
```

would give a result of 34 to uc_lit.

Numeric variables are converted to 9 byte character variables. Since the number is right justified in the field, with padding on the left with spaces when the receiving variable is longer than the actual number, a displacement modifier may point to a position containing a space. Positive displacement values are always calculated from the leftmost position of the field.

A negative displacement is specified by preceding it with a minus (-). When a negative displacement is specified, the processing for numeric variables differs from that for character variables.

For numeric and character variables, the displacement is calculated from the rightmost position and proceeds backwards for the length specified. For example:

- If the user defined variable, *un_number*, contained the value 123, then the following:

  ```
  un_number(-2,2)
  ```

  would give a result of 23.

For a character variable:

- If *uc_name* contained the value ABCDE, then the following:

  ```
  uc_name(-2,2)
  ```

  would give a result of DE.

Assigning a variable using special functions

```
LET variable-name = function
```

where *function* can be one of the following:

```
ABBREV (string abbrev length)
ABS (n)
ADDOUT (string row col attr)
CENTRE (string length pad)
CHANGESTR (string1 string2 new)
COMPARE (string1 string2 pad)
COPIES (string n)
COUNTSTR (string1 string2)
DELSTR (string n length)
DELWORD (string n length)
D2X (variable)
ENTDATA (fieldnum extdata row-col-prefix)
ENTRY_IO ('operation' ddname other_parms)
EXTRAUTH (command_variable keyword_variable)
FORMATMSG (nnnn 'SEND'|'SENDNO' 'MSGID'|'ID'|'NOID' P1 P2...P9)
INQUIRE ( )
INSERT (new string n length pad)
KEYPOS (string words n)
LASTPOS (string1 string2 n)
LEFT (string length pad)
LENGTH (string)
MAX (n1 ... nx)
MIN (n1 ... nx)
OVERLAY (new-string n length pad)
PASSIN (session)
PASSOUT (session)
POS (string1 string2 n)
REVERSE (string)
RIGHT (string length pad)
SIGN (number)
SPACE (string n pad)
SPLXLOCUSER ('U' | 'T', object, stem <,'ALL'>)
SPLXLOG (stem)
SPLXNODES (stem)
STRIP (string 'B' | 'L' | 'T' char)
SUBSTR (string n length pad)
```

```
SUBWORD (string n length)
TMSG (variable)
UPPER (string)
VERIFY (string chars 'N' | 'M' n)
WORD (string n)
WORDINDEX (string n)
WORDLENGTH (string n)
WORDPOS (string1 string2 n)
WORDS (string)
X2D (hexstring)
```

Note All the function arguments can be enclosed in parentheses (like REXX functions), although this is not mandatory.

Function arguments can be separated by spaces or commas.

Function specifications have the following meanings:

ABBREV

```
ABBREV (string,abbrev,length)
```

If the *abbrev* specified is equal to the leading characters of *string* ***and*** the length of the *abbrev* is not less than the *length* specified, then the ABBREV function returns 1. If either of these conditions is not met, then it returns 0.

length, if specified, must be a non-negative integer. The default for *length* is the number of characters in *abbrev*.

For example:

```
Let uc_var = 'START'
Let un_a = ABBREV (uc_var,'STA')       /* sets un_a to 1
Let un_a = ABBREV (uc_var,'STA',4)     /* sets un_a to 0
Let un_a = ABBREV (uc_var,'')          /* sets un_a to 1
```

Note A null string always matches if a length of 0 (or the default) is used. This allows a default keyword to be selected automatically, if desired.

ABS

```
ABS (n)
```

The absolute value of the number *n* specified is returned by the ABS function. The result returned has no sign.

For example:

```
Let un_var = '-2'
Let un_n = ABS (un_var)       /* sets un_n to '2'
```

ADDOUT

```
ADDOUT (string row col attr)
```

The ADDOUT function adds *string* to a 3270 output datastream, for displaying at the position identified by a start row, *row*, and start col, *col*. If there is no existing application output datastream, the function will generate a new 3270 write datastream for the data.

string can be a variable or a literal. An attribute, *attr*, is optional; if supplied, its maximum length is 16 characters.

For example:

```
Let lc_start_attr = X'1DE8'
Let sub1 = ADDOUT ('Replace',4,29,lc_start_attr)
```

modifies an application screen by writing 'Replace' at row 4, column 30, and changes the current attribute at column 29 to highlite, pen detectable. `sub1` is a numeric variable which is set to zero.

After execution, the function sets a return code in the variable t_rc. Possible values of the return code are:

Code	Description
0	Successful; either the supplied data has been added to an existing queued application datastream or a new 3270 write datastream has been created.
8	The attribute is longer than 16 characters.
12	One of the following: ▪ Incorrect number of arguments – more than 4 or less than 3. ▪ Invalid row or column value – either non-numeric or the value is invalid for the model of screen.
16	The function was not called from an application script.

CENTRE/CENTER

`CENTRE (string length pad)`

The CENTRE function can be used to cause the value of the string specified to be centred, or truncated, within the *length* specified. It is then stored. The default pad character is blank.

For example:

```
Let uc_a = 'tOp'
Let uc_x = CENTRE (uc_a,7)           /* sets uc_x to '  tOp  '
```

If the length of the pad argument is greater than 1, only the first digit is used. If the string is longer than the length specified, then the string is truncated at both ends. If odd numbers of characters are truncated or added, the rightmost end of the string loses or gains one more character than the leftmost end.

For example:

```
let uc_x = CENTRE (uc_a,8,'-')        /* sets uc_x to '--tOp---'
let uc_x = CENTRE ('The blue sea',8)  /* sets uc_x to 'e blue s'
let uc_x = CENTRE ('The blue sea',7)  /* sets uc_x to 'e blue '
```

If the function is assigned to a numeric variable, then the result is usually zero. The only exception is if the resultant string was numeric and less than 999999999.

CHANGESTR

`CHANGESTR (string1 string2 new)`

The CHANGESTR function replaces each occurrence of the string, *string1*, within the string, *string2*, with the string *new*. The old string, *string1*, and the *new* string may be of unequal length, in which case the length of *string2* is adjusted. If *new* is a null string, the effect will be to delete all occurrences of *string1*. Overlapping occurrences of *string1* are not replaced.

For example:

```
Let uc_a = 'abcabcabc'
Let uc_x = CHANGESTR ('c' uc_a 'cd')    /* sets uc_x to
                                        /* 'abcdabcdabcd'
```

COMPARE

COMPARE (*string1 string2 pad*)

The COMPARE function compares the two strings, *string1* and *string2*, after adding as many pad characters as are necessary to equalize the lengths of the two strings (padding is added to the right of the shorter string). The default pad character is blank. A value of 0 (zero) is returned if they are identical, otherwise the position of the first character that does not match is returned.

For example:

```
Let uc_a = 'ab '
Let un_compare = COMPARE (uc_a 'ab')     /* sets un_compare to 0
Let un_compare = COMPARE (uc_a 'ab' 'x') /* sets un_compare to 3
```

COPIES

COPIES (*string n*)

The COPIES function can be used to produce a specified number of concatenated copies of the *string* specified. *n* must be a non-negative whole number.

For example:

```
Let uc_a = 'TODAY '
Let uc_x = COPIES (uc_a,2)          /* sets uc_x to 'TODAY TODAY'
```

If the function is assigned to a numeric variable, then the result is usually zero. The only exception is if the resultant string was numeric and less than 999999999.

COUNTSTR

COUNTSTR (*string1 string2*)

The COUNTSTR function counts the number of times one string, *string1*, is present in another, *string2*. Overlapping occurrences of *string1* are not counted.

For example:

```
Let uc_a = 'Saturday, Sunday, Monday'
Let un_x = COUNTSTR ('day' uc_a)       /*sets un_x to 3
```

DELSTR

DELSTR (*string n length*)

The DELSTR function returns *string* after deleting the substring that starts at the *n*'th character of *string* and is of the specified *length*. *n* must be a positive whole number. If *length* is specified it must be a positive whole number or zero.

string is returned unchanged if *n* is greater than the length of *string*. If *length* is omitted, or if *length* is greater than the number of characters from *n* to the end of *string*, the characters deleted are those from the *n*'th position to the end of the target *string*.

For example:

```
Let uc_a = 'abcde'
Let uc_x = DELSTR (uc_a 3)                /* sets uc_x to 'ab'
Let uc_x = DELSTR (uc_a 3 2)              /* sets uc_x to 'abe'
Let uc_x = DELSTR (uc_a 6)                /* sets uc_x to
'abcde'
```

DELWORD

DELWORD (*string n length*)

The DELWORD function returns *string* after deleting the substring that starts at the *n*'th blank-delimited word and is of the number of words specified by *length*. *n* must be a positive whole number. If *length* is specified it must be a positive whole number or zero.

string is returned unchanged if *n* is greater than the number of words in *string*. If *length* is omitted, or if *length* is greater than the number of words from *n* to the end of *string*, the words deleted are those from the *n*'th position to the end of the target *string*.

The substring which is deleted includes any blanks following the final deleted word but no blanks preceding the first deleted word.

For example:

```
Let uc_a = 'Now is the  time'
Let uc_x = DELWORD (uc_a 2 2)    /* sets uc_x to 'Now time'
Let uc_x = DELWORD (uc_a 3)      /* sets uc_x to 'Now is '
Let uc_x = SUBWORD (uc_a 5)      /* sets uc_x to
                                 /* 'Now is the  time'
```

D2X

D2X (*variable*)

The D2X function returns a string in character format, representing a decimal number contained in *variable*, converted to hexadecimal. The returned string uses uppercase alphabetics for values A-F and does not include blanks.

See the X2D description for the inverse function.

For example:

```
Let un_num = 42
Let uc_hex = D2X (un_num)        /* sets uc_hex to '2A'
```

If the function is assigned to a numeric variable, then the result is usually zero. The only exception is if the resultant string was numeric and less than 999999999.

ENTDATA

ENTDATA (*fieldnum extdata row-col-prefix*)

The ENTDATA function extracts the data which the user has entered into an input field and places it in one or more variables, depending on its length. *fieldnum* is the number of the required input field. The extracted data is placed in the variable *extdata* which is subscripted; the first variable will contain the first 256 characters of the data entered in the field, the second the next 256 characters, etc. The maximum length of the name supplied as *extdata* is 11 characters.

The function also creates two variables containing the starting row and column of the input field, and names these variables from the specified prefix, *row-col-prefix*, followed by '_row' and '_col' respectively. The maximum length of the prefix is 12 characters.

For example:

```
Let insub = 3
Let ln_indata_nbr = ENTDATA (insub,'lc_indata','ln_inpos')
```

extracts the data in the third input field and places it in the subscripted variable lc_indata. If the input data consists of 300 characters, lc_indata.1 contains the first 256 characters and lc_indata.2 contains the remaining 44 characters; ln_indata_nbr is then set to 2, the number of lc_indata variables. The starting row and column of the input field are placed in two variables named ln_inpos_row and ln_inpos_col.

The ENTDATA function can be used within a loop to process fields entered by a user.

After execution, the function sets a return code in the variable t_rc. Possible values of the return code are:

Code	Description
0	Successful; the selected field has been found. The number of subscripted data variables has been stored in the target variable, and the subscripted data variables and the row and column variables have been created.
4	Selected field not found.
8	Internal error – the sba does not match the device block.
12	Error in one of the parameters.
16	The function was not called from an application script.

ENTRY_IO

```
ENTRY_IO ('operation' ddname other_parms)
```

A new format for the configuration was introduced in Session Manager 1.105 in which configuration data is stored in several PDS(E)s (for details, see the product's *Online and Batch Administration* manual). The ENTRY_IO function allows access to these PDS(E)s allocated to the Session Manager started task; the component of Session Manager that handles the dataset I/O operations is the TCB Manager.

Notes

1. I/O is performed in a pool of separate z/OS operating system tasks, so has little impact on Session Manager processing.
2. The maximum number of lines in a configuration member is 5000.

The different forms of the ENTRY_IO function are shown below; *operation* must be enclosed in quotes as shown.

`ENTRY_IO ('READ' ddname membername stem)`	Read member of PDS(E)
`ENTRY_IO ('READ' ddname stem)`	Read sequential dataset
`ENTRY_IO ('LIST' ddname stem [maxmem mask srules audit])`	List members of PDS(E)
`ENTRY_IO ('WRITE' ddname membername stem)`	Write member of PDS(E)
`ENTRY_IO ('REPLACE' ddname membername stem)`	Replace member of PDS(E)
`ENTRY_IO ('DELETE' ddname membername)`	Delete member of PDS(E)

Parameters

ddname

Is the DDNAME associated with the dataset.

membername

Specifies the member name when a READ, WRITE, REPLACE or DELETE operation is being performed on a PDS(E).

stem

Is the variable stem used to construct the 'number of values' variable names (*stem*_NBR and *stem*_SET) and the subscripted variable names (*stem*_VAL and *stem*_V*mm*). For details, see 'Stem variables' on page 168.

As the maximum length of a subscripted variable name is 14 characters and the suffix (_NBR, _SET, _VAL or _V*mm*) is 4 characters, the maximum length of the stem is 10 characters.

Optional parameters for LIST operation

maxmem

Specifies the maximum number of members to include in the list.

mask

Species a mask value to limit selection against an 8-character member name. These special characters can be included in the mask:

- * An asterisk indicates any number of any characters.
- \+ A plus sign indicates any single character in that position.

srules

A string, that specifies security rules for checking access to PDS(E) members. This string consists of one or more keywords separated by a single blank; optionally, each keyword can have a qualifier.

The keywords, which must be entered in uppercase, are ALL, NONE, DISPLAY, DELETE, NODELETE, CREATE, NOCREATE, MODIFY, NOMODIFY. The qualifier is enclosed in brackets immediately following a keyword, and is a mask value to limit selection against an 8-character member name. These special characters can be included in the mask:

- * An asterisk indicates any number of any characters.
- \+ A plus sign indicates any single character in that position.

Example:

```
NONE ALL(ABC*) NOCREATE(+++++++X) NONE(ABCX*)
```

This would give full access to members prefixed ABC, but disallow creation of members with X as the eighth character, and disallow any access to members prefixed by ABCX.

audit

Specifies whether or not to audit PDS(E) members that fail the security rules. Allowed values are:

`Y` Audit members that fail the security rules. A WTO will be generated if a member fails the security rules – that is, security failures appear only on the z/OS console; they do not appear in the audit dataset.

`N` Do not audit members that fail the security rules.

Stem variables

For a successful `READ` or `LIST` operation, a number of variables are generated automatically by the ENTRY_IO function; for a `WRITE` or `REPLACE` operation, these variables should be set up before the function is called. For details, see:

- 'READ or LIST operation' on page 168
- 'WRITE or REPLACE operation' on page 169

READ or LIST operation

TPSL has a maximum subscript limit of 9999. If the total number of values is less than or equal to 9999 then, for a successful `READ` or `LIST` operation, these stem variables are generated automatically:

Variable	Description
*stem*_NBR	Total number of lines read (`READ` operation) or total number of member names read (`LIST` operation). The result of the function will also contain this number.
*stem*_VAL.*nnnn*	Values of individual lines read (`READ` operation) or values of individual member names read (`LIST` operation). *stem*_VAL.0=*stem*_NBR.

However, if the total number of values is greater than 9999 then these stem variables are also generated automatically:

Variable	Description
*stem*_V*mm*.*nnnn*	Further sets of values of individual lines read (`READ` operation) or values of individual member names read (`LIST` operation): *mm*=01 set – stores values 10000 to 19998, *mm*=02 set – stores values 19999 to 29997, and so on. *stem*_V*mm*.0=total number of values in set *mm*.
*stem*_SET	Number of sets of *stem*.V*mm* variables.

In this case:

- *stem*_VAL.0=9999.
- *stem*_NBR=*stem*_VAL.0+*stem*_V*mm*.0 (*mm*=01, 02, … *stem*_SET).

  The result of the function will also contain this number.

For example, if *stem* is LST and the total number of values is 30000, these variables are generated automatically, where 'Item' is Line or Member as appropriate:

Variable	Value	Set number
LST_NBR	30000	-
LST_VAL.0	9999	-
LST_VAL.1	Item 1	-
…	…	-
LST_VAL.9999	Item 9999	-
LST_V01.0	9999	01
LST_V01.1	Item 10000	01
…	…	01
LST_V01.9999	Item 19998	01
LST_V02.0	9999	02
LST_V02.1	Item 19999	02
…	…	02
LST_V02.9999	Item 29997	02
LST_V03.0	3	03
LST_V03.1	Item 29998	03
…	…	03
LST_V03.3	Item 30000	03
LST_SET	3	-

WRITE or REPLACE operation

For a WRITE or REPLACE operation, the stem variables used are the same as those used for a READ or LIST operation (see page 168). However, for a WRITE or REPLACE operation:

- The variables should be set up before the ENTRY_IO function is called to contain the data for the member (one value per line).
- The script should limit the total number of values in each set to the TPSL maximum subscript limit of 9999.
- Variable *stem*_SET is not used.

For example, if *stem* is WRT and the total number of values is 300, in the script logic it might be convenient to limit each set of subscripted variables to 100:

Variable	Value	Set number
WRT_NBR	300	-
WRT_VAL.0	100	-
WRT_VAL.1	Value 1	-
…	…	-
WRT_VAL.100	Value 100	-

Variable	Value	Set number
WRT_V01.0	100	01
WRT_V01.1	Value 101	01
…	…	01
WRT_V01.100	Value 200	01
WRT_V02.0	100	02
WRT_V02.1	Value 201	02
…	…	02
WRT_V02.100	Value 300	02

Usage note

In a script, the default loop maximum value is 100. However, a new loop maximum value can be set by specifying a `DOMAX` value at the start of the script. When working with stem variables for `READ`, `LIST`, `WRITE` or `REPLACE` operations, ensure that the script specifies `DOMAX 9999` (the maximum allowed value).

Return and reason codes

After execution, the ENTRY_IO function sets a return code in the variable t_rc and may also set a reason code in the variable t_result.

Return codes

Possible values of the return code (t_rc) are:

Code	Description
0	Operation successful. For a `READ` or `LIST` operation, a number of stem variables are generated automatically (see page 168).
4	Operation failed.
12	Invalid parameters supplied to function.

Reason codes

Possible values of the reason code (t_result) are:

`READ` or `DELETE` operation:

Code	Description
4	Member not found in any of the PDS(E)s making up the concatenation.
56	Record length not 80.
84	(`DELETE` operation only.) Member found in the second or subsequent PDS(E) making up the concatenation, but it was *not* deleted.

`LIST` operation:

Code	Description
68	Member list capped at *maxmem*.
72	Some members failed security check.

Code	Description
76	Member list capped at *maxmem* and some members failed security check.
80	Invalid security rules supplied to function.

WRITE or REPLACE operation:

Code	Description
4	(REPLACE operation only.) Existing member not found, but the specified member will still be added to the PDS(E).
8	*stem*_NBR=0 on write.
48	Directory full, not stored.
52	Attempt to write out a null member.
56	Record length not 80.
60	(WRITE operation only.) Member already exists.

All operations:

Code	Description
12	Open failed.
16	Operating system macro failed member. For details, see these variables: lc_ops_macro - macro name. ln_ops_r15 - return code. ln_ops_r0 - reason code.
20	I/O error.
24	Maximum DDNAMEs in use.
28	TCB Manager inactive.
32	TCB Manager closing.
64	TCB Manager TCB abended (error undetermined).

ESMPROF

ESMPROF (*stem*)

Allows scripts and panels to determine which profiles would be assigned to a user who is not currently logged on to Session Manager.

For example:

```
let lc_something = esmprofs(stem)
```

The stem must begin with uc_, lc_ or gc_ and not be greater than eleven characters in length. Before issuing the function the variable *stem*_user must be set to the user name to be interrogated.

On return from the function the following variables will have been set:

Variable	Len	Format	Description
stem.0000			The number of profiles assigned to the user, up to a maximum of eighteen.
stem.0001			The first profile name assigned to the user.
stem.0002			The second profile name assigned to the user.
to:			
stem.0018			The eighteenth profile name assigned to the user.

A return code from the function will be set in t_rc:

Code	Description
0	Successful, ESM assigned profile(s) returned.
4	Default profile name assigned. User is unknown to the ESM.
8	Default profile name assigned. User does not have authority to any profiles.
12	Default profile name assigned. Error encountered with the ESM.
16	Default profile name assigned. Session Manager not configured for ESM assigned profiles.
20	Invalid stem name.

EXTRAUTH

`EXTRAUTH (command_variable keyword_variable)`

Returns the authorization level required to issue a command or the keyword of a command.

Parameters

command_variable

A variable containing the name of a Session Manager command.

keyword_variable

An optional parameter containing the name of a keyword of the command specified in *command_variable*.

Return and reason codes

After execution, the EXTRAUTH function sets a return code in the variable t_rc and may also set a reason code in the variable t_result.

Return codes

Possible values of the return code (t_rc) are:

Code	Description
0	Operation successful. (The variable will contain the authorization level necessary to use the command or keyword of the command.)
4	The specified command was not found, or the keyword does not exist or is not valid for the specified command.

Reason codes

Possible values of the reason code (t_result) are:

Code	Description
4	The command specified is a script. The name of the script will be returned in lc_cmdscript. If a keyword has been specified, the authorization level of the keyword returned will be the same as that of the command.

FORMATMSG

`FORMATMSG (nnnn 'SEND'|'SENDNO' 'MSGID'|'ID'|'NOID' P1 P2...P9)`

Enables a script or panel to issue Session Manager messages:

- defined conventionally within the product

  or

- defined externally using `MESSAGE` statements.

Parameters

nnnn

Message number as defined on a `MESSAGE` statement.

`'SEND' | 'SENDNO'`

Specify `'SEND'` to format the message in the target variable and send it to the Audit log and/or the console, or `'SENDNO'` to just format the message in the target variable.

`'MSGID' | 'ID' | 'NOID'`

Specify `'MSGID'` to format the message with the message prefix if the user has `MSGID` set on, `'ID'` to format the message with the message prefix, or `'NOID'` to format the message without the message prefix.

Notes

a If the target of the function will subsequently be used in the TMSG function (see page 187) then specify `'ID'`.

b If `'SEND'` is specified (see above) then this set of parameters have no affect on the message sent – they just affect the formatted message stored in the target variable.

`P1, P2...P9`

Optional substitution variables.

Return and reason codes

After execution, the FORMATMSG function sets a return code in the variable t_rc and may also set a reason code in the variable t_result.

Return codes

Possible values of the return code (t_rc) are:

Code	Description
0	Operation successful. (The function has formatted the message in the target variable.)
12	Invalid parameters supplied to function.

Reason codes

Possible values of the reason code (t_result) are:

Code	Description
4	Message number either not numeric or not in the range 1-9999.
8	Second parameter not `'SEND'` or `'SENDNO'`.
12	Third parameter not `'MSGID'`, `'ID'` or `'NOID'`.
16	Too few parameters (that is, less than 3).
20	Too many parameters (that is, greater than twelve).

INQUIRE

`INQUIRE( )`

The INQUIRE function, which has no parameters, is used in a command script to perform VTAM inquiries immediately. VTAM inquiries are normally performed under the control of the `INQINTERVAL` parameter on the SYSTEM statement (see the *Session Manager Technical Reference* manual)

For example:

```
Let lc_dummy = INQUIRE( )
```

INSERT

`INSERT (new string n length pad)`

The INSERT function inserts the string *new*, padded or truncated to *length*, into the target *string* after the *n*'th character. If specified, *n* and *length* must be positive whole numbers or zero. By default, *new* is inserted before the beginning of *string* (*n* is of the value 0). If *n* is greater than the length of the target *string*, padding is added before the *new* string also. *length* defaults to the length of *new*. The default pad character is blank.

For example:

```
Let uc_a = 'abc'
Let uc_x = INSERT (' ' uc_a 2)         /* sets uc_x to ‘ab c’
Let uc_x = INSERT ('123' uc_a 5 6)     /* sets uc_x to ‘123abc’
Let uc_x = INSERT ('3' uc_a 4 2 '+')   /* sets uc_x to ‘abc+3+’
```

KEYPOS

`KEYPOS (string words n)`

The KEYPOS function returns the number of the blank-delimited word within *words* which matches the *n*'th blank-delimited word in string. 0 is returned if no match is found, if *string* is a null string, or if n is greater than the number of words in *string*.

By default, the search starts at the first blank-delimited word of *string* (*n* is of the value 1). This can be overridden by specifying *n*, where *n* is the number of the word at which the search is to start. If specified, *n* must be a positive whole number.

For example:

```
Let uc_words = 'DLST DHLD DRDR'
Let un_a = KEYPOS ('DHLD' uc_words)       /* sets un_a to 2
Let un_a = KEYPOS ('DPUN' uc_words)       /* sets un_a to 0
Let un_a = KEYPOS ('DLST' uc_words 2)     /* sets un_a to 0
```

Case is not considered when matching. words must be in uppercase and the minimum abbreviation of each word can be marked with a hyphen. The selected word from *string* is first converted to uppercase before trying to find a match, and will match against that part of the word that precedes the hyphen plus any number of characters following the hyphen.

For example:

```
Let uc_words = 'DL-ST DH-LD DR-DR'
Let un_a = KEYPOS ('DH' uc_words)         /* sets un_a to 2
Let un_a = KEYPOS ('DRd' uc_words 2)      /* sets un_a to 3
```

LASTPOS

`LASTPOS (string1 string2 n)`

The LASTPOS function returns the position of the last occurrence of one string *string1*, in another string, *string2*. If *string1* is not found, or is a null string, 0 is returned.

By default, the search starts at the last character of *string2* and scans backwards. This may be overridden by *n*, which specifies the point at which the scan is to start the backwards scan. *n* must be an integer. It defaults to the length of *string2* if larger than that value, or omitted.

See the POS description for the reverse function.

For example:

```
Let uc_a = 'top,mid,bot'
Let un_lstpos = LASTPOS (',' uc_a)      /* sets un_lstpos to 8
Let un_lstpos = LASTPOS (',' uc_a,7)    /* sets un_lstpos to 4
```

LEFT

`LEFT (string length pad)`

The LEFT function returns the value of *string* left-adjusted within the specified length and with pad characters added as necessary to make up the length. *length* must be a positive whole number or zero. The default pad character is blank. If the length of the pad argument is > 1, only the first character is used. If *string* is longer than the length specified, the string is truncated on the right to fit.

For example:

```
Let uc_a = 'xYz'
Let uc_x = LEFT (uc_a 6 '.')         /* sets uc_x to 'xYz...'
Let uc_x = LEFT ('The blue sea' 7)   /* sets uc_x to 'The blu'
```

Invalid values of *length* return a null string.

Note The LEFT function is exactly equivalent to the SUBSTR function specified with a position of 1.

LENGTH

LENGTH (*string*)

The LENGTH function returns the length of *string*.

For example:

```
Let uc_a = 'hardcopy'
Let un_length = LENGTH (uc_a)        /* sets un_length to 8
```

MAX

MAX (*n1 ... nx*)

The MAX function returns the largest number from the list specified. Up to 9 numbers can be specified.

For example:

```
Let un_num = '12'
Let un_max = MAX (un_num,5,11,9)  /* sets un_max to 12
```

MIN

MIN (*n1 ... nx*)

The MIN function returns the smallest number from the list specified. Up to 9 numbers can be specified.

For example:

```
Let un_num = '12'
Let un_min = MIN (un_num,6,7,9)   /* sets un_min to 6
```

OVERLAY

OVERLAY (*new-string string n length pad*)

The OVERLAY function overlays the target *string*, starting at the *n*th character, with *new-string*, padded or truncated to *length*. If *length* is specified it must be an integer or zero. If *n* is greater than the length of the target *string*, padding is added before the *new-string* string.

If the length of the pad argument is greater than 1, only the first digit is used. The default pad character is blank and the default value for *n* is 1. If specified, *n* must be an integer.

For example:

```
Let uc_a = 'ab,cdef'
Let uc_x = OVERLAY (' ',uc_a,4)     /* sets uc_x to 'ab, def'
```

If the function is assigned to a numeric variable, then the result is usually zero. The only exception is if the resultant string was numeric and less than 999999999.

PASSIN

`PASSIN (session)`

The PASSIN function directs the next terminal/script input data to another sessions. The specified session must be active. *session* must be a session selection value, for example 'PF4' if the session has been defined as KEY PF4, or '20' if the session has been defined as SESSION 20.

This function can be used to direct terminal input to an application builder script. It can only be used in an application script.

For example:

```
Let lc-sess = 'PF6'
Let ln_passin = PASSIN (lc_sess)
```

directs all terminal input to session PF6. ln_passin is a numeric variable which is set to zero.

After execution, the function sets a return code in the variable t_rc. Possible values of the return code are:

Return code	Description
0	Successful; the next input will be sent to the specified session.
12	One of the following: ▪ Incorrect number of arguments – none or more than one. ▪ Specified session not active. ▪ Specified session not an application session.
16	The function was not called from an application script.

PASSOUT

`PASSOUT (session)`

The PASSOUT function directs output from an application to another session. The specified session must be executing an INITSCRIPT; this script will receive the output as application data. *session* must be a session selection value, for example 'PF4' if the session has been defined as KEY PF4, or '20' if the session has been defined as SESSION 20.

This function can be used to direct application output from an application builder script to another session. It can only be used in an application script.

For example:

```
Let lc-sess = '88'
Let ln_pass = PASSOUT (lc_sess)
```

sends application output data to session 88. ln_pass is a numeric variable which is set to the number of buffers sent.

After execution, the function sets a return code in the variable t_rc. Possible values of the return code are:

Return code	Description
0	Successful; the application output has been sent to the specified session.
8	No data to send.
12	One of the following: ▪ Incorrect number of arguments – none or more than one. ▪ Specified session not an application session. ▪ Specified session not executing an INITSCRIPT.
16	The function was not called from an application script.

POS

`POS (string1 string2 n)`

The POS function returns the position of one string, *string1*, in another, *string2*. 0 is returned if *string1* is a null string, not found, or if n is greater than the length of string2. By default, each search starts at the first character (i.e., *n* is of the value 1). This can be overridden by specifying *n*, where *n* is the point at which the search is to start. If specified, *n* must be an integer.

For example:

```
Let uc_a = 'Hardcopy'
Let un_x = POS ('copy',uc_a)     /* sets un_x to 5
```

REVERSE

`REVERSE (string)`

The REVERSE function swaps the specified string end for end.

For example:

```
Let uc_a = 'TODAY.'
Let uc_rev = REVERSE (uc_a)      /* sets uc_rev to '.YADOT'
```

If the function is assigned to a numeric variable, then the result is usually zero. The only exception is if the resultant string was numeric and less than 999999999.

RIGHT

`RIGHT (string length pad)`

The RIGHT function returns the value of *string* right-adjusted within the specified length and with pad characters added as necessary to make up the length. *length* must be a positive whole number or zero. The default pad character is blank. If the length of the pad argument is > 1, only the first character is used. If *string* is longer than the length specified, the string is truncated on the left to fit.

For example:

```
Let uc_a = 'xYz'
Let uc_x = RIGHT (uc_a 6 '.')         /* sets uc_x to '...xYz'
Let uc_x = RIGHT ('The blue sea' 7)   /* sets uc_x to 'lue sea'
```

Invalid values of *length* return a null string.

SIGN

`SIGN (number)`

The SIGN function returns a number that indicates the sign of *number*. *number* is first rounded, as though the operation *number* + 0 had been carried out. The number returned is:

- -1 if *number* is less than 0
- 0 if *number* is equal to 0
- 1 if *number* is greater than 0

For example:

```
Let un_numpos = '12'
Let un_sign = SIGN (un_numpos) /* sets un_sign to 1
Let un_numneg = '-307'
Let un_sign = SIGN (un_numneg)    /* sets un_sign to -1
Let un_numzero = '0'
Let un_sign = SIGN (un_numzero)/* sets un_sign to 0
```

SPACE

`SPACE (string n pad)`

The SPACE function can be used to format the blank-delimited words in *string* with a specified number of pads between each word. The *n* must be non-negative. If it is 0 all blanks are removed. Leading and trailing blanks are always removed. If the length of the pad argument is greater than 1, only the first digit is used. The default for *n* is 1. The default pad character is a blank.

For example:

```
Let uc_a = '   top bot   '
Let uc_sp = SPACE (uc_a,2,+)         /* sets uc_sp to 'top++bot'
```

If the function is assigned to a numeric variable, then the result is usually zero. The only exception is if the resultant string was numeric and less than 999999999.

SPLXLOCUSER

`SPLXLOCUSER ('U' | 'T', object, stem <,'ALL'>)`

The SPLXLOCUSER function enables a script executing in a Sysplex Session Manager system to identify whether a particular user, or `SIGNON NO` terminal, is active in the Sysplex group.

A Sysplex Session Manager system is one that is running in a Sysplex and has the SYSTEM statement parameter `SYSPLEXGROUP` specified, with the `USERSTRUCTURE` sub-parameter set to Y (if `USERSTRUCTURE` is not specified, the default is Y).

If the user or `SIGNON NO` terminal is active, a number of variables will be set by the function indicating the number of users, or `SIGNON NO` terminals with the specified name, that are active and which Session Manager node each user is active on. The information contained in these variables can then be used to issue various commands concerning the users via the Session Manager `SEND` *nodename command* operator command.

Provided the users are not SHARE, SHARESESS or SHAREDISC users and are not connected via the Session Manager Telnet Server then only one user will be active in the Sysplex group. If a number of SHARE, SHARESESS or SHAREDISC users with the same name are active then provided they are not connected via the Session Manager Telnet Server they will all be on the same Session Manager node. If users are connected via the Session Manager Telnet Server (not recommended in a Session Manager Sysplex) then users with the same name can be active on any node simultaneously.

object is either a user name or a terminal as identified by 'U' or 'T'.

stem is the stem, the prefix, of the variables the SPLXLOCUSER function is to create. The stem must specify a value with a prefix of LC, UC or GC that is specifying local, user or global user-defined variables. Each value must be no shorter than two, and no longer than nine characters.

'ALL' (optional) - includes signing on entries.

The function will return a return code, in t_rc and the target variable, of:

Return code	Description
0	The user or terminal name has been located on this Sysplex node.
4	The user or terminal name has been located on a remote Session Manager node in this instance's Sysplex group.
8	The user or terminal name was not located on any node in the Sysplex.
12	First parameter not 'U' or 'T', third parameter length error, not LC, UC or GC or second parameter is blank.
16	This Session Manager instance has not been configured to be part of a Session Manager Sysplex group; that is, the SYSTEM statement parameter SYSPLEXGroup has not been specified, the instance failed to join the specified group or the SYSTEM statement USERStructure parameter has been specified with a value of NO or OFF.

If t_rc is 16, then t_result is one of the following:

Return code	Description
4	This Session Manager instance has not been configured to be part of a Session Manager Sysplex group; that is, the SYSTEM statement parameter SYSPLEXGroup has not been specified.
8	The SYSTEM statement USERStructure parameter has been specified with a value of NO or OFF.
12	Logic error.

If the function returns a return code of 0 or 4 then four arrays named *stem_N*, *stem_Q*, *stem_I* and *stem_T* will be set up and *stem_N.0* will contain the number of entries in each array. There will be one entry for each user or terminal found. If multiple users are signed on using the same name (using the SHARE, SHARESESS or SHAREDISC options) or there are multiple terminals with the same name and the SIGNON NO option is applied then there will be multiple entries. If multiple entries are returned each entry can be uniquely identified by the *stem_Q* value. If there are more than 512 entries then t_result will be set to 4 but only 512 entries are stored in each array.

Each entry in the *stem_N* array contains the name of the user or terminal as specified by the *object*.

Each entry in the stem_Q array contains the *object*'s unique qualifier. The value of the qualifier varies dependent on the type of user or terminal.

Type	Value of qualifier
1	For users not using the SHARE, SHARESESS or SHAREDISC options the qualifier will be blank and the *stem_T* value will be 'U'.
2	For users using the SHARE, SHARESESS or SHAREDISC options the value of the qualifier depends on the SYSTEM statement parameter MULTUSER. If the MULTUSER parameter is TERML4 or not specified, then the value will be the last four characters of the user's terminal name and the *stem_T* value will be 'M'. If the MULTUSER parameter is COUNT then the value will be the user's assigned number and the *stem_T* value will be 'C'.
3	For terminals with the SIGNON NO option active the value will be a unique non-blank value and the *stem_T* value will be 'T'.
4	For signing on users, only returned if 'ALL' specified, the value will be a unique non-blank value and the *stem_T* value will be 'S'.

Each entry in the *stem_I* array contains the node name of the Session Manager on which the *object* is active.

For example, the following code locates the Session Manager node in this instance's Sysplex group that user JOE is signed on to, if any. After the execution of the second 'Let' ln_rc will contain the SPLXLOCUSER's return code. If the return code is 0 or 4 then four arrays named LC_OP_N, LC_OP_Q, LC_OP_I and LC_OP_T will be set up and LC_OP_N.0 will contain the number of entries in each array.

```
Let lc_user = 'JOE'

Let ln_rc = SPLXLOCUSER('U', lc_user, 'LC_OP')
```

SPLXLOG

SPLXLOG (*stem*)

The SPLXLOG function can be used to return records from the Sysplex global log.

For example:

```
Let lc_something = SPLXLOG(stem)
```

The stem must begin with `uc`, `lc` or `gc` and not be greater than seven characters in length.

Issued in this form, without setting any variable parameters, it will return the last 100 records written to the Sysplex log in the following subscripted variables:

Variable	Description
stem.0	The number of records returned.
*stem*_blk.n	The block id of the record.
stem.n	The record data, containing the following data elements, each delimited by a blank character: Date of log record – 8 characters (yyyymmdd) Time of log record – 6 characters (hhmmss) z/OS system name – 1 to 8 characters Job name – 1 to 8 characters Node name – 1 to 8 characters Message id– 8 characters Message text – 1 to 256 characters
*stem*_fpos	Returned from a find request (see below). Contains the position of the 'found' string within stem.1

There are additional parameter variables which can be used to tailor the request:

Variable	Description
*stem*_nrec	The number of records to return. Maximum = 200. Default = 100.
*stem*_blki	The block id of the record at which to begin reading.
*stem*_drct	Direction of read. 'F' forward (default). 'B' backward.
*stem*_scrl	Used in conjunction with *stem*_drct. The number of records to scroll back or forward to before returning data. A 1-4 byte character numeric. Maximum 9999. Mutually exclusive with *stem*_find.

Variable	Description
*stem*_find	Used in conjunction with *stem*_drct. A character string to search for, within the each record. The first record containing the string will be the first record returned. Mutually exclusive with *stem*_scrl.

Returned records will ALWAYS be in the order oldest to youngest.

A return code from the function will be set in t_rc:

0	All OK.
4	'find' string not found.
8	Invalid stem name.
12	Invalid variable parm.
16	Mutually exclusive variable parms.
20	Could not access the Sysplex Group audit log. An associated message, containing return and reason code, will be written to the job's audit log.
24	Sysplex log inactive.

SPLXNODES

`SPLXNODES (stem)`

The SPLXNODES function can be used to return information about nodes within a Sysplex group.

For example:

```
Let lc_something = SPLXNODES(stem)
```

Where *stem* is a two to seven character variable stem name which must begin with the characters `gc`, `lc` or `uc`.

This will return the following un-subscripted variables:

Variable	Len	Format	Description
stem.grp	8	Character numeric	The Sysplex group name.
*stem*_bcn	8	Character numeric	The broadcast node name.
stem.std	8	Character numeric	Date that the Standby timer was started.
stem.stt	8	Character numeric	Time that the Standby timer was started.
stem.stv	1-4	Character numeric	The Standby timer value (in seconds) specified on the `STANDBYTAKETIME` parameter in the configuration.

And the following subscripted variables:

Variable	Len	Format	Description
*stem*_acb.n	8	Character numeric	The ACB name.
stem.css.n	2	Character numeric	The Standby/Controller status of this instance. See the table below for values.
stem.mbr.n	1-16	Character numeric	Member name of the Session Manager instance.
*stem*_mvs.n	8	Character numeric	The z/OS name.
*stem*_nod.n	8	Character numeric	The node name.
*stem*_job.n	8	Character numeric	The Session Manager job name.
stem.sbn.n	1-8	Character numeric	Subplex name of instance.
stem.sgn.n	1-8	Character numeric	Sysplex group name of instance.
*stem*_sta.n	1	Numeric	A one-byte numeric character which indicates the node status. Where: '0' = non-Sysplex node. '1' = inactive Sysplex node. '2' = active Sysplex node. '3' = terminating Sysplex node.
stem.stg.n	1-8	Character numeric	Standby's target instance nodename.
stem.stn.n	1-8	Character numeric	This instances Standby nodename.
stem.vln.n	1-8	Character numeric	VTAM link name of instance.

The following table shows the Standby/Controller (*stem*.css.n) values:

Value	Instance type	Description.
20	Normal instance	Not configured as a Standby.
30	Normal instance	Normal status (has synchronized with its target).
31	Normal instance	Synchronizing with its target.
32	Normal instance	Waiting for its target to start.
33	Normal instance	Transferring from its target instance.

Value	Instance type	Description.
34	Normal instance	Recovering from its targets failure.
50	Standby/Controller	Normal status (has synchronized with the Controller).
51	Standby/Controller	Synchronizing with the Controller.
52	Standby/Controller	Waiting for the Controller to start.
53	Standby/Controller	Transferring from the Controller.
54	Standby/Controller	Recovering from a Controller failure.
60	Controller	Normal status.
61	Controller	Draining to Standby (Transferring).

On return from the function, `t_rc` will contain a return code as follows:

0 Successful.

4 Stem name does not begin with `gc`, `lc` or `uc`.

8 Stem name exceeds seven characters.

STRIP

`STRIP (string 'B' | 'L' | 'T' char)`

The `STRIP` function removes leading characters, trailing characters, or both, from the specified string based on the option specified. The options are:

Option	Description
`'B'`	removes both leading and trailing characters from *`string`*. If an option is not specified, this is the default.
`'L'`	removes leading characters from *`string`*.
`'T'`	removes trailing characters from *`string`*.

The option must be enclosed in quotes as shown. No other delimiters are valid.

The third argument, *`char`*, specifies the character to be removed. The default is a blank. If specified, *`char`* should be exactly one character long. If more than one character is specified, only the first is used. If *`char`* is ***not*** 0 to 9 then it must be enclosed in quotes. If it is 0 to 9 then quotes are optional.

For example:

```
Let uc_a = '   ab c   '
Let uc_st = STRIP (uc_a,'L')        /* sets uc_st to ‘ab c   ’
```

If the function is assigned to a numeric variable, then the result is usually zero. The only exception is if the resultant string was numeric and less than 999999999.

SUBSTR

`SUBSTR (string n length pad)`

The SUBSTR function returns the substring that starts at the *n*'th character of the target *string* and is of the specified *length*, with pad characters added as necessary. *n* must be a positive whole number. If *length* is specified it must be a positive whole number or zero.The default pad character is blank. If the length of the pad argument is > 1, only the first digit is used.

If *n* is greater than the length of the target *string*, pad characters only are returned. If *length* is omitted, the substring returned is from the *n*'th character to the end of the target *string*.

For example:

```
Let uc_a = 'abcdef'
Let uc_x = SUBSTR (uc_a 3)              /* sets uc_x to 'cdef'
Let uc_x = SUBSTR (uc_a 4 6 '.')        /* sets uc_x to 'def...'
```

Note A SUBSTR function specifying a value of 1 for *n* is equivalent to the LEFT function.

SUBWORD

`SUBWORD (string n length)`

The SUBWORD function returns the substring that starts at the *n*'th blank-delimited word of the target *string* and is of the number of words specified by *length*. *n* must be a positive whole number. If *length* is omitted, the substring returned is from the *n*'th word to the end of the target *string*.

The returned string has no leading or trailing blanks but includes all blanks between the selected words.

For example:

```
Let uc_a = 'Now is the  time'
Let uc_x = SUBWORD (uc_a 2 2)     /* sets uc_x to 'is the'
Let uc_x = SUBWORD (uc_a 3)       /* sets uc_x to 'the  time'
Let uc_x = SUBWORD (uc_a 5)       /* sets uc_x to ''
```

T_RECOVER_DET

`T_RECOVER_DET (stem)`

The T_RECOVER_DET function will return a group of variables containing various counts relating to the `RECOVERYLEVEL` state of USERs, ACBs and Sessions.

For example:

`Let lc_something = T_RECOVER_DET(stem)`

Where *stem* is a two to seven character variable stem name which must begin with the characters `gc`, `lc` or `uc`.

Each variable contains a one to five byte character numeric value with leading zeroes removed. The variables returned are dependant on the type of Session Manager Instance as follows:

For a Session Manager Instance or Standby Instance:

Name	Description
stem_pusrs	Number of users signed on to the Primary Instance

Name	Description
*stem*_pnacb	Number of open `RECOVERYLEVEL NONE` ACBs
*stem*_purhs	Number of users with `RECOVERYLEVEL HIGH` Primary Sessions
*stem*_puris	Number of users with `RECOVERYLEVEL INTERMEDIATE` Primary Sessions
*stem*_purns	Number of users with `RECOVERYLEVEL NONE` Sessions
*stem*_susrs	Number of users signed on to the Standby Instance
*stem*_surhs	Number of users with `RECOVERYLEVEL HIGH` Standby Sessions
*stem*_puris	Number of users with `RECOVERYLEVEL INTERMEDIATE` Standby Sessions

For a Session Manager Instance or Controller Instance:

Name	Description
*stem*_phacb	Number of Primary open `RECOVERYLEVEL HIGH` ACBs
*stem*_piacb	Number of Primary open `RECOVERYLEVEL INTERMEDIATE` ACBs
*stem*_parhs	Number of Primary active `RECOVERYLEVEL HIGH` Sessions
*stem*_paris	Number of Primary active `RECOVERYLEVEL INTERMEDIATE` Sessions
*stem*_parns	Number of Primary active `RECOVERYLEVEL NONE` Sessions

For a Session Manager Instance, Standby Instance or Standby Controller Instance:

Name	Description
*stem*_shacb	Number of Standby open `RECOVERYLEVEL HIGH` ACBs
*stem*_siacb	Number of Standby open `RECOVERYLEVEL INTERMEDIATE` ACBs
*stem*_sarhs	Number of Standby active `RECOVERYLEVEL HIGH` Sessions
*stem*_saris	Number of Standby active `RECOVERYLEVEL INTERMEDIATE` Sessions

On return from the function, t_rc will contain a return code as follows:

0 successful

4 stem name does not begin `gc`, `lc` or `uc`

8 stem name exceeds nine characters

TMSG

`TMSG (variable)`

Enables a script or panel to set the t_message variable.

Parameters

variable

Either a variable name or a literal containing either blanks or a formatted message in the form:

xxxnnnns text

where:

xxx

Is the product id.

nnnn

Is the message number.

s

Is the severity of the message ('I' for an information message, 'W' for a warning message or 'E' for a serious error).

text

Is the message text.

Usage note

The TMSG function would normally be used with the FORMATMSG function (see page 173) – for example:

```
TMSG(FORMATMSG(1234 'SENDNO' 'ID' ...))
```

Return and reason codes

After execution, the TMSG function sets a return code in the variable t_rc and may also set a reason code in the variable t_result.

Return codes

Possible values of the return code (t_rc) are:

Code	Description
0	Operation successful. (The function has formatted the message in the target variable.)
12	Invalid parameter supplied to function.

Reason codes

Possible values of the reason code (t_result) are:

Code	Description
4	Message number not numeric.
8	Severity level not I, W or E.
12	The ninth character is not a space.

UPPER

`UPPER (string)`

The UPPER function causes any lower case characters in the specified string to be converted to upper case.

For example:

```
Let uc_a = 'TODAy'
Let uc_x = UPPER (uc_a)          /* sets uc_x to 'TODAY'
Let uc_x = UPPER 'lower case'    /* sets uc_x to 'LOWER CASE'
```

If the function is assigned to a numeric variable, then the result is usually zero. The only exception is if the resultant string was numeric and less than 999999999.

VACCESS

`VACCESS (stem)`

Allows scripts and panels to determine if a user has ESM authority to access a specific resource.

For example:

```
let lc_something = vaccess(stem)
```

The stem must begin with `uc_`, `lc_` or `gc_` and not be greater than ten characters in length. Before issuing the function the following variables must be set:

Variable	Len	Format	Description
stem.user			Must be set to the user name.
stem.res			Must be set to the name of the resource against which to check the user's access.
stem.class			Must be set to the ESM class name in which the resource resides.

A return code from the function will be set in t_rc:

Code	Description
0	The user is allowed access.
4	The ESM could not make a decision.
8	The user is denied access.
12	The user is unknown to the ESM.
16	The class name is unknown to the ESM.
20	Invalid stem name.

VERIFY

`VERIFY (string chars 'N' | 'M' n)`

The VERIFY function checks whether *string* is composed of the characters specified by *chars*, and returns a number depending on the option ('N' or 'M') specified. The options are:

Option	Description
'N'	returns 0 if all the characters in *string* are also in *chars*, or returns the position of the first character in *string* that does not match any of the *chars*. If *chars* is null, the value of *n* is returned. This is the default.
'M'	returns the position of the first character in *string* that matches one of the *chars*, or returns 0 if none of the *chars* is found or if *chars* is null.

The option must be enclosed in quotes as shown. No other delimiters are valid.

By default, the search starts at the first character of *string* (*n* is of the value 1). This can be overridden by specifying *n*, where *n* is the point at which the search is to start. If specified, *n* must be a positive whole number.

If *string* is null, or *n* is greater than the length of *string*, 0 is returned (regardless of whether 'N' or 'M' is specified).

For example:

```
Let uc_chars = '1234567890'
Let un_a = VERIFY ('123' uc_chars)            /* sets un_a to 0
Let un_a = VERIFY ('1Z3' uc_chars)            /* sets un_a to 2
Let un_a = VERIFY ('AB4T' uc_chars 'M')       /* sets un_a to 3
Let un_a = VERIFY ('1P3Q4' uc_chars,,3)      /* sets un_a to 4
```

WORD

```
WORD (string n)
```

The WORD function returns the *n*th blank delimited word in *string*, or returns the null string if fewer than *n* words are in *string*. The *n* must be an integer and has a default of 1.

For example:

```
Let uc_str = 'one two three four'
Let uc_word = WORD (uc_str,3)        /* sets uc_word to 'three'
```

Alternatively, a variable may be used to index the string:

```
Let un_index = 3
Let uc_word = WORD (uc_str,un_index)  /* sets uc_word to 'three'
```

If the function is assigned to a numeric variable, then the result is usually zero. The only exception is if the resultant string was numeric and less than 999999999.

WORDINDEX

`WORDINDEX (string n)`

The WORDINDEX function returns the position of the first character in the *n*'th blank-delimited word in *string*, or returns 0 if fewer than *n* words are in *string*. *n* must be a positive whole number.

For example:

```
Let uc_a = 'Now is the time'
Let un_wordix = WORDINDEX (uc_a 3)    /* sets un_wordix to 8
Let un_wordix = WORDINDEX (uc_a 6)    /* sets un_wordix to 0
```

WORDLENGTH

`WORDLENGTH (string n)`

The WORDLENGTH function returns the length of the *n*'th blank-delimited word in *string*, or returns 0 if fewer than *n* words are in *string*. *n* must be a positive whole number.

For example:

```
Let uc_a = 'Now is the time'
Let un_wordlen = WORDLENGTH (uc_a 2)     /* sets un_wordlen to 2
Let un_wordlen = WORDLENGTH (uc_a 6)     /* sets un_wordlen to 0
```

WORDPOS

`WORDPOS (string1 string2 n)`

The WORDPOS function returns the word number within *string2* of the first word of *string1* if *string1* is contained in *string2*. If *string1* is not found, or is a null string, 0 is returned. Multiple blanks between words in either string are treated as a single blank, otherwise the words must match exactly.

By default, the search starts at the first word of *string2*. This may be overridden by *n*, which specifies the word at which the search is to start. *n* must be a positive whole number.

For example:

```
Let uc_a = 'now is the time'
Let un_wordpos = WORDPOS ('the' uc_a)    /* sets un_wordpos to 3
Let un_wordpos = WORDPOS ('The' uc_a)    /* sets un_wordpos to 0
Let un_wordpos = WORDPOS ('is   the' uc_a)/*sets un_wordpos to 2
```

WORDS

`WORDS (string)`

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

For example:

```
Let uc_str = 'one two three four'
Let un_words = WORDS (uc_str)            /* sets un_words to 4
```

X2D

`X2D (hexstring)`

The X2D function returns the decimal representation of *hexstring*. *hexstring* is a string of hexadecimal characters.

For example:

```
let uc_hex = '2A'
let un_num2 = X2D (uc_hex)              /* sets un_num2 to 42
```

If the supplied value does not contain hexadecimal characters, or null, the function returns a value of -1 (a text string -1 if in a character variable).

The ITERATE statement

The ITERATE statement is used to alter the flow within a repetitive DO loop. Execution of the group of instructions stops and control is passed to the DO instruction, just as though the bottom of the group of instructions had been reached.

The DO condition is evaluated as normal. See the previous section, 'The DO Statement' on page 153 for further details.

The ITERATE statement has no parameters:

```
ITERATE
```

The LEAVE statement

The LEAVE statement causes immediate exit from a DO loop. Execution of the group of instructions is terminated and control is passed to the instruction following the END keyword, just as though the END keyword had been encountered and the termination condition had been met normally.

The LEAVE statement has no parameters:

```
LEAVE
```

The RETURN statement

The RETURN statement is used for *early* termination of the current logic processing. This may be either from the current panel section (HEADER, CONTENT, TRAILER and PROCESS) or from a script. In the case of a panel it causes the next panel section to be processed.

Note Section processing is normally terminated when the last line is encountered, in which case the RETURN Statement is not required.

In the case of a script, the current script processing is terminated. Control is returned to the next higher level script (if any). For example:

```
SCRIPT ONE SCRIPT TWO
   ..
   ..
   Call two Return
   .
```

When RETURN is executed in Script Two, Script Two terminates. Control is returned to the statement immediately following 'Call two' parameter in Script One.

The RETURN statement has no parameters:

```
RETURN
```

CHAPTER 5

Session Manager variables

Session Manager provides a wide range of variables which may be used in several of the configuration control statements. This chapter gives a brief description of each variable, and indicates whether or not it is modifiable. For a fuller description of the use of variables in the appropriate context, see:

- 'Input variable substitution' on page 30
- 'Panel variables' on page 62
- 'Hardcopy variable substitution' in the *User and Administrator* manual.

Overview

The control statements and parameters in which the variables may be used for substitution of current values are:

Statement	Parameters
APPL statement	DATA, and DESC and ACB parameterrs.
HCFORMAT statement	HEADER and TRAILER parameters.
HCROUTE statement	various parameters.
PANEL statement	LET assignments.
SCRIPT statement	LET assignments, AUDITMSG, and USERMSG parameters.
SESSION parameter	ACB, DATA, and DESC subparameters.

When variables are used to substitute a current value in a configuration control statement parameter, for example a user name, the variable name must be preceded immediately by an ampersand (&). The variable should be followed immediately either by an ampersand, or a space. For example:

```
Data &t_user
Data &t_user&
```

To use a portion of a variable, i.e., displacement and length, specify:

```
Data '&t_user(2,3)&'
```

When two or more variables are to be concatenated, there must be two ampersands between each. One to terminate the variable name, the second to indicate that a new variable name is beginning. If only one ampersand is specified, then the second variable is taken as a literal.

Variables used in logic statements, or in field by field PANEL definitions, do not require any preceding special character, except when the variable is contained in a literal, for example, in a LET statement.

The variables fall into two major groups, those supplied by Session Manager, and those that are user definable. The Session Manager supplied variables have predefined names, while user-definable variables must conform to certain naming conventions described at the end of this chapter in 'User definable variables' on page 249.

Session Manager supplied variables

Session Manager provides a number of variables with predefined meanings. Many of them refer to the parameters defined, or assumed, for the terminal user and initially reflect values obtained from the configuration data. Some of the variables can be updated, either by a user typing in a new value in a panel; by logic executed in a SCRIPT or PANEL; or by the User exit calling the Variable Access Routine.

Many of the Session Manager supplied variables are either useful or essential for PANEL definitions, for example the 't_command' variable.

Prefixes for variables

The supplied variables are generally prefixed by the letter t and an underscore character (t_), except for SESSION variables, which are prefixed with an s and an underscore (s_), window variables, which are prefixed with a w and an underscore (w_), and exit script variables, which are prefixed with either ec or en and an underscore (ec_, en_). There are also some special variables which are unprefixed.

Groupings for variables

The Session Manager supplied variables are tabulated next and are grouped under headings which indicate where each set of variables may be used. Certain variables are also only applicable in certain operating environments. The Session Manager products to which they apply are listed where this is the case.

'M' heading in tables

In the tables that follow, the second column – headed **M** –contains either R/O for read-only variables, R/W for read-write variables which are modifiable, or Y/N for variables which are modifiable only under certain conditions. Where this is the case, the instances where the variable may be updated are explained in the corresponding description text.

Global variables

Global variables may be used on any parameter that accepts variable substitution, although on certain parameters their use would be nonsensical. Their main use would be in the definition of panels.

Variable	M	Format	Description
t_applid	R/O	xxxxxxxx	This is the ACB name from the `SYSTEM` statement.
t_authclass	R/O	xxxxxxxx	Used by the E21 exit. Contains the ESM class name which will be used to determine a user's `AUTH` and `OLACLASS` settings.
t_authresn	R/O	xxx ... xxx	Used by the E21 exit. Contains the resource name which when appended with the `AUTH` value will be used to determine a user's `AUTH` setting. The resource name can be made up of any alphanumeric characters, including periods ('.'), and can be up to the length permitted by the defined ESM class, minus one character.
t_company	R/O	xxx ... xxx	The company name: `Macro 4 Limited - a division of UNICOM Systems, Inc.`
t_copyr	R/O	xxx ... xxx	The copyright notice: `(c) Copyright 1982-`*`yyyy`*` All Rights Reserved. Macro 4 Limited - a division of UNICOM Systems, Inc.`
t_config_suf	R/O	xx	The current Session Manager instance's configuration suffix value. To use it in the configuration, prefix and suffix the variable with two consecutive percent signs: `%%t_config_suf%%`.
t_dapplcheck	R/O	x	Set to 'Y' or 'N'. Used when adding a dynamic session and if set to 'Y' will check to ensure that an `APPL` definitions exists for the application.
t_date	R/O	dd/mm/yy or mm/dd/yy	Today's date. Format depends on `SYSTEM` statement parameters.

Variable	M	Format	Description
t_date_l	R/O	dd/mm/yyyy or mm/dd/yyyy	Same as t_date but with 4 digit year.
t_dynmalog	R/O	x	Used by E22 exit. Contains the log attribute value 'Y' or 'N'.
t_dynmautsthid	R/O	x	Used by E22 exit. Contains the autostart or autoselect value 'Y' or 'N'. See 'SYSTEM statement' chapter in the *Technical Reference*.
t_dynmclass	R/O	xxxxxxxx	Used by the E22 exit. Contains the ESM class name which will be used to determine which applications the user has authority to access.
t_dynmdropsess	R/O	x	Used by E22 exit. Contains the drop attribute value 'Y' or 'N'. See 'SYSTEM statement' chapter in the Technical *Reference*.
t_dynmhide	R/O	x	Used by E22 exit. Contains the hide attribute value 'Y' or 'N'. See 'SYSTEM statement' chapter in the *Technical Reference*.
t_dynmlogmax	R/O	nnnn	Used by E22 exit. Contains the maximum permitted number of log entries, from 0 to 9999. See 'SYSTEM statement' chapter in the *Technical Reference*.
t_dynmresnm	R/O	xxx ... xxx	Used by the E22 exit. Contains the resource name, which will be appended with either the APPL name or APPLID name, will be used to determine which applications the user has authority to access when dynamic menus are being used.
t_dynmtype	R/O	xxxxxxxx	Used by E22 exit. Contains the resource type value APPL or VTAMAPPL to indicate whether the appl name or the VTAM applid should be used when checking a user's access authorization with the ESM. See 'SYSTEM statement' chapter in the *Technical Reference*.

Variable	M	Format	Description
t_esmprfacc	R/O	x	Created by the SECURITY parameter of the SYSTEM statement. Contains the value 'Y' or 'N', and is used when the ESM cannot determine if a user would have access to the generated resource name (a combination of the setting in ESMPRFRSNM and the Session Manager PROFILE name). It determines the user's Session Manager PROFILEs. A setting of Yes or ON will allow the PROFILE to be assigned to the user. A setting of No or OFF will not allow the PROFILE to be assigned to the user.
t_esmprfclnm	R/O	xxxxxxxx	Created by the SECURITY parameter of the SYSTEM statement. Contains the ESM class name which will be used when checking if the user has access to the Session Manager PROFILE.
t_esmprfrsnm	R/O	xxx ... xxx	Created by the SECURITY parameter of the SYSTEM statement. Contains the ESM PROFILE resource name, which will be prefixed to the Session Manager PROFILE name to provide the resource name to be checked by the ESM. The resource name can be made up of any alphanumeric characters, including periods ('.'), and can be up to the length permitted by the defined ESM class, minus the length of the Session Manager PROFILE name.
t_feature	R/O	nnnn ... nnnn	Sixteen-character list of feature bits set. All feature bits should be set to 'on' – that is, '1111111111111111'.
t_genresname	R/O	xxxxxxxx	If set, is used by the E21 Exit when issuing a VERIFY to the ESM. Determines whether the Session Manager generic resource name, stored in the SYSTEM statement parameter GENRESNAME, is to be used with the Express Logon Facility instead of the Session Manager instance name.

Variable	M	Format	Description
t_global_msg	R/W	x	Controls the scope of the BROADCAST and MSG commands – either global (sent to all other Session Manager instances in same Sysplex group) or local. The default, if t_global_msg hasn't been set for the task, or if it has been set to 'Y', is whatever is set by the GLOBALMESSAGES sub-parameter of the SYSPLEXGROUP parameter. Default for the sub-parameter is 'Y'. If the value of t_global_msg is set to 'N' it overrides GLOBALMESSAGES and means the scope is local.
t_global_msgdef	R/O	x	Reflects the value ('Y' or 'N') of the GLOBALMESSAGES sub-parameter of the SYSPLEXGROUP parameter on the SYSTEM statement.
t_hardenu	R/O	x	'Y' or 'N' to indicate the value of the HARDENUSER parameter that determines whether a new non-generic user is created when a generic userid is used for signon.
t_mxcpass	R/O	x	'Y' or 'N' to indicate whether or not an external security manager, such as RACF, supports mixed case passwords.
t_n	R/O	xxxxx	S/MGR ('short' program name)
t_olaresn	R/O	xxx ... xxx	Used by the E21 exit. Contains the resource name which when appended with the OLACLASS value will be used to determine a user's OLACLASS setting. The resource name can be made up of any alphanumeric characters, including periods ('.'), and can be up to the length permitted by the defined ESM class, minus two characters.
t_opsys	R/O	z/OS	
t_p	R/O	xxx	ISZ (program prefix)
t_pgmname	R/O	xxx ... xxx	5697-N61 IBM Session Manager for z/OS *v.rmmX* ('long' program name)

Variable	M	Format	Description
t_security	R/O	x	'Y' or 'N' to indicate whether security is activated for the system, or whether there will be no security restrictions in place. The value of this variable is only meaningful if the ISZE21SF security exit is being used.
t_signonaccess	R/O	x	Used by the E21 exit. Contains the value 'Y' or 'N', and is used when the ESM cannot determine if a user would have access to the generated resource name (a combination of the setting in SIGNONRESNAME and the Session Manager ACB name). It determines if a user is allowed to log on to the Session Manager application. A setting of Yes or ON will allow the user to be logged on. A setting of No or OFF will cause the logon to be revoked.
t_signonclass	R/O	xxxxxxxx	Used by the E21 exit. Contains the ESM class name which will be used when checking if the user has access to the Session Manager application name.
t_signonresname	R/O	xxx ... xxx	Used by the E21 exit. Contains the sign on resource name, which will be appended with the Session Manager ACB name. The resource name can be made up of any alphanumeric characters, including periods ('.'), and can be up to the length permitted by the defined ESM class, minus the length of the Session Manager ACB name.
t_sos_msg	R/O	xxx ... xxx	The Short-on-Storage message, if an S-O-S condition exists, otherwise blank.
t_sysname	R/O	xxxxxxxx	The z/OS system name that Session Manager is running on.
t_tcp_stack	R/O	xxx ... xxx	The type of TCP/IP stack that Session Manager is using: HPNS.

Variable	M	Format	Description
t_terminalaccess	R/O	x	Used by the E21 exit. Contains the value 'Y' or 'N', and is used when the ESM cannot determine if a user would have access to the generated resource name (a combination of the setting in `TERMINALRESNAME` and the terminal name). It determines if a user is allowed to log on to the Session Manager application from this terminal. A setting of `Yes` or `ON` will allow the user to be logged on. A setting of `No` or `OFF` will cause the logon to be revoked.
t_terminalclass	R/O	xxxxxxxx	Used by the E21 exit. Contains the ESM class name which will be used when checking if the terminal being used by the user has access to Session Manager.
t_terminalresnam	R/O	xxx ... xxx	Used by the E21 exit. Contains the terminal resource name, which will be appended with the terminal name, which will be used when checking if the terminal being used by the user has access to Session Manager. The resource name can be made up of any alphanumeric characters, including periods ('.'), and can be up to the length permitted by the defined ESM class, minus the length of the terminal name.
t_time	R/O	hh.mm.ss	Time of day
t_tvnode	R/O	xxxxxxxx	The name specified by LOCALNODE in the system where the script or panel is running.

z/OS system symbols

During startup of Session Manager for z/OS, all z/OS system symbols are identified, and made available as Session Manager global variables.

Each variable has the form `GC_MVS_`*`symbolname`*.

For example, if a z/OS system symbol has the name `&SYSNAME`, and a value '`M4ZOSA`', then a global variable is created during startup, called `GC_MVS_SYSNAME`, and has the value '`M4ZOSA`'.

The reading of system symbols and creation of global variables takes place before other configuration parameters are read. This means that all z/OS system symbols are available as Session Manager variables during configuration processing.

Panel variables

Common panel variables

Common panel variables may be used on any panel definition, and some of them must be defined for certain panels.

Variable	M	Format	Description

t_aid	R/W	x	The last terminal aid character. It is the one byte code which defines the attention or PF key last pressed by the user. It may be compared or set to key names such as ENTER, PA1, REQ, PF1, PF01, OID, MAGNET, PEN but these should not be enclosed in delimiters, for example:
			
			`If t_aid = pa1`
			`  then let t_aid = enter`
			`End`

t_aid_c	R/O	xxxx	Character representation of the last terminal aid character. The value may be from PF1 to PF24, or PA1 to PA3, ENTER, CLEAR, MAGNET, REQ, PEN, or OID. It should be enclosed in delimiters when used in TPSL statements, for example: `If t_aid_c = 'PA1'`
t_cmd_ok	R/W	x	'Y' or 'N' to indicate whether or not the user's entered command is removed from t_command or left in it. It may be set to 'N' by a command script when the command script also sets the value of t_command and can be useful if the command script establishes an error (possibly in the entered parameters) and wishes to leave the user's entered command in t_command.

Variable	M	Format	Description

t_command	R/W	xxx ... xxx	Defines the command input field on PANELs. It is advisable to place it in the HEADER or TRAILER portion to ensure that a command field is always available. It may be lost if placed in the pageable portion. The field may usefully be updated in the PROCESS sub-section, for example:
			`If t_aid = pa2`
			`  then let t_command =`
			`  'Q ISZMGR'`
			The field is set to spaces, after the command has been processed, if the command input is valid.

t_cursor_name	R/W	xxxxxxxx	Returns the variable name of the field where the cursor was in the Process section and is cleared before a Header section is executed. The name of a field may be assigned to it in a HEADER, CONTENT, or TRAILER section to cause the cursor to be positioned. For example: `Let t_cursor_name = 't_command'`
t_cursfld	R/O	xxxxxxxx	Returns the name of the field where the cursor was in input, without any subscript.
t_curssub	R/O	x	Returns the subscript value of the field name where the cursor was on input.
t_lastsess	R/O	nnnn	Contains the menu session number of the last session accessed by the user. Please note that this is a consecutive number derived from the menu order and is not necessarily the same as the session number as defined in the config.
t_menutop	R/W	nnnn	Contains the line number of the session currently being displayed at the top of the session list on the menu panel. Modifying this value will result in the session list being scrolled so that the line whose number is contained within t_menutop is the first line displayed on the session list. This variable only applies to the menu panel.

Variable	M	Format	Description
t_menuntop	R/O	nnnn	When a scrolling command is issued, t_menutop (see above) is updated to the line number for the new top line displayed on the menu. However, this doesn't happen until the PROCESS section is executed. To obtain this value whilst building the HEADER, CONTENT or TRAILER section, t_menuntop must be used. This variable only applies to the menu panel.
t_message	R/W	xxx ... xxx	Defines the field to contain Session Manager error messages. It is advisable to place it on most PANEL definitions, preferably in the HEADER or TRAILER portion. It is set to spaces as soon as the panel containing the error message has been displayed. It can be assigned a value in either a panel or a script, e.g., it may be set by a Command script.
t_more_lines	R/O	x	'Y' or 'N' to indicate whether or not there are more lines to display from the CONTENT section of the PANEL statement.
t_panel	R/W	xxxxxxxx	Returns the name of the PANEL statement defining the current panel display. It may be updated by direct input from the terminal, or by a LET operation, but any change will be ignored if the new value is not the name of a PANEL statement. Care must be taken to ensure that any new PANEL is capable of supporting the same function as the original PANEL.
t_reccnt	R/O	nnn	Contains the count of recorded data streams. This is not restricted to Replay panels; it can be accessed by all panels and scripts.
t_repdof	R/W	nnn	Decimal offset to start the display of a PLAYHEX, PLAYDS REPLAY data stream. Replay panels only.
t_rephdr	R/O	xxx ... xxx	Data extracted from the buffer to provide an identifying header on REPLAY screens. Replay panels only.
t_replen	R/O	nnnn	Length of the buffer currently being replayed. Replay panels only.

Signon panel variables

These variables are available for definition on signon panels. The values are also made available to the E21 exit point of the User exit.

Variable	M	Format	Description
t_affinity	R/W	xxxxxxxx	Defines the system-acbname of a secondary Session Manager system. If it is defined, the signon is passed directly to this second Session Manager, along with all signon data. The value of this variable is made available to the E21 routine in the User exit in the primary Session Manager and may be modified within the routine.
t_npass	R/W	xxxxxxxx	Defines the new password field, and when defined on a panel should be specified with an attribute of NONDISP. The value of this variable is made available to the E21 routine in the User exit.
t_nprof	R/W	xxxxxxxx	Defines the new profile field. The value of this variable is made available to the E21 routine in the User exit, and may be modified within the routine.

Special Session Manager displays

Special displays are those which display Session Manager preformatted data, such as those used for QUERY command replies, displaying the Audit file, replaying recorded data, and displaying broadcasts.

Note Entering a paging command on a panel that uses a set of t_data variables causes the values of the t_data variable to be scrolled; the format of the panel does not change. This differs from other supplied panels such as the Menu and Help panels where paging commands cause the format of the panel to be moved.

Variable	M	Format	Description
t_data	R/O	xxx ... xxx	Defines the area on a screen which is to contain preformatted Session Manager data, such as query reply lines or broadcast message details. t_data is normally placed in the CONTENT lines of the PANEL statement. Panels using t_data variables control paging internally. For this reason, the logic used to display t_data in the Content section should not be altered as some data may then not be displayed.

Variable	M	Format	Description
t_tskid	R/O	xxx ... xxx	Defines the function that is using the panel. It identifies the Session Manager task identifier of the task displaying the panel. Most tasks have the format *xxxnnnnn*, where *xxx* is the Session Manager task type and *nnnnn* is the task sequence number. A user's tasks can be displayed by entering the QTASK command.

Terminal variables

Terminal description variables

Terminal description variables contain values pertaining to the terminal, and may not be updated.

Variable	M	Format	Description
t_apl	R/O	x	'Y' or 'N' to indicate whether or not the terminal supports the APL character set.
t_ccsid	R/O	xxxx	Terminal code page (285, for example).
t_colour	R/O	x	'Y' or 'N' to indicate whether or not the terminal supports 7 colours.
t_dbcs	R/O	x	'Y' or 'N' to indicate whether or not the terminal supports double-byte character sets.
t_depth_alt_part	R/O	nnn	The number of lines on the alternate screen of the terminal.
t_depth_def_part	R/O	nnn	The number of lines on the default screen of the terminal.
t_exthi	R/O	x	'Y' or 'N' to indicate whether or not the terminal supports extended highlighting.
t_ge	R/O	x	'Y' or 'N' to indicate whether or not the terminal supports the graphic escape character.
t_ipaddr	R/O	xxx.xxx.xxx.xxx	The TCP/IP IP address for terminals connected via the IBM CS TN3270 Server (not the Session Manager TN3270).
t_ipport	R/O	xxxxx	The TCP/IP port number for terminals connected via the IBM CS TN3270 Server (not the Session Manager TN3270).
t_logm	R/O	xxxxxxxx	The logmode entry name.
t_logm_bf	R/O	xxxxxxxx	The best fit logmode entry name for terminals with REBIND YES specified.
t_luname	R/O	xxxxxxxx	VTAM® luname (same as t_termid).
t_model	R/O	x	Terminal 3270 model id, can be 2, 3, 4, 5, or X.

Variable	M	Format	Description
t_mts_modl	R/O	xxxxxxxx	MTS model name.
t_mts_prt1	R/O	xxxxxxxx	MTS primary luname.
t_mts_prt2	R/O	xxxxxxxx	MTS secondary luname.
t_netid	R/O	xxxxxxxx	Terminal network name.
t_pss	R/O	x	'Y' or 'N' to indicate whether or not the terminal supports programmable symbol sets.
t_rtermcls	R/O	xxxx	The real terminal class – LU or TN32.
t_rtermid	R/O	xxxx	The VTAM luname.
t_sna	R/O	x	SNA indicator, S for SNA sessions, N for non-SNA.
t_termcls	R/O	xxxx	The terminal class known to Session Manager – LU or TN32.
t_termid	R/O	xxxxxxxx	VTAM luname.
t_termtyp	R/O	xxxx	The terminal type, 3277 or 3278.
t_width_alt_part	R/O	nnn	The number of columns on the alternate screen of the terminal.
t_width_def_part	R/O	nnn	The number of columns on the default screen of the terminal.
t_wsfrpq	R/O	x	'Y' or 'N' to indicate whether or not the terminal supports Write Structured Field Read Partition Query.

Terminal cursor variables

Terminal cursor variables provide the current position of the cursor on screen. Their main use is in the panel processing section of Session Manager panels.

See also the cursor positioning variables on page 208.

Variable	M	Format	Description
t_csr_row	R/O	nnn	Row number of current cursor position.
t_csr_col	R/O	nnn	Column number of current cursor position.
t_csr_ofs	R/O	nnnn	Screen offset of current cursor position.

User associated variables

User associated variables contain user related information, much of which is taken from the configuration data. Some variables may be updated.

Variable	M	Format	Description
t_actcmd	R/W	PF1-PF24, or NONE	Enables user to add `CMDACTIONKEY` to a menu as an input field, allowing user to view and change their command action key. See *Technical Reference* for details.
t_actprf	R/W	x	Enables user to add `COMMANDPRFXVAL` to a menu as an input field, allowing user to view and change their command prefix value. See *Technical Reference* for details.
t_appcdata	R/O	xxx ... xxx	The CICS agent signon data.
t_auth	Y/N	n	Authorization level associated with this user. This can be passed to the `E22` User exit for update, if required.
t_deauthmsg	Y/N	n	If set to 'Y', and an unauthorized script calls an authorized script, message 4007 is issued identifying the unauthorized script that has de-authorized the previously authorized script(s).
t_dsrng_from	R/O	nnnn	Contains a numeric value derived from the 'from' value defined in the `DSESSRANGE` parameter. Used when adding dynamic sessions.
t_dsrng_to	R/O	nnnn	Contains a numeric value derived from the 'to' value defined in the `DSESSRANGE` parameter. Used when adding dynamic sessions.
t_fmtopt	R/O	x	Indicates whether user has the authority to replay recorded screens in Dump/3270 Data Stream format, or just as the screens were recorded.
t_hcprof	R/W	xxxxxxxx	Hardcopy profile name. See also Hardcopy Option Variables.
t_logdata	R/O		Provides the VTAM logon data. A maximum of 256 characters has been allowed for this.
t_menu	R/O	xxxxxxxx	The current user's menu name.

Variable	M	Format	Description
t_msgid	R/O	x	'Y' or 'N' according to the current setting of the MSGID command. The user's default value being set by the common enduser parameter MSGID.
t_lang	R/W	xx	The user language.
t_node	R/O	xxxxxxxx	The name specified by LOCALNODE in the system where the user is logged on (i.e., some other system for remote sessions).
t_pass	Y/N	xxxxxxxx	The password entered on the Signon screen. This variable may only be modified on the Signon screen. On the Signon PANEL definition, it is advisable to define the password field with a NONDISP attribute. The variable value is available to the E21 routine in the User exit. It is set only when the User exit is used or the USER statement specifies PASSWORD.
t_prof	R/O	xxxxxxxx	User profile name. For Session Manager 1.1.05 and higher, a user can be associated with multiple profiles, so t_prof is subscriptable.
t_record	R/O	xxxx	Type of record being performed, either TERM, APPL, or blanks if not recording.
t_script	R/W	xxxxxxxx	Autoscript name. This may be set to any valid script name, or various trigger values, i.e: '.A' - use AUTOSCRIPT (default value) '.E' - use ENDSCRIPT '.I' - use script named in input 'script-name' - name of script specified on AUTOSEQ or SAUTSEQ statement.
t_security_class	Y/N	xx	The user's OLA security class. This variable should only be modified during signon processing (the E21 and E22 exits).
t_share	R/O	x	'Y' or 'N' according to the value of the parameter SHARE on a USER, TERMINAL, PROFILE or SYSTEM statement.

Variable	M	Format	Description
t_shareacb	R/O	xxxxxxxx	Name of the user's shared virtual terminal ACB, selected for one of the currently active sessions whose definitions nominated the APPL statement specified on the SHAREAPPL SYSTEM statement parameter.
t_signed_on	R/O	x	'Y' or 'N' according to the value of the parameter SIGNON on the TERMINAL statement.
t_signonpanel	R/O		Provides name of signon panel. Please note this variable will not be correct until after control is returned to Session Manager from the E11exit.
t_tn3270e	R/O	x	If Session Manager has successfully negotiated with the TN3270 client to use TN3270E protocols then this variable returns a value of 'Y'. Otherwise, and for all other terminal types, this variable returns a value of 'N'.
t_tn3270e_name	R/O	xxxxxxxx	Returns the device name, if any, supplied by the TN3270E client at connection time.
t_user	Y/N	xxxxxxxx	Userid. This may only be updated on the Signon screen.
t_user_acb	R/W	xxxxxxxx	The name of the current user's virtual terminal ACB, if any.
t_user_appl	R/O	xxxxxxxx	The name of the APPL definition used to nominate the ACB range from which the virtual terminal ACB was allocated.
t_user_qual	R/O	xxxxxxxx	Depending on the value of the MULTUSER parameter on the SYSTEM statement (see the *Technical Reference*), the 'user qualifier' can be created from the last four characters of the terminal LU name, or it is an eight-digit number in the range 1-99999999.
t_userdata1 - t_userdata5	R/O	xxx ... xxx	Fields (up to 32 bytes) for general use by user (for example, in scripts, exits and so on). No validation. The 5th field will be displayed as asterisks in an Online Administration display.

Variable	M	Format	Description
ucsautoparm	R/O	xxxxxxxx	If PARM Y is specified on the SAUTOSEQ parameter then Session Manager will scan for a parameter of up to eight characters. If one is found then the variable ucsautoparm will contain the value.
ucrestesc*n*	R/O	xxxxxxxx	When the escape sequence to invoke a script has been removed by the SAUTOSEQ REMESC parameter, this variable contains the escape sequence removed, followed by the Set Buffer Address, followed by an Insert Cursor. *n* will be replaced by the session number of the session in which the variable is created.
unrestescrow*n*	R/O	xxx	When the escape sequence to invoke a script has been removed by the SAUTOSEQ REMESC parameter, this variable contains the starting row for the escape sequence removed. *n* will be replaced by the session number of the session in which the variable is created.
unrestesccol*n*	R/O	xxx	When the escape sequence to invoke a script has been removed by the SAUTOSEQ REMESC parameter, this variable contains the starting column for the escape sequence removed. *n* will be replaced by the session number of the session in which the variable is created.

User escape command variables

For all escape commands, the case is significant for input. Thus, if t_esc is defined as 'HH', then if 'hh' is keyed as input, it will not be recognized as the menu escape command.

Variable	M	Format	Description
t_auto	R/W	xxxxxxxx	Auto script sequence/aid.
t_bwd	R/W	xxxxxxxx	Backward sequence/aid.
t_cut	R/W	xxxxxxxx	Cut sequence/aid.
t_esc	R/W	xxxxxxxx	Escape sequence/aid.
t_fwd	R/W	xxxxxxxx	Forward sequence/aid.
t_hcmd	R/W	xxxxxxxx	Hardcopy request sequence/aid.

Variable	M	Format	Description
t_paste	R/W	xxxxxxxx	Paste sequence/aid.
t_prev	R/W	xxxxxxxx	Previous sequence/aid.
t_pull	R/W	xxxxxxxx	Pull command sequence/aid.
t_push	R/W	xxxxxxxx	Push command sequence/aid.

Session detail variables

All the variables relating to details of the session definition are prefixed by s_. When using session detail variables to define a Menu screen, the variables should either be subscripted within a loop using one of the special subscript variables sub1 to sub9, or should be referred to explicitly by a session subscript number appended to the variable name, for example, s_appl.*nnn*, where *nnn* reflects the position of the session definition on the Menu screen. For example, if two sessions were defined on a Menu screen, the first for session 100, the second for PF13, then a subscript with a value of 1 would be associated with session detail number 13, and a subscript with a value of 2 would be associated with session detail number 100.

In SCRIPT and PANEL statements, unsubscripted session variables are assumed to refer to the current session. If a session variable for an alternative session is intended, then that variable should be subscripted, or referenced by a specific subscript number. The variable s_s may be used to refer to the position of a session definition on the Menu screen.

Logon data may contain variables, i.e., s_acb, s_logd and s_desc and these may or may not be subscripted.

For example, a SESSION parameter may contain:

```
DATA INIT&t_user&
```

to define logon data to an application. If &t_user& is substituted by FRED, then s_logd, after substitution, gives INITFRED.

Any updates that are made to s_ variables (other than s_start, s_desc, s_auto_script, s_auto_seq, and s_brdvar) while the session is processing, including the Initscript and E31 phases, and when the session is active, last for the duration of the session only.

Variable	M	Format	Description
s_a	R/O	nnn	Status (numeric).
s_aa	R/O	xxx	Status (character).

The two status fields are directly linked and reflect application or session status as follows:

For inactive sessions, describing the application

s_a	s_aa	
0	???	availability unknown
10	(X)	unavailable
20	(B)	blocked
25	(R)	remote session - remote node is active
30	(A)	available

For active sessions, describing the session itself

s_a	s_aa	
40	-E-	terminating
50	-A-	active (no read/write outstanding)
60	-I-	initiating
65	-S-	session ended
70	-Q-	read/write data outstanding - busy sent (SNA only)
80	***	write data outstanding
90	*Q*	write data outstanding - work queued
100	!!!	read outstanding
110	!Q!	read outstanding - work queued

Or to identify a command session

200	CMD	command session (that is, s_cmd is defined for the session)

The s_aa values are defined (as shown above) in messages 404I to 416I, 505I, and 729I and may be modified using the configuration MESSAGE statement. If the value specified exceeds 30 characters, it is truncated on the right.

Variable	M	Format	Description
s_acb	Y/N	xxxxxxxx	ACB name, after substitution. This field is the 8-character ACB name derived from s_acb_x at the time of display or reference, or, for an active session, the name of the ACB in use. This field is not modifiable for active sessions, but is modifiable for inactive sessions. Modification sets s_acb_x to the same value.
s_acb_x	Y/N	xxx ... xxx	ACB name, before substitution. For active sessions, the name of the ACB in use, and thus not updateable. For inactive sessions the ACB name as defined in the configuration, which may contain references to other variables. It may be updated by terminal input, or by a LET operation. It is not restricted to 8 characters.
s_allowesc	R/W	x	Specified as Y or N to signify whether Session Manager escape sequences are recognized or not

Variable	M	Format	Description
s_appl	Y/N	xxxxxxxx	APPL statement name or VTAM application name. Specified originally by using the SESSION/KEY configuration parameter APPLID. If s_ref is 'R' s_appl is the APPL statement name. If s_ref is not 'R' or there is no APPL statement, s_appl is taken to be the applid of the session. It is not updateable for active sessions, or when s_cmd is defined for the session, or in an Initscript.
s_applid	Y/N	xxxxxxxx	Application name. This is the name specified on the APPL statement APPLID parameter if REFAPPL is set on for the session, and so may be different to s_appl. If REFAPPL is off, then s_applid is taken from the SESSION sub-statement APPLID parameter. This field cannot be updated directly on the menu, only indirectly by changing s_appl. It may be updated by an Initscript without affecting the value of s_appl.
s_auto_seq	R/W	xxxxxxxx	This allows an SAUTOSEQ to be specified dynamically for each session. It is initialized to null. When set by a script, exit, or panel, the sequence or key assigned to s_auto_seq is checked before SAUTOSEQ definitions. The script name or its location must be assigned to the s_auto_script variable.
s_auto_script	R/W	xxxxxxxx	This identifies the script name or script name location '.A', '.E', or '.I'., when the sequence or key in variable s_aut_seq has been entered for this session or the session identified by the subscript. If a value of '.A' is stored, then the script specified by the AUTOSCRIPT parameter in operation for this session is used. If a value of '.E' is stored, then the script specified by the ENDSCRIPT parameter in operation for this session is used. If a value of '.I' is stored, then the script name entered following the s_auto_seq value is used. If s_auto_seq specifies a function key then only the script name is entered.
s_blnkscrpt	R/O	xxxxxxxx	Returns the script named in the BLANKSCript configuration parameter.

Variable	M	Format	Description
s_brdvar	R/W	xxx ... xxx	BROADCAST command id-pattern. This is set by the BRDVAR parameter of the APPL or PROFILE statement. It can also be set in a script.
s_cmd	Y/N	xxx ... xxx	Session Manager command. Specified originally by using the SESSION/KEY configuration sub-statement CMD parameter. If defined for a session, the variables associated with a session are not updateable.
s_cmdret	R/W	x	Contains the value of 'Y' or 'N'. Following the exit from a command dialog invoked by an ISZCMD/A session script verb, this determines whether the user returns to the invoking session (if 'Y') or to the menu (if the default, 'N').
s_cols	R/O	nnn	Session screen width (columns) - normal size.
s_colsa	R/O	nnn	Session screen width (columns) - alternate size.
s_conceal	R/W	x	Contains the value of 'Y' or 'N'. The default is 'N'. If s_conceal is set to 'Y', the session will not be visible on the user's menu.
s_desc	R/W	xxx ... xxx	Session description.
s_desc_d	R/O	xxx ... xxx	Session description but treated as display only. Thus any function performed on this variable, where it contains sensitive data (that is, t_pass), will return '********' and not the actual value.
s_dropsess	R/O	x	Contains the value of 'Y' or 'N' according to the value of common session parameter DROP_SESSION.
s_escape	R/W	x	Contains the value of 'Y' or 'N'. The default is 'Y'. If s_escape is set to 'Y', all escapes, including CUT and PASTE, will be enabled for the session. If s_escape is set to 'N', all escapes will be disabled for the session.
s_flash	Y/N	xxx ... xxx	Application flash. This field is modifiable only with the 'FLASH APPL' command.
s_hidden	R/O	x	'Y' or 'N' according to the value of the common session parameter HIDE.

Variable	M	Format	Description
s_logd	Y/N	xxx ... xxx	Logon data, after substitution. The logon data string derived from s_logd_x at the time of display or reference, or, for an active session, the logon data in use. It is not updateable for active sessions. It may be modified for inactive sessions. If the value is updated then s_logd_x is set to the same value.
s_logd_x	Y/N	xxx ... xxx	Logon data, before substitution. For active sessions, the logon data in use, and thus not updateable. For inactive sessions, it contains the logon data as defined in the configuration DATA parameter for the session. It may contain references to other variables and is modifiable. Modification also sets s_logd.
s_logm	Y/N	xxxxxxxx	Logmode entry name.This may only be updated for an inactive session. It is only validated by VTAM, when the session is selected from the menu.
s_miser	R/O	n	The value of the s_miser variable indicates the status of MISER for the session. Possible values are: 0 MISER OFF 1 MISER PENDING 2 MISER ON Note that the value of s_miser will always be 0 until the session is established.Values: 3 to 6 indicate that MISER is temporarily disabled. Possible values are: 3 A data stream contained a create partition structure, other than a create explicit partition zero the same size as the screen. 4 A data stream contained an asynchronous read. 5 A data stream contained graphic structures. 6 User exit E33 generated the data stream.

Variable	M	Format	Description
s_model	R/O	x	Session 3270 model code. It is not directly modifiable since it is derived from the session logmode. It is a character variable taking one of the values 2,3,4,5 or X.
s_mts_modl	R/W	xxxxxxxx	Session MTS model name
s_mts_prt1	R/W	xxxxxxxx	Session MTS primary luname
s_mts_prt2	Y/N	xxxxxxxx	Session MTS secondary luname Note: If all three s_mts variables are set to '0' then no MTS information is passed to| the session. If data is defined at session level, it replaces that set at user level (t_mts). If no data is defined at this level the values of t_mts_xxx are used. The fields supplied must start with an alphabetic character to be acceptable to VTAM. Note: As all the MTS values are used by VTAM at session start, modifying the contents once the session has started will have no effect.
s_netid	R/W	xxxxxxxx	The name of the network where the application is running, if different from the network where Session Manager is running.
s_node	Y/N	xxxxxxxx	The name specified by LOCALNODE in the system where the session should start. The same as REMOTE specification on the session. This variable can only be updated before session start, e.g., in menu processing or in the Initscript.
s_n	R/O	nnn	Session detail number. This is the number following the SESSION sub-statement, or the PF key number in the KEY sub-statement. It does not necessarily equal the subscript value, s_s, which is associated with the session. E.g., s_n.2 could be 101 if SESSION 101 is the second session defined for the user in question.
s_pstkappl	R/O	xxxxxxxx	If set, is used by the passticket exit to generate a passticket for an application.
s_pstkuser	R/O	xxxxxxxx	If set, is used by the passticket exit to generate a passticket for an application.

Variable	M	Format	Description
s_ref	Y/N	x	The REFAPPL option. Specified originally by using the SESSION or KEY configuration parameter REFAPPL. It may have the value R if REFAPPL is YES, or N for REFAPPL NO. It may be modified to R or Y to set REFAPPL on; a value of N turns REFAPPL off. It cannot be updated for active sessions, or when s_cmd is defined for the session, or in an Initscript. It can be updated at the Menu panel.
s_rows	R/O	nnn	Session screen depth (rows) - normal size.
s_rowsa	R/O	nnn	Session screen depth (rows) - alternate size.
s_runinitsc	R/O	x	'Y' or 'N' according to whether the running of the session 'initialization' script has been disabled or (re-)enabled using the INITSC command. For details, see the *Technical Reference*.
s_runstartsc	R/O	x	'Y' or 'N' according to whether the running of the session start script has been disabled or (re-)enabled using the STARTSC command. For details, see the *Technical Reference*.
s_s	R/O	nnn	Session subscript number. This is the position of the session definition on the PROFILE, USER, or TERMINAL statement. In scripts, a LET statement must be used to assign this value to a subscript for a user variable – LET SUB1=s_s, for example. This is not needed for session variables. In panel definitions, the subscripts sub1 to sub9 can be used.

Variable	M	Format	Description
s_script_cmds	R/W	x	'Y' or 'N' according to whether ISZCMD/A supports command scripts, (as well as native Session Manager commands), and whether WAITAPPL, WAITDATA and WAITTERM supports SAUTOSEQ scripts. If set to 'Y' then ISZCMD/A will support command scripts by converting the ISZCMD/A into the equivalent of a TPSL call, and similarly WAITAPPL, WAITDATA and WAITTERM will support SAUTOSEQ scripts by temporarily converting the wait into a TPSL call.
s_sel	R/W	xxxxxxxx	The last selection command. s_sel only provides the last configured selection command for the session. If it is modified, by terminal input, for example, it becomes the only selection command. It may be defined as a PA or a PF key, or as a synonym for the application. With a synonym selection command, the case as input is not significant; for example, upper case matches with lower case.
s_sequence	R/O	nnnn	Contains the menu sequence number for the session, which is used by Session Manager to order sessions.
s_sescount	R/O	nnnnn	Contains the current number of active sessions with the application identified in the session's s_applid variable. s_sescount can be subscripted.
s_sessdata1 - s_sessdata5	R/O	xxx ... xxx	Fields (up to 32 bytes) for general use by user (for example, in scripts, exits and so on). May optionally be defined with subscripts. No validation. The 5th field will be displayed as asterisks in an Online Administration display.
s_sestype	R/O	nnnn	Contains the session type, which is used by Session Manager to eliminate duplicate sessions.
s_shareterm	R/O	xxxxxxxx	Contains the name of the terminal owning a session started by a SHARESESS or SHAREDISC user.
s_size	R/O	nnnnn	Session screen size (number of characters) - normal size. This is obtained my multiplying s_cols by s_rows.

Variable	M	Format	Description
s_sizea	R/O	nnnnn	Session screen size (number of characters) - alternate size. This is obtained my multiplying s_colsa by s_rowsa.
s_sna	R/O	x	SNA indicator. 'S' for active SNA sessions, 'N' for active non-SNA sessions, otherwise blank. It is not directly modifiable since it is derived from the session logmode.
s_start	Y/N	x	Signifies whether the session is autostarted or not, as denoted by 'Y' or 'N'. Can be set dynamically in the E21 exit to autostart a session.
s_status_update	Y/N	x	Default is 'N'. If a session script sets the value to 'Y' then control will return to the verb that follows the session script's WAITDATA verb when the following events occur (in addition to the normal WAITDATA events): ▪ Status of a VTAM application defined in Session Manager changes (t_sendval.1 will be set to APPL). ▪ Status of a Sysplex node changes (t_sendval.1 will be set to NODE). ▪ Status of a Sysplex node or conventional VTAM node changes (t_sendval.1 will be set to APPL). In the case of Sysplex node therefore, WAITDATA (potentially) gets satisfied twice, once with a NODE and then with an APPL value in t_sendval.1. These events will set t_waitdata to 'S'.
s_tran	R/W	xxxxxxxx	The first transaction id. s_tran only provides the first configured transid command for the session. If it is modified, by terminal input, for example, it becomes the only transid. It may be defined as a PA or a PF key, or as a synonym for the application. With a synonym transid, the case as input is not significant, e.g., upper case matches with lower case.
s_standby	R/O	x	Session standby status. 'Y' indicates that the session is an HLA Standby session. 'N' indicates that the session is not a Standby session.

TCP/IP variables

TELNET panel variables

The following variables are used to provide TCP/IP TELNET support in Session Manager.

Variable	M	Format	Description
s_tcp_url	R/O	nnn	The Telnet Server's URL
s_telnet_hidec	R/O	x	Specifies whether the user is prompted for a password. If set to 'Y' the user is prompted to enter a password. and the TELNET panel should have the t_command variable set to non-display. This is set by the HIDECMD action, or if Session Manager automatically detects a password is being requested. If set to 'N', the user is not prompted for a password. The TELNET panel should display the value of t_command.

TCP/IP session variables

Variable	M	Format	Description
s_telnet_port	R/O	nnnn	Contains the port number in use for the session.
s_telnet_user	R/O	nnnn	Contains the userid for the Telnet session. This is taken from the DATA parameter input.
s_telnet_pswd	R/O	nnnn	Contains the password for the Telnet session. This is taken from the DATA parameter input
s_telnet_host	R/O	xxxxxx	Contains the host IP name or address.
s_tcp_stype	R/O	xxxxxx	Identifies the Telnet mode of operation in use: ▪ TNFULL – Telnet in full screen mode ▪ TNLINE – Telnet in line mode
s_telnet_output	R/O	xxx ... xxx	The next line after a WAITAPPL or WAITDATA from the line mode Telnet Server. (This allows you to develop your own line mode Telnet operation and displays.)
s_telnet_lupd	R/O	x	Qualifies the value in the s_telnet_output variable. If set to 'Y', the line is a new line, If set to 'N' it is an update to, or replacement for, the previous line.

Variable	M	Format	Description
s_telnet_lmore	R/O	x	Indicates whether the value in s_telnet_output variable is the last line (set to 'N') or if there are more lines to follow, (set to 'Y').
s_telnet_more	R/O	nnn	Indicates the number of output| lines queued. The default maximum queued, or unseen, is 150. This can be overridden using ln_telnet_wrkq variable. The range can be 10-300. If more than this maximum is received, the oldest lines are lost and replaced by message 744.
s_tn3270e	R/O	x	If Session Manager has successfully negotiated with the TN3270 Server to use TN3270E protocols then this variable returns a value of 'Y'. Otherwise, and for all other terminal types, this variable returns a value of 'N'.
s_tn3270e_dev	R/O	xxxxxxxx	Returns the device name, if any, supplied by the TN3270E Server at connection time.

Session statistics and response time monitor variables

The following variables are available only if the session has STATS YES **specified.** They can be used by panels, scripts and exit scripts. The E39 exit script is the place where the totals for a a session should be recorded.

Session statistics variables

Variable	M	Format	Description
s_stat_sict	R/O	nnnn	The count of the session inputs.
s_stat_siby	R/O	nnnnnnnn	The count of session input bytes.
s_stat_soct	R/O	nnnn	The count of session outputs.
s_stat_soby	R/O	nnnnnnnn	The count of session output bytes.
s_stat_ioct	R/O	nnnn	The count of MISER outputs.
s_stat_ioin	R/O	nnnnnnnn	The count of outputs in bytes before MISER.
s_stat_ioon	R/O	nnnnnnnn	The count of outputs in bytes after MISER.
s_stat_iict	R/O	nnnn	The count of MISER inputs.
s_stat_iiin	R/O	nnnnnnnn	The count of inputs in bytes before MISER.
s_stat_iion	R/O	nnnnnnnn	The count of inputs in bytes after MISER.
s_stat_ifct	R/O	nnnn	The count of application reads emulated by MISER.
s_stat_ifon	R/O	nnnnnnnn	The count of application reads emulated by MISER in bytes.
s_stat_ifct	R/O	nnnn	The count of internal reads emulated by MISER
s_stat_ibon	R/O	nnnnnnnn	The count of internal reads emulated by MISER in bytes.
s_stat_cmct	R/O	nnnn	The count of application output datastreams compressed.
s_stat_cmin	R/O	nnnnnnnn	The count of application data-streams before compression in bytes.
s_stat_cmon	R/O	nnnnnnnn	The count of application datastreams after compression in bytes

Response time monitor variables

Variable	M	Format	Description
s_rtm_rtmt1	R/O	nn.nn	RTM threshold from RTMT1 SYSTEM statement parameter
s_rtm_rtmt2	R/O	nn.nn	RTM threshold from RTMT2 SYSTEM statement parameter.
s_rtm_strtt	R/O	hh:mm:ss	Session start time.
s_rtm_strtd	R/O	mm/dd/yy or as varied in the Configuration	Session start date
s_rtm_restt	R/O	hh:mm:ss	Time RTM statistics were reset
s_rtm_restd	R/O	mm/dd/yy or as varied in the Configuration	Date RTM statistics were reset
s_rtm_restu	R/O	nnnnnnnn	Userid of user who reset RTM statistics.
s_rtm_restm	R/O	nnnnnnnn	Terminal id where RTM statistics were reset
s_rtm_restp	R/O	nnnnnnnn	Terminal type where RTM statistics were reset.
s_rtm_tot1	R/O	nnnn	Total responses that were within the RTMT1 threshold
s_rtm_tot2	R/O	nnnn	Total responses greater than RTMT1 threshold but within RTMT2 threshold.
s_rtm_tot3	R/O	nnnn	Total responses greater than RTMT2 threshold
s_rtm_totav	R/O	nn.nn	Average Total response time
s_rtm_totlg	R/O	nn.nn	Longest Total response time
s_rtm_totlt	R/O	hh:mm:ss	Time at which the longest Total response time occurred.
s_rtm_totld	R/O	mm/dd/yy or as varied in the Configuration	Date at which the longest Total response time occurred
s_rtm_net1	R/O	nnnn	Number of Network responses within RTMT1 threshold.
s_rtm_net2	R/O	nnnn	Number of Network responses greater than RTMT1 threshold but within RTMT2 threshold
s_rtm_net3	R/O	nnnn	Number of Network responses greater than RTMT2 threshold.
s_rtm_netav	R/O	nn.nn	Average Network response time

Variable	M	Format	Description
s_rtm_netlg	R/O	nn.nn	Longest Network response time
s_rtm_netlt	R/O	hh:mm:ss	Time at which the longest Network response time occurred.
s_rtm_netld	R/O	mm/dd/yy or as varied in the Configuration	Date at which the longest Network response time occurred.
s_rtm_app1	R/O	nnnn	Number of Application responses within RTMT1 threshold.
s_rtm_app2	R/O	nnnn	Number of Application responses greater than RTMT1 threshold but within RTMT2 threshold
s_rtm_app3	R/O	nnnn	Number of Application responses greater than RTMT2 threshold.
s_rtm_appav	R/O	nn.nn	Average Application response time.
s_rtm_applg	R/O	nn.nn	Longest Application response time.
s_rtm_applt	R/O	hh:mm:ss	Time at which the longest Application response time occurred.
s_rtm_appld	R/O	mm/dd/yy or as varied in the Configuration	Date at which the longest Application response time occurred

Facility-related variables

Hardcopy option variables

The various values are obtained from the HCOPTION parameters of the HCPROFILE configured for the user. The HCPROFILE name is contained in the User Associated variable, t_hcprof.

Subscripting may be used. For example, for a PANEL to display all HCOPTIONS available to the user. If no subscript is specified, the variable contains the value associated with the current hardcopy option. The HCOPTION command is used to select the hardcopy option. None of the hardcopy option variables are modifiable.

Variable	M	Format	Description
t_hcop	R/O	xxx ... xxx	Hardcopy option description.
t_hcop_n	R/O	nn	Number of the current hardcopy option.
t_hcformat	R/O	xxxxxxxx	Hardcopy format name
t_hcroute	R/O	xxxxxxxx	Hardcopy route name.
hmax	R/O	nnn	Number of hardcopy options defined in a profile.

Demonstration and Viewer variables

Demo-View variables contain values pertaining to the use of the Demonstration and View feature.

Variable	M	Format	Description
t_daut	R/O	x	Autocopy status. Possible values are ‘Y’ or ‘N’
t_dint	R/O	x	Current setting for Session Manager internal screens. This indicates if the screens are sent to viewers. Possible values are ‘Y’ or ‘N''
t_dkey	R/W	xxxxxxxx	Current demokey sequence.
t_dnview	R/O	nnn	The number of users currently viewing this demonstration.
t_dtermid	R/O	xxxxxxxx	An array containing the luname of each viewer.
t_duserid	R/O	xxxxxxxx	An array containing the userid of each viewer.
t_inview	R/W	nnn	Assisted input eligible viewer number.
t_inview_n	R/W	xxxxxxxx	Assisted input eligible viewer userid.
t_unview	R/W	nnn	Unassisted input eligible viewer number.

Variable	M	Format	Description
t_unview_n	R/W	xxxxxxxx	Unassisted input eligible viewer userid.

Application Builder variables

The following variables are used by Application Builder scripts.

Variable	M	Format	Description
t_sendval.n	R/O	xxxxxxxx	Each variable contains the individual value sent to a session using the SENDDATA/SENDVAL verb. A maximum of 10 can be sent.
t_sendvals	R/O	xxx	The total number of values sent using the SENDDATA/SENDVAL verb.

Window variables

Window variables - user level

The following variables give general information regarding the status of windowing for a user. All variables except t_w_status contain a default value:

Zero - For numeric-type variables

Blanks - For character-type variables

Variable	M	Format	Description
t_w_curses	R/O	nnn	Number of the session whose window currently contains the cursor.
t_w_esc	R/W	xxxxxxxx	Subscripted variable with up to 24 elements. Each element may contain a windowing escape sequence or key whose value and meaning is defined within the window script. If one of these escape sequences is entered at the terminal while window script processing has been suspended (following execution of the WAITEVENT verb), event 'ESCAPE' is recognized, and window script processing resumes. The value of the escape sequence is placed in variable t_w_input for inspection.
t_w_event	R/O	x	Name of the event that caused control to be passed back to the window script after a WAITEVENT command was executed. Possible values are: E = escape M = message/broadcast S = session end I = switch
t_w_id	R/O	nnn	Identifier of the currently active window.
t_w_input	R/O	xxxxxxxx	Most recently entered window escape sequence (see above).
t_w_msg	R/O	xxxxxxxx	Text of the message or broadcast which caused window script processing to resume with event MSG following execution of the WAITEVENT verb.

Variable	M	Format	Description
t_w_mstime	R/O	xxxxxxxx	Contains the time that the window broadcast or message was sent. The format is: hh:mm:ss.
t_w_msuser	R/O	xxxxxxxx	Contains the userid of the originator of the window broadcast or message.
t_w_name	R/O	xxxxxxxx	Name of the currently active window. Contains blanks if the currently active window is unnamed.
t_w_status	R/O	x	Current status of windowing. Possible values are: A = active I = inactive S = suspended
t_w_zoom	R/O	nnn	Identifier of the currently zoomed window. Set to zero if no window is zoomed.
t_windows	R/O	nnn	Number of windows and window control areas currently open.

Window variables - window level

The following variables are subscripted, with one copy of each per window. The subscript for each window is its window identifier; for the currently active window this value is held in the variable t_w_id. When no subscript is specified, the currently active window is assumed.

All variables contain a default value:

Zero	For numeric-type variables.
Blanks	For character-type variables and for both character and numeric-type variables when windowing is inactive, or the subscript does not refer to an open window, or no value was assigned.

Variable	M	Format	Description
w_active	R/O	x	Indicates whether the window is currently active. Possible values are: Y or N.
w_aid	R/O	x	Last AID value returned to the window.

Variable	M	Format	Description
w_border	R/O	xxxxxxxx	Name of the border-set in use with the window. Blanks if no border set is in use. A border set name of BOX indicates that the window was defined with the BORDER BOX parameter.
w_cdepth	R/O	nnn	Content depth of the window. Number of rows available for session output or window message.
w_csr_col	R/O	nnn	Column number of the cursor (position within the window, relative to the origin of the window).
w_csr_ofs	R/O	nnnn	Screen offset of the cursor (position within the window, relative to the origin of the window).
w_csr_row	R/O	nnn	Row number of the cursor (position within the window, relative to the origin of the window).
w_cwidth	R/O	nnn	Content width of the window. Number of columns available for session output or window message.
w_depth	R/O	nnn	Depth of the window (rows).
w_end_col	R/O	nnn	Column number of bottom right-hand corner of the window.
w_end_ofs	R/O	nnnn	Screen offset of bottom right-hand corner of the window.
w_end_row	R/O	nnn	Row number of bottom right-hand corner of the window.
w_name	R/O	xxxxxxxx	Name of the window. Blanks if window is unnamed.
w_org_col	R/O	nnn	Column number of top left-hand corner of the window.
w_org_ofs	R/O	nnnn	Screen offset of top left-hand corner of the window.
w_org_row	R/O	nnn	Row number of top left-hand corner of the window.
w_output	R/O	x	Indicates whether or not session output is reflected to the window when it is inactive. Possible values are: Y or N.
w_owned	R/O	nnn	Identifier of the owned window. Zero if window is not an owner.
w_owner	R/O	nnn	Identifier of the owning window. Zero if window is not owned.

Variable	M	Format	Description
w_scrollh	R/O	nnn	Horizontal scroll offset of the window.
w_scrollv	R/O	nnn	Vertical scroll offset of the window.
w_session	R/O	nnn	Number of the session running in the window. Zero if no session is running in the window.
w_visible	R/O	x	Indicates how much of the window is currently visible. Possible values are: Y window is fully visible P window is partly visible N window is not visible
w_wca	R/O	x	Indicates whether or not the window is a window control area. Possible values are: Y or N.
w_width	R/O	nnn	Width of the window (columns).
w_zoom	R/O	x	Indicates whether the window is zoomed or hidden. Possible values are: Z window is zoomed H window is hidden N window is normal size

Window variables - session level

The following variables are subscripted, with a copy of each one per window session.

Variable	M	Format	Description
s_w_id	R/O	nnn	Identifier of the window in which the session is running, or most recently opened window if the session is running in more than one window. Zero if the session is not running in a window.
s_w_name	R/O	xxxxxxxx	Name of the window in which the session is running, or most recently opened window if the session is running in more than one window. Blanks if the session is not running in a window, or if the window is not named.

Special variables

Variable	M	Format	Description
smax	R/W	nnn	The number of sessions defined for the user. It is useful in defining the DO loop which processes a user's sessions for Menu display. For Session Manager 1.1.05 and higher, smax can be written as well as read, and its value can be updated to a value less than or equal to the number of sessions defined.
sub1,sub2, ..., sub9		nnn	Subscript variables. These are special numeric variables which may be used as subscripts on user definable variables, and as subscripts on session and hardcopy option variables and windows.

TPSL and SCRIPT variables

Return code variables

These variables are set by the execution of some script statements and TPSL statements. They indicate the success or failure of the execution of that statement. Refer to the description of each statement for the values placed in the return codes.

Variable	M	Format	Description
t_rc	R/W	nnnnn	Return code. This variable is set by certain script verbs to indicate the success or failure of the operation.
t_result	R/W	nnnnn	Supplementary return code. This variable is set by certain script verbs to give further information for a particular return code. All other verbs set it to zero.

Script verb variables

The following variables are set by the execution of certain 'script verbs', for example OUTSCAN.

Variable	M	Format	Description
s_inscan_col	R/O	nnn	If the text specified by an INSCAN parameter is found in | the datastream, s_inscan_col is set to the column number of the beginning of the matching text.
s_inscan_row	R/O	nn	If the text specified by an INSCAN parameter is found in the datastream, s_inscan_row is set to the row number of the beginning of the matching text.
s_outscan_col	R/O	nnn	If the text specified by an OUTSCAN parameter is found in the data stream, s_outscan_col is set to the column number of the beginning of the matching text.
s_outscan_row	R/O	nn	If the text specified by an OUTSCAN parameter is found in the data stream, s_outscan_row is set to the row number of the beginning of the matching text.

Variable	M	Format	Description
t_inplen	R/O	nnnn	This is set to the length of the unprocessed data entered at the terminal at the last WAITTERM PASS or WAITDATA PASS. It is set to zero by the next INPUT PASS, or when there is no input from a WAITTERM or WAITDATA PASS.
t_recur_script	R/W	x	Set to 'Y' or 'N' by a script depending on whether the next TPSL CALL or VCALL to another script can be recursive (that is, script to be called is same as script that directly or indirectly called the current script). Value is set to 'N' (the default) after each CALL or VCALL. Scripts are aborted after the original calling script exceeds its DOMAX value in terms of number of calls.
t_scparms	R/W	xxx ... xxx	Contains the entire input that caused a command script to be called. For Session Manager 1.1.05 and higher, t_scparms can be written as well as read, so another script can set up these values and call the command script.
t_waitdata	R/O	x	When a WAITDATA verb completes, this variable is set to 'A' to indicate application output sent, to 'T' to indicate terminal input received, or to 'S' to indicate SENDDATA received. Note: This variable is not subscriptable.

User Exit variables

The following variables are used by Exit scripts to access User exit parameters. As far as possible, these are equivalent to what is parsed to the Assembler/COBOL User exit. See 'The Session Manager User Exit' in the *Installation and Customization* manual for further details.

Note The Timer Interval Exit E08 can be used as an exit script but it only uses the general variables applicable to all exit scripts.

All exits

Variable	M	Parm	Format	Description
ec_rcode	R/O	1	nnn	Call indicator *'Enn'*.
ec_reason	R/O	1	xxxxxxxx	The reason code. This is different for each exit.
ec_julian	R/O	1	xxxxxxxx	Julian date (yyddd) in decimal. (Note this is not packed as it is in the User exit).

Configuration statement processing exit point E05

Variable	M	Parm	Format	Description
ec05_name	R/O	4	xxx ... xxx	Member name.
ec05_keywd	R/O	4	xxx ... xxx	Statement keyword, or blank Maximum length 16 bytes.
ec05_record	R/O	5	xxx ... xxx	Logical record image. May be modified by the Exit script.

Application status change exit point E06

Variable	M	Parm	Format	Description
ec06_appl	R/O	4	xxxxxxxx	Application name.
ec06_nstatus	R/O	4	x	New status. 'Y' if available, 'N' if unavailable.
ec06_ostatus	R/O	4	x	Previous status. 'Y' if it was previously available, 'N' if it was unavailable, blank if status was not known.

Logon exit point E11

Variable	M	Parm	Format	Description
ec11_ttype	R/O	4	xxxx	Terminal task type.
ec11_term	R/O	4	xxxxxxxxx	Terminal name.
ec11_logd	R/O			Provides the VTAM logon data within the E11 exit.
ec11_logm	R/O	4	xxxxxxxxx	Logmode name.
ec11_bind	R/O	4	xxx ... xxx	Bind image. (Access by using MASKxx TPSL condition).
ec11_prof	R/O	5	xxxxxxxxx	Profile name.
ec11_sign	R/O	5	x	Signon required, 'Y' or 'N'.
ec11_signp	R/W			Provides the name of the signon panel within the E11 exit.
ec11_user	R/W	6	xxxxxxxxx	Userid.
ec11_pass	R/W	6	xxxxxxxxx	Password.
ec11_npass	R/W	6	xxxxxxxxx	New password.
ec11_nprof	R/W	6	xxxxxxxxx	New profile.
ec11_nsign	R/W	6	x	Signon required. 'Y' or 'N'.

Signon validation exit point E21

Variable	M	Parm	Format	Description
ec21_ttype	R/O	4	xxxx	Terminal task type.
ec21_term	R/O	4	xxxxxxxxx	Terminal name.
ec21_logm	R/O	4	xxxxxxxxx	Logmode name.
ec21_bind	R/O	4	xxx ... xxx	Bind image. (Access by using MASKxx TPSL condition).
ec21_euser	R/O	5	xxxxxxxxx	Entered userid.
ec21_epass	R/O	5	xxxxxxxxx	Entered password.
ec21_enpass	R/O	5	xxxxxxxxx	Entered new password.
ec21_eprof	R/O	5	xxxxxxxxx	Entered profile.
ec21_enode	R/O	5	xxxxxxxxx	Entered logon node.
ec21_config	R/O	6	x	User definition in configuration. 'Y' or 'N'.
ec21_cauth	R/O	6	xx	Configuration authority.
ec21_cpass	R/O	6	xxxxxxxxx	Configuration password.

Variable	M	Parm	Format	Description
ec21_cprof	R/O	6	xxxxxxxxx	Configuration profile.
ec21_cnode	R/O	6	xxxxxxxxx	Configuration affinity.
ec21_npass	R/W	7	xxxxxxxxx	New password.
ec21_nprof	R/W	7	xxxxxxxxx	New profile.
en21_nauth	R/W	7	xx	New authority.
ec21_ruser	R/W	7	x	Redisplay user value flag, 'Y' or 'N'.
ec21_rpass	R/W	7	x	Redisplay password flag, 'Y' or 'N'.
ec21_rnpass	R/W	7	x	Redisplay new password flag, 'Y' or 'N'.
ec21_rnprof	R/W	7	x	Redisplay new profile flag, 'Y' or 'N'.
ec21_nnode	R/W	7	xxxxxxxxx	New Session Manager node/affinity.
ec21_ndata	R/W	7	xxx ... xxx	Logon data. Maximum 256.

Signon completion exit point E22

Variable	M	Parm	Format	Description
ec22_ttype	R/O	4	xxxx	Terminal task type.
ec22_term	R/O	4	xxxxxxxxx	Terminal name.
ec22_logm	R/O	4	xxxxxxxxx	Logmode name.
ec22_bind	R/O	4	xxx ... xxx	Bind image. (Access by using MASKxx TPSL condition).
ec22_user	R/O	5	xxxxxxxx	Userid.
ec22_pass	R/O	5	xxxxxxxx	Password.
ec22_prof	R/O	5	xxxxxxxx	Profile name.
en22_auth	R/W	5	xx	Authority level. May be modified by the Exit script.
en22_autos	R/W	6	nn	Autoselect session detail number.

IDLEDISC/IDLELOGOFF/IDLELOCK timer expiry exit point E26

Variable	Parm	M	Format	Description
ec26_ttype	4	R/O	xxxx	Terminal task type.
ec26_term	4	R/O	xxxxxxxxx	Terminal name.

Variable	Parm	M	Format	Description
ec26_logm	4	R/O	xxxxxxxxx	Logmode name.
ec26_bind	4	R/O	xxx ... xxx	Bind image (access by using MASK*xx* TPSL condition).
ec26_user	5	R/O	xxxxxxxx	Userid.
ec26_prof	5	R/O	xxxxxxxx	Profile name.

Signoff exit point E29

Variable	M	Parm	Format	Description
ec29_ttype	R/O	4	xxxx	Terminal task type.
ec29_term	R/O	4	xxxxxxxxx	Terminal name.
ec29_logm	R/O	4	xxxxxxxxx	Logmode name.
ec29_bind	R/O	4	xxx ... xxx	Bind image (access by using MASK*xx* TPSL condition).
ec29_user	R/O	5	xxxxxxxx	Userid.
ec29_pass	R/O	5	xxxxxxxx	Password.
ec29_prof	R/O	5	xxxxxxxx	Profile name.
en29_auth	R/O	5	xx	Authority level.

Slave session pre-initiation exit point E31

Variable	M	Parm	Format	Description
ec31_ttype	R/O	4	xxxx	Terminal task type.
ec31_term	R/O	4	xxxxxxxxx	Terminal name.
ec31_logm	R/O	4	xxxxxxxxx	Logmode name.
ec31_bind	R/O	4	xxx ... xxx	Bind image. (Access by using MASKxx TPSL condition).
ec31_user	R/O	5	xxxxxxxx	Userid.
ec31_pass	R/O	5	xxxxxxxx	Password.
ec31_prof	R/O	5	xxxxxxxx	Profile name.
en31_auth	R/O	5	xx	Authority level.
ec31_taskida	R/O	6	SAAnnnnn	Session task, where nnnnn is the unique session number.
ec31_taskidb	R/O	6	xxxxxxxx	Session taskid, either userid or terminal name.
ec31_appl	R/W	6	xxxxxxxx	Applid.
ec31_acb	R/W	6	xxxxxxxx	ACB name.

Variable	M	Parm	Format	Description
ec31_slogm	R/W	6	xxxxxxxx	Logmode name.
ec31_script	R/W	6	xxxxxxxx	Startscript name.
ec31_data	R/W	6	xxx ... xxx	Logon data. Maximum of 256 bytes.

Slave session post-initiation exit point E33

Variable	M	Parm	Format	Description
ec33_ttype	R/O	4	xxxx	Terminal task type.
ec33_term	R/O	4	xxxxxxxxx	Terminal name.
ec33_logm	R/O	4	xxxxxxxxx	Logmode name.
ec33_bind	R/O	4	xxx ... xxx	Bind image. (Access by using MASKxx TPSL condition).
ec33_user	R/O	5	xxxxxxxx	Userid.
ec33_pass	R/O	5	xxxxxxxx	Password.
ec33_prof	R/O	5	xxxxxxxx	Profile name.
en33_auth	R/O	5	xx	Authority level.
ec33_taskida	R/O	6	SAAnnnnn	Session task, where nnnnn is the unique session number.
ec33_taskidb	R/O	6	xxxxxxxx	Session taskid, either userid or terminal name.
ec33_applid	R/O	6	xxxxxxxx	Applid.
ec33_acb	R/O	6	xxxxxxxx	ACB name.
ec33_appl	R/O	6	xxxxxxxx	Session Manager APPL name.
ec33_dstream	R/W	7	xxx ... xxx	Terminal data stream. Maximum length 256 bytes.

SIDLTIME timer expiry exit point E36

Variable	M	Parm	Format	Description
ec36_ttype	R/O	4	xxxx	Terminal task type.
ec36_term	R/O	4	xxxxxxxxx	Terminal name.
ec36_logm	R/O	4	xxxxxxxxx	Logmode name.
ec36_bind	R/O	4	xxx ... xxx	Bind image (access by using MASK*xx* TPSL condition).
ec36_user	R/O	5	xxxxxxxx	Userid.
ec36_prof	R/O	6	xxxxxxxx	Profile name.

Variable	M	Parm	Format	Description
ec36_taskida	R/O	6	SAAnnnnn	Session task, where nnnnn is the unique session number
ec36_taskidb	R/O	6	xxxxxxxx	Session taskid, either userid or terminal name.
ec36_applid	R/O	6	xxxxxxxxx	Application name.
ec36_acb	R/O	6	xxxxxxxx	ACB name.
ec36_slogm	R/O	6	xxxxxxxx	Logmode name

Slave session termination exit point E39

Variable	M	Parm	Format	Description
ec39_ttype	R/O	4	xxxx	Terminal task type.
ec39_term	R/O	4	xxxxxxxxx	Terminal name.
ec39_logm	R/O	4	xxxxxxxxx	Logmode name.
ec39_bind	R/O	4	xxx ... xxx	Bind image. (Access by using MASKxx TPSL condition).
ec39_user	R/O	5	xxxxxxxx	Userid.
ec39_pass	R/O	5	xxxxxxxx	Password.
ec39_prof	R/O	5	xxxxxxxx	Profile name.
en39_auth	R/O	5	xx	Authority level.
ec39_ taskida	R/O	6	SAAnnnnn	Session task, where nnnnn is the unique session number.
ec39_taskidb	R/O	6	xxxxxxxx	Session taskid, either userid or terminal name.
ec39_applid	R/O	6	xxxxxxxx	Applid.
ec39_acb	R/O	6	xxxxxxxx	ACB name.
ec39_slogm	R/O	6	xxxxxxxx	Logmode name.

Session switch exit point E79

Variable	M	Parm	Format	Description
ec79_ttype	R/O	4	xxxx	Terminal task type.
ec79_term	R/O	4	xxxxxxxxx	Terminal name.
ec79_logm	R/O	4	xxxxxxxxx	Logmode name.
ec79_bind	R/O	4	xxx ... xxx	Bind image. (Access by using MASKxx TPSL condition).

Variable	M	Parm	Format	Description
ec79_user	R/O	5	xxxxxxxx	Userid.
ec79_pass	R/O	5	xxxxxxxx	Password.
ec79_prof	R/O	5	xxxxxxxx	Profile name.
en79_auth	R/O	5	xx	Authority level.
ec79_ taskida	R/O	6	xxxnnnnn	Session task, where xxx is the function id, and nnnnn is the unique task number.
ec79_taskidb	R/O	6	xxxxxxxx	Session taskid, either userid or terminal name.
ec79_applid	R/O	6	xxxxxxxx	Applid.
ec79_acb	R/O	6	xxxxxxxx	ACB name.
ec79_appl	R/O	6	xxxxxxxx	Session Manager APPL name.
ec79_oapplid	R/O	6	xxxxxxxx	Old application name for a call reason of PASS, otherwise the current/latest APPL name if a CLSDST PASS has been issued by the application.
ec79_napplid	R/O	6	xxxxxxxx	New application name. (Blank if call reason is not PASS).
ec79_dstream	R/W	7	xxx ... xxx	Terminal data stream. Maximum length is 256 bytes.

User definable variables

Any number of user definable variables may be used. A user definable variable is defined when it first has data assigned to it. They are distinguished from Session Manager supplied variables by the variable name prefix, the first character of which indicates the scope of the variable as follows:

Character	Scope	Description
G	global	The variable takes the same value throughout the system and is available until system shutdown.
U	user	The variable has a different value for each user and is available while a user is active.
L	local	The variable is temporary, it is only applicable within the current PANEL or SCRIPT definition and any called sub-definitions or scripts.

The second character of the prefix defines the data type, that is, C for 'character', or N for 'numeric'. The rest of the name, which may be up to 14 more characters, may be any alphanumeric character or an underscore character (_).

Numeric user variables always have a length of nine and are held as signed binary. The maximum value that can be held in a numeric variable is 999999999.

Numeric variables that have a zero value are displayed as a zero (0) unless displayed on a panel with the BLANK attribute, in which case a blank is displayed. Numeric variables that have a negative value are displayed with a minus sign, for example, -90.

Character variables have a maximum length of 32K. To prevent storage being used unnecessarily, a length modifier should be used on the definition of character variables. Note also that not all functions support variables longer than 256.

Data conversion between character and numeric is allowed. the contents must be numeric.

Examples of user definable variable names are:

```
gcnote
UN_COUNT_1
lc_last_panel
```

One useful and general application of GC-prefix variables is in their use as general system 'notice-boards'. The content of any GC-prefix variable may be changed by the FLASH command and this will be reflected on any screen where the variable is displayed.

The same effect can be achieved by making the screen field definitions modifiable. A field modified on one screen will show its new value on other screens, with the same field defined, when the screen is rewritten.

A special variable, gcflash, is used by Session Manager. This is the system flash area, and is updated by default by the FLASH command. It is useful for general information messages, and may also be updated by terminal input, or by the TPSL statement, LET.

Subscripting user definable variables

User definable variables may be subscripted to form one-dimensional arrays. The special variables sub1 to sub9 may be used for subscripting, or any element may be referenced by an absolute value. The user definable variable name is restricted to 11 characters so that the subscript name can be accommodated in the maximum total length of 16 characters. For example:

```
untotalmsgs.sub3
```

A subscript must not exceed 9999. Any processing which attempts to access a variable using an invalid subscript is returned a null field. As with unsubscripted variables, subscripted variables do not occupy storage until they are assigned a value.

A user definable variable which is unsubscripted, but which has the same name as a subscripted variable, is treated as a completely separate variable. There is no association between the variable and the elements of the array.

CHAPTER 6

Tracing PANEL and SCRIPT processing

This chapter describes the use of the TTPSL command which provides the facility to trace Session Manager panels and scripts. Trace records can either be printed as each statement is processed, or stored in a wrap-around internal table which can be browsed.

Before considering the use of TTPSL, the AUDITMSG script parameter and FORMATMSG panel function should be explored. These facilities enable messages to be written to the audit file or to the screen during script and panel processing, and often provide more straightforward assistance than that offered by TTPSL.

For more information on AUDITMSG, see 'Session script parameters' on page 97.
For more information on FORMATMSG, see page 173.

Overview

When the trace records are displayed in the internal trace table, the latest record is at the bottom of the screen. Earlier records can be displayed by paging backwards. However, if a large number of statements have been traced, then the initial trace records may be overwritten when the table wraps around.

A sample of the first page of a trace display is shown below.

```
 Session Manager        Panel/Script Tracing Display         xx/yy/200z 14:33:08
 LU A1234                                                                PL

 17.39.48.26 SCRIPT TXSSOLOG STARTING
 17.39.48.26 CALL RSETSPAN
 17.39.48.27 SCRIPT RSETSPAN STARTING
 17.39.48.27 LET SUB1
 17.39.48.27 DO UNTIL
 17.39.48.27 LET UC_SCRIPT  .   .
 17.39.48.27 LET SUB1
 17.39.48.27 END (FOR DO)
 17.39.48.27 RESULT = FALSE
 17.39.48.27 LET UC_SCRIPT  .   .
 17.39.48.27 LET SUB1
 17.39.48.27 END (FOR DO)
 17.39.48.27 RESULT = FALSE
 17.39.48.27 LET UC_SCRIPT  .   .
 17.39.48.27 LET SUB1
 17.39.48.27 END (FOR DO)
 17.39.48.27 RESULT = FALSE
 17.39.48.27 LET UC_SCRIPT  .   .
 17.39.48.27 LET SUB1

===>

 PF1:Help  PF3:Quit  PF4:Return           PF7:Bwd  PF8:Fwd
```

The TTPSL COMMAND

Syntax

```
TTpsl [[Yes | No | ON | OFF | Display]
      Task task-name | [Selection seln-id
        [User userid | LU luname]]
      [PAnel | SCript]
      [PRint | NOPrint [DUmp | NODump]]
      [Internal | NOInternal]
      [All | TRaceverb]
      [SIze nn]]
```

Parameters

`[Yes | No | ON | OFF | Display]`

YES and ON cause tracing to start.

When the NO or OFF parameters are specified, tracing is stopped and the internal trace table is cleared of any trace records.

The DISPLAY option causes the internal trace table to be browsed. This is the default when tracing is already started. Any other parameters previously specified will be used. If a Selection parameter is also specified, but tracing has not been started, then tracing is started.

This option can only be entered at a terminal, it cannot be entered at the console. Details contained in the internal trace table are shown on the DATA panel, as shown in the previous screen example.

`Task task-name | [Selection seln-id [User userid | LU luname]]`

Determines what is to be traced.

TASK *task-name* identifies a running task when the script or panel is traced. The task-name can be found by issuing the QTASK command.

SELECTION specifies the user session to be traced. The *seln-id* can be specified as either a session number or a session selection sequence.

USER causes the user task for *userid* to be traced.

LU is the *luname* of the terminal to be traced.

USER and LU options can only be specified on the SELECTION parameter; they identify which user or LU is to be traced. The default is USER with the userid of the user who invoked the command. When TTPSL is invoked from the console, and Selection is specified, either USER or LU must also be specified.

`[PAnel | SCript]`

If PANEL is specified then the panel logic is traced. Alternatively, if SCRIPT is specified, the script logic is traced. SCRIPT is the default.

`[PRint | NOPrint [DUmp | NODUmp]]`

These options specify the destination of the trace records.

PRINT causes the trace record to be sent to the Session Manager Audit file, or the console, or both, depending on the routing that has been specified for the TTPSL messages.

To avoid excessive amounts of output being sent to the console log, the following MESSAGE statement should be present in the configuration file:

```
MESSAGE 476 LOG NO AUDIT YES
```

NOPRINT indicates that the trace records are not to be sent to the Session Manager Audit file. NOPRINT is the default.

DUMP causes the trace records, stored in the internal wrap-around table, to be sent to the Session Manager Audit file when either the trace ends or when the TTPSL OFF command is issued.

NODUMP indicates that the trace records are not to be sent to the Session Manager Audit file when tracing ends. This is the default.

`[Internal | NOInternal]`

Indicates whether the internal trace table should be used.

INTERNAL specifies that the internal trace table should be used to store the trace records. The internal trace table is a wrap-around table and its size is determined by the SIZE parameter below. The trace table can be browsed by using the TTPSL DISPLAY command.

NOINTERNAL specifies that no internal trace table is to be used. This is the default.

`[All | TRaceverb]`

Indicates what level of tracing is to be performed.

ALL causes all SCRIPT and PANEL statements to be traced. This is the default.

TRACEVERB causes only the SCRIPT and PANEL statements between the TRACEON and TRACEOFF parameters to be traced.

`SIze` *nn*

Specifies the size of the internal trace table, and is only valid if the trace table is being used. The size is specified in kilobytes. Any value between 1 and 31 can be used; the default is 12K.

CHAPTER 7

Script examples

The following script examples are provided in the member ISZSCRIP on the product tape.

The following examples are provided:

VM logon script example

The following is s simple startscript for entering the Session Manager userid and password as machine id and password to VM using VSCS.

```
SCRIPT vmlogon
   DOMAX 5 APPLTIME 5
   USERMSG TEXT 'Script vmlogon now running. Please wait'
   waitappl
   let found = false
   do until found                      /* Locate logon screen */
      input key clear
      waitappl
      outscan 'CP READ'
   end
   input text 'LOGON &t_user& &t_pass&' key enter
   waitappl
   input key clear
```

TSO logon script example

The following example is a simple panel definition and a startscript, which takes the TSO userid, password, proc name and account as input from the TSO panel.

```
PANEL tso
Define mode alt width 80

Header lines 3
Let lc_pandesc = 'TSO startscript'
Call m4head

CONTENT
LET sub1 = s_s
TEXTSTART
<   Description field ===>?s_desc <
<
<   Please overtype the fields you require. Press Enter to
<   continue.
<
<             TSO userid                    ===>{uc_tsousr.sub1 <
<             TSO password                  ===>%uc_tsopass.sub1<
<             TSO procedure name   (opt)    ===>%uc_tsoproc.sub1<
<             TSO account code     (opt)    ===>%uc_tsoacct.sub1<
TEXTEND
Trailer lines 5
Col 1 Field uc_msg    refattr '$'
Col 1 Field T_message refattr '$'
Nline 2
Col 1 Field T_sos_msg refattr '$'
If uc_msg = ' '
Textstart
<          PF3:Quit  PF4:Return
Textend
Else
    Nline
End
SCRIPT tsostrt APPLTIME 15
LET sub2 = s_s
Let uc_msg = ' '
USERMSG PANEL tso
WAITTERM
IF t_aid = PF3 or PF4
    ENDSESS /* END session WHEN PF3 or PF4 elected */
RETURN
END
If uc_tsousr.sub2 = SPACES
    LET uc_msg = 'USERID not supplied. Press any key to quit'
    USERMSG PANEL tso
    waitterm
    Endsess
    Return
End
```

```
LET uc_msg = 'Scanning for USERID prompt.'
USERMSG PANEL tso
DO UNTIL found
    WAITAPPL DISPLAY NO
    IF noresp THEN
        LET uc_msg = 'USERID prompt not found. Press any key to
                      quit'
        USERMSG PANEL tso
        WAITTERM
        ENDSESS
        RETURN
    END
    OUTSCAN 'ENTER USERID'
END
LET uc_msg = 'USERID prompt found. '
USERMSG PANEL tso
INPUT TEXT uc_tsousr.sub2 KEY ENTER
DO UNTIL found
    WAITAPPL DISPLAY no
    IF noresp THEN
        LET uc_msg = 'TSO/E LOGON SCREEN not found. Press any key
                  to quit'
        USERMSG PANEL tso
        waitterm
        ENDSESS
        RETURN
    END
    outscan 'LOGON'
END
LET uc_msg = 'TSO/E LOGON screen found.'
USERMSG PANEL tso
IF uc_tsopass.sub2 ¬= ' '
    INPUT KEY no sba 08 20 text uc_tsopass.sub2
End
IF uc_tsoproc.sub2 ¬= ' '
    INPUT KEY no sba 10 20 text uc_tsoproc.sub2
End
IF uc_tsoacct.sub2 ¬= ' '
    INPUT KEY no sba 12 20 text uc_tsoacct.sub2
End
Input Key enter
waitappl
```

CICS logon script example

The following example is a simple panel definition and a startscript, which takes the username and password for CESN and the initial transaction from the CICS panel.

```
PANEL cics
Define mode alt width 80
Header lines 3
Let lc_pandesc = 'CICS startscript'
Call m4head
CONTENT
LET sub1 = s_s
TEXTSTART
<   Description field ===>?s_desc                                <
<
<   Please overtype the fields you require. Press Enter to
<   continue.
<
<             Name parameter for CESN         ===>{uc_uname.sub1<
<             Password for CESN               ===>%uc_pass.sub1 <
<             Initial transaction id          ===>%uc_tr.sub1   <
TEXTEND
Trailer lines 5
Col 1 Field uc_msg    refattr '$'
Col 1 Field T_message refattr '$'
Nline 2
Col 1 Field T_sos_msg refattr '$'
If uc_msg = ' '
Textstart
<          PF3:Quit  PF4:Return
Textend
Else
    Nline
End
SCRIPT cicsstrt DOMAX 24 APPLTIME 30
LET sub2 = s_s       /* set session subscript no. for uservars */
Let uc_msg = ' '
USERMSG PANEL CICS
waitterm
If t_aid = PF3 or PF4 Endsess Return End
WAITAPPL                     /* wait for cics output */
INPUT KEY CLEAR              /* CLEAR gmmsg */
WAITAPPL
INPUT KEY CLEAR              /* just make sure */
WAITAPPL
If uc_uname.sub2 = SPACES Return End
Let uc_msg = 'Performing sign-on for &uc_uname.sub2'
USERMSG PANEL CICS
INPUT text 'CESN'            /* run CESN */
DO UNTIL found               /* found set by sucessful outscan */
    WAITAPPL
    IF noresp THEN
```

```
        Let uc_msg = 'CESN logon screen not received. Press any
                      key to quit'
        USERMSG PANEL CICS
        Waitterm
        Endsess
        Return
    END
    outscan 'CICS Signon'         /* look for this in outbound
                                     datastream */
    END
    Let uc_msg = 'CESN logon screen received '
    USERMSG PANEL CICS
INPUT sba 6 24 text uc_uname.sub2 /* change if necessary */
      sba 7 24 text uc_pass.sub2
      sba 8 24                    /* blank language ( default) */
KEY enter
LET found = false                 /* reset found */
DO UNTIL found
    WAITAPPL
    IF noresp THEN
        Let uc_msg = 'Sign-on complete msg not received.'-
                     'Press any key to quit'
        USERMSG PANEL CICS
        waitterm
        endsess
        return
    END
outscan 'SIGNON IS COMPLETE'
END
If uc_tr.sub2 = SPACES Return End
Let uc_msg = 'Entering transaction &uc_tr.sub2'
USERMSG PANEL CICS
INPUT KEY CLEAR
WAITAPPL
INPUT sba 1 1 text uc_tr.sub2        /* enter transaction code */
WAITAPPL DISPLAY
```

VTAM status inquiry script example

The following is a command script to perform a VTAM inquiry. The processing is the same as that provided by the SYSTEM parameter INQINTERVAL except that it is one off and no time interval is set to repeat the inquiry.

```
script inquire
let lc_dummy = inquire( )
```

Users of a classic system will need to add a COMMAND definition to the configuration of the form:

```
command inquire cmdscript y sname inquire
```

OLA users will be able to define the same through Online Administration.

The user can then perform a VTAM inquiry of all applications by typing INQUIRE on the Menu command line.

HOME/TAB/ERASE functions

The HOME/TAB/ERASE functions allow a user to write scripts without the need to specify SBA (Set Buffer Address) positions on the `INPUT` statements. The functions work by formatting `WAITAPPL` screens received from the application and retaining information on the location of unprotected input fields. Note that ERASE/TAB/HOME cannot be used until a `WAITAPPL` has been received from the application.

The following is a STARTSCRIPT that logs on to TSO, goes into EDIT and enters the name of a library and member (whilst ensuring that all other fields are clear):

```
script test9
 appltime 10
 do until found
    waitappl display
    outscan 'ENTER'
 end
 input text t_user key enter
 do until found
    waitappl display
    outscan 'TSO/E LOGON'
 end
 erase eof
 input text t_pass key enter
 do until found
    waitappl display
    outscan '***'
 end
 do until found
    waitappl display
    outscan '***'
 end
 input key enter
 do until found
    waitappl display
    outscan 'Licensed'
 end
  waitappl display
  input text '2' key enter
  waitappl display
  do for 6
     tab b
  end
  erase eof
  input text 'prod' key no
  tab
  erase eof
```

```
  input text 'test' key no
  tab
  do for 3
     erase eof
     tab
  end
  input text 'cntl' key no
  tab
  input text 'dummy' key enter
return
```

Window scripts

Controlling window script logic

The top-level script is invoked when a user enters the WINDOWS command. The Windows script then runs continuously until the user quits from Windows. Most of the time is spent in the WAITEVENT state, waiting until one of four events occurs:

- a window escape sequence is entered;
- a message arrives from a MSG or BROADCAST command;
- a windowed session terminates;
- a windowed session switches to a DLOG, QUERY, QTASK, SPY or VIEW.

When one of these is detected, the script continues processing. Other scripts are called from the top-level script as shown in the diagram 'Overview' on page 36.

A synopsis of the logic used by the main supplied scripts follows. The logic used by the sub-scripts is not expanded as they are generally smaller and the logic flows in a similar way to normal scripts, i.e., from top to bottom. A full explanation of each window verb can be found in 'Window verbs' on page 119.

Initialization

Action	Script reference
Define the window pfkeys	T_W_ESC.nnn
Define the background character	WBACKGROUND
Define the inactive window border	WBORDER DIM
Define the active window border	WBORDER BRIGHT
Initialize script variables	–
Create the top line of the screen	WOPEN £ACTBAR
Start of main loop	DO FOREVER
If only 1 window on screen	T_WINDOWS = 1
Open application list on row 2	WOPEN £OPENLST
Show detail lines on row 2 and on last row	WOPEN £WHOAMI WOPEN £PFSHOW
Else	
Ensure active windows border is correct	WALTER BRIGHT
End	
If the active window is in normal mode	W_ZOOM = N
Set up its border contents	
Ensure the active window is completely visible	WPOP
Ensure the detail lines are visible	WPOP £WHOAMI WPOP £PFSHOW
End	
Wait for something to happen	WAITEVENT
When a window escape was entered	T_W_EVENT =E
If no input, it's from a WCA - top line	T_W_INPUT = ''

Action	Script reference
Else test the pfkeys and act as required	T_W_ESC.nnn
End	
When a Session Manager command was entered	T_W_EVENT = I
Update variables to change window name	DLOG,VIEW,SPY
and count the active commands in windows	QUERY,QTASK
When a broadcast or message has arrived	T_W_EVENT = M
Update variables with contents of message	T_W_MSG
to display later	T_W_MSUSER T_W_MSTIME
If we are in zoomed mode	W_ZOOM = Z
unzoom and remember whence we came	WNORM
End	
Create broadcast window or update it	WOPEN/WALTER £BMSGS
When a session has ended	T_W_EVENT = S
If the zoomed session ended	
restore the pfkeys	T_W_ESC.nnn
End	
When its an application session	W_NAME <> M'£*'
Scroll to the origin of the window	WSCROLL
Display session ended message	WALTER £END.nn
When its the broadcast window	£BMSGS
rezoom if necessary	WZOOM
When its the update pfkeys window	£PFNEW
Set up possible new pfkeys	T_W_ESC.nnn
When its the scroll options window	£SCRLOPT
Scroll by the required amount	WSCROLL
When its the application list window	£OPEN..
If an application has been selected to	WOPEN in
put into the window, do it	that posn
Else	
Update variables noting the change	
End	
End	
Ensure the top line is complete	WPOP £ACTBAR
Return to start of main loop	ITERATE
End	

Creating facilities using panels and scripts

Terminal lock facility example

This facility enables a user to temporarily leave their terminal without the need to logoff. As other users cannot use the terminal while it is locked, this facility should only be used during a short absence.

When this facility is invoked by entering the command LOCKIT on the MENU or the transid LOCKIT in as session, a panel which is non-escapable is displayed on the terminal, showing the time the lock was initiated. The only way to gain access to the sessions is to key the user's password.

The following definition should be added to the Profiles which require access to this facility:

```
Session nn
lockit
applid lockit
transid locked
```

If the session where the user entered LOCKIT has the transid defined, the user is returned to that session when the terminal is unlocked.

The source of the terminal lock panel and the Lockitscript is in ISZCSAMP and is also reproduced below.

Terminal Lock Facility: If you wish to return to the session whence you came, it must have a transid. A session definition must be included in your profile/user configuration; e.g., Session 99 lockit applid lockit transid locked.

If your users are using CMDACTIONKEY then use the definition below with ACTKEY Y on. If your users are using COMMANDPRFXVAL then use the SAUTOSEQ definition below with COMMANDPRFX Y on:

```
command lockit cmdscript y sname cmdlock
*command lockit cmdscript y sname cmdlock actkey y   /*
CMDACTIONKEY required

SYSTEM SAUTOSEQ lockit n lockit
*SYSTEM SAUTOSEQ lockit n lockit commandprfx y

PANEL lock
DEFINE MODE alt   WIDTH 80  DOMAX=50

ATTR '#' unprot,nodisp,in,ic
HEADER LINES 4
Let lc_pandesc = 'Terminal Lock Facility'
Call iszhead
```

```
CONTENT
IF un_flag = 0 THEN
   LET lc_time = t_time
   LET un_flag = 1
END
TEXTSTART
<              This terminal has been locked since ===>&lc_time
<
<              Please enter password to unlock ===>#uc_lock <
TEXTEND

TRAILER LINES 4
Col 1 Field T_message refattr '½'
Nline 2
Col 1 Field T_sos_msg refattr '½'
nline

Process
Let t_command = ' '
Let t_aid = ENTER
 IF uc_lock = t_pass
    LET un_flag = 0
 END

script cmdlock
if t_tskid like 'SAA*'
   Let un_sess = s_s
else
   Let un_sess = 0
end
if t_actcmd ne ''
   let lcsetpfkey =  t_actcmd
   call setpfkey
   iszcmd '&t_actprf&locked'
else
   iszcmd 'locked'
end

script lockit
Let un_sess = s_s
if t_actcmd ne ''
   let lcsetpfkey =  t_actcmd
   call setpfkey
   iszcmd '&t_actprf&locked'
else
   iszcmd 'locked'
end
```

```
script setpfkey
case
   when lcsetpfkey = 'pf1' or 'pf01' let t_aid = PF1
   when lcsetpfkey = 'pf2' or 'pf02' let t_aid = PF2
   when lcsetpfkey = 'pf3' or 'pf03' let t_aid = PF3
   when lcsetpfkey = 'pf4' or 'pf04' let t_aid = PF4
   when lcsetpfkey = 'pf5' or 'pf05' let t_aid = PF5
   when lcsetpfkey = 'pf6' or 'pf06' let t_aid = PF6
   when lcsetpfkey = 'pf7' or 'pf07' let t_aid = PF7
   when lcsetpfkey = 'pf8' or 'pf08' let t_aid = PF8
   when lcsetpfkey = 'pf9' or 'pf09' let t_aid = PF9
   when lcsetpfkey = 'pf10' let t_aid = PF10
   when lcsetpfkey = 'pf11' let t_aid = PF11
   when lcsetpfkey = 'pf12' let t_aid = PF12
   when lcsetpfkey = 'pf13' let t_aid = PF13
   when lcsetpfkey = 'pf14' let t_aid = PF14
   when lcsetpfkey = 'pf15' let t_aid = PF15
   when lcsetpfkey = 'pf16' let t_aid = PF16
   when lcsetpfkey = 'pf17' let t_aid = PF17
   when lcsetpfkey = 'pf18' let t_aid = PF18
   when lcsetpfkey = 'pf19' let t_aid = PF19
   when lcsetpfkey = 'pf20' let t_aid = PF20
   when lcsetpfkey = 'pf21' let t_aid = PF21
   when lcsetpfkey = 'pf22' let t_aid = PF22
   when lcsetpfkey = 'pf23' let t_aid = PF23
   when lcsetpfkey = 'pf24' let t_aid = PF24
   else auditmsg "SETPFKEY - bad value '&lcsetpfkey&'"
end

script locked
LET uc_lock = ' '
Auditmsg '&t_user& &t_termcls& &t_termid& Lock facility
initiated'
Let sub1 = un_sess
Let s_allowesc = 'N'
DO UNTIL uc_lock = T_PASS
   USERMSG PANEL lock
   WAITTERM
   If uc_lock <> t_pass
      Auditmsg '&t_user& &t_termcls& &t_termid& Invalid unlock
attempt'
   End
END
Let s_allowesc = 'Y'
if  sub1 ne 0
   if  t_actcmd ne ''
```

```
      let lcsetpfkey = t_actcmd
      call setpfkey
      iszcmd '&t_actprf&&s_n.sub1&'
   else
      iszcmd '&s_n.sub1&'
   end
   Let un_sess = 0
end
let t_message = "Terminal unlocked"
Endsess

appl lockit
     desc 'lock terminal'
     initscript locked
```

Notice board facility

The Notice facility provides a common facility where users can view text messages by entering the NOTICE command on the Menu.

Unlike the BCASTEDIT facility, the messages can be redisplayed on the terminal and are available until the next restart of Session Manager.

The Notice facility can be updated by issuing the NOTICEUP command. This displays 30 lines of text in the panel. Session Manager panel paging is then used to display the remaining lines that are not displayed on the terminal display area.

Notice board panel definitions

The first panel uses LC_ prefixed variables (local character) which are displayed as unprotected fields. This allows messages to be altered. The second panel displays the variables as GC_prefixed (global character) after they have been copied from the LC_ variables. The GC_ prefixed variables are displayed as protected fields.

The source of the Notice Board panel definition is in ISZCSAMP and is also reproduced below.

```
                              Notice Update Panel

PANEL noticeup
DEFINE MODE alt   WIDTH 80  DOMAX=50
HEADER LINES 3
Let lc_pandesc = 'Notice Board Update'
Call ISZhead
CONTENT
LET sub1 = 1
DO FOR 30
    FIELD lc_notice.sub1(79) refattr '%'
    LET sub1 = sub1 + 1
END

TRAILER LINES 4
call ISZtrail
Textstart
<PF1:Help PF3:Quit PF4:Ret PF7:Bwd PF8:Fwd PF10:Updat PF11:Retr
Textend
PROCESS
Select
When t_aid = PF1   Let t_command = 'Help'
```

```
When t_aid = PF3   Let t_command = ' ' Let t_panel = uc_oldpan
When t_aid = PF4   Let t_command = ' ' Let t_panel = uc_oldpan
When t_aid = pf7   Let t_command = 'Bwd'
When t_aid = pf8   Let t_command = 'Fwd'
When t_aid = pf10  Let sub1 = 1
               DO for 30
                  LET gc_notice.sub1 =CENTRE(lc_notice.sub1,79
                  LET sub1 = sub1 + 1
               END
               LET T_PANEL = 'notice'
When t_aid = pf11   Let sub1 = 1
               DO for 30
                  LET lc_notice.sub1 = gc_notice.sub1'a
                  LET sub1 = sub1 + 1
               END
END
Let t_aid = ENTER
```

Notice Display Panel

```
PANEL notice
DEFINE MODE alt   WIDTH 80  DOMAX=50

HEADER LINES 3
Let lc_pandesc = 'Notice Board Display'
Call ISZhead

CONTENT
LET sub1 = 1
DO for 30
    FIELD gc_notice.sub1(79) refattr '?'
    LET sub1 = sub1 + 1
END
TRAILER LINES 4
call ISZtrail
Textstart
<PF1:Help  PF3:Quit  PF4:Return  PF7:Bwd  PF8:Fwd
Textend

PROCESS
Select
When t_aid = PF1     Let t_command = 'Help'
When t_aid = PF3     Let t_command = ' ' Let t_panel = uc_oldpan
When t_aid = PF4     Let t_command = ' ' Let t_panel = uc_oldpan
When t_aid = pf7     Let t_command = 'Bwd'
When t_aid = pf8     Let t_command = 'Fwd'
Else                 Let t_command = ' '
End
Let t_aid = ENTER
```

BCASTED facility

The BCASTED command is a facility to broadcast up to three pages of text to either one or all users of the Session Manager system.

This command expands the standard Session Manager BROADCAST command by allowing messages of more than 256 characters to be sent.

The Process section processes the PF keys and copies the variables between global GC and local LC variables. It also issues a Session Manager BROADCAST message with the text message of 'broadcast'. This is used in the panel content section.

BCASTED command - PANEL definition

```
                    Broadcast Full Screen Edit Panel

PANEL bcasted
DEFINE MODE alt   WIDTH 80  DOMAX=50

HEADER LINES 3
let lc_pandesc = 'Broadcast Screen Edit'
callISZ4head
CONTENT
LET SUB1 = 1
DO FOR 30
    FIELD lc_bcast.SUB1(79) refattr '%'
    LET SUB1 = SUB1 + 1
END
col 1 field 'END of INPUT area' refattr '<'
TRAILER LINES 6
nline
IF lc_bcast_dest = ' '
    LET lc_bcast_dest = 'user &t_user'
END
COL 1 FIELD 'PF12: Broadcast to ==>' refattr '<'
      field lc_bcast_dest(20)  refattr '%'
call ISZtrail
Textstart
<PF1:Help PF3:Quit PF4:Return PF6:Retr last Bro PF7:Bwd PF8:Fwd
PF10:Bro All
Textend
PROCESS
Select
When t_aid = PF1     Let t_command = 'Help'
When t_aid = PF3     Let t_command = ' ' Let t_panel = uc_oldpan
When t_aid = PF4     Let t_command = ' ' Let t_panel = uc_oldpan
When t_aid = PF6     Let t_command = ' '
        LET sub1 = 1              /* read global variables */
        DO FOR 30                 /* into local variables */
        LET lc_bcast.sub1 = gc_bcast.sub1
        LET sub1 = sub1 + 1
END
When t_aid = pf7     Let t_command = 'Bwd'
When t_aid = pf8     Let t_command = 'Fwd'
When t_aid = pf10    LET sub1 = 1
        DO FOR 30
           If lc_bcast.sub1 <> SPACES
              LET gc_bcast.sub1 = CENTRE(lc_bcast.sub1,79)
              LET sub1 = sub1 + 1
           Else
              leave
           End
        END
        Let sub1 = sub1 - 1
        LET t_command = "br 'broadcast &sub1&' all"
When t_aid = pf12 LET sub1 = 1     /* set new global variables */
        DO FOR 30
```

```
            If lc_bcast.sub1 <> SPACES
               LET gc_bcast.sub1 = CENTRE(lc_bcast.sub1,79)
               LET sub1 = sub1 + 1
            Else
               leave
            End
         END
         Let sub1 = sub1 - 1
         LET t_command = "br 'broadcast &sub1&' &lc_bcast_dest"
End
LET t_aid = ENTER
```

BROADCAST command - panel definition

When this panel is displayed, the T_DATA variable is checked to see if it contains the text message ‘broadcast’ which was issued in the Process section.

If the T_DATA variable does contain the text 'broadcast' then the GC_BCAST variables are displayed, otherwise the standard panel format is used. The number of these variables has also been setup in the broadcast text. The standard format uses the T_DATA variable which is not pageable. Paging is controlled by the panel and not by Session Manager.

```
Broadcast PANEL
This PANEL replaces the boroadcast PANEL ‘BCAST’ supplied in ISZCPANS
```

```
PANEL BCAST
DEFINE MODE alt   WIDTH 80  DOMAX=50

HEADER LINES 3
let lc_pandesc = 'Broadcasts & Messages'
call ISZhead
CONTENT
If sub5 = 0
   Select
      When t_model = '2' Let sub9 = 17
      When t_model = '3' Let sub9 = 25
      When t_model = '4' Let sub9 = 36
      When t_model = '5' Let sub9 = 20
      When t_model = 'x' Let sub9 = 41
   End
   Let sub1 = 1
   Let lc_bcast = word t_data.2 1
   If lc_bcast = 'broadcast'
      Let lc_bcast.sub1 = t_data.1
      Let sub1 = 2
      Let sub2 = 1
      Let sub8 = word t_data.2 2 /*how many lines from bcasted */
      Do for sub8
         Let lc_bcast.sub1 = gc_bcast.sub2
         Let sub1 = sub1 + 1
         Let sub2 = sub2 + 1
      End
   End
   If sub8 = 0 Let sub2 = 1
   Else        Let sub2 = 3
   End
   Do while t_data.sub2 <> SPACES
      Let lc_bcast.sub1 = t_data.sub2
      Let sub1 = sub1 + 1
      Let sub2 = sub2 + 1
   End
   Let sub1 = sub1 - 1       /* how many lines of total bcast */
   Let sub5 = 3
Else
   If t_data.sub2 <> SPACES  /* new broadcast arrived */
      Let sub1 = sub1 + 1
      Do while t_data.sub2 <> SPACES
         Let lc_bcast.sub1 = t_data.sub2
         Let sub1 = sub1 + 1
         Let sub2 = sub2 + 1
      End
      Let sub1 = sub1 - 1    /* how many lines of total  bcast */
   End
End
```

```
IF sub1 <= sub9              /* paging impossible */
   Let sub4 = 1
Else
   Select
      When sub5 = 3 Let sub4 = 1             /* first time   */
      When sub5 = 1 Let sub4 = sub4 + 1      /* page forward */
      When sub5 = 2 Let sub4 = sub3 - sub9   /* page backward */
                    Let sub4 = MAX(1,sub4)   /* minimum 1 */
      Else          Let sub4 = sub3          /* same as before */
   End
End
Let sub3 = sub4              /* remember the start */
Do for sub9
   field lc_bcast.sub4(79) refattr '|'
   Let sub4 = sub4 + 1
   If sub4 > sub1  Leave End                 /* last one */
End

TRAILER LINES 4
call ISZtrail
Textstart
<PF1:Help  PF3:Quit  PF4:Return PF7:Bwd  PF8:Fwd
Textend

PROCESS
Select
   When t_aid = PF1    Let t_command = 'Help'
   When t_aid = PF3    Let t_command = 'return'
   When t_aid = PF4    Let t_command = 'return'
   When t_aid = pf7    Let t_command = ' ' Let sub5 = 2
   When t_aid = pf8    Let t_command = ' ' Let sub5 = 1
   else            Let t_command = ' ' Let sub5 = 4 /*no change/*
END
Let t_aid = ENTER
```

BCASTED command - PROCESS section (edit)

This section processes the PF keys and copies the variables between global **GC** and user **UC**.

This section issues a Session Manager BROADCAST command with the text message of 'broadcast'. This is used in the panel content section.

```
PROCESS
   SELECT
      WHEN t_aid = pf8 THEN LET t_command ='F'
      WHEN t_aid = pf7 THEN LET t_command ='b'
   END
   IF t_aid = pf3 THEN
      LET t_command = ' '
      LET t_PANEL = uc_oldpan
   END
   LET sub2 = 31
   IF t_aid = pf6
      LET sub1 = 1
```

```
      DO WHILE sub1 ¬= sub2
         LET lc_bcast.sub1 = gc_bcast.sub1    /* read global
                                                 variables
         LET sub1 = sub1 + 1                  /* into local
                                                 variables.
      END
      LET t_command = ' '
   END
   IF t_aid = pf12
      LET sub1 = 1
      DO WHILE sub1 ¬= sub2
         LET  gc_bcast.sub1 = lc_bcast.sub1   /* set global
                                                 variables
         LET sub1 = sub1 + 1                  /* from new local
                                                 values
      END
      LET t_command = "br 'broadcast' &lc_bcast_dest"
   END
   IF t_aid = pf10
      LET sub1 = 1
      DO WHILE sub1 ¬= sub2
         LET gc_bcast.sub1 = lc_bcast.sub1
         LET sub1 = sub1 + 1
      END
      LET t_command = "br 'broadcast' all" /* Session Manager
                                              broadcast
   END
   LET T_aid = enter
```

Exit scripts

The following example is supplied on the distribution tape as ISZESAMP (contains both E05 and E06 samples). The script can be included at startup so that applications which were available and change to unavailable cause the first Broadcast message to be issued to users of profile prodpro1. When an application becomes available which was previously unavailable, the second message is issued to users of profile prodpro1.

The following changes should be made to the system configuration to run this script.

To change the system dynamically, you must:

- Create or update this member to define the script in Session Manager, e.g., UPDATE M ISZESAMP
- Issue the Update command so that exit point E06 runs the exit script EXIT06, i.e., UPDATE E06 S

Alternatively, if the ISZESAMP member is included at startup, you do not need to issue the UPDATE E06 because the OPTION statement coded will take effect.

Script example

```
OPTION E06 S
SCRIPT EXIT06
   If ec06_nstatus = 'N' and ec06_ostatus = 'Y'
      ISZCMD (Br 'Application &ec06_appl& is no longer available'
             profile prodpro1)
   Else
      If ec06_nstatus = 'Y' and ec06_ostatus = 'N'
         ISZCMD(Br 'Application &ec06_appl& is now available'
               profile prodpro1)
      End
   End
```

Index

Numerics

A

B

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Bibliography

IBM Session Manager library

The following publications contain information about IBM Session Manager.

Installation and Customization	GC34-7066
Technical Reference	SC34-7067
User and Administrator	SC34-7069
Panels, Scripts and Variables	SC34-7068
Messages and Codes	GC34-7070
Quick Reference	SC34-7071
Online and Batch Administration	SC34-7072
Program Directory	GI13-0520

Accessibility

Accessibility for people with disabilities

The following features make it easier for disabled people to use Session Manager:

- Operation by keyboard alone
- Optional font enlargement
- High-contrast display settings
- Can be used with screen readers
- Absence of audio prompts.

Changing font, color and display settings

Session Manager can be controlled using a 3270 emulator such as IBM Personal Communications or Hummingbird HostExplorer. Refer to the emulator documentation for guidance on adjusting font and color settings.

Using Session Manager with a screen reader

Screen readers can be used to provide accessible output for blind users. Session Manager has been tested with the following screen readers:

- Jaws version 4.5, using Hummingbird HostExplorer and the script file for Hummingbird HostExplorer
- WindowEyes 4.2, using Hummingbird HostExplorer and the set file for Hummingbird HostExplorer.

Contact the screen reader manufacturer for information about the availability of set and script files.

Documentation

Softcopy PDF documentation is shipped with Session Manager. The documentation supports optional font enlargement, high-contrast display settings, and may be operated by the keyboard alone. Alternative text is not provided for screen-reader users. Fully accessible softcopy documentation, with alternative text for diagrams, will be made available on request. Contact your IBM service representative for information.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. 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 IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply in the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM United Kingdom Laboratories, MP151, Hursley Park, Winchester, Hampshire, England, SO21 2JN. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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, IBM International Programming License Agreement, or any equivalent agreement between us.

Trademarks

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To ask questions, make comments about the functions of IBM products or systems, or to request additional publications, contact your IBM representative or your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom

- By fax:
  - From outside the U.K., after your international access code use 44–1962–816151
  - From within the U.K., use 01962–816151
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink™ : HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

SC34-7068-01