CA IDMS™

Callable Services Guide

Release 18.5.00, 2nd Edition

CA Technologies Product References

This document references the following CA Technologies products:

- CA ADSTM
- CA IDMSTM
- CA IDMSTM/DC (DC)
- CA IDMSTM/DC or CA IDMSTM UCF (DC/UCF)

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Documentation Changes

The following documentation updates were made for the 18.5.00, 2nd Edition release of this documentation:

- Mandatory Parameters (see page 137), Optional Parameter (see page 138), INREC Format (see page 140), OUTREC Format (see page 140)—Moved these sections to the correct locations in the Invoking System Tasks from Programs chapter.
- RHDCSNON parameters (see page 149)—Added information on PassTickets to the description of parameter 2.

Contents

Chapter 1: Introduction

This section contains the following topics:

Syntax Diagram Conventions

The syntax diagrams presented in this guide use the following notation conventions:

`UPPERCASE OR SPECIAL CHARACTERS`

Represents a required keyword, partial keyword, character, or symbol that must be entered completely as shown.

`lowercase`

Represents an optional keyword or partial keyword that, if used, must be entered completely as shown.

`italicized lowercase`

Represents a value that you supply.

`lowercase bold`

Represents a portion of the syntax shown in greater detail at the end of the syntax or elsewhere in the document.

Points to the default in a list of choices.

Indicates the beginning of a complete piece of syntax.

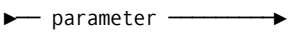
Indicates the end of a complete piece of syntax.

Indicates that the syntax continues on the next line.

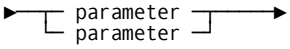
Indicates that the syntax continues on this line.

Indicates that the parameter continues on the next line.

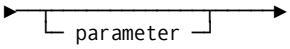
Indicates that a parameter continues on this line.



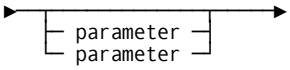
Indicates a required parameter.



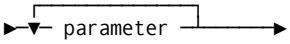
Indicates a choice of required parameters. You must select one.



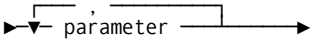
Indicates an optional parameter.



Indicates a choice of optional parameters. Select one or none.



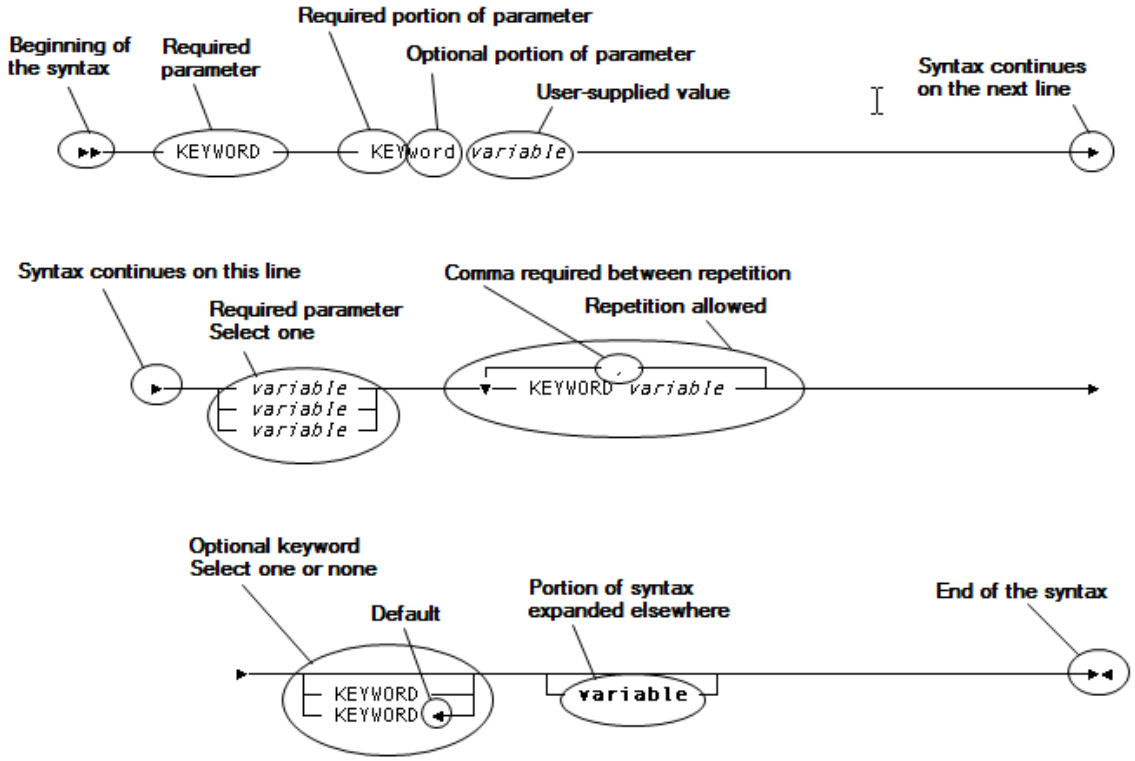
Indicates that you can repeat the parameter or specify more than one parameter.



Indicates that you must enter a comma between repetitions of the parameter.

Sample Syntax Diagram

The following sample explains how the notation conventions are used:



Chapter 2: IDMSCALC

The IDMSCALC utility is a subroutine which can be called from a user-written program to determine the target page of a record, based on a user-supplied CALC key.

It is typically used to optimize the loading of data by allowing you to sort input in target page sequence.

IDMSCALC is implemented as a called subroutine. The utility returns to a user-written program a target page number for storage of a CALC record, based on a page range and CALC key value supplied by the program. The user program, which can be written in any language supporting a call statement, must build a single five-field fullword-aligned argument as outlined in the following table, then call IDMSCALC, passing the argument. IDMSCALC must be link edited with the calling program.

This section contains the following topics:

Calling the IDMSCALC Routine

The following example shows how to call the IDMSCALC routine from a user-written program.

```
01  CALC-PARMS.
    05  CALC-PAGE-TARGET     PIC S9(9) COMP.
    05  CALC-PAGE-RANGE-HIGH PIC S9(9) COMP.
    05  CALC-PAGE-RANGE-LOW  PIC S9(9) COMP.
    05  CALC-KEY-LENGTH      PIC S9(4) COMP.
    05  CALC-KEY             PIC X(16).

    .
    .
    .
    .

    MOVE 75001   TO CALC-PAGE-RANGE-LOW.
    MOVE 75101   TO CALC-PAGE-RANGE-HIGH.
    MOVE 16      TO CALC-KEY-LENGTH.
    MOVE 'SMITH' TO CALC-KEY.
    CALL 'IDMSCALC' USING CALC-PARMS.
    DISPLAY 'TARGET PAGE IS ' CALC-PAGE-TARGET.
```

The IDMSCALC Argument

The following table outlines the five-field argument that a calling program must pass to IDMSCALC.

Field	Usage	Size (bytes)	COBOL Picture	Field Description
1 (Output)	Binary	4	PIC S9(9) COMP	Specifies the target page number for storage of the record.
2 (Input)	Binary	4	PIC S9(9) COMP	Specifies the number of the highest page on which the record can be stored.
3 (Input)	Binary	4	PIC S9(9) COMP	Specifies the number of the lowest page on which the record can be stored.
4 (Input)	Binary	2	PIC S9(4) COMP	Specifies the length, in bytes, of the CALC key value.
5 (Input)	Character	1-256	PIC X(nnn)	Specifies the value of the CALC key.

Note: The information in fields 2 and 3 of IDMSCALC must match the database definition for the record type as specified in the schema.

Input

Input to the IDMSCALC utility consists of the IDMSCALC argument with fields 2-5 initialized by the calling program.

Output

The IDMSCALC utility returns a target page number for storage of a CALC record.

More Information:

For more information about how the CALC location mode works, see the *Database Administration Guide*.

Chapter 3: IDMSIN01

This section contains the following topics:

Overview

The IDMS module contains an IDMSIN01 entry point which provides miscellaneous CA IDMS functions to user programs. The parameters passed depend on what service is being called.

The following service functions are available by calling IDMSIN01:

- Activate/deactivate the DML or SQL trace.
- Establish/retrieve user profile information.
- Retrieve SQL error messages into a user buffer.
- Translate an internal 8-byte DATETIME stamp to a displayable format.
- Return current DATE and TIME in a displayable format.
- Translate an external 26-character DATE to an 8-byte DATETIME stamp.
- Translate an internal 8-byte TIME stamp to an 8-character display format.
- Translate an external 8-character TIME to an 8-byte TIME stamp.
- Translate an internal 8-byte DATE stamp to a 10-character display format.
- Translate an external 10-character DATE to an 8-byte DATE stamp.
- Retrieve the current USERID that is signed on.
- Establish the SYSCTL DDNAME to use for batch/CV processing.
- Turn transaction sharing on or off for the current task.
- Extract or set a private RRS context (CV only).
- Convert strings to and from EBCDIC and ASCII.
- Format dbkey as character string 'page number:line index'.

- IDMSIN01 enables you to programmatically override many SYSIDMS parameters, which are as follows:
  - Activating or de-activating a DML or SQL trace.
  - DBNAME.
- Return a block of runtime environment information (described in the following COBOL layout format)

```
01 EVBLOCK.
  02  EV$SIZE  PIC S9(4) COMP VALUE +31. -- length of amount of data
                                            to be returned
  02  EV$MODE  PIC X.                    -- runtime mode
      88  LOCAL-MODE        VALUE 'L'.   -- batch local
      88  BATCH-TO-CV-MODE  VALUE 'B'.   -- batch to CV
      88  ONLINE-DC-MODE    VALUE 'D'.   -- DC online
      88  CICS-MODE         VALUE 'C'.   -- CICS
  02  EV$TAPE# PIC X(6).                 -- CA IDMS tape volser
  02  EV$REL#  PIC X(6).                 -- CA IDMS release number
  02  EV$SPACK PIC X(2).                 -- CA IDMS service pack number
  02  EV$DMCL  PIC X(8).              -- DMCL name (blank for batch to CV mode)
  02  EV$NODE  PIC X(8).               -- System node name (CICS and DC online,
                                            blank for batch local and batch to
CV jobs)
```

Calling IDMSIN01 from an Assembler Program

Assembler programs can call for IDMSIN01 services by using the IDMSIN01 macro.

An Assembler program can gain access to the IDMSIN01 functions by using the IDMSIN01 macro.

Notes:

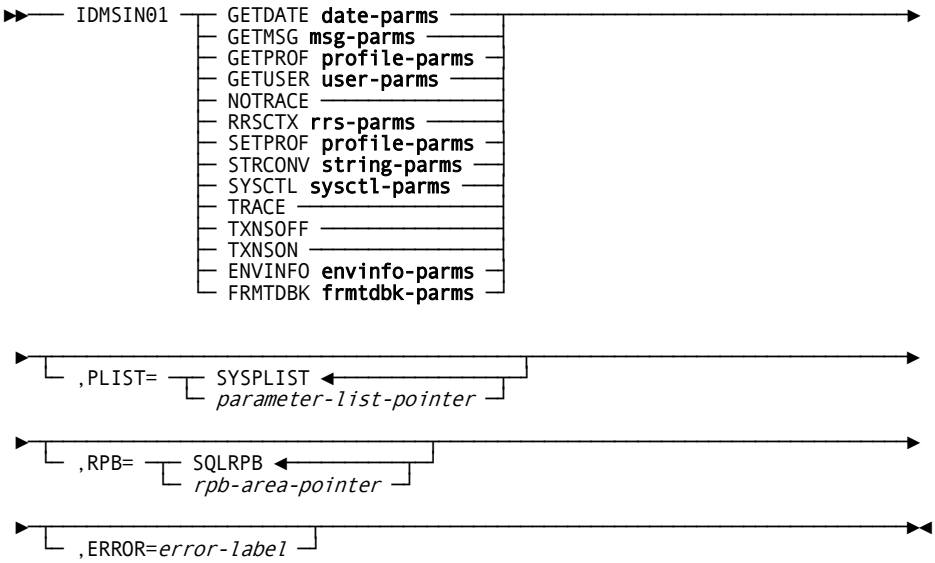
- Ensure that R13 points to a standard register save area when calling IDMSIN01 from an Assembler program.
- A return code is returned in R15. You should check errors with the ERROR= parameter of the IDMSIN01 macro.

- The syntax does not show Assembler column conventions (label starts in column 1; statement in column 10; continuation line in column 16; continuation character in column 72).
- Data field or register notation can be used for all function-specific parameters, *except* those requiring a keyword value.

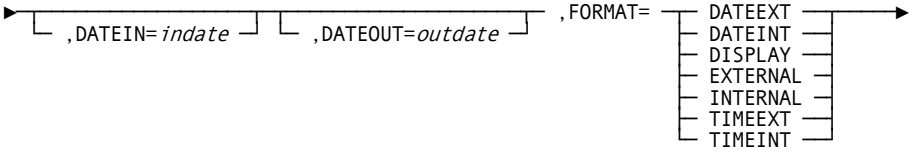
  If using data field notation, the program specifies the name of a variable field containing the parameter value.

  When using register notation, the program specifies a register containing the address of the variable field that contains the parameter value. General registers 2 through 15 can be used and the register reference must be enclosed in parentheses.

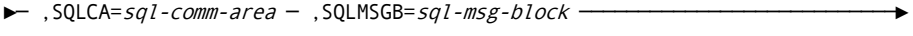
IDMSIN01 Macro Syntax



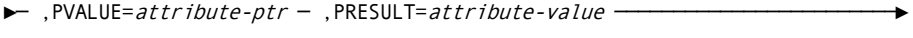
Expansion of date-parms



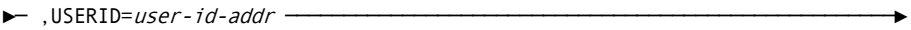
Expansion of msg-parms



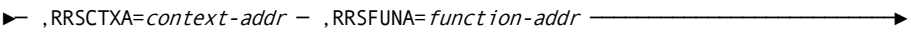
Expansion of profile-parms



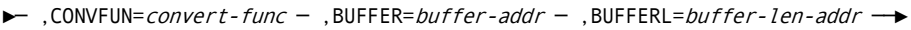
Expansion of user-parms



Expansion of rrs-parms



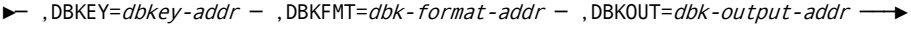
Expansion of string-parms



Expansion of sysctl-parms

Expansion of envinfo-parms

Expansion of frmtdbk-parms



Parameters

IDMSIN01 Indicates a request for the IDMSIN01 function specified by the keyword that follows. The following are the valid functions for IDMSIN01:

- GETDATE
- GETMSG
- GETPROF
- GETUSER
- NOTRACE
- RRSCTX
- SETPROF
- STRCONV
- SYSCTL
- TRACE
- TXNSOFF
- TXNSON
- ENVINFO
- FRMTDBK

GETDATE

Returns date and time in a display format.

,DATEIN=*indate*

Specifies the address of the 8-byte internal DATETIME stamp.

,DATEOUT=*outdate*

Specifies the 26-byte output field into which the display format of the DATETIME value is returned. This parameter is required for GETDATE processing. This parameter is optional.

,FORMAT=

Specifies the type of GETDATE function being requested.

DATEEXT

Specifies that a 10-byte external DATE display is returned as an 8-byte DATE stamp.

DATEINT

Specifies that an 8-byte internal DATE stamp is returned as a displayable 10-character DATE display.

DISPLAY

Specifies that the current date and time are returned as a 26-character date-time display.

EXTERNAL

Specifies that a 26-byte external DATETIME display is returned as an 8-byte DATETIME stamp.

INTERNAL

Specifies that the internal format of the DATETIME value specified by DATEIN= is returned.

TIMEEXT

Specifies that an 8-byte external time display is returned as an 8-byte TIME stamp.

TIMEINT

Specifies that an 8-byte internal TIME stamp is returned as a displayable 8-character TIME display.

GETMSG

Retrieves SQL error messages and places them in a user buffer that is displayed to the user.

,SQLCA=*sql-comm-area*

Specifies the address of the SQL communications area.

,SQLMSGB=*sql-msg-block*

Specifies the address of the SQL message control block.

GETPROF

Returns session profile information.

,PVALUE=*attribute-ptr*

Supplies the attribute keyword for the GETPROF function. *attribute-ptr* must identify an 8-byte character field.

,PRESULT=*attribute-value*

Contains the attribute value for the GETPROF function.

attribute-value must identify a 32-byte character field.

GETUSER

Requests the current user-id established by the executed JCL information when running batch, or by the SIGNON USER *xxxxxxxx* when running under CV.

,USERID=*user-id-addr*

Specifies the address of the 32-byte USERID returned value.

NOTRACE

Turns navigational DML or SQL DML tracing off.

RRSCTX

Extracts or sets a private RRS context (CV only).

,RRSCTXA=*context-addr*

Specifies the address of a 16-byte field for the RRS context token. Depending upon the function, this field is input, output, or both.

,RRSFUNA=*function-addr*

Specifies the address of a 1-byte field that contains the function to execute. The following are the valid function codes and their return codes:

X'01'

Get RRS context. The following are the valid return codes:

- 00—An RRS context exists; the field pointed to by RRSCTXA contains the current RRS context.
- 04—No RRS context exists; the field pointed to by RRSCTXA is cleared.
- 12—Invalid parameter list passed to IDMSIN01.

X'02'

Set RRS context. If the field pointed to by RRSCTXA contains binary zeros, a new RRS context is created and returned; if the field is not binary zeros, it must contain an RRS context token which is saved by the CA IDMS transaction manager. No attempt is made to validate the RRS context token. The following are the valid return codes:

- 00—The RRS context token was successfully saved by the CA IDMS transaction manager.
- 08—An active RRS context already exists or there has been an internal error.
- 12—Invalid parameter list passed to IDMSIN01.
- Any other return codes—An error has occurred. Return codes 103-107, 301, 701, 756, F00, and FFF are from context services. Their descriptions can be found in the IBM guide *MVS Programming Resource Recovery* in the topic "Begin_Context."

X'03'

End RRS context. The field pointed to by RRSCTXA must contain the token of the RRS context to be ended. The following are the valid return codes:

- 00—The RRS context was successfully terminated. The field pointed to by RRSCTXA is set to binary zeroes.
- 12—Invalid parameter list passed to IDMSIN01.
- Any other return codes—An error has occurred. Return codes 103-107, 301, 701, 756, F00, and FFF are from context services. Their descriptions can be found in the IBM guide *MVS Programming: Resource Recovery* in the topic "End_Context."

SETPROF

Changes session profile information.

,PVALUE=*attribute-ptr*

Supplies the attribute keyword for the SETPROF function. *attribute-ptr* must identify an 8-byte character field.

,PRESULT=*attribute-value*

Contains the attribute value for the SETPROF function.

attribute-value must identify a 32-byte character field.

STRCONV

Converts strings to and from EBCDIC and ASCII.

,CONVFUN=*convert-func*

Specifies the function to execute. To convert a string from ASCII to EBCDIC, specify 'ATOE'. To convert a string from EBCDIC to ASCII, specify 'ETOA'.

,BUFFER=*buffer-addr*

Specifies the name of the area that contains the string to convert.

,BUFFERL=*buffer-len-addr*

Specifies the name of a fullword field that contains the length in bytes of the string.

Converts strings to and from EBCDIC and ASCII.

SYSCTL

Establishes the SYSCTL's DDname for running CV jobs in batch.

,DDNAME=*ddname-addr*

Specifies the address of the 8-byte SYSCTL that is passed.

TRACE

Turns navigational DML or SQL DML tracing on.

TXNSOFF

Turns transaction sharing OFF for the current task.

TXNSON

Turns transaction sharing ON for the current task.

ENVINFO

Returns runtime environment information.

,EVBLOCK=*returned-environment-information-area*

Specifies the address where the environment information is returned. The first halfword contains the maximum length of the data to be returned, followed by the return area.

FRMTDBK

Formats the dbkey as character string 'page number:line index'. Leading zeros are removed from both numbers. If the dbkey has a value of zero, then the output field displays the character '0'. If the dbkey has a value of null, then the output field displays the character string '<NULL>'. If the database-key format is invalid, then the output field displays blanks and the return code is set to 8.

,DBKEY=*dbkey-addr*

Specifies the address of the dbkey to be formatted.

,DBKFMT=*dbk-format-addr*

Specifies the address of the halfword containing the database-key format associated with the dbkey. If the value provided is invalid (out of the range 2 through 12), then the output field displays blanks and the return code is set to 8.

,DBKOUT=dbk-output-addr

Specifies the address of the output field where the formatted character string will be stored. The field must have a length of 12 bytes.

PLIST=

Specifies the name of the parameter list to be used for the macro expansion. The parameter list must be at least 12 fullwords in length. This parameter can be used with all functions.

SYSPLIST

Specifies the default parameter. If PLIST= is not specified, the default value of *parameter-list-pointer* is SYSPLIST.

parameter-list-pointer

Specifies the name of the parameter list to be used for the macro expansion.

RPB=

Specifies the name of the parameter list to be used for the macro RPB= is required for user-mode programs. If RPB= is not specified, the default value of *rpb-area-pointer* is SQLRPB. This parameter can be used with all functions.

rpb-area-pointer

Specifies the name of a 36-byte work area that is modified during function execution. The RPB work area must be fullword aligned.

ERROR=*error-label*

Specifies a program label to which control should be passed in the event an error is detected during processing. This parameter can be used with all functions.

Assembler Program Calling IDMSIN01

Assembler programs can use standard calling conventions. The following are some examples of calling IDMSIN01 from an Assembler program:

```
*************************************************************************
*********************  Assembler work fields  ***************************

         #SQLCA  CSECT
SYSPLIST DC    10F'0'                 Standard PLIST
SQLRPB   DC    XL36'00'               RPB used by IDMSIN01 macro
*
SQLMSGB  DS    0F                     SQL error messages control block
SQLMMAX  DC    F'6'                   Max. number of SQL error lines
SQLMSIZE DC    F'80'                  Error line size
SQLMCNT  DC    F'0'                   Act. number of messages returned
SQLMLINE DC    6CL80' '               Allow for 6 error messages
*
```

```
XTRAPKEY DS    CL8                    Key value for SETPROF + GETPROF
XTRAPVAL DS    CL32                   Variable for SETPROF + GETPROF
DATETIME DS    XL8                    Internal date/time stamp
DATEFLD  DS    CL26                   Edited date/time used by GETDATE
TIMESTMP DS    XL8                    Internal TIME stamp
TIMESHOW DS    CL8                    External TIME display
DATESTMP DS    XL8                    Internal DATE stamp
DATESHOW DS    CL10                   External DATE display
USERID   DS    CL32                   Current user id returned by USERID
DDSYSCTL DS    CL8                    DDNAME for SYSCTL
BLANKS   DC    CL133' '               Blanks for all
RRSCTX   DC    XL16'00'               16-byte context token
RRSFUNC  DS    X                      RRS context function:
RRSFNGET EQU   X'01'                  - Get RRS context
RRSFNSET EQU   X'02'                  - Set RRS context
CONVFUNC DS    CL4                    STRCONV function:
CONVFE2A EQU   'ETOA'                 - EBCDIC -> ASCII
CONVFA2E EQU   'ATOE'                 - ASCII  -> EBCDIC
STRING   DC    C'String to convert'   String to convert
STRINGL  DC    A(L'STRING)            String length
          SPACE
         DS    0F                     Align on a fullword boundary
EVBLOCK  DS    XL(EV$DSLEN)           Runtime environment return area
         COPY  #ENVINFO               Copy in runtime environment
                                      return area dsect
DBKEYFLD DS    F                      Dbkey to be formatted
DBKEYFMT DS    H                      Associated database-key format
DBKCHAR  DS    CL12                   Output field for formatted dbkey
************************************************************
*  Call IDMSIN01 to deactivate the DML trace or SQL trace
*  which was originally activated by the corresponding
*  SYSIDMS parm (DMLTRACE=ON or SQLTRACE=ON).
************************************************************

         IDMSIN01 NOTRACE                  Deactivate the DML or SQL trace

************************************************************
*  Call IDMSIN01 to request a 'GETPROF' to get the user
*  profile default DBNAME, which was established by the
*  SYSIDMS parm DBNAME=xxxxxxxx when running batch, or
*  by the DCUF SET DBNAME xxxxxxxx when running under CV.
*
*    PVALUE is the address of the 8 byte GETPROF keyword.
*    PRESULT is the address of the 32 byte GETPROF returned value.
************************************************************
```

```
         MVC   XTRAPKEY,=CL8'DBNAME'       Establish GETPROF keyval
         IDMSIN01 GETPROF,
               PVALUE=XTRAPKEY,
               PRESULT=XTRAPVAL,
               ERROR=ERROROUT
         MVC   WORKLINE,BLANKS             Clear print work line
         MVC   WORKLINE+5(6),=C'DBNAME'    Move out GETPROF keyval
         MVC   WORKLINE+11(17),=C'         is set to BLANKS'
         CLC   XTRAPVAL,BLANKS             Was variable set to blanks
         BE    *+4+6                       Yes, all set
         MVC   WORKLINE+22(32),XTRAPVAL    Move out variable
         $PRNT WORKLINE                    Print the GETPROF results

*********************************************************
*  Call IDMSIN01 to activate Transaction Sharing for this
*  task.
*********************************************************

         IDMSIN01 TXNSON                   Activate Transaction Sharing

*********************************************************
*  Call IDMSIN01 to deactivate Transaction Sharing for this
*  task.
*********************************************************

         IDMSIN01 TXNSOFF                  Deactivate Transaction Sharing

*********************************************************
*  Call IDMSIN01 to request a 'RRSCTX' to set a private
*  context.
*
*    RRSFUNA
*    Specifies the address of a 1-byte field that contains
*    the function to execute.  Valid values are:
*       X'01' Get RRS context.
*       X'02' Set RRS context.
*
*    RRSCTXA
*    Specifies the address of a 16-byte field for the RRS
*    context token.  Depending upon the function, this field is
*    input, output, or both.
*********************************************************

         MVI   RRSFUNC,RRSFNGET            Function: get RRS context
         IDMSIN01 RRSCTX,
               RRSFUNA=RRSFUNC,
               RRSCTXA=RRSCTX,
               ERROR=ERROROUT
```

```
*********************************************************
*  Call IDMSIN01 to request a 'STRCONV' to convert a string
*  from EBCDIC to ASCII
*    CONVFUN - specifies which conversion:
*       ETOA - EBCDIC to ASII
*       ATOE - ASCII to EBCDIC
*
*    BUFFER - SPECIFIES STRING TO CONVERT
*    BUFFERL- SPECIFIES LENGTH OF STRING
*********************************************************
         MVC   CONVFUNC,=A(CONVFE2A)        Convert EBCDIC to ASCII
         IDMSIN01 STRCONV,
               BUFFER=STRING,
               BUFFERL=STRINGL,
               ERROR=ERROROUT

*********************************************************
*   Call IDMSIN01 to convert STRING (now in ASCII) back to EBCDIC
*********************************************************

         MVC   CONVFUNC,=A(CONVFA2E)        Convert ASCII to EBCDIC
         IDMSIN01 STRCONV,
               BUFFER=STRING,
               BUFFERL=STRINGL,
               ERROR=ERROROUT

*********************************************************
*  Call IDMSIN01 to request a 'SETPROF' to set the user
*  profile default SCHEMA to the value 'SYSTEM'.
*
*    PVALUE is the address of the 8 byte SETPROF keyword.
*    PRESULT is the address of the 32 byte SETPROF value.
*********************************************************
```

```
         MVC   XTRAPKEY,=CL8'SCHEMA'        Est. SETPROF keyval
         MVC   XTRAPVAL,BLANKS              Init SETPROF variable
         MVC   XTRAPVAL(8),=CL8'SYSTEM'     Save SETPROF variable
         IDMSIN01 SETPROF,
               PVALUE=XTRAPKEY,
               PRESULT=XTRAPVAL,
               ERROR=ERROROUT

*********************************************************
*  Call IDMSIN01 to request the current USERID established
*  by the executed JCL information when running batch, or
*  by the SIGNON USER xxxxxxxx when running under CV.
*
*    USERID is the address of the 32 byte USERID returned value.
*********************************************************

         IDMSIN01 GETUSER,
               USERID=USERID,
               ERROR=ERROROUT
         MVC   WORKLINE,BLANKS              Clear print work line
         MVC   WORKLINE+10(17),=C'Current user --> '
         MVC   WORKLINE+27(32),USERID       Display current user id
         $PRNT WORKLINE                     Print the user id

*********************************************************
*  Call IDMSIN01 to establish the SYSCTL DDNAME to be used
*  when running a Batch/CV job.
*
*    DDNAME is the address of the 8 byte SYSCTL DDNAME passed.
*********************************************************

         MVC   DDSYSCTL,=C'SYSCTL73'        Est. DDNAME for SYSCTL file
         IDMSIN01 SYSCTL,
               DDNAME=DDSYSCTL,
               ERROR=ERROROUT

*********************************************************
*  Call IDMSIN01 to have an 8 byte internal DATETIME stamp
*  returned as a displayable 26 character DATE/TIME display.
*
*    DATEIN is the address of the 8 byte internal DATETIME stamp.
*    DATEOUT is the address of the 26 byte DATE/TIME returned.
*********************************************************
```

```
         IDMSIN01 GETDATE,
               DATEIN=DATETIME,
               DATEOUT=DATEFLD,
               FORMAT=INTERNAL,
               ERROR=ERROROUT
         MVC   WORKLINE,BLANKS              Clear print work line
         MVC   WORKLINE+10(14),=C'DATETIME ---> '
         MVC   WORKLINE+24(26),DATEFLD      Displayable date/time
         $PRNT WORKLINE                     Print the date/time

*********************************************************
*  Call IDMSIN01 to have the current DATE and TIME
*  returned as a displayable 26 character DATE/TIME display.
*
*    DATEOUT is the address of the 26 byte DATE/TIME returned.
*********************************************************

         IDMSIN01 GETDATE,
               DATEOUT=DATEFLD,
               FORMAT=DISPLAY,
               ERROR=ERROROUT
         MVC   WORKLINE,BLANKS              Clear print work line
         MVC   WORKLINE+10(22),=C'Current DATETIME ---> '
         MVC   WORKLINE+32(26),DATEFLD      Displayable date/time
         $PRNT WORKLINE                     Print the current date/time

*********************************************************
*  Call IDMSIN01 to have a 26 byte external DATE/TIME display
*  returned as an 8 byte DATETIME stamp.
*
*    DATEIN is the address of the 26 byte DATE/TIME.
*    DATEOUT is the address of the 8 byte DATETIME stamp returned.
*********************************************************

         MVC   DATEFLD,=C'1994-07-18-12.01.18.458382'
         IDMSIN01 GETDATE,
               DATEIN=DATEFLD,
               DATEOUT=DATETIME,
               FORMAT=EXTERNAL,
               ERROR=ERROROUT

*********************************************************
*  Call IDMSIN01 to have a 8 byte external TIME display
*  returned as an 8 byte TIME stamp.
*
*    DATEIN is the address of the 8 byte external TIME.
*    DATEOUT is the address of the 8 byte TIME stamp returned.
*********************************************************
```

```
         MVC   TIMESHOW,=C'13.58.11'
         IDMSIN01 GETDATE,
               DATEIN=TIMESHOW,
               DATEOUT=TIMESTMP,
               FORMAT=TIMEEXT,
               ERROR=ERROROUT

*********************************************************
*  Call IDMSIN01 to have an 8 byte internal TIME stamp
*  returned as a displayable 8 character TIME display.
*
*    DATEIN is the address of the 8 byte internal TIME stamp.
*    DATEOUT is the address of the 8 byte external TIME returned.
*********************************************************

         IDMSIN01 GETDATE,
               DATEIN=TIMESTMP,
               DATEOUT=TIMESHOW,
               FORMAT=TIMEINT,
               ERROR=ERROROUT
         MVC   WORKLINE,BLANKS              Clear print work line
         MVC   WORKLINE+10(10),=C'TIME ---> '
         MVC   WORKLINE+20(8),TIMESHOW      Displayable time
         $PRNT WORKLINE                     Print the time

*********************************************************
*  Call IDMSIN01 to have a 10 byte external DATE display
*  returned as an 8 byte DATE stamp.
*
*    DATEIN is the address of the 10 byte external DATE.
*    DATEOUT is the address of the 8 byte DATE stamp returned.
*********************************************************

         MVC   DATESHOW,=C'2003-03-10'
         IDMSIN01 GETDATE,
               DATEIN=DATESHOW,
               DATEOUT=DATESTMP,
               FORMAT=DATEEXT,
               ERROR=ERROROUT

*********************************************************
*  Call IDMSIN01 to have an 8 byte internal DATE stamp
*  returned as a displayable 10 character DATE display.
*
*    DATEIN is the address of the 8 byte internal DATE stamp.
*    DATEOUT is the address of the 10 byte external DATE returned.
*********************************************************
```

```
         IDMSIN01 GETDATE,
               DATEIN=DATESTMP,
               DATEOUT=DATESHOW,
               FORMAT=DATEINT,
               ERROR=ERROROUT
         MVC   WORKLINE,BLANKS              Clear print work line
         MVC   WORKLINE+10(10),=C'DATE ---> '
         MVC   WORKLINE+20(10),DATESHOW     Displayable date
         $PRNT WORKLINE                     Print the date

*********************************************************
*  Call IDMSIN01 to retrieve SQL error messages into a user
*  buffer that will then be displayed back to the user.
*  Whats passed is the SQLCA block and a message control
*  block consisting of the following fields:
*
*          - Maximum number of lines in user buffer.
*          - The size (width) of one line in the user buffer.
*          - The actual number of lines returned from IDMSIN01.
*          - The user buffer where the message lines are returned
*
*  A return code of 4 means that there were no SQL error messages.
*  A return code of 8 means that there were more SQL error lines
*  in the SQLCA than could fit into the user buffer, meaning
*  truncation has occurred.
*
*    SQLCA is the address of the SQLCA block.
*    SQLMSGB is the address of the message control block.
*********************************************************

         IDMSIN01 GETMSG,
               SQLCA=SQLCA,
               SQLMSGB=SQLMSGB
         CH    R15,=H'4'                    Were there any SQL errors returned
         BE    NOMSGS                       No, well thats okay with me
         MVC   WORKLINE,BLANKS              Clear print work line
         MVC   WORKLINE+11(27),=C'Buffer returned from XTRA  '
         $PRNT WORKLINE                     Print the heading
         $PRNT BLANKS                       Print 1 blank line
         L     R3,SQLMCNT                   Get number of message lines returned
         LA    R5,SQLMLINE                  Point at first message line
         MVC   WORKLINE,BLANKS              Clear print work line
MSGLOOP  MVC   WORKLINE+3(80),0(R5)          Move SQL error message to print line
         $PRNT WORKLINE                     Print SQL error message
         LA    R5,80(R5)                    Bump to next SQL error message
         BCT   R3,MSGLOOP                   Print all SQL error messages
```

```
*********************************************************
*  Call IDMSIN01 to reactivate the DML trace or SQL trace
*  which was originally activated by the corresponding
*  SYSIDMS parm (DMLTRACE=ON or SQLTRACE=ON), that has
*  been previously deactivated earlier on in this job.
*********************************************************

         IDMSIN01 TRACE                    Reactivate the DML or SQL trace

*********************************************************************
*  Call IDMSIN01 to request that it return runtime environment
*  information. The layout of the information returned is described
*  in #ENVINFO dsect.
*
*  EVBLOCK is the address of where to return the information. The
*  first halfword contains the length of the data you want returned.
*********************************************************************

          LA       R5,EVBLOCK             Get address of return area
          USING  EV$INFO,R5               -> EV$INFO
           MVC   EV$SIZE,=AL2(EV$MAXL)    Return all environment information
           IDMSIN01 ENVINFO,
              EVBLOCK= EV$INFO,
              ERROR=ERROROUT
           MVC   WORKLINE,BLANKS          Clear print work line
           MVC   WORKLINE+5(5),=C'Mode='
           MVC   WORKLINE+10(1), EV$MODE  Show runtime mode
           MVC   WORKLINE+13(5),=C'Tape='
           MVC   WORKLINE+18(6), EV$TAPE# Show CA IDMS tape volser
           MVC   WORKLINE+26(8),=C'Release='
           MVC   WORKLINE+34(6), EV$REL#  Show CA IDMS release number
           MVC   WORKLINE+42(13),=C'Service Pack='
           MVC   WORKLINE+55(2), EV$SPACK Show CA IDMS tape service pack number
           MVC   WORKLINE+59(5),=C'DMCL='
           MVC   WORKLINE+64(8), EV$DMCL  Show DMCL name
           MVC   WORKLINE+74(5),=C'Node='
           MVC   WORKLINE+79(8), EV$NODE  Show system node name
           $PRNT WORKLINE                 Print the ENVINFO results

*********************************************************
*   Call IDMSIN01 to format dbkey
*********************************************************
         IDMSIN01 FRMTDBK,
               DBKEY=DBKEYFLD,
               DBKFMT=DBKEYFMT,
               DBKOUT=DBKCHAR
```

Calling IDMSIN01 from a CA ADS Dialog

When you are using CA ADS, issue calls to IDMSIN01 using this CA ADS convention:

```
LINK TO 'IDMSIN01' USING (RPB, REQ-WK, parm-3, ... parm-n)
```

Calling IDMSIN01 from a COBOL Program

When calling IDMSIN01 from a COBOL program, the first two parameters passed are always the address of an RPB block and the address of the function REQUEST-CODE and RETURN-CODE fields. The rest of the parameters depend on what service is being called.

COBOL Program Calling IDMSIN01

COBOL programs can use standard calling conventions. The following is an example of calling all IDMSIN01 functions from a COBOL program.

```
 ENVIRONMENT DIVISION.

 DATA DIVISION.
 WORKING-STORAGE SECTION.

*************************************************************
*     The following is the 1st parameter on all IDMSIN01 calls
*************************************************************
 01  RPB.
     02  FILLER             PIC X(36).

*************************************************************
*     The following is the 2nd parameter on all IDMSIN01 calls
*************************************************************
 01  REQ-WK.
     02  REQUEST-CODE      PIC S9(8) COMP.
         88  IN01-FN-TRACE            VALUE 00.
         88  IN01-FN-NOTRACE          VALUE 01.
         88  IN01-FN-GETPROF          VALUE 02.
         88  IN01-FN-SETPROF          VALUE 03.
         88  IN01-FN-GETMSG           VALUE 04.
         88  IN01-FN-GETDATE          VALUE 05.
         88  IN01-FN-GETUSER          VALUE 08.
```

```
           88  IN01-FN-SYSCTL           VALUE 10.
           88  IN01-FN-TRINFO           VALUE 16.
           88  IN01-FN-TXNSON           VALUE 28.
           88  IN01-FN-TXNSOFF          VALUE 29.
           88  IN01-FN-RRSCTX           VALUE 30.
           88  IN01-FN-STRCONV          VALUE 34.
           88  IN01-FN-ENVINFO          VALUE 36.
           88  IN01-FN-FRMTDBK          VALUE 40.
       02  REQUEST-RETURN     PIC S9(8) COMP.

*************************************************************
*     The following work fields are used by a variety of
*     IDMSIN01 calls
*************************************************************
 01  WORK-FIELDS.
       02  WK-DTS-FORMAT          PIC S9(8) COMP VALUE 0.
       02  LINE-CNT               PIC S9(4) COMP.
       02  WK-DTS                 PIC X(8).
       02  WK-CDTS                PIC X(26).
       02  WK-KEYWD               PIC X(8).
       02  WK-VALUE               PIC X(32).
       02  WK-DBNAME              PIC X(8).
       02  WK-USERID              PIC X(32).
       02  WK-SYSCTL              PIC X(8).
       02  WK-TIME-INTERNAL       PIC X(8).
       02  WK-TIME-EXTERNAL       PIC X(8).
       02  WK-DATE-INTERNAL       PIC X(8).
       02  WK-DATE-EXTERNAL       PIC X(10).
       02  WK-RRS-FAKE-FUNCTION   PIC S9(4) COMP.
           88  IN01-FN-RRSCTX-GET       VALUE 01.
           88  IN01-FN-RRSCTX-SET       VALUE 02.
       02  WK-RRS-FUNCTION-REDEF    REDEFINES WK-RRS-FAKE-FUNCTION.
           03  WK-RRS-FAKE-FILLER PIC X.
           03  WK-RRS-FUNCTION    PIC X.
       02  WK-RRS-CONTEXT         PIC X(16).
       02  WK-STRING-FUNCTION     PIC X(4).
           88  CONVERT-EBCDIC-TO-ASCII  VALUE 'ETOA'.
           88  CONVERT-ASCII-TO-EBCDIC  VALUE 'ATOE'.
       02  WK-STRING              PIC X(17)
                                        VALUE 'String to convert'.
       02  WK-STRING-LENGTH       PIC S9(8) COMP VALUE 17.
       02  WK-DBKEY-OUTPUT        PIC X(12).
```

```
*************************************************************
*     The following group item is only used by the call that
*     returns runtime environment information.
*************************************************************
 01  EVBLOCK.
     02  EV$SIZE            PIC S9(4) COMP VALUE +31.
     02  EV$MODE            PIC X.
     02  EV$TAPE#           PIC X(6).
     02  EV$REL#            PIC X(6).
     02  EV$SPACK           PIC X(2).
     02  EV$DMCL            PIC X(8).
     02  EV$NODE            PIC X(8).

*************************************************************
*     The following group item is only used by the call that
*     retrieves SQL error messages
*************************************************************
 01  SQLMSGB.
     02  SQLMMAX            PIC S9(8) COMP VALUE +6.
     02  SQLMSIZE           PIC S9(8) COMP VALUE +80.
     02  SQLMCNT            PIC S9(8) COMP.
     02  SQLMLINE           OCCURS 6 TIMES PIC X(80).

*************************************************************
*     The following SQL include statement is needed only for
*     the call that retrieves SQL error messages, and is only
*     required if the program contains no other SQL statements.
*************************************************************
 EXEC SQL
     INCLUDE SQLCA
 END-EXEC.

*********************************************************
 PROCEDURE DIVISION.
*********************************************************

*********************************************************
*  Call IDMSIN01 to deactivate the DML trace or SQL trace
*  which was originally activated by the corresponding
*  SYSIDMS parm (DMLTRACE=ON or SQLTRACE=ON).
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*********************************************************
```

```
     SET IN01-FN-NOTRACE TO TRUE.
     CALL 'IDMSIN01' USING RPB REQ-WK.

*********************************************************
*  Call IDMSIN01 to request a 'GETPROF' to get the user
*  profile default DBNAME, which was established by the
*  SYSIDMS parm DBNAME=xxxxxxxx when running batch, or
*  by the DCUF SET DBNAME xxxxxxxx when running under CV.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*    Parm 3 is the address of the 8 byte GETPROF keyword.
*    Parm 4 is the address of the 32 byte GETPROF returned value.
*********************************************************

     SET IN01-FN-GETPROF TO TRUE.
     MOVE 'DBNAME' TO WK-KEYWD
     CALL 'IDMSIN01' USING RPB REQ-WK WK-KEYWD
       WK-VALUE.
     MOVE WK-VALUE TO WK-DBNAME.
     IF WK-DBNAME = SPACES
       DISPLAY 'DBNAME is set to BLANKS'
              ELSE
       DISPLAY 'DBNAME is set to ' WK-DBNAME.

*********************************************************
*  Call IDMSIN01 to activate Transaction Sharing for this
*  task.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*********************************************************

     SET IN01-FN-TXNSON  TO TRUE.
     CALL 'IDMSIN01' USING RPB REQ-WK.

*********************************************************
*  Call IDMSIN01 to deactivate Transaction Sharing for this
*  task.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*********************************************************

     SET IN01-FN-TXNSOFF TO TRUE.
     CALL 'IDMSIN01' USING RPB REQ-WK.
```

```
*********************************************************
*  Call IDMSIN01 to request a 'SETPROF' to set the user
*  profile default SCHEMA to the value 'SYSTEM'.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*    Parm 3 is the address of the 8 byte SETPROF keyword.
*    Parm 4 is the address of the 32 byte SETPROF value.
*********************************************************

     SET IN01-FN-SETPROF TO TRUE.
     MOVE 'SCHEMA' TO WK-KEYWD
     MOVE 'SYSTEM' TO WK-VALUE
     CALL 'IDMSIN01' USING RPB REQ-WK WK-KEYWD
       WK-VALUE.
     IF REQUEST-RETURN NOT = 0
       DISPLAY 'SETPROF returned error ' REQUEST-RETURN.

*********************************************************
*  Call IDMSIN01 to request the current USERID established
*  by the executed JCL information when running batch, or
*  by the SIGNON USER xxxxxxxx when running under CV.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*    Parm 3 is the address of the 32 byte USERID returned value.
*********************************************************

     SET IN01-FN-GETUSER TO TRUE.
     CALL 'IDMSIN01' USING RPB REQ-WK WK-USERID.
     IF WK-USERID = SPACES
       DISPLAY 'USERID is set to BLANKS'
              ELSE
       DISPLAY 'USERID is set to ' WK-USERID.

*********************************************************
*  Call IDMSIN01 to establish the SYSCTL DDNAME to be used
*  when running a Batch/CV job.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*    Parm 3 is the address of the 8 byte SYSCTL DDNAME passed.
*********************************************************
```

```
    SET IN01-FN-SYSCTL  TO TRUE.
    MOVE 'SYSCTL73' TO WK-SYSCTL.
    CALL 'IDMSIN01' USING RPB REQ-WK WK-SYSCTL.

*****************************************************************
* Call IDMSIN01 to retrieve the current RRS context token.
* Uses an alternate method to set the function by using the
* SET statement, which allows exploiting the LEVEL 88 definitions.
*****************************************************************

     SET IN01-FN-RRSCTX       TO TRUE.
     SET IN01-FN-RRSCTX-GET   TO TRUE.
     CALL 'IDMSIN01' USING RPB,
                           REQ-WK,
                           WK-RRS-FUNCTION,
                           WK-RRS-CONTEXT.

*****************************************************************
* Call IDMSIN01 to request string conversion from EBCDIC to ASCII.
*****************************************************************

     SET IN01-FN-STRCONV           TO TRUE.
     SET CONVERT-EBCDIC-TO-ASCII   TO TRUE.
     CALL 'IDMSIN01' USING RPB,
                           REQ-WK,
                           WK-STRING-FUNCTION,
                           WK-STRING,
                           WK-STRING-LENGTH.

*****************************************************************
* Call IDMSIN01 to request string conversion from ASCII to EBCDIC.
*****************************************************************

     SET IN01-FN-STRCONV           TO TRUE.
     SET CONVERT-ASCII-TO-EBCDIC   TO TRUE.
     CALL 'IDMSIN01' USING RPB,
                           REQ-WK,
                           WK-STRING-FUNCTION,
                           WK-STRING,
                           WK-STRING-LENGTH.
```

```
*********************************************************
*  Call IDMSIN01 to have an 8 byte internal DATETIME stamp
*  returned as a displayable 26 character DATE/TIME display.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*    Parm 3 is the address of the 4 byte format indicator (0).
*    Parm 4 is the address of the 8 byte internal DATETIME stamp.
*    Parm 5 is the address of the 26 byte DATE/TIME returned.
*********************************************************

     SET IN01-FN-GETDATE TO TRUE.
        MOVE 0 TO WK-DTS-FORMAT
        MOVE 'UNKNOWN' TO WK-CDTS
     CALL 'IDMSIN01' USING RPB REQ-WK
           WK-DTS-FORMAT WK-DTS WK-CDTS.
        DISPLAY 'THE DATE AND TIME IS -->  ' WK-CDTS.

*********************************************************
*  Call IDMSIN01 to have the current DATE and TIME
*  returned as a displayable 26 character DATE/TIME display.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*    Parm 3 is the address of the 4 byte format indicator (1).
*    Parm 4 is the address of the 26 byte DATE/TIME returned.
*********************************************************

     SET IN01-FN-GETDATE TO TRUE.
        MOVE 1 TO WK-DTS-FORMAT
     CALL 'IDMSIN01' USING RPB REQ-WK
           WK-DTS-FORMAT WK-CDTS.
        DISPLAY 'THE DATE AND TIME IS -->  ' WK-CDTS.

*********************************************************
*  Call IDMSIN01 to have a 26 byte external DATE/TIME display
*  returned as an 8 byte DATETIME stamp.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*    Parm 3 is the address of the 4 byte format indicator (2).
*    Parm 4 is the address of the 26 byte DATE/TIME.
*    Parm 5 is the address of the 8 byte DATETIME stamp returned.
*********************************************************
```

```
     SET IN01-FN-GETDATE TO TRUE.
        MOVE 2 TO WK-DTS-FORMAT
        MOVE '1994-07-18-12.01.18.458382' TO WK-CDTS
     CALL 'IDMSIN01' USING RPB REQ-WK
           WK-DTS-FORMAT WK-CDTS WK-DTS.

*********************************************************
*  Call IDMSIN01 to have a 8 byte external TIME display
*  returned as an 8 byte TIME stamp.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*    Parm 3 is the address of the 4 byte format indicator (4).
*    Parm 4 is the address of the 8 byte external TIME.
*    Parm 5 is the address of the 8 byte TIME stamp returned.
*********************************************************

     SET IN01-FN-GETDATE TO TRUE.
        MOVE 4 TO WK-DTS-FORMAT
        MOVE '13.58.11' TO WK-TIME-EXTERNAL
     CALL 'IDMSIN01' USING RPB REQ-WK WK-DTS-FORMAT
           WK-TIME-EXTERNAL WK-TIME-INTERNAL.

*********************************************************
*  Call IDMSIN01 to have an 8 byte internal TIME stamp
*  returned as a displayable 8 character TIME display.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*    Parm 3 is the address of the 4 byte format indicator (3).
*    Parm 4 is the address of the 8 byte internal TIME stamp.
*    Parm 5 is the address of the 8 byte external TIME returned.
*********************************************************

     SET IN01-FN-GETDATE TO TRUE.
        MOVE 3 TO WK-DTS-FORMAT
     CALL 'IDMSIN01' USING RPB REQ-WK WK-DTS-FORMAT
           WK-TIME-INTERNAL WK-TIME-EXTERNAL.
        DISPLAY 'THE EXTERNAL TIME IS -->  ' WK-TIME-EXTERNAL.
```

```
*********************************************************
*  Call IDMSIN01 to have a 10 byte external DATE display
*  returned as an 8 byte DATE stamp.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*    Parm 3 is the address of the 4 byte format indicator (6).
*    Parm 4 is the address of the 10 byte external DATE.
*    Parm 5 is the address of the 8 byte DATE stamp returned.
*********************************************************

     SET IN01-FN-GETDATE TO TRUE.
        MOVE 6 TO WK-DTS-FORMAT
        MOVE '2003-03-10' TO WK-DATE-EXTERNAL
     CALL 'IDMSIN01' USING RPB REQ-WK WK-DTS-FORMAT
           WK-DATE-EXTERNAL WK-DATE-INTERNAL.

*********************************************************
*  Call IDMSIN01 to have an 8 byte internal DATE stamp
*  returned as a displayable 10 character DATE display.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*    Parm 3 is the address of the 4 byte format indicator (5).
*    Parm 4 is the address of the 8 byte internal DATE stamp.
*    Parm 5 is the address of the 10 byte external DATE returned.
*********************************************************

     SET IN01-FN-GETDATE TO TRUE.
        MOVE 5 TO WK-DTS-FORMAT
     CALL 'IDMSIN01' USING RPB REQ-WK WK-DTS-FORMAT
           WK-DATE-INTERNAL WK-DATE-EXTERNAL.
        DISPLAY 'THE EXTERNAL DATE IS -->  ' WK-DATE-EXTERNAL.

*********************************************************
*  Call IDMSIN01 to retrieve SQL error messages into a user
*  buffer that will then be displayed back to the user.
*  Whats passed is the SQLCA block and a message control
*  block consisting of the following fields:
*
*           - Maximum number of lines in user buffer.
*           - The size (width) of one line in the user buffer.
*           - The actual number of lines returned from IDMSIN01.
*           - The user buffer where the message lines are returned.
*
```

```
*  A return code of 4 means that there were no SQL error messages.
*  A return code of 8 means that there were more SQL error lines
*  in the SQLCA than could fit into the user buffer, meaning
*  truncation has occurred.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*    Parm 3 is the address of the SQLCA block.
*    Parm 4 is the address of the message control block.
************************************************************

     SET IN01-FN-GETMSG  TO TRUE.
     CALL 'IDMSIN01' USING RPB, REQ-WK,
           SQLCA, SQLMSGB.
     IF REQUEST-RETURN NOT = 4
       MOVE 1 TO LINE-CNT
       PERFORM DISP-MSG UNTIL LINE-CNT > SQLMCNT.

 DISP-MSG SECTION.
     DISPLAY SQLMLINE (LINE-CNT).
     ADD 1 TO LINE-CNT.

************************************************************
*  Call IDMSIN01 to reactivate the DML trace or SQL trace
*  which was originally activated by the corresponding
*  SYSIDMS parm (DMLTRACE=ON or SQLTRACE=ON), that has
*  been previously deactivated earlier on in this job.
*
*    Parm 1 is the address of the RPB.
*    Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
************************************************************

     SET IN01-FN-TRACE   TO TRUE.
     CALL 'IDMSIN01' USING RPB REQ-WK.

************************************************************
*  Call IDMSIN01 to request that it return runtime
*  environment information.
*
*  Parm 1 is the address of the RPB.
*  Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*  Parm 3 is the address of the ENVINFO return area.
************************************************************
```

```
           SET IN01-FN-ENVINFO TO TRUE.
           CALL 'IDMSIN01' USING RPB REQ-WK EVBLOCK.
           DISPLAY 'Runtime mode is  ' EV$MODE.
           DISPLAY 'CA IDMS tape volser is  ' EV$TAPE#.
           DISPLAY 'CA IDMS release number is  ' EV$REL#.
           DISPLAY 'CA IDMS service pack number is  ' EV$SPACK.
           DISPLAY 'DMCL name is  ' EV$DMCL.
           DISPLAY 'System node name is  ' EV$NODE.

      ********************************************************
      *  Call IDMSIN01 to format dbkey stored in SUBSCHEMA-CTRL
      ********************************************************

            SET IN01-FN-FRMTDBK TO TRUE.
            CALL 'IDMSIN01' USING RPB,
                                  REQ-WK,
                                  DBKEY,
                                  PAGE-INFO-DBK-FORMAT,
                                  WK-DBKEY-OUTPUT.
            DISPLAY ‘DBKEY = ‘ WK-DBKEY-OUTPUT.
```

Calling IDMSIN01 from a PL/I Program

The following is an example of calling IDMSIN01 functions from a PL/I program:

```
/* Declare IDMSIN01 entry */
 DCL IDMSIN01 ENTRY OPTIONS(INTER,ASSEMBLER);
/* Definition of IDMSIN01 variables: */
DCL 1 REQ_WK,
     2 REQUEST_CODE   FIXED BINARY(31),
     2 REQUEST_RETURN FIXED BINARY(31);
```

```
/* Definition of IDMSIN01 functions: */
DCL IN01_FN_TRACE    FIXED BINARY(31)  VALUE(00);
DCL IN01_FN_NOTRACE  FIXED BINARY(31)  VALUE(01);
DCL IN01_FN_GETPROF  FIXED BINARY(31)  VALUE(02);
DCL IN01_FN_SETPROF  FIXED BINARY(31)  VALUE(03);
DCL IN01_FN_GETMSG   FIXED BINARY(31)  VALUE(04);
DCL IN01_FN_GETDATE  FIXED BINARY(31)  VALUE(05);
DCL IN01_FN_GETUSER  FIXED BINARY(31)  VALUE(08);
DCL IN01_FN_SYSCTL   FIXED BINARY(31)  VALUE(10);
DCL IN01_FN_TRINFO   FIXED BINARY(31)  VALUE(16);
DCL IN01_FN_TXNSON   FIXED BINARY(31)  VALUE(28);
DCL IN01_FN_TXNSOFF  FIXED BINARY(31)  VALUE(29);
DCL IN01_FN_RRSCTX   FIXED BINARY(31)  VALUE(30);
DCL IN01_FN_STRCONV  FIXED BINARY(31)  VALUE(34);
DCL IN01_FN_ENVINFO  FIXED BINARY(31)  VALUE(36);
DCL IN01_FN_FRMTDBK  FIXED BINARY(31)  VALUE(40);

/* The following work fields are used by a variety of */
/* IDMSIN01 calls */
DCL 1 WORK_FIELDS,
     2 WK_DTS_FORMAT      FIXED BINARY(31) INIT(0),
     2 LINE_CNT           FIXED BINARY(31),
     2 WK_DTS             CHAR(8),
     2 WK_CDTS            CHAR(26),
     2 WK_KEYWD           CHAR(8),
     2 WK_VALUE           CHAR(32),
     2 WK_DBNAME          CHAR(8),
     2 WK_SYSCTL          CHAR(8),
     2 WK_TIME_INTERNAL   CHAR(8),
     2 WK_TIME_EXTERNAL   CHAR(8),
     2 WK_DATE_INTERNAL   CHAR(8),
     2 WK_DATE_EXTERNAL   CHAR(10),
     2 WK_USERID          CHAR(32);
     2 WK_DBKEY_OUTPUT    CHAR(12);

DCL 1 WK_RRS_FUNCTION     FIXED BINARY (7);

/* Definition of WK_RRS_FUNCTION functions: */

DCL  IN01_FN_RRSCTX_GET   FIXED BINARY (7)  VALUE (1);
DCL  IN01_FN_RRSCTX_SET   FIXED BINARY (7)  VALUE (2);
```

```
DCL 1 WK_RRS_CONTEXT      BIT (128);
DCL 1 WK_STRING_FUNCTION  CHAR (4);

/* Definition of WK_STRING_FUNCTION functions: */

DCL  CONVERT_EBCDIC_TO_ASCII  CHAR (4) VALUE ('ETOA');
DCL  CONVERT_ASCII_TO_EBCDIC  CHAR (4) VALUE ('ATOE');

DCL 1 WK_STRING           CHAR (17) INIT('String to convert');
DCL 1 WK_STRING_LENGTH    FIXED BINARY(31) INIT(17);

DCL 1 SNAP_TITLE,
    3 SNAP_TITLE_TEXT CHAR (14) INIT (' PLIIN01 snap '),
    3 SNAP_TITLE_END  CHAR (1)  INIT (' ');

/* ************************************************************** */
/* The following group item is only used by the call that         */
/* retrieves runtime environment information.                     */
/* ************************************************************** */

DCL 1 EVBLOCK,
     2 EV$SIZE   FIXED BINARY(15) INIT(31),
     2 EV$MODE   CHAR(1),
     2 EV$TAPE#  CHAR(6),
     2 EV$REL#   CHAR(6),
     2 EV$SPACK  CHAR(2),
     2 EV$DMCL   CHAR(8),
     2 EV$NODE   CHAR(8);

/* ************************************************************** */
/* The following group item is only used by the call that         */
/* retrieves SQL error messages.                                  */
/* ************************************************************** */

DCL 1 SQLMSGB,
     2 SQLMMAX  FIXED BINARY(31) INIT(6),
     2 SQLMSIZE FIXED BINARY(31) INIT(80),
     2 SQLMCNT  FIXED BINARY(31),
     2 SQLMLINE (6) CHAR(80);

/* ************************************************************** */
/*  The following SQL include statement is needed only for        */
/*  the call that retrieves SQL error messages, and is only       */
/*  required if the program contains no other SQL statements.     */
/* ************************************************************** */
```

```
   EXEC SQL   INCLUDE SQLCA ;
/* ************************************************************** */
/* BEGIN MAINLINE ...                                             */
/* ************************************************************** */

/*
     ****************************************************************
     * Call IDMSIN01 to deactivate the DML trace or SQL trace
     * which was originally activated by the corresponding
     * SYSIDMS parm (DMLTRACE=ON or SQLTRACE=ON).
     *
     * Parm 1 is the address of the RPB.
     * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
     ****************************************************************
*/
  REQUEST_CODE = IN01_FN_NOTRACE;
  CALL IDMSIN01 ( RPB,
                  REQ_WK);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;
/*
     ****************************************************************
     * Call IDMSIN01 to request a 'GETPROF' to get the user
     * profile default DBNAME, which was established by the
     * SYSIDMS parm DBNAME=xxxxxxxx when running batch, or
     * by the DCUF SET DBNAME xxxxxxxx when running under CV.
     *
     * Parm 1 is the address of the RPB.
     * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
     * Parm 3 is the address of the 8 byte GETPROF keyword.
     * Parm 4 is the address of the 32 byte GETPROF returned value.
     ****************************************************************
*/
  REQUEST_CODE = IN01_FN_GETPROF;
  WK_KEYWD = 'DBNAME';
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  WK_KEYWD,
                  WK_VALUE);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;
  WK_DBNAME = SUBSTR(WK_VALUE,1,8);
/*
     ****************************************************************
     * Call IDMSIN01 to activate Transaction Sharing for this task.
     *
```

```
     * Parm 1 is the address of the RPB.
     * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
     ****************************************************************
*/
  REQUEST_CODE = IN01_FN_TXNSON;
  CALL IDMSIN01 ( RPB,
                  REQ_WK);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;
/*
     ****************************************************************
     * Call IDMSIN01 to deactivate Transaction Sharing for this task.
     *
     * Parm 1 is the address of the RPB.
     * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
     ****************************************************************
*/
  REQUEST_CODE = IN01_FN_TXNSOFF;
  CALL IDMSIN01 ( RPB,
                  REQ_WK);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;
/*
     ****************************************************************
     * Call IDMSIN01 to request a 'SETPROF' to set the user
     * profile default SCHEMA to the value 'SYSTEM'.
     *
     * Parm 1 is the address of the RPB.
     * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
     * Parm 3 is the address of the 8 byte SETPROF keyword.
     * Parm 4 is the address of the 32 byte SETPROF value.
     ****************************************************************
*/
  REQUEST_CODE = IN01_FN_SETPROF;
  WK_KEYWD = 'SCHEMA';
  WK_VALUE = 'SYSTEM';
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  WK_KEYWD,
                  WK_VALUE);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;
/*
```

```
     ****************************************************************
     * Call IDMSIN01 to request the current USERID established
     * by the executed JCL information when running batch, or
     * by the SIGNON USER xxxxxxxx when running under CV.
     *
     * Parm 1 is the address of the RPB.
     * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
     * Parm 3 is the address of the 32 byte USERID returned value.
     ****************************************************************
*/
  REQUEST_CODE = IN01_FN_GETUSER;
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  WK_USERID);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;
  DISPLAY ('USERID is set to ' || WK_USERID);
/*
     ****************************************************************
     * Call IDMSIN01 to establish the SYSCTL DDNAME to be used
     * when running a Batch/CV job.
     *
     * Parm 1 is the address of the RPB.
     * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
     * Parm 3 is the address of the 8 byte SYSCTL DDNAME passed.
     ****************************************************************
*/
  REQUEST_CODE = IN01_FN_SYSCTL;
  WK_SYSCTL = 'SYSCTL73';
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  WK_SYSCTL);
/*
     ****************************************************************
     * Call IDMSIN01 to retrieve the current RRS context token.
     * Note: this call requires an operating mode of IDMS_DC
     * Note: use of SNAP requires an operating mode of IDMS_DC
     ****************************************************************
*/
```

```
  REQUEST_CODE = IN01_FN_RRSCTX;
  WK_RRS_FUNCTION = IN01_FN_RRSCTX_GET;
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  WK_RRS_FUNCTION,
                  WK_RRS_CONTEXT);
  IF (REQUEST_RETURN = 0)
  THEN
    SNAP TITLE (SNAP_TITLE)
         FROM (WK_RRS_CONTEXT) LENGTH (16);
  ELSE
    IF (REQUEST_RETURN = 4)
    THEN
      DISPLAY ('No RRS context active yet.');
    ELSE GO TO IN01_ERROR;
/*
     ****************************************************************
     * Call IDMSIN01 to convert WK_STRING from EBCDIC to ASCII.
     * Note: use of SNAP requires an operating mode of IDMS_DC
     ****************************************************************
*/
  REQUEST_CODE = IN01_FN_STRCONV;
  WK_STRING_FUNCTION = CONVERT_EBCDIC_TO_ASCII;
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  WK_STRING_FUNCTION,
                  WK_STRING,
                  WK_STRING_LENGTH);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;
  SNAP TITLE (SNAP_TITLE)
       FROM (WK_STRING) LENGTH (WK_STRING_LENGTH);
/*
     ****************************************************************
     * Call IDMSIN01 to convert WK_STRING from ASCII to EBCDIC.
     ****************************************************************
*/
  REQUEST_CODE = IN01_FN_STRCONV;
  WK_STRING_FUNCTION = CONVERT_ASCII_TO_EBCDIC;
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  WK_STRING_FUNCTION,
                  WK_STRING,
                  WK_STRING_LENGTH);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;
/*
```

```
     ****************************************************************
     * Call IDMSIN01 to have an 8 byte internal DATETIME stamp
     * returned as a displayable 26 character DATE/TIME display.
     *
     * Parm 1 is the address of the RPB.
     * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
     * Parm 3 is the address of the 4 byte format indicator (0).
     * Parm 4 is the address of the 8 byte internal DATETIME stamp.
     * Parm 5 is the address of the 26 byte DATE/TIME returned.
     ****************************************************************
*/
  REQUEST_CODE = IN01_FN_GETDATE;
  WK_DTS_FORMAT = 0;
  WK_CDTS = 'UNKNOWN';
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  WK_DTS_FORMAT,
                  WK_DTS,
                  WK_CDTS);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;
  DISPLAY ('THE DATE AND TIME IS --> ' || WK_CDTS);
/*
     ****************************************************************
     * Call IDMSIN01 to have the current DATE and TIME
     * returned as a displayable 26 character DATE/TIME display.
     *
     * Parm 1 is the address of the RPB.
     * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
     * Parm 3 is the address of the 4 byte format indicator (1).
     * Parm 4 is the address of the 26 byte DATE/TIME returned.
     ****************************************************************
*/
  REQUEST_CODE = IN01_FN_GETDATE;
  WK_DTS_FORMAT = 1;
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  WK_DTS_FORMAT,
                  WK_CDTS);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;
  DISPLAY ('THE DATE AND TIME IS --> ' || WK_CDTS);
/*
```

```
     ****************************************************************
     * Call IDMSIN01 to have a 26 byte external DATE/TIME display
     * returned as an 8 byte DATETIME stamp.
     *
     * Parm 1 is the address of the RPB.
     * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
     * Parm 3 is the address of the 4 byte format indicator (2).
     * Parm 4 is the address of the 26 byte DATE/TIME.
     * Parm 5 is the address of the 8 byte DATETIME stamp returned.
     ****************************************************************
*/
  REQUEST_CODE = IN01_FN_GETDATE;
  WK_DTS_FORMAT = 2;
  WK_CDTS = '1994-07-18-12.01.18.458382';
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  WK_DTS_FORMAT,
                  WK_CDTS,
                  WK_DTS);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;
/*
     ****************************************************************
     * Call IDMSIN01 to have a 8 byte external TIME display
     * returned as an 8 byte TIME stamp.
     *
     * Parm 1 is the address of the RPB.
     * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
     * Parm 3 is the address of the 4 byte format indicator (4).
     * Parm 4 is the address of the 8 byte external TIME.
     * Parm 5 is the address of the 8 byte TIME stamp returned.
     ****************************************************************
*/
  REQUEST_CODE = IN01_FN_GETDATE;
  WK_DTS_FORMAT = 4;
  WK_TIME_EXTERNAL = '13.58.11';
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  WK_DTS_FORMAT,
                  WK_TIME_EXTERNAL,
                  WK_TIME_INTERNAL);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;
/*
```

```
     ****************************************************************
     * Call IDMSIN01 to have an 8 byte internal TIME stamp
     * returned as a displayable 8 character TIME display.
     *
     * Parm 1 is the address of the RPB.
     * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
     * Parm 3 is the address of the 4 byte format indicator (3).
     * Parm 4 is the address of the 8 byte internal TIME stamp.
     * Parm 5 is the address of the 8 byte external TIME returned.
     ****************************************************************
*/
  REQUEST_CODE = IN01_FN_GETDATE;
  WK_DTS_FORMAT = 3;
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  WK_DTS_FORMAT,
                  WK_TIME_INTERNAL,
                  WK_TIME_EXTERNAL);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;
  DISPLAY ('THE EXTERNAL TIME IS --> ' || WK_TIME_EXTERNAL);
/*
     ****************************************************************
     * Call IDMSIN01 to have a 10 byte external DATE display
     * returned as an 8 byte DATE stamp.
     *
     * Parm 1 is the address of the RPB.
     * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
     * Parm 3 is the address of the 4 byte format indicator (6).
     * Parm 4 is the address of the 10 byte external DATE.
     * Parm 5 is the address of the 8 byte DATE stamp returned.
     ****************************************************************
*/
  REQUEST_CODE = IN01_FN_GETDATE;
  WK_DTS_FORMAT = 6;
  WK_DATE_EXTERNAL = '2003-03-10';
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  WK_DTS_FORMAT,
                  WK_DATE_EXTERNAL,
                  WK_DATE_INTERNAL);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;
/*
```

```
    ****************************************************************
    * Call IDMSIN01 to have an 8 byte internal DATE stamp
    * returned as a displayable 10 character DATE display.
    *
    * Parm 1 is the address of the RPB.
    * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
    * Parm 3 is the address of the 4 byte format indicator (5).
    * Parm 4 is the address of the 8 byte internal DATE stamp.
    * Parm 5 is the address of the 10 byte external DATE returned.
    ****************************************************************
*/
  REQUEST_CODE = IN01_FN_GETDATE;
  WK_DTS_FORMAT = 5;
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  WK_DTS_FORMAT,
                  WK_DATE_INTERNAL,
                  WK_DATE_EXTERNAL);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;
  DISPLAY ('THE EXTERNAL DATE IS --> ' || WK_DATE_EXTERNAL);
/*
    ****************************************************************
    * Call IDMSIN01 to retrieve SQL error messages into a user
    * buffer that will then be displayed back to the user.
    * Whats passed is the SQLCA block and a message control
    * block consisting of the following fields:
    *
    * - Maximum number of lines in user buffer.
    * - The size (width) of one line in the user buffer.
    * - The actual number of lines returned from IDMSIN01.
    * - The user buffer where the message lines are returned.
    *
    * A return code of 4 means that there were no SQL error messages.
    * A return code of 8 means that there were more SQL error lines
    * in the SQLCA than could fit into the user buffer, meaning
    * truncation has occurred.
    *
    * Parm 1 is the address of the RPB.
    * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
    * Parm 3 is the address of the SQLCA block.
    * Parm 4 is the address of the message control block.
    ****************************************************************
*/
```

```
  REQUEST_CODE = IN01_FN_GETMSG;
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  SQLCA,
                  SQLMSGB);
  IF (REQUEST_RETURN = 4)
  THEN
    DO;
    DISPLAY ('No SQL error message');
    END;
  ELSE
    IF ((REQUEST_RETURN = 0) | (REQUEST_RETURN = 8))
    THEN
      DO LINE_CNT=1 TO SQLMCNT;
        DISPLAY (SQLMLINE(LINE_CNT));
      END;
    ELSE GO TO IN01_ERROR;
/*
     ****************************************************************
     * Call IDMSIN01 to reactivate the DML trace or SQL trace
     * which was originally activated by the corresponding
     * SYSIDMS parm (DMLTRACE=ON or SQLTRACE=ON), that has
     * been deactivated earlier on in this job.
     *
     * Parm 1 is the address of the RPB.
     * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
     ****************************************************************
*/
  REQUEST_CODE = IN01_FN_TRACE;
  CALL IDMSIN01 ( RPB,
                  REQ_WK);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;

/*
     ****************************************************************
     * Call IDMSIN01 to retrieve the runtime environment information.
     *
     * Parm 1 is the address of the RPB.
     * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
     * Parm 3 is the address of the runtime environment returned
     * information.
     ****************************************************************
*/
```

```
  REQUEST_CODE = IN01_FN_ENVINFO;
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  EVBLOCK);
  IF (REQUEST_RETURN ¬= 0) THEN GO TO IN01_ERROR;
  DISPLAY ('Runtime mode is ' || EV$MODE);
  DISPLAY ('CA IDMS tape volser is ' || EV$TAPE#);
  DISPLAY ('CA IDMS release number is ' || EV$REL#);
  DISPLAY ('CA IDMS service pack  number is ' || EV$SPACK);
  DISPLAY ('DMCL name is ' || EV$DMCL);
  DISPLAY ('System node name is ' || EV$NODE);
/*

    ****************************************************************
    * Call IDMSIN01 to format dbkey stored in SUBSCHEMA_CTRL.
    *
    * Parm 1 is the address of the RPB.
    * Parm 2 is the address of the REQUEST_CODE and RETURN_CODE.
    * Parm 3 is the address of the DBKEY.
    * Parm 4 is the address of the database-key format.
    * Parm 5 is the address of output field for formatted dbkey.
    ****************************************************************
*/


  REQUEST_CODE = IN01_FN_FRMTDBK;
  CALL IDMSIN01 ( RPB,
                  REQ_WK,
                  DBKEY,
                  PAGE_INFO_DBK_FORMAT,
                  WK_DBKEY_OUTPUT);


RETURN;

IN01_ERROR:
   DISPLAY ('IDMSIN01 function' || REQUEST_CODE);
   DISPLAY ('IDMSIN01 return code ' || REQUEST_RETURN);

RETURN;
```

Chapter 4: TCP/IP API Support

This section contains the following topics:

Using TCP/IP with CA IDMS

TCP/IP is an industry standard communications protocol. In order to understand this section, you should be familiar with the terminology and base concepts of TCP/IP. Tutorials on TCP/IP can be found on the Internet by doing a search on a general search web site with keywords "TCP/IP" and "tutorial."

CA IDMS exploits TCP/IP in the following ways:

- An online application can use the TCP/IP socket program interface to communicate with another TCP/IP application, possibly on another platform.
- Remote applications can directly access a central version and start an online task.

A "communication" consists of two socket programs exchanging messages. The program that initiates a service request is the **client**. The program receiving incoming requests is the **server**.

Typically, the client communicates with one server at a time. However, a server processes requests from multiple clients. The server type depends on how the client requests are processed:

- Iterative server—Accepts a single client request, processes it and returns the result to the client and waits for the next client request.
- Concurrent server—Accepts a client requests and spawns a "child" task to process it.

CA IDMS TCP/IP functionality is available for these operating systems:

- z/OS
- z/VSE
- z/VM

Note: The following limitations are associated with the z/VSE implementation of TCP/IP:

- Domains—Only AF_INET is supported.
- Protocol—Only TCP is supported.
- Sockets—Only streaming sockets are supported.

TCP/IP Programming Support for Online Applications

TCP/IP programming support within CA IDMS allows an application to communicate through TCP/IP protocols with a second application. The second application can reside on the same platform or another platform.

The socket program interface depends upon the programming language used to write the application:

- Programs written in Assembler use the #SOCKET macro interface.
- Programs written in COBOL or PL/I use a call interface to IDMSOCKI.
- Applications written in CA ADS can use the SOCKET built-in function or the call interface to IDMSOCKI.

For sample TCP/IP programs written in these programming languages, see TCP/IP Programming Examples.

More Information:

- A TCP/IP trace facility is available to assist you in debugging socket programs. It is enabled using the DCMT VARY LTERM command. For more information, see the *CA IDMS System Tasks and Operator Commands Guide*.
- IDMSIN01 provides a string translation function to convert data from EBCDIC to ASCII format and from ASCII to EBCDIC format. For more information, see the chapter IDMSIN01 (see page 15).

Socket Macro Interface for Assembler Programs

Programs written in the Assembler language use the #SOCKET macro to exploit TCP/IP sockets. The #SOCKET macro takes the following general form:

```
label    #SOCKET function,                                  X
               RETCODE=return-code,                         X
               ERRNO=errno,                                 X
               RSNCODE=reason-code,                         X
               PLIST=parameter-list-area,                   X
               RGSV=(rgsv),                                 X
               CALL=call-value,                             X
               function-specific-parameters
```

Parameter	Description
label	Specifies the optional Assembler label.
function	Specifies the name of the function to execute. A detailed description of the supported functions can be found in Function Descriptions (see page 78).
return-code	Specifies the name of a fullword that receives the outcome of the operation. The following are the possible values: ■ 0—No error occurred. ■ -1—A socket error was encountered; the *errno* and *reason-code* fields contain more detailed information about the error.
errno	Specifies the name of a fullword that receives the ERRNO value when return-code is -1. For more information, see Return, Errno, and Reason Codes (see page 159).
reason-code	Specifies the name of a fullword that receives the reason code value when *return-code* is -1. For more information, see Return, Errno, and Reason Codes (see page 159).
parameter-list-area	Specifies the name of an area or register pointing to the area that is used to build the #SOCKET parameter list. The default is SYSPLIST. The length of the *parameter-list-area* used by the macro depends on the #SOCKET function that is called; the longest parameter-list currently needed for a #SOCKET call is 16 fullwords.
rgsv	Specifies the registers to be saved. This parameter applies only to system mode programs. The default is (R2-R8).

Parameter	Description
call-value	Indicates whether to generate the parameter list and/or execute the function. The following are the possible values: ■ YES—Generates the parameter list and executes the function. This is the default. ■ NO—Generates the parameter list, but does not execute the function. ■ ONLY—Executes the function for which a parameter list is pre-built.

Notes

- The syntax does not show Assembler column conventions (label starts in column 1; statement in column 10; continuation line in column 16; continuation character in column 72).
- On return from the #SOCKET call, R15 is always 0, except in cases of a parameter-list error where the RETCODE field cannot be found; in this case R15 is set to -1.
- The parameter values assigned to the three return code parameters (RETCODE, ERRNO and RSNCODE) and to all the function-specific-parameters can be specified in data field notation or in register notation.

  In data field notation, the program specifies the name of a variable field containing the parameter value.

  In register notation, the program specifies a register containing the address of the variable field containing the parameter value (not the value itself). General registers 2 to 15 can be used in this notation; the register reference must be enclosed in parentheses.
- Some parameters also accept a value in the form of an absolute expression. Where applicable, this is mentioned under the corresponding parameter's description.
- Some parameters from the #SOCKET macro are optional. There are two ways to address an optional parameter:
  - Omit the parameter on the #SOCKET macro call.
  - Assign a null value to the parameter. For example, HOSTNAME=NULL.

  Both ways are equivalent.
- The #SOCKET macro uses the following registers when building its parameter list:
  - R0—A work register to build the parameter list.
  - R1—Address the parameter list.
  - R14 and R15—The branch and link register for the call sequence to socket services.

- #SOCKET TCPIPDEF generates DSECTs and EQUates needed to write a TCP/IP program.
- #SOCKET ERRNOS generates all EQUates for CA IDMS specific errno values.

Assembler Structure Description

In Assembler programs, the following DSECTs can be generated by coding a #SOCKET TCPIPDEF statement:

- SOCK@IN—Describes the SOCKADDR structure for IPv4.
- SOCK@IN6—Describes the SOCKADDR structure for IPv6.
- HOSTENTD—Describes the HOSTENT structure.
- SERVENTD—Describes the SERVENT structure.
- TIMEVAL—Describes the TIMEVAL structure.
- ADDRINFO—Describes the ADDRINFO structure.

Each of these structures is described in Socket Structure Descriptions (see page 166).

The CA ADS Socket Interface

Applications written in CA ADS can use one of two methods to exploit TCP/IP sockets:

- Using a CA ADS system-supplied built-in function, SOCKET. It follows the same general rules as other CA ADS built-in functions. The following is an example of the code required to invoke the SOCKET built-in function in your CA ADS dialog:

```
SOCKET(function,
       return-code,
       errno,
       reason-code,
       function-dependent-parameter1,
       ...)
```

  Parameters can be records or record elements.

- Using a CA ADS control statement to invoke the socket call interface, IDMSOCKI. IDMSOCKI is the same socket call interface that can be used with COBOL programs. In this scenario, the LINK control statement is used to invoke IDMSOCKI:

  ```
  LINK TO PROGRAM 'IDMSOCKI' USING
        (function,
         return-code,
         errno,
         reason-code,
         function-dependent-parameter1,
         ...)
  ```

  Each parameter must be a separate record.

For both methods, the first four parameters are identical except that if linking to IDMSOCKI, each parameter must be defined as a record whose first element is a field described in the following table. If using the SOCKET built-in function the parameters can be records or record elements.

Parameter	Description
function	Specifies a 4-byte, fullword-aligned integer field that the program sets to the desired socket function. A detailed description of the supported functions can be found in Function Descriptions (see page 78).
return-code	Specifies a 4-byte, fullword-aligned integer field that receives the outcome of the operation. The following are the returned values: ■ 0—No errors occurred. ■ 20—A parameter list error was encountered. ■ -1—A socket error was encountered; the *errno* and *reason-code* fields contain more detailed information about the error.
errno	Specifies a 4-byte, fullword-aligned integer field that receives the ERRNO value when *return-code* is -1. For more information, see Return, Errno, and Reason Codes (see page 159).
reason-code	Specifies a 4-byte, fullword-aligned integer field that receives the reason code value when *return-code* is -1. For more information, see Return, Errno, and Reason Codes (see page 159).

Depending on the function, zero or more parameters can follow.

Comparing IDMSOCKI and SOCKET

While either of these methods allows you to utilize the TCP/IP API functionality, there are benefits to using the SOCKET built-in function:

- Parameters can be a record element. When IDMSOCKI is used, each parameter must be defined as a separate record.
- It is easier to use.
- It provides optimum performance. Calling a system-defined built-in function is more efficient than LINKing to another program type.
- You can use the system-defined record SOCKET-CALL-INTERFACE, which contains the definition of the first four parameters. To use this record, add it to the dialog as a work record.
- SOCKET supports omitted parameters.

Because of these advantages, use of the SOCKET built-in function is recommended.

Notes

- To omit an optional parameter in the parameter list, replace the parameter with the @ symbol.
- A CA ADS dialog associated with a server task (a task started by a generic listener):
  - Must be a mapless dialog.
  - Should include SOCKET-LISTENER-PARMS as a work record.
- The following pre-defined records are provided during installation and can be attached to a dialog as work records:
  - SOCKET-CALL-INTERFACE—Describes the socket functions, return codes, and errno values used to issue all socket requests.
  - SOCKET-MISC-DEFINITIONS—Describes options and flags specific to individual functions.
  - SOCKET-MISC-DEFINITIONS-2—Describes the flags specific to the IOCTL function.
  - SOCKET-SOCKADDR-IN, SOCKET-SOCKADDR-IN6, SOCKET-HOSTENT, SOCKET-SERVENT, SOCKET-TIMEVAL, and SOCKET-ADDRINFO—Describe structures that may be useful for certain socket applications.

- The SOCKET-CALL-INTERFACE record contains fields that can be used for SOCKET built-in function common parameters:
  - *function*
  - *return-code*
  - *errno*
  - *reason-code*

  Each supported function is represented by a field, whose value is the function number. The following example illustrates how to issue a READ socket request using the SOCKET built-in function and fields within the SOCKET-CALL-INTERFACE record:

```
IF (SOCKET (SOCKET-FUNCTION-READ,
            SOCKET-RETCD,
            SOCKET-ERRNO,
            SOCKET-RESNCD,. . .) = 0)
```

- For more information about CA ADS and built-in functions, see the *CA ADS Reference Guide*.

CA ADS Structure Description

The following records are installed to describe structures related to SOCKET processing:

- SOCKET-SOCKADDR-IN—Describes the SOCKADDR structure for IPv4.
- SOCKET-SOCKADDR-IN6—Describes the SOCKADDR structure for IPv6.
- SOCKET-HOSTENT—Describes the HOSTENT structure.
- SOCKET-SERVENT—Describes the SERVENT structure.
- SOCKET-TIMEVAL—Describes the TIMEVAL structure.
- SOCKET-ADDRINFO—Describes the ADDRINFO structure.

Each of these structures is described in Socket Structure Descriptions (see page 166).

Socket Call Interface for COBOL

Programs written in COBOL use the CALL statement to exploit TCP/IP sockets:

```
CALL 'IDMSOCKI' USING
         function,
         return-code,
         errno,
         reason-code,
         function-dependent-parameter1,
           . . .
```

A call to IDMSOCKI must pass the following four parameters:

Parameter	Description
function	Specifies a 4-byte, fullword-aligned, integer field that the program sets to the desired socket function. The following is the sample definition of a *function* field: 01 SOCKET-FUNCTION PIC S9(8) COMP. A detailed description of the supported functions can be found in Function Descriptions (see page 78).
return-code	Specifies a 4-byte, fullword-aligned, integer field that receives the outcome of the operation. The following are the returned values: ■ 0—No errors occurred. ■ 20—A parameter list error was encountered. ■ -1—A socket error was encountered; the *errno* and *reason-code* fields contain more detailed information about the error. The following is the sample definition of a *return-code* field: 01 SOCKET-RETCD PIC S9(8) COMP.
errno	Specifies a 4-byte, fullword-aligned, integer field that receives the ERRNO value when *return-code* is -1. The following is the sample definition of an *errno* field: 01 SOCKET-ERRNO PIC S9(8) COMP. For more information, see Return, Errno, and Reason Codes (see page 159).
reason-code	Specifies a 4-byte, fullword-aligned, integer field that receives the reason code value when *return-code* is -1. The following is the example definition of a *reason-code* field: 01 SOCKET-RSNCD PIC S9(8) COMP. For more information, see Return, Errno, and Reason Codes (see page 159).

Depending on the function, zero or more parameters can follow.

Notes

- If an optional parameter is not specified in the parameter list, it should be replaced by a parameter that depends on the COBOL compiler:
  - For COBOL for z/OS, specify reserved keyword OMITTED.
  - For ANSI COBOL85, specify BY VALUE dummy-variable; dummy-variable should be set to 0.

- The following pre-defined records are provided during installation to assist in writing socket applications:
  - SOCKET-CALL-INTERFACE—Describes the socket functions, return codes, and errno values used to issue all socket requests.
  - SOCKET-MISC-DEFINITIONS—Describes options and flags specific to individual functions.
  - SOCKET-MISC-DEFINITIONS-2—Describes the flags specific to the IOCTL function.
  - SOCKET-SOCKADDR-IN, SOCKET-SOCKADDR-IN6, SOCKET-HOSTENT, SOCKET-SERVENT, SOCKET-TIMEVAL, and SOCKET-ADDRINFO—Describe structures that may be useful for certain socket applications.
- The SOCKET-CALL-INTERFACE record contains fields that can be used for the socket call common parameters:
  - *function*
  - *return-code*
  - *errno*
  - *reason-code*

  Each supported function is represented by a field, whose value is the function number. The following example illustrates how to issue a READ socket request using the fields within the SOCKET-CALL-INTERFACE record:

```
CALL 'IDMSOCKI' USING SOCKET-FUNCTION-READ,
                      SOCKET-RETCD,
                      SOCKET-ERRNO,
                      SOCKET-RESNCD,
                      . . .
```

  Note: The SOCKET-CALL-INT record is identical to theSOCKET-CALL-INTERFACE record except that functions values are defined as condition names instead of fields. Unless storage is critical, the SOCKET-CALL-INTERFACE record should be used.
- The program associated with a server task (a task started by a generic listener) must specify the information in the following sections:
  - In the LINKAGE SECTION:

```
01  SOCKET-PARMS        PIC X(80).
01  SOCKET-DESCRIPTOR   PIC S9(8) COMP.
01  SOCKET-RESUME-COUNT PIC S9(8) COMP.
```

  - In the PROCEDURE DIVISION:

```
PROCEDURE DIVISION USING
          SOCKET-PARMS,
          SOCKET-DESCRIPTOR,
          SOCKET-RESUME-COUNT.
```

COBOL Structure Description

The following records are installed to describe structures related to SOCKET processing:

- SOCKET-SOCKADDR-IN—Describes the SOCKADDR structure for IPv4.
- SOCKET-SOCKADDR-IN6—Describes the SOCKADDR structure for IPv6.
- SOCKET-HOSTENT—Describes the HOSTENT structure.
- SOCKET-SERVENT—Describes the SERVENT structure.
- SOCKET-TIMEVAL—Describes the TIMEVAL structure.
- SOCKET-ADDRINFO—Describes the ADDRINFO structure.

Each of these structures is described in Socket Structure Descriptions (see page 166).

Socket Call Interface for PL/I

Programs written in PL/I use the CALL statement to exploit TCP/IP sockets:

```
CALL IDMSOCKI
       (function,
       return_code,
       errno,
       reason_code,
       function_dependent_parameter1,
       ...);
```

A call to IDMSOCKI must pass the following first four parameters:

Parameter	Description
function	Specifies a 4-byte, fullword-aligned, integer field that the program sets to the desired socket function. The following is the sample definition of a *function* field: DCL SOCKET_FUNCTION FIXED BINARY(31); A detailed description of the supported functions can be found in Function Descriptions (see page 78).

Parameter	Description
return_code	Specifies a 4-byte, fullword-aligned, integer field that receives the outcome of the operation. The following are the returned values: ■ 0—No errors occurred. ■ 20—A parameter list error was encountered. ■ -1—A socket error was encountered; the *errno* and *reason_code* fields contain more detailed information about the error. The following is the sample definition of a *return_code* field: DCL SOCKET_RETCD FIXED BINARY(31);
errno	Specifies a 4-byte, fullword-aligned, integer field that receives the ERRNO value when *return_code* is -1. The following is the sample definition of an *errno* field: DCL SOCKET_ERRNO FIXED BINARY(31); For more information, see Return, Errno, and Reason Codes (see page 159).
reason_code	Specifies a 4-byte, fullword-aligned, integer field that receives the reason code value when *return_code* is -1. The following is the sample definition of a *reason_code* field: DCL SOCKET_RSNCD FIXED BINARY(31); For more information, see Return, Errno, and Reason Codes (see page 159).

Depending on the function, zero or more parameters can follow.

Notes

- Some PL/I compilers limit the length of an external name to 7 characters. Since IDMSOCKI contains 8 characters, this can lead to errors at compile time. These errors can be solved in the following ways:
  - Use the compile option LIMITS(EXTNAME(8)).
  - Use entry point IDMSOCK, which is defined as a synonym to IDMSOCKI.
- If an optional parameter is not to be specified in the parameter list, replace it by an asterisk (*).
- The following pre-defined records are provided during installation to assist in writing socket applications:
  - SOCKET_CALL_INTERFACE—Describes the socket functions, return codes and errno values used to issue all socket requests.
  - SOCKET_MISC_DEFINITIONS—Describes options and flags specific to individual functions.

- SOCKET_MISC_DEFINITIONS_2—Describes the flags specific to the IOCTL function.
- SOCKET_SOCKADDR_IN, SOCKET_SOCKADDR_IN6, SOCKET_HOSTENT, SOCKET_SERVENT, SOCKET_TIMEVAL, and SOCKET_ADDRINFO—Describe structures that may be useful for certain socket applications.

Note: Some of these records contain condition names. To generate the appropriate declare statements, specify the following pre-compiler option:

```
EXPAND88=YES
```

- The SOCKET_CALL_INTERFACE record contains fields that can be used for socket call common parameters:
  - *function*
  - *return_code*
  - *errno*
  - *reason_code*

  Each supported function is represented by a field whose value is the function number. The following example illustrates how to issue a READ socket request using the fields within the SOCKET_CALL_INTERFACE record:

```
CALL 'IDMSOCKI' USING (SOCKET_FUNCTION_READ,
                       SOCKET_RETCD,
                       SOCKET_ERRNO,
                       SOCKET_RESNCD,
                       . . .);
```

  Note: The SOCKET_CALL_INT record is identical to the SOCKET_CALL_INTERFACE record except that functions values are defined as condition names instead of fields. Unless storage is critical, the SOCKET_CALL_INTERFACE record should be used.

- The program associated with a server task (a task started by a generic listener) must specify the following:

```
PROCEDURE (P1, P2, P3)
  OPTIONS (REENTRANT,FETCHABLE);

DCL (P1,P2,P3)          POINTER;
DCL SOCKET_PARMS        CHAR(80)          BASED (ADDR(P1));
DCL SOCKET_DESCRIPTOR   FIXED BINARY(31)  BASED (ADDR(P2));
DCL SOCKET_RESUME_COUNT FIXED BINARY(31)  BASED (ADDR(P3));
```

PL/I Structure Description

The following records are installed to describe structures related to SOCKET processing:

- SOCKET_SOCKADDR_IN—Describes the SOCKADDR structure for IPv4.
- SOCKET_SOCKADDR_IN6—Describes the SOCKADDR structure for IPv6.
- SOCKET_HOSTENT—Describes the HOSTENT structure.
- SOCKET_SERVENT—Describes the SERVENT structure.
- SOCKET_TIMEVAL—Describes the TIMEVAL structure.
- SOCKET_ADDRINFO—Describes the ADDRINFO structure.

Each of these structures is described in Socket Structure Descriptions (see page 166).

Generic Listener Service

The generic listener service facilitates the implementation of concurrent servers quickly and easily. Generic listening performs the following tasks:

- Creates a stream socket on a given port, optionally on a specific TCP/IP stack.
- Listens on the socket.
- Accepts connection requests, acquires a PTERM/LTERM pair and attaches a server task on it. This continues until the service is stopped.
- Waits for input on the socket if a server task ends normally without closing its socket. This allows implementation of suspend/resume processing, which is useful when a client application wants to keep the connection alive without tying up a CA IDMS/DC task. Whenever the client application is ready to proceed, it sends another message over the connection. When the generic listener service receives this message it attaches a new server task on the same PTERM/LTERM pair. The task code that is invoked on a resume can be specified in the prior task by using the NEXT TASK clause of the DC RETURN statement. If the next task code is not set, the task code specified in the listener PTERM definition is invoked.

Implementation

Generic listening is a service provided by the SOCKET line driver. The parameters that control the listener service are defined using the following methods:

- A listener PTERM: it defines the port on which to listen, the backlog, the task code to invoke when a connection is established and the mode in which to invoke the task. Optionally, if running on a multi-homed host, the TCP/IP stack can be selected. Also optionally, a character string can be defined to pass to the attached task.
- A task and associated program definition.

  Note: The task and program should be defined to the security system so that anyone can execute them.

The program associated with the server task receives control with a parameter list containing the following addresses:

- The address of an 80-byte character string set to the value of the string specified in the listener PTERM definition or blanks if none was specified.
- The address of the socket descriptor.
- The address of a 4-byte field named the resume counter. The resume counter is provided for suspend/resume processing.

Notes:

- If the listener program is written in CA ADS the parameters are passed in the SOCKET-LISTENER-PARMS record. This record must be included as work record in the dialog definition.
- If MODE IS SYSTEM is specified in the LISTENER PTERM definition, the listener program must be written using DC/UCF calling convention conventions as described in *CA IDMS System Operations Guide*.

The program associated with the server task responds to the message sent from the client application. In addition to performing the required business function, it is also responsible for the following services:

- Security—When the program receives control, no user has been signed onto the system. For security purposes, the executing program must immediately signon to the system. To provide signon capabilities you must link to the RHDCSNON program. Or, for Assembler programs, you can code a #SECSGON macro.

  More Information:

  For more information about linking to RHDCSNON, see the *CA IDMS System Tasks and Operator Commands Guide*. For more information about #SECSGON, see the *CA IDMS Security Administration Guide*.

- Character conversion—If the remote host sends text messages in a character set other than the one used on the central version, these text messages might need translation. The program is responsible for performing this translation and IDMSIN01 functions are provided to assist in this process.
- Closing the socket—Once the conversation is over, the socket should be closed. Closing the socket causes a sign off when the task terminates. If the task ends normally without closing the socket, generic listening starts a "receive" on the socket because it interprets this situation as a suspend. As a result, the LTERM/PTERM pair remains in use and long-term resources, such as the signon element, remain allocated. These resources are subject to CA IDMS time-out processing and can be deleted with the DCMT VARY LTERM ... RESOURCE DELETE command.

  Note: If the task abends, CA IDMS closes the socket and the PTERM/LTERMpair is signed off automatically.

More Information:

For more information about implementing the generic listening service required for TCP/IP integration, see the *CA IDMS System Generation Guide*.

Application Design Considerations

The TCP/IP socket program interface is available only to CA IDMS/DC applications running under a central version. A batch program trying to use the interface receives a socket return code of RNOSLIND.

Server tasks started by a generic listener cannot do any terminal I/O such as #LINEIN, #LINEOUT, #TREQ and so on. If written in CA ADS, they should be mapless dialogs.

Using Stream Sockets

TCP allows for arbitrary amounts of data to be sent and received over a stream socket. Because a stream is interpreted as a sequence of bits, TCP cannot identify the organization, content or amount of data being processed. Therefore, a TCP application should use its own protocol to logically divide a stream into messages. The most common way of doing this is to prefix the data with the data length.

Receiving Data

TCP determines whether to break a block of data into pieces and transmit each piece separately or to accumulate data in its buffer and send it in one block. However, the receiving application can receive the data of multiple send requests in a single receive. TCP receives data until the expected message is completely received.

Sending Data

As with receiving data, there is no guarantee that a send request is completely serviced. TCP determines if the amount of data in the send request is too large. If so, it returns the amount of data already processed and the application must re-issue the send with an updated data length and buffer pointer. TCP sends data until the message is completely sent.

TCP/IP Coding Samples

The CA IDMS installation media contains these sample programs, which are intended for demonstration purposes only:

- TCPASM01—An Assembler program that tests the #SOCKET API calls. TCPASM01 can be invoked in one of two ways:
  - As a user task code at the "ENTER NEXT TASK CODE" prompt. Depending on the command line parameters, a client or server program is initiated. The program converses with a partner program using SEND and RECV calls. If no parameters are specified, a HELP screen containing the full syntax and its options is displayed.
  - As a server program started by a listener PTE.

    Note: The listener's PTERM definition should specify MODE is USER.
- TCPADS01—A TCP/IP client program written in CA ADS.
- TCPCOB01—A TCP/IP generic listener server program written in COBOL.
- TCPPLI01—A TCP/IP generic listener server program written in PL/I.

TCPADS01 and TCPASM01 (as a client) provide the same functionality: they connect to a port number that matches a port number of a generic listener PTERM. TCPPLI01, TCPCOB01, and TCPASM01 (as a server) can be invoked by the task code associated with the generic listener PTERM.

Note: The header section of each sample program contains compiler option information that is required to successfully compile the program.

Miscellaneous TCP/IP Considerations

Using the TCP/IP Trace Facility

To help debug socket programs, a TCP/IP trace facility is available. It is activated using the DCMT VARY LTERM command.

More Information:

For more information about this command, see the *CA IDMS System Tasks and Operator Commands Guide*.

Using Multiple TCP/IP Stacks

In a multiple TCP/IP stack environment, a CA IDMS system is able to use several available TCP/IP stacks concurrently. Only z/OS and z/VM support multiple TCP/IP stacks.

In the z/OS environment, the Common INET (CINET) configuration is required to run multiple TCP/IP stacks concurrently. CA IDMS uses special system calls to get a list of the available TCP/IP stacks in the system. For more information about the CINET feature, see the IBM's *z/OS Communication Server IP Configuration Guide*.

For z/VM, multiple TCP/IP stacks are implemented by starting each stack in its own virtual machine.

Limit the TCP/IP stacks available in a CA IDMS system

In a multiple TCP/IP stack environment, you can control or limit the stacks that are usable by the socket applications running in the CA IDMS system. This enhancement is primarily for CA IDMS systems running on z/OS with CINET active and on z/VM. It is useful in an environment where certain applications need to use secured sockets or some TCP/IP stacks are for testing only.

You can control or limit the TCP/IP stacks using the following methods:

- At startup, through the INCLUDE STACK or EXCLUDE STACK clause from the TCP/IP system generation statement
- At startup, through the INCLUDE_TCP/IP_STACK or EXCLUDE_TCP/IP_STACK SYSIDMS parameters
- Dynamically, through the DCMT VARY TCP/IP command

Note: On z/VM, the definition of the TCP/IP stacks to use through the SYSTCPD file and the 8 SYSIDMS parameters (TCP/IP_STACK_1 -> TCP/IP_STACK_8) as used in r16 is still available and is kept for compatibility reasons.

More Information:

- For more information about the TCP/IP system generation statement, see the *CA IDMS System Generation Guide*.
- For more information about the SYSIDMS parameters, see the *CA IDMS Common Facilities Guide*.
- For more information about the DCMT VARY TCP/IP command, see the *CA IDMS System Tasks and Operator Commands Guide*.
- For more information about the SYSTCPD file, see the *CA IDMS System Operations Guide*.

Default TCP/IP stack

In a multiple stacks environment, the assignment of the default stack also depends on the operating system where the CA IDMS system is running as follows:

- On z/OS, the system always assigns a specific stack as the default stack. If TCP/IP stacks are excluded from the system list, either using SYSGEN or using SYSIDMS parameters, then the default stack for the socket environment will be assigned to the following:
  - the default stack from the system, if it has not been explicitly excluded.
  - the first active stack from the resulting list, if the default stack from the system has been excluded
- On z/VM, the default stack will always be the first stack from the resulting list of stacks.

There is a possibility to overwrite this default stack assignment in the system:

- At startup, through the DEFAULT STACK clause from the TCP/IP system generation statement
- Dynamically, through the DCMT VARY TCP/IP command.

Current TCP/IP stack for a DC task

When a DC task is started, the current TCP/IP stack for the DC task is the default TCP/IP stack from the CA IDMS system. The SETSTACK function can be used to assign another value to the current TCP/IP stack or to restore the default value for the DC task.

Stack affinity

This concept refers to sockets. When a socket is created and it is exclusively attached to a specific TCP/IP stack, it is said to have "stack affinity." The stack affinity is equal to the value of the current TCP/IP stack when the socket was created. A socket that is not attached to a specific TCP/IP stack has no stack affinity.

Default stack affinity

When a socket is created in the default DC task environment, that is, no specific SETSTACK calls have been issued in the task yet, the default stack affinity is the default TCP/IP stack.

When a socket has set stack affinity to *ALL and the application issues an ACCEPT socket call with the IP address equal to INADDR_ANY in the corresponding socket address structure, then the ACCEPT request is propagated to all available TCP/IP stacks, and therefore the application can accept connections from clients specifying an IP address from any of the available TCP/IP stacks. This configuration is possible on z/OS only. If the accepting socket is assigned a specific stack affinity, the client must specify the IP address corresponding to that specific stack.

If a socket application is accepting connections from all the TCP/IP stacks available in the system, that is its current stack affinity is set to *ALL, and if some of the TCP/IP stacks have been excluded by the user, then the current processing for the #SOCKET ACCEPT socket function will reject all connections that were explicitly addressed to one of the excluded stacks. This processing is done internally. The user applications do not require any changes. The output of the DCMT DISPLAY TCP/IP STATISTICS shows the number of accepted connections that have been rejected for that reason.

Two socket functions return values that are influenced by which TCP/IP stack is current on the DC task.

- GETHOSTID—Returns the IP address of the current TCP/IP stack.
- GETHOSTNAME—Returns the hostname of the current TCP/IP stack.

The output from the DCMT DISPLAY TCP/IP STACK TABLE command shows the available TCP/IP stacks with their associated IP address and hostname.

More Information:

For more information about DCMT DISPLAY TCP/IP, see the *CA IDMS System Tasks and Operator Commands Guide*.

Associating Time-outs to Sockets

In the standard POSIX socket interface, time-out conditions can only be detected through the use of the SELECT socket function. The socket interface provided by CA IDMS offers an extension that assigns a time-out value to each socket that created in the DC/UCF environment. The FCNTL socket function enables you to specify or retrieve a socket's time-out value.

When a socket is created, a default time-out value is assigned. The default time out value depends on the type of socket:

- For a socket created by the SOCKET function, or "client socket," the default time-out value is set to the corresponding DC task's INACTIVE INTERVAL parameter.
- For a socket created by the ACCEPT function, or "serversocket," the default time-out value is set to the corresponding DC task's EXTERNAL WAIT parameter.

The following socket functions check the time-out value at runtime:

- ACCEPT
- CONNECT
- READ
- RECV
- RECVFROM
- SEND
- SENDTO
- WRITE

When a time-out condition occurs, the socket function returns a ETIMEDOUT errno code to the application.

More Information:

For more information about the FCNTL socket function, see FCNTL (see page 82). For more information about the INACTIVE INTERVAL or EXTERNAL WAIT parameters of the TASK statement, see the *CA IDMS System Generation Guide*.

Function Descriptions

This section describes the socket functions that are supported by CA IDMS. The following information is provided for each function:

- An Assembler #SOCKET macro invocation showing all of the parameters that can be specified.
- A list of parameters that can be passed when invoking the function in COBOL, PL/I, and CA ADS. The first of these parameters is the function name as defined in the SOCKET-CALL-INTERFACE record.
- A description of the function-dependent parameters.
- Additional notes if applicable to a specific function.

ACCEPT

ACCEPT accepts the first connection request on the queue of pending connection requests. If the queue is empty, the call waits until the first connection request arrives or fails with an EWOULDBLOCK condition if the socket had been marked as non-blocking. If successful, a new socket descriptor is returned.

Assembler

```
label    #SOCKET ACCEPT,
                 RETCODE=return-code,
                 ERRNO=errno,
                 RSNCODE=reason-code,
                 SOCK=socket-descriptor,
                 SOCKADDR=sockaddr,
                 SOCKADDL=sockaddr-length,
                 NEWSOCK=new-socket-descriptor,
                 PLIST=parameter-list-area,
                 RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-ACCEPT,
return-code,
errno,
reason-code,
socket-descriptor,
sockaddr,
sockaddr-length,
new-socket-descriptor
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor that was used on the BIND and LISTEN functions.
sockaddr	Specifies the name of an area in which to return the sockaddr structure of the connecting client. The format of that structure depends on the domain of the corresponding socket. This parameter can be assigned to NULL if the caller is not interested in the connector's address.
sockaddr-length	Specifies the name of a fullword field containing the length of *sockaddr*. If SOCKADDR is assigned to NULL, *sockaddr-length* must be 0. On return, *sockaddr-length* contains the size required to represent the connecting socket. If the value is 0, the contents of *sockaddr* are unchanged. If the *sockaddr* is too small to contain the full sockaddr structure, it is truncated. The maximum value for this parameter is 4096.
new-socket-descriptor	Specifies the name of a fullword field where the socket descriptor of the new connection is returned.

Notes

When the time-out value associated with the socket expires, the socket function terminates with the ETIMEDOUT errno code. For more information about socket time-outs, see Associating Time-outs to Sockets (see page 77).

BIND

BIND assigns a local name to an unnamed socket.

Assembler

```
label    #SOCKET BIND,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               SOCKADDR=sockaddr,
               SOCKADDL=sockaddr-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-BIND,
return-code,
errno,
reason-code,
socket-descriptor,
sockaddr,
sockaddr-lengt
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor to bind.
sockaddr	Specifies the name of an area that contains the sockaddr structure to be bound to the socket. The format of the sockaddr structure depends on the domain of the corresponding socket. **z/VSE systems:** Only the domain AF_INET is supported.
sockaddr-length	Specifies the name of a fullword field containing the length of sockaddr. *sockaddr-length* can be specified as an absolute expression. The maximum value for this parameter is domain dependent. If the domain is AF_INET, it is the length of the SOCKET-SOCKADDR-IN record (SIN#LEN for Assembler). If the domain is AF_INET6, it is the length of the SOCKET-SOCKADDR-IN6 record (SIN6#LEN for Assembler).

CLOSE

CLOSE deletes the socket descriptor from the internal descriptor table maintained for the application program and terminates the existence of the communications endpoint. If the socket was connected, the connection is terminated in an orderly fashion.

Assembler

```
label    #SOCKET CLOSE,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-CLOSE,
return-code,
errno,
reason-code,
socket-descriptor
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor to close.

CONNECT

CONNECT initiates a connection on a socket.

Assembler

```
label    #SOCKET CONNECT,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               SOCKADDR=sockaddr,
               SOCKADDL=sockaddr-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-CONNECT,
return-code,
errno,
reason-code,
socket-descriptor,
sockaddr,
sockaddr-length
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword containing the socket descriptor to which to connect.
sockaddr	Specifies the name of an area that contains the sockaddr structure to which to connect. The format of the sockaddr structure depends on the domain of the corresponding socket. **z/VSE systems:** Only the domain AF_INET is supported. Specify family *AF@INET* when building the sockaddr structure.
sockaddr-length	Specifies the name of a fullword field containing the length of *sockaddr*. *sockaddr-length* can be specified as an absolute expression. The maximum value for this parameter is domain dependent. If the domain is AF_INET, it is the length of the SOCKET-SOCKADDR-IN record (SIN#LEN for Assembler). If the domain is AF_INET6, it is the length of the SOCKET-SOCKADDR-IN6 record (SIN6#LEN for Assembler).

Notes

- When the time-out value associated with the socket expires, the socket function terminates with the ETIMEDOUT errno code. For more information about socket time-outs, see Associating Time-outs to Sockets (see page 77).
- After a CONNECT error, including a time-out condition, the corresponding socket cannot be used. For the application to continue processing, it must close the current socket and create a new socket.

FCNTL

FCNTL provides control over a socket descriptor. Depending on the command, it retrieves or sets control information.

Assembler

```
label    #SOCKET FCNTL,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               COMMAND=command,
               ARGUMENT=argument,
               RETVAL=returned-value,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-FCNTL,
return-code,
errno,
reason-code,
socket-descriptor,
command,
argument,
returned-value
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor to process.
command	Specifies the name of a fullword field containing the command to perform on the socket. *command* can be specified as an absolute expression.
argument	Specifies the name of a fullword field containing the argument that applies to some commands. *argument* can be specified as an absolute expression. While *argument* is optional, it must be specified for setting functions.
returned-value	Specifies the name of a fullword field that contains the returned information from any retrieval commands. While *returned-value* is optional, it must be specified for retrieval function.

Notes

- **z/VSE systems:** The F@GETFL and F@SETFL commands are not supported.
- The following table lists the commands and arguments that can be specified. The EQUate symbol is generated by #SOCKET macro and the field names are associated with the SOCKET-MISC-DEFINITIONS record.

EQUate Symbol	Field Name	Description
F@GETFL	SOCKET-FCNTL-GET	Gets file status command
F@SETFL	SOCKET-FCNTL-SET	Sets file status command
F@GETIMO	SOCKET-FCNTL-GETIMO	Gets time-out value associated with a socket
F@SETIMO*	SOCKET-FCNTL-SETIMO	Associates a time-out value with a socket
NONBLOCK	SOCKET-FCNTL-NONBLOCK	Sets socket in non-blocking mode

EQUate Symbol	Field Name	Description
* Acceptable argument values for the F@SETIMO command: ■ 1 through 32767—The time-out value in seconds ■ 0—No time-out processing is wanted, but a task abend ■ -1—Indefinite wait (equivalent for FOREVER)		

PL/I programs: The SOCKET_MISC_DEFINITIONS is used and the dashes are replaced by underscores.

FD_CLR

FD_CLR clears a socket descriptor's bit in a bit list.

Assembler

```
label    #SOCKET FD_CLR,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               BITLIST=bit-list,
               BITLISTL=bit-list-length,
               BITORDER=bit-order,
               PLIST=parameter-list-area
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor whose bit must be cleared (set to zero) in the bit list.
bit-list	Specifies the name of the area containing the bit list.
bit-list-length	Specifies the name of a fullword field containing the length of the bit-list in bytes. *bit-list-length* can be specified as an absolute expression. *bit-list-length* must be a multiple of 4.

Parameter	Description
bit-order	Specifies the name of the fullword containing the order in which the bits are addressed in the bit list. This order should be the same as the value specified on the *option* parameter of the SELECT or SELECTX function. *bit-order* can be specified as an absolute expression. The following are the accepted values: ■ SEL@BBKW (default) ■ SEL@BFWD

Notes

- This function is only available to the Assembler interface.
- For performance reasons, FD_CLR does not call the RHDCSOCK processor to execute the function. Instead, the corresponding code is expanded in your program. This code is substantial, so it is best to code the function call in a subroutine.

FD_ISSET

FD_ISSET tests a socket descriptor's bit in a bit list to see if it is ON or OFF.

Assembler

```
label    #SOCKET FD_ISSET,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               BITLIST=bit-list,
               BITLISTL=bit-list-length,
               BITORDER=bit-order,
               RETVAL=returned-bit-status,
               PLIST=parameter-list-area
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor whose bit needs testing in the bit list.
bit-list	Specifies the name of the area containing the bit list.

Parameter	Description
bit-list-length	Specifies the name of a fullword field containing the length of the bit-list in bytes. *bit-list-length* can be specified as an absolute expression. *bit-list-length* must be a multiple of 4.
bit-order	Specifies the name of a fullword containing the order in which the bits are addressed in the bit list. This order should be the same as the value specified on the *option* parameter of the SELECT or SELECTX function. *bit-order* can be specified as an absolute expression. The following are the accepted values: ■ SEL@BBKW (default) ■ SEL@BFWD
returned-bit-status	Specifies the name of a fullword field that will contain the status of the tested bit: ■ 0—OFF ■ 1—ON

Notes

- This function is only available to the Assembler interface.
- For performance reasons, FD_ISSET does not call the RHDCSOCK processor to execute the function. Instead, the corresponding code is expanded in your program. This code is substantial, so it is best to code the function call in a subroutine.

FD_SET

FD_SET sets a socket descriptor's bit in a bit list ON.

Assembler

```
label    #SOCKET FD_SET,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               BITLIST=bit-list,
               BITLISTL=bit-list-length,
               BITORDER=bit-order,
               PLIST=parameter-list-area
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor whose bit must be set ON in the bit list.
bit-list	Specifies the name of the area containing the bit list.
bit-list-length	Specifies the name of a fullword field containing the length of the bit-list in bytes. *bit-list-length* can be specified as an absolute expression. *bit-list-length* must be a multiple of 4.
bit-order	Specifies the name of the fullword containing the order in which the bits are addressed in the bit list. This order should be the same as the value specified on the *option* parameter of the SELECT or SELECTX function. *bit-order* can be specified as an absolute expression. The following are the accepted values: ■ SEL@BBKW (default) ■ SEL@BFWD

Notes

- This function is only available to the Assembler interface.
- For performance reasons, FD_SET does not call the RHDCSOCK processor to execute the function. Instead, the corresponding code is expanded in your program. This code is substantial, so it is best to code the function call in a subroutine.

FD_ZERO

FD_ZERO clears all bits in a bit list.

Assembler

```
label    #SOCKET FD_ZERO,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               BITLIST=bit-list,
               BITLISTL=bit-list-length,
               PLIST=parameter-list-area
```

Parameters

Parameter	Description
bit-list	Specifies the name of the area containing the bit list.
bit-list-length	Specifies the name of a fullword field containing the length of the bit-list in bytes. *bit-list-length* can be specified as an absolute expression. *bit-list-length* must be a multiple of 4.

Notes

- This function is only available to the Assembler interface.
- For performance reasons, FD_ZERO does not call the RHDCSOCK processor to execute the function. Instead, the corresponding code is expanded in your program. This code is substantial, so it is best to code the function call in a subroutine.

FREEADDRINFO

FREEADDRINFO frees the ADDRINFO structure that has been allocated by the system during the processing of a previous call to the GETADDRINFO #SOCKET function.

Assembler

```
label    #SOCKET FREEADDRINFO,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               AINFOIN=pointer-to-addrinfo-structure,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-FREEADDRINFO,
return-code,
errno,
reason-code,
pointer-to-addrinfo-structure
```

Parameters

Parameter	Description
pointer-to-addrinfo-structure	Specifies the name of a fullword field containing the address of the ADDRINFO structure to release.

Notes

- The FREEADDRINFO function is supported as of z/OS V1R4.
- The FREEADDRINFO function *is not* supported in these operating environments:
  - z/VSE
  - z/VM

GETADDRINFO

GETADDRINFO converts a host name and/or a service name into a set of socket addresses and other associated information. This information can be used to open a socket and connect to the specified service.

Assembler

```
label    #SOCKET GETADDRINFO,
                 RETCODE=return-code,
                 ERRNO=errno,
                 RSNCODE=reason-code,
                 HOSTNAME=hostname,
                 HOSTNAML=hostname-length,
                 SERVNAME=service-name,
                 SERVNAML=service-name-length,
                 AINFOIN=pointer-to-input-addrinfo-structure,
                 AINFOOUT=pointer-to-output-addrinfo-structure,
                 CANONAML=canonical-name-length,
                 PLIST=parameter-list-area,
                 RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETADDRINFO,
return-code,
errno,
reason-code,
hostname,
hostname-length,
service-name,
service-name-length,
pointer-to-input-addrinfo-structure,
pointer-to-output-addrinfo-structure,
canonical-name-length
```

Parameters

Parameter	Description
hostname	Specifies the name of an area containing the name of the host to resolve.
hostname-length	Specifies the name of a fullword field containing the length of *hostname*. *hostname-length* can be specified as an absolute expression. *hostname* and *hostname-length* are optional. If they are not specified, *service-name* and *service-name-length* must be specified. The maximum value for this parameter is 256.
service-name	Specifies the name of an area containing the name of the service.
service-name-length	Specifies the name of a fullword field containing the length of *service-name*. *service-name-length* can be specified as an absolute expression. *service-name* and *service-name-length* are optional. If they are not specified, *hostname* and *hostname-length* must be specified. The maximum value for this parameter is 32.
pointer-to-input-addrinfo-structure	Specifies the name of a fullword field containing the address of an input ADDRINFO structure. The following fields in the ADDRINFO structure can be set: flags, family, socket type, and protocol. If this pointer is assigned to NULL, it is equivalent to an ADDRINFO structure where all fields are set to 0.
pointer-to-output-addrinfo-structure	Specifies the name of a fullword field that contains the address of the output ADDRINFO structure returned by the system. This structure has to be explicitly released by the user using the FREEADDRINFO #SOCKET call.

Parameter	Description
canonical-name-length	Specifies the name of a fullword field in which the system returns the length of the canonical name. The system returns the canonical name in the first output ADDRINFO structure if *hostname* is specified and the AI_CANONNAMEOK flag is set in the input ADDRINFO structure. If the canonical name length is not needed, *canonical-name-length* can be omitted.

Notes

- For more information about the ADDRINFO structure, see Socket Structure Descriptions (see page 166).
- On z/VSE and z/VM, the GETADDRINFO function is supported by CA IDMS's internal DNS and services resolvers. For more information, see the *CA IDMS System Operations Guide*.
- The following table lists the flags that can be set or returned in the ADDRINFO structure. The EQUate symbol is generated by the #SOCKET TCPIPDEF macro call and the field names are associated with the SOCKET-MISC-DEFINITIONS record.

EQUate Symbol	Field Name	TCP Protocol Value
AI@PASSV	SOCKET-AIFLAGS-PASSIVE	AI_PASSIVE
AI@CANOK	SOCKET-AIFLAGS-CANONNAMEOK	AI_CANONNAMEOK
AI@NHOST	SOCKET-AIFLAGS-NUMERICHOST	AI_NUMERICHOST
AI@NSERV	SOCKET-AIFLAGS-NUMERICSERV	AI_NUMERICSERV
AI@V4MAP	SOCKET-AIFLAGS-V4MAPPED	AI_V4MAPPED
AI@ALL	SOCKET-AIFLAGS-ALL	AI_ALL
AI@ADDRC	SOCKET-AIFLAGS-ADDRCONFIG	AI_ADDRCONFIG

PL/I programs: The SOCKET_MISC_DEFINITIONS is used and the dashes are replaced by underscores.

GETHOSTBYADDR

GETHOSTBYADDR takes an IP address and domain and tries to resolve it through a name server. If successful, it returns the information in a HOSTENT structure.

Assembler

```
label    #SOCKET GETHOSTBYADDR,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               IPADDR=ip-address,
               IPADDRL=ip-address-length,
               DOMAIN=domain,
               HOSTENTP=hostentp,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETHOSTBYADDR,
return-code,
errno,
reason-code,
ip-address,
ip-address-length,
domain,
hostentp
```

Parameters

Parameter	Description
ip-address	Specifies the name of a fullword field containing the binary format IP address to resolve.
ip-address-length	Specifies the name of a fullword field containing the length of *ip-address*. *ip-address-length* can be specified as an absolute expression. The maximum value for this parameter is defined by IPADDR4L in Assembler and SOCKET-IPADDR4L in other languages.
domain	Specifies the name of a fullword field containing the domain. *domain* can be specified as an absolute expression. Currently, only AF_INET is supported.
hostentp	Specifies the name of a fullword field in which the system returns the address of a HOSTENT structure containing the information about the host.

Notes

- The HOSTENT structure area is allocated by the system at the CA IDMS task level, and freed at task termination. It is reused by subsequent calls to a DNS function: GETHOSTBYADDR or GETHOSTBYNAME.
- For more information about the HOSTENT structure, see Socket Structure Descriptions (see page 166).

z/VM systems: The DNS socket functions are supported by CA IDMS's internal DNS resolver. For information about configuring the DNS resolver, see the TCP/IP Considerations section of the *CA IDMS System Operations Guide*.

z/VSE systems: The DNS socket functions can be supported by CA IDMS's internal DNS resolver. If the socket functions are supported by CA IDMS, see the TCP/IP Considerations section of *CA IDMS System Operations Guide* for information about configuring the DNS resolver. If the socket functions are supported by Barnard Software Inc. or Connectivity Systems Inc., see the appropriate TCP/IP stack documentation for configuring DNS support.

GETHOSTBYNAME

GETHOSTBYNAME takes a host name and tries to resolve it through a name server. If successful, it returns the information in a HOSTENT structure.

Assembler

```
label    #SOCKET GETHOSTBYNAME,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               HOSTNAME=hostname,
               HOSTNAML=hostname-length,
               HOSTENTP=hostentp,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETHOSTBYNAME,
return-code,
errno,
reason-code,
hostname,
hostname-length,
hostentp
```

Parameters

Parameter	Description
hostname	Specifies the name of an area containing the name of the host to resolve.
hostname-length	Specifies the name of a fullword field containing the length of *hostname*. *hostname-length* can be specified as an absolute expression. The maximum value for this parameter is 256.
hostentp	Specifies the name of a fullword field where the system returns the address of a HOSTENT structure containing the information about the host.

Notes

- The HOSTENT structure area is allocated by the system at the CA IDMS task level, and freed at task termination. It is reused by subsequent calls to a DNS function: GETHOSTBYADDR or GETHOSTBYNAME.
- For more information about the HOSTENT structure, see Socket Structure Descriptions (see page 166).

z/VM systems: The DNS socket functions are supported by CA IDMS's internal DNS resolver. For information about configuring the DNS resolver, see the TCP/IP Considerations section of the *CA IDMS System Operations Guide*.

z/VSE systems: The DNS socket functions can be supported by CA IDMS's internal DNS resolver. If the socket functions are supported by CA IDMS, see the TCP/IP Considerations section of *CA IDMS System Operations Guide* for information about configuring the DNS resolver. If the socket functions are supported by Barnard Software Inc. or Connectivity Systems Inc., see the appropriate TCP/IP stack documentation for configuring DNS support.

GETHOSTID

GETHOSTID retrieves the IP address of the local host corresponding to the current TCP/IP stack.

Assembler

```
label    #SOCKET GETHOSTID,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               IPADDR=ip-address,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETHOSTID,
return-code,
errno,
reason-code,
ip-address
```

Parameters

Parameter	Description
ip-address	Specifies the name of a fullword field in which the service returns the IP address in binary format.

Notes

This service only supports IPv4.

GETHOSTNAME

GETHOSTNAME retrieves the name of the local host corresponding to the current TCP/IP stack.

Assembler

```
label    #SOCKET GETHOSTNAME,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               HOSTNAME=hostname,
               HOSTNAML=hostname-length,
               RETLEN=returned-hostname-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETHOSTNAME,
return-code,
errno,
reason-code,
hostname,
hostname-length,
returned-hostname-length
```

Parameters

Parameter	Description
hostname	Specifies the name of an area in which the service returns the host name.
hostname-length	Specifies the name of a fullword field containing the length of *hostname. hostname-length* can be specified as an absolute expression. The maximum value for this parameter is 256.
returned-hostname-length	Specifies the name of a fullword field in which the actual length of the host name is returned.

GETNAMEINFO

GETNAMEINFO resolves a socket address into a hostname and a service name.

Assembler

```
label    #SOCKET GETNAMEINFO,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCKADDR=sockaddr,
               SOCKADDL=sockaddr-length,
               SERVNAME=service-name,
               SERVNAML=service-name-length,
               RETSNAML=returned-service-name-length,
               HOSTNAME=hostname,
               HOSTNAML=hostname-length,
               RETHNAML=returned-hostname-length,
               FLAGS=flags,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETNAMEINFO,
return-code,
errno,
reason-code,
sockaddr,
sockaddr-length,
service-name,
service-name-length,
returned-service-name-length,
hostname,
hostname-length,
returned-hostname-length,
flags
```

Parameters

Parameter	Description
sockaddr	Specifies the name of the sockaddr structure containing the information that must be resolved: the domain (or socket family), the port number and the IP address.
sockaddr-length	Specifies the name of a fullword field containing the length of *sockaddr*. *sockaddr-length* can be specified as an absolute expression. The maximum value for this parameter is domain dependent. If the domain is AF_INET, it is the length of the SOCKET-SOCKADDR-IN record (SIN#LEN for Assembler). If the domain is AF_INET6, it is the length of the SOCKET-SOCKADDR-IN6 record (SIN6#LEN for Assembler).
service-name	Specifies the name of an area where the system returns the service name corresponding to the port number specified in the sockaddr structure.
service-name-length	Specifies the name of a fullword field containing the length of *service-name*. *service-name-length* can be specified as an absolute expression. The maximum value for this parameter is 4096. .row
returned-service-name-length	Specifies the name of a fullword field into which the actual length of the service name is returned. *service-name*, *service-name-length* and *returned-service-name-length* are optional; specify all three parameters, or none of them. If none of these parameters are specified, *hostname, hostname-length,* and *returned-hostname-length* must be specified.
hostname	Specifies the name of an area where the system returns the *hostname* corresponding to the IP address specified in the sockaddr structure.
hostname-length	Specifies the name of a fullword field containing the length of *hostname*. *hostname-length* can be specified as an absolute expression. The maximum value for this parameter is 4096.

Parameter	Description
returned-hostname-length	Specifies the name of a fullword field into which the length of the host name is returned. *hostname*, *hostname-length* and *returned-hostname-length* are optional; specify all three parameters, or none of them. If none of these parameters are specified, *service-name, service-name-length,* and *returned-service-name-length* must be specified.
flags	Specifies the name of a fullword field containing flags to control the resolution of the socket address.

Notes

- The GETNAMEINFO function is supported as of z/OS V1R4.
- On z/VSE and z/VM, the GETNAMEINFO function is supported by CA IDMS's internal DNS and services resolvers. For more information, see the CA IDMS *System Operations Guide*.
- The following table lists the flags that can be passed. The EQUate symbol is generated by the #SOCKET TCPIPDEF macro call and the field names are associated with the SOCKET-MISC-DEFINITIONS.

EQUate Symbol	Field Name	Description
NI@NFDQN	SOCKET-NIFLAGS-NOFQDN	Returns the node name portion only
NI@NREQD	SOCKET-NIFLAGS-NAMEREQD	Returns an error if the host is not located
NI@NHOST	SOCKET-NIFLAGS-NUMERICHOST	Returns the numeric form of the host
NI@NSERV	SOCKET-NIFLAGS-NUMERICSERV	Returns the numeric form of the server
NI@DGRAM	SOCKET-NIFLAGS-DGRAM	Specifies that the service is a datagram service

PL/I programs: The SOCKET_MISC_DEFINITIONS is used and the dashes are replaced by underscores.

GETPEERNAME

GETPEERNAME retrieves the name of the peer connected to a socket.

Assembler

```
label    #SOCKET GETPEERNAME,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               SOCKADDR=sockaddr,
               SOCKADDL=sockaddr-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETPEERNAME,
return-code,
errno,
reason-code,
socket-descriptor,
sockaddr,
sockaddr-length
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor from which to retrieve the peer name.
sockaddr	Specifies the name of an area in which to return the sockaddr structure of the peer. The format of this structure depends on the domain of the corresponding socket. This parameter can be assigned to NULL if the caller is not interested in the peer's address.
sockaddr-length	Specifies the name of a fullword field containing the length of *sockaddr*. If SOCKADDR is assigned to NULL, *sockaddr-length* must be 0. On return, *sockaddr-length* contains the size required to represent the peer. If the size of *sockaddr* is too small to contain the full sockaddr structure, it is truncated. The maximum value for this parameter is 4096.

GETSERVBYNAME

GETSERVBYNAME takes a service name and a protocol and tries to resolve them using the services file. If successful, it returns the information in a SERVENT structure.

Assembler

```
Label    #SOCKET GETSERVBYNAME,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SERVNAME=service-name,
               SERVNAML=service-name-length,
               PROTNAME=protocol-name,
               PROTNAML=protocol-name-length,
               SERVENTP=serventp,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETSERVBYNAME,
return-code,
errno,
reason-code,
service-name,
service-name-length,
protocol-name,
protocol-name-length,
serventp
```

Parameters

Parameter	Description
service-name	Specifies the name of an area containing the name of the service to resolve.
service-name-length	Specifies the name of a fullword field containing the length of *service-name*. *service-name-length* can be specified as an absolute expression. Specifies the maximum value for this parameter is 256.
protocol-name	Specifies the name of an area containing the name of the protocol to use.
protocol-name-length	Specifies the name of a fullword field containing the length of *protocol-name*. *protocol-name-length* can be specified as an absolute expression. Specifies the maximum value for this parameter is 256.

Parameter	Description
serventp	Specifies the name of a fullword field where the system returns the address of a SERVENT structure containing the information about the service.

Notes

- The services socket functions are supported by CA IDMS's internal services resolver. For more information, see the *CA IDMS System Operations Guide*.
- The SERVENT structure area is allocated by the system and associated with a CA IDMS task. It is freed at task termination. It is reused by subsequent calls to a services function: GETSERVBYNAME or GETSERVBYPORT.

  Note: For more information about the SERVENT structure, see Socket Structure Descriptions (see page 166).
- When the CASE sub-clause in the SERVICES FILE clause is defined as SENSITIVE, then the *service-name* and the *protocol-name* must be specified exactly as they are defined in the services file.

  If it is defined as INSENSITIVE, the internal services resolver always tries to first retrieve the *service-name* and *protocol-name* as they are coded in the socket function call. If they are not found, the first entry where the uppercase versions of the service names and protocol names match are returned. In all cases, all the strings returned in the SERVENT structure are always coded as they appear in the services file.

GETSERVBYPORT

GETSERVBYPORT takes a port number and a protocol number and tries to resolve them using the services file. If successful, it returns the information in a SERVENT structure.

Assembler

```
Label    #SOCKET GETSERVBYPORT,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               PORT=port-number,
               PROTNAME=protocol-name,
               PROTNAML=protocol-name-length,
               SERVENTP=serventp,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETSERVBYPORT,
return-code,
errno,
reason-code,
port-number,
protocol-name,
protocol-name-length,
serventp
```

Parameters

Parameter	Description
port-number	Specifies the name of a fullword field containing the *port-number* to resolve.
protocol-name	Specifies the name of an area containing the name of the protocol to use.
protocol-name-length	Specifies the name of a fullword field containing the length of protocol-name.*protocol-name-length* can be specified as an absolute expression. The maximum value for this parameter is 256.
serventp	Specifies the name of a fullword field where the system returns the address of a SERVENT structure containing the information about the service.

Notes

- The services socket functions are supported by CA IDMS's internal services resolver. For more information, see the *CA IDMS System Operations Guide*.
- The SERVENT structure area is allocated by the system and associated with a CA IDMS task. It is freed at task termination. It is reused by subsequent calls to a services function: GETSERVBYNAME or GETSERVBYPORT.

  More Information:

  For more information about the SERVENT structure, see Socket Structure Descriptions (see page 166).
- When the CASE sub-clause in the SERVICES FILE clause is defined as SENSITIVE, then the *service-name* and the *protocol-name* must be specified exactly as they are defined in the services file.

  If it is defined as INSENSITIVE, the internal services resolver always tries to first retrieve the *service-name* and *protocol-name* as they are coded in the socket function call. If they are not found, the first entry where the uppercase versions of the service names and protocol names match are returned. In all cases, all the strings returned in the SERVENT structure are always coded as they appear in the services file.

GETSOCKNAME

GETSOCKNAME retrieves the current name of a socket into a sockaddr structure.

Assembler

```
label    #SOCKET GETSOCKNAME,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               SOCKADDR=sockaddr,
               SOCKADDL=sockaddr-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETSOCKNAME,
return-code,
errno,
reason-code,
socket-descriptor,
sockaddr,
sockaddr-length
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor from which to retrieve the name.
sockaddr	Specifies the name of an area in which to return the sockaddr structure of the socket. The format of this structure depends on the domain of the corresponding socket. This parameter can be assigned to NULL if the caller is not interested in the socket's address.
sockaddr-length	Specifies the name of a fullword field containing the length of *sockaddr*. If SOCKADDR is assigned to NULL, *sockaddr-length* must be 0. On return, *sockaddr-length* contains the size required to represent the socket. If the size of *sockaddr* is too small to contain the full sockaddr structure, it is truncated. The maximum value for this parameter is 4096.

GETSOCKOPT

GETSOCKOPT retrieves the options currently associated with a socket.

Assembler

```
label    #SOCKET GETSOCKOPT,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               LEVEL=level,
               OPTNAME=option-name,
               OPTVAL=option-value,
               OPTLEN=option-value-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETSOCKOPT,
return-code,
errno,
reason-code,
socket-descriptor,
level,
option-name,
option-value,
option-value-length
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor for which the service is to be performed.
level	Specifies the name of a fullword field containing the level for the *option*. *level* can be specified as an absolute expression.
option-name	Specifies the name of a fullword field indicating the *option* to retrieve. *option-name* can be specified as an absolute expression.
option-value	Specifies the name of an area that will contain the requested data.
option-value-length	Specifies the name of a fullword field that contains the length of *option-value*. On return, *option-value-length* contains the size of the data returned in *option-value*. The maximum value for this parameter is 4096.

Notes

- **z/VSE systems:** The GETSOCKOPT function is not supported.
- The following table lists the options that can be specified. The EQUate symbol is generated by the #SOCKET TCPIPDEF macro call and the field names are associated with the SOCKET-MISC-DEFINITIONS.

EQUate Symbol	Field Names	Description
S@SOCKET	SOCKET-SOCKOPT-SOLSOCKET	Specifies level number for socket options
SO@REUSE	SOCKET-SOCKOPT-REUSEADDR	Allows local address reuse
SO@KEEPA	SOCKET-SOCKOPT-KEEPALIVE	Activates the keep-alive mechanism
SO@OOBIN	SOCKET-SOCKOPT-OOBINLINE	Accepts out-of-band data
SO@SNBUF	SOCKET-SOCKOPT-SNDBUF	Reports send buffer size information
SO@RCBUF	SOCKET-SOCKOPT-RCVBUF	Reports receive buffer size information
TCP@NODL	SOCKET-SOCKOPT-NODELAY	Specifies TCP_NODELAY option

PL/I programs: The SOCKET_MISC_DEFINITIONS is used and the dashes are replaced by underscores.

GETSTACKS

GETSTACKS retrieves the list of all the TCP/IP stacks currently defined in the system.

Assembler

```
label    #SOCKET GETSTACKS,
                 RETCODE=return-code,
                 ERRNO=errno,
                 RSNCODE=reason-code,
                 BUFFER=buffer,
                 BUFFERL=buffer-length,
                 FORMAT=output-format
                 RETLEN=output-length,
                 RETNSTKS=stacks-count,
                 PLIST=parameter-list-area,
                 RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-GETSTACKS,
return-code,
errno,
reason-code,
buffer,
buffer-length,
output-format,
output-length,
stacks-count
```

Parameters

Parameter	Description
buffer	Specifies the name of a buffer that receives the list of all the stacks. This parameter is optional.
buffer-length	Specifies the name of a fullword field containing the length of *buffer*. *buffer-length* can be specified as an absolute expression. This parameter is optional. If the size of *buffer* is too small to contain the full output, it is truncated. The maximum value for this parameter is 4096.
output-format	Specifies the name of a fullword field indicating the format desired for the output. *output-format* can be specified as an absolute expression. If the *output-format* value is 1, all the names of the different stacks are listed in a sequence of 8-byte character string. If *output-format* value is 2, all the names of the different stacks are listed in a sequence of the following structure: a 1-byte field containing the length of the name followed by the name itself. This is an optional parameter. If it is not specified, output-format 1 is assumed.
output-length	Specifies the name of a fullword field containing the actual length required to hold all the output in the requested format..
stacks-count	Specifies the name of a fullword field containing the number of TCP/IP stacks currently defined (but not necessarily active) in the system.

Notes

- The *buffer* and *buffer-length* parameters are optional. If these parameters are not specified, only the *output-length* and *stacks-count* values are returned.
- For more information, see Using Multiple TCP/IP Stacks (see page 74).

HTONL

HTONL converts a fullword integer from host byte order to network byte order. Within CA IDMS, host and network byte order are the same. Therefore, the HTONL function does not apply to the mainframe environment; it is implemented for the application programmer's convenience.

Assembler

```
label    #SOCKET HTONL,
               FIELDIN=input-field,
               FIELDOUT=output-field,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-HTONL,
input-field,
output-field
```

Parameters

Parameter	Description
input-field	Specifies the name of a fullword field containing the integer to convert.
output-field	Specifies the name of a fullword field that receives the converted integer.

HTONS

HTONS converts a halfword integer from host byte order to network byte order. Within CA IDMS, host and network byte order are the same. Therefore, the HTONS function does not apply to the mainframe environment; it is implemented for the application programmer's convenience.

Assembler

```
label    #SOCKET HTONS,
               FIELDIN=input-field,
               FIELDOUT=output-field,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-HTONS,
input-field,
output-field
```

Parameters

Parameter	Description
input-field	Specifies the name of a halfword field containing the integer to convert.
output-field	Specifies the name of a halfword field that receives the converted integer.

INET_ADDR

INET_ADDR translates an IP address in standard dotted string format into its binary format.

Assembler

```
label    #SOCKET INET_ADDR,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               IPADDRS=ip-address_string,
               IPADDRSL=ip-address-string-length,
               IPADDR=ip-address,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-INETADDR,
return-code,
errno,
reason-code,
ip-address-string,
ip-address-string-length,
ip-address
```

Parameters

Parameter	Description
ip-address-string	Specifies the name of an area containing the IP address in standard dotted string format.

Parameter	Description
ip-address-string-length	Specifies the name of a fullword field containing the length of *ip-address-string*, which can be specified as an absolute expression. The maximum value for this parameter is defined by IPADDS4L in Assembler and SOCKET-IPADDS4L in other languages.
ip-address	Specifies the name of a fullword field that will contain the IP address in binary format.

INET_NTOA

INET_NTOA translates an IP address in binary format into standard dotted string format. The IP address is in IPv4 format.

Note: INET_NTOA does not support IPv6 format. For new applications, you can use INET_NTOP, which supports IPv6 and IPv4 formats.

Assembler

```
label    #SOCKET INET_NTOA,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               IPADDR=ip-address,
               IPADDRS=ip-address-string,
               IPADDRSL=ip-address-string-length,
               RETIPASL=returned-ip-address-string-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-INETNTOA,
return-code,
errno,
reason-code,
ip-address,
ip-address-string,
ip-address-string-length,
returned-ip-address-string-length
```

Parameters

Parameter	Description
ip-address	Specifies the name of a fullword field containing the IP address in binary format.
ip-address-string	Specifies the name of an area in which to return the IP address in standard dotted string format.
ip-address-string-length	Specifies the name of a fullword field containing the length of *ip-address-string*. *ip-address-string-length* can be specified as an absolute expression. The maximum value for this parameter is 4096.
returned-ip-address-string-length	Specifies the name of a fullword field in which the actual length of the IP address string is returned.

INET_NTOP

INET_NTOP translates an IP address in binary format into standard string format.

Assembler

```
label    #SOCKET INET_NTOP,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               DOMAIN=domain,
               IPADDR=ip-address,
               IPADDRS=ip-address-string,
               IPADDRSL=ip-address-string-length,
               RETIPASL=returned-ip-address-string-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-INETNTOP,
return-code,
errno,
reason-code,
domain,
ip-address,
ip-address-string,
ip-address-string-length,
returned-ip-address-string-length
```

Parameters

Parameter	Description
domain	Specifies the name of a fullword field containing the domain. *domain* can be specified as an absolute expression. Possible values are AF@INET and AF@INET6.
ip-address	Specifies the name of an area containing the IP address in binary format: a fullword for an Ipv4 address, or a 16-byte area for an Ipv6 address.
ip-address-string	Specifies the name of an area in which to return the IP address in standard string format.
ip-address-string-length	Specifies the name of a fullword field containing the length of *ip-address-string*. *ip-address-string-length* can be specified as an absolute expression. The maximum value for this parameter is 4096.
returned-ip-address-string-length	Specifies the name of a fullword field in which the actual length of the IP address string is returned.

INET_PTON

INET_PTON translates an IP address in standard string format into its binary format.

Assembler

```
label    #SOCKET INET_PTON,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               DOMAIN=domain,
               IPADDRS=ip-address_area,
               IPADDRSL=ip-address-string-length,
               IPADDR=ip-address,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-INETPTON,
return-code,
errno,
reason-code,
domain,
ip-address-string,
returned-ip-address-string-length,
ip-address
```

Parameters

Parameter	Description
domain	Specifies the name of a fullword field containing the domain. *domain* can be specified as an absolute expression. Possible values are AF@INET and AF@INET6.
ip-address-string	Specifies the name of an area containing the IP address in standard string format.
ip-address-string-length	Specifies the name of a fullword field containing the length of *ip-address-string*. *ip-address-string-length* can be specified as an absolute expression. The maximum value for this parameter is determined by the type of address: ■ IPv4 address—IPADDS4L in Assembler and SOCKET-IPADDS4L in other languages ■ IPv6 address—IPADDS6L in Assembler and SOCKET-IPADDS6L in other languages
ip-address	Specifies the name of an area in which to return the IP address in binary format: a fullword for an IPv4 address, or a 16-byte area for an IPv6 address.

IOCTL

IOCTL controls certain characteristics of a socket. Depending on the command, it can retrieve or set control information.

Assembler

```
Label    #SOCKET IOCTL,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               COMMAND=command,
               ARGUMENT=argument,
               ARGUMENL=argument-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-IOCTL,
return-code,
errno,
reason-code,
socket-descriptor,
command,
argument,
argument-length
```

Parameters

Parameter	Description
socket-descriptor	The name of a fullword field containing the *socket-descriptor* to process.
command	The name of a fullword field containing the *command* to perform on the socket. *command* can be specified as an absolute expression.
argument	The name of a fullword field containing the address of the *argument* area that is used by the corresponding *command*. The *argument* area usually contains input and output fields.
argument-length	The name of a fullword field that contains the length of the *argument* area.

The different commands and arguments allowed usually depend on the operating system where the CA IDMS system is running. For a full description of these parameters, see the corresponding socket API manual.

Notes

- z/VSE systems-The IOCTL function is not supported.
- The following table lists the commands that can be specified. The EQUate symbol is generated by #SOCKET macro and the field names are associated with the SOCKET-MISC-DEFINITIONS-2 record.

EQUate Symbol	Field Name	Description
IO@NREAD	SOCKET-IOCTL-FIONREAD	Sets or resets socket in non-blocking mode
IO@NBIO	SOCKET-IOCTL-FIONBIO	Retrieves the number of readable bytes available
IO@CTTLS	SOCKET-IOCTL-SIOCTTLSCTL	Allows an application to query or control AT-TLS

PL/I programs: The SOCKET_MISC_DEFINITIONS_2 is used and the dashes are replaced by underscores.

More Information:

For more information about socket functions, see the *CA IDMS Callable Services Guide*.

LISTEN

LISTEN indicates that an application is ready to accept client connection requests and defines the maximum length of the connection request queue.

Assembler

```
label    #SOCKET LISTEN,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               BACKLOG=backlog,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-LISTEN,
return-code,
errno,
reason-code,
socket-descriptor,
backlog
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor on which to listen.
backlog	Specifies the name of a fullword field containing the backlog value. *backlog* can be specified as an absolute expression. It defines the maximum number of pending connections that may be queued. The value cannot exceed the maximum number of connections allowed by the installed TCP/IP. **z/VSE systems:** The BACKLOG parameter is ignored. The installed TCP/IP determines the backlog value for a given socket.

NTOHL

NTOHL converts a fullword integer from network byte order to host byte order. Within CA IDMS, host and network byte order are the same. Therefore, the NTOHL function does not apply to the mainframe environment; it is implemented for the application programmer's convenience.

Assembler

```
label    #SOCKET NTOHL,
               FIELDIN=input-field,
               FIELDOUT=output-field,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-NTOHL,
input-field,
output-field
```

Parameters

Parameter	Description
input-field	Specifies the name of a fullword field containing the integer to convert.
output-field	Specifies the name of a fullword field that receives the converted integer.

NTOHS

NTOHS converts a halfword integer from network byte order to host byte order. Within CA IDMS, host and network byte order are the same. Therefore, the NTOHS function does not apply to the mainframe environment; it is implemented for the application programmer's convenience.

Assembler

```
label    #SOCKET NTOHS,
               FIELDIN=input-field,
               FIELDOUT=output-field,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-NTOHS,
input-field,
output-field
```

Parameters

Parameter	Description
input-field	Specifies the name of a halfword field containing the integer to convert.
output-field	Specifies the name of a halfword field that receives the converted integer.

READ

READ reads a number of bytes from a socket into an area.

Assembler

```
label    #SOCKET READ,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               BUFFER=buffer,
               BUFFERL=buffer-length,
               RETLEN=read-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-READ,
return-code,
errno,
reason-code,
socket-descriptor,
buffer,
buffer-length,
read-length
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor to read from.
buffer	Specifies the name of the area where the data is to be placed.
buffer-length	Specifies the name of a fullword field containing the length of the *buffer. buffer-length* can be specified as an absolute expression.
read-length	Specifies the name of a fullword field in which the actual length of the data read is returned.

Notes

When the time-out value associated with the socket expires, the socket function terminates with the ETIMEDOUT errno code. For more information about socket time-outs, see Associating Time-outs to Sockets (see page 77).

RECV

RECV reads a number of bytes from a connected socket into an area.

Assembler

```
label    #SOCKET RECV,
                 RETCODE=return-code,
                 ERRNO=errno,
                 RSNCODE=reason-code,
                 SOCK=socket-descriptor,
                 BUFFER=buffer,
                 BUFFERL=buffer-length,
                 FLAGS=flags,
                 RETLEN=read-length,
                 PLIST=parameter-list-area,
                 RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-RECV,
return-code,
errno,
reason-code,
socket-descriptor,
buffer,
buffer-length,
flags,
read-length
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor from which to read.
buffer	Specifies the name of the area where the data is to be placed.
buffer-length	Specifies the name of a fullword field containing the length of the *buffer. buffer-length* can be specified as an absolute expression.
flags	Specifies the name of a fullword field containing information on how the data is to be received. **z/VSE systems:** MSG@PEEK is the only flag value that is supported. The remaining flags are not supported and returns an error if specified. When the MSG@PEEK flag is specified only the first byte of the RECV buffer is returned, even if a larger buffer size is specified. **z/VM systems:** MSG@WALL is not supported.
read-length	Specifies the name of a fullword field in which the actual length of the data read is returned.

Notes

- When the time-out value associated with the socket expires, the socket function terminates with the ETIMEDOUT errno code. For more information about socket time-outs, see Associating Time-outs to Sockets (see page 77).
- The following table lists the flags that can be specified. The EQUate symbol is generated by the MSGFLAGS DSECT by the #SOCKET TCPIPDEF macro call and the field names are associated with the SOCKET-MISC-DEFINITIONS.

EQUate Symbol	Field Name	Description
MSG@DROU	SOCKET-MSGFLAGS-DONTROUTE	Send without network routing

EQUate Symbol	Field Name	Description
MSG@OOB	SOCKET-MSGFLAGS-OOB	Send and receive out-of-band data
MSG@PEEK	SOCKET-MSGFLAGS-PEEK	Peek at incoming data
MSG@WALL	SOCKET-MSGFLAGS-WAITALL	Wait until all data returned

PL/I programs: The SOCKET_MISC_DEFINITIONS is used and the dashes are replaced by underscores.

RECVFROM

RECVFROM reads a number of bytes from a datagram socket into an area.

Assembler

```
label    #SOCKET RECVFROM,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               BUFFER=buffer,
               BUFFERL=buffer-length,
               FLAGS=flags,
               SOCKADDR=sockaddr,
               SOCKADDL=sockaddr-length,
               RETLEN=read-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-RECVFROM,
return-code,
errno,
reason-code,
socket-descriptor,
buffer,
buffer-length,
flags,
sockaddr,
sockaddr-length,
read-length
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor from which to read.
buffer	Specifies the name of the area where the data is to be placed.
buffer-length	Specifies the name of a fullword field containing the length of the *buffer. buffer-length* can be specified as an absolute expression.
flags	Specifies the name of a fullword field containing information on how the data is to be received. The list of the different flags supported can be found in the MSGFLAGS DSECT generated by the #SOCKET TCPIPDEF macro call and in the SOCKET-MISC-DEFINITIONS record for other languages. For an explanation of flags that can be specified, see RECV function description.
sockaddr	Specifies the name of an area in which to return the sockaddr structure of the sender of the data. The format of this structure depends on the domain of the corresponding socket. This parameter can be assigned to NULL if the caller is not interested in the sender's address.
sockaddr-length	Specifies the name of a fullword field containing the length of *sockaddr*. If SOCKADDR is assigned to NULL, *sockaddr-length* must be 0. On return, *sockaddr-length* contains the size required to represent the socket. If the size of *sockaddr* is too small to contain the full sockaddr structure, it is truncated. The maximum value for this parameter is 4096.
read-length	Specifies the name of a fullword field in which the actual length of the data read is returned.

Notes

- When the time-out value associated with the socket expires, the socket function terminates with the ETIMEDOUT errno code. For more information about socket time-outs, see Associating Time-outs to Sockets (see page 77).
- **z/VSE systems:** The RECVFROM function is not supported.

SELECT and SELECTX

SELECT synchronizes processing of several sockets operating in non-blocking mode. Sockets that are ready for reading, ready for writing, or have a pending exceptional condition can be selected. If no sockets are ready for processing, SELECT can block indefinitely or wait for a specified period of time (which may be zero) and then return.

SELECT examines the socket descriptors specified by *read-list*, *write-list*, and *exception-list* to see if some are ready for reading, ready for writing, or have an exceptional condition pending, respectively. On return, SELECT updates each of the lists to indicate which socket descriptors are ready for the requested operation. The total number of ready descriptors in all the lists is returned.

SELECTX has the same functionality as SELECT with the additional capability of waiting on one or more ECBs in addition to a time interval. This allows interruption of a wait if an external event occurs.

Assembler

```
label    #SOCKET SELECT,
                 RETCODE=return-code,
                 ERRNO=errno,
                 RSNCODE=reason-code,
                 NFDS=number-of-socket-descriptors,
                 READLST=read-list,
                 READLSTL=read-list-length,
                 WRITLST=write-list,
                 WRITLSTL=write-list-length,
                 EXCELST=exception-list,
                 EXCELSTL=exception-list-length,
                 OPTION=option,
                 TIMEOUT=timeval-structure,
                 RETNFDS=returned-number-of-descriptors,
                 PLIST=parameter-list-area,
                 RGSV=(rgsv)
```

```
label    #SOCKET SELECTX,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               NFDS=number-of-socket-descriptors,
               READLST=read-list,
               READLSTL=read-list-length,
               WRITLST=write-list,
               WRITLSTL=write-list-length,
               EXCELST=exception-list,
               EXCELSTL=exception-list-length,
               OPTION=option,
               TIMEOUT=timeval-structure,
               ECB=ecb,
               ECBLIST=ecb-list,
               RETNFDS=returned-number-of-descriptors,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-SELECT,
return-code,
errno,
reason-code,
number-of-socket-descriptors,
read-list,
read-list-length,
write-list,
write-list-length,
exception-list,
exception-list-length,
option,
timeval-structure,
returned-number-of-descriptors
```

```
SOCKET-FUNCTION-SELECTX,
return-code,
errno,
reason-code,
number-of-socket-descriptors,
read-list,
read-list-length,
write-list,
write-list-length,
exception-list,
exception-list-length,
option,
timeval-structure,
ecb,
ecb-list,
returned-number-of-descriptors
```

Parameters

Parameter	Description
number-of-socket-descriptors	Specifies the name of a field containing the highest socket descriptor specified in any of the lists + 1. Only socket descriptors whose value is less than *number-of-socket-descriptors* are considered in servicing the request.
read-list	Specifies the name of an area containing a bit list identifying the socket descriptors to be examined for a "ready to read" condition. Only socket descriptors whose corresponding bit in the bit list is on are considered. On return, the bits that are set indicate the descriptors that are ready to read. Specify NULL if the *read-list* is to be ignored.
read-list-length	Specifies the name of a fullword field containing the length in bytes of *read-list*. *read-list-length* can be specified as an absolute expression. *read-list-length* must be a multiple of 4; specify 0 if the *read-list* is to be ignored.
write-list	Specifies the name of an area containing a bit list identifying the socket descriptors to be examined for a "ready to write" condition. Only socket descriptors whose corresponding bit in the bit list is on are considered. On return, the bits that are set indicate the descriptors that are ready to write. Specify NULL if the *write-list* is to be ignored.
write-list-length	Specifies the name of a fullword field containing the length in bytes of *write-list*. *write-list-length* can be specified as an absolute expression. *write-list-length* must be a multiple of 4; specify 0 if the *write-list* is to be ignored.

Parameter	Description
exception-list	Specifies the name of an area containing a bit list identifying the socket descriptors to be examined for an exception condition. Only socket descriptors whose corresponding bit in the bit list is on are considered. On return, the bits that are set indicate the descriptors that have had exceptions. Specify NULL if the *exception-list* is to be ignored.
exception-list-length	Specifies the name of a fullword field containing the length in bytes of *exception-list*. *exception-list-length* can be specified as an absolute expression. *exception-list-length* must be a multiple of 4; specify 0 if the *exception-list* is to be ignored.
option	Specifies the name of a fullword field containing the way the different bits are interpreted in the different bit-lists. *option* can be specified as an absolute expression. For a list of *options* that can be specified, see Notes (see page 124).
timeval-structure	Specifies the name of the area containing the TIMEVAL structure. If the parameter is assigned to NULL, SELECT waits until at least one of the descriptors is ready. If the time-out value (number of seconds + number of microseconds) is 0, SELECT checks the descriptors and returns immediately without waiting. The TIMEVAL structure is generated by the #SOCKET TCPIPDEF macro call and described in the SOCKET-TIMEVAL record.
returned-number-of-descriptors	Specifies the name of a fullword field in which the total number of ready descriptors is returned.
ecb	Specifies the name of an area containing a CA IDMS ECB.
ecb-list	Specifies the name of an area containing a CA IDMS ECB list. Each entry in the ECB list is represented by two fullwords: ■ The first fullword is a pointer to the ECB. ■ The second fullword is zero, except for the last entry in the list. In this case the high-order bit is turned ON to identify the end of the ECB list.

Notes

- For more information about manipulating bits in bit lists, see FD_ZERO, FD_CLR, FD_SET, and FD_ISSET #SOCKET function.
- For programming languages like COBOL and CA ADS where it is difficult to manipulate bits in bit lists, byte lists can be used by specifying a SOCKET-SELECT-BYTELIST for *option*. In this case, the *read-list*, *write-list*, and *exception-list* are byte lists instead of bit lists. In byte lists, each byte represents one socket descriptor. A socket descriptor will be processed if its corresponding byte is set to the character '1'. A socket descriptor's corresponding byte is the *n*th byte relative to 1 in the list, where *n* is equal to the value of socket descriptor + 1.

- ECB and ECBLIST are mutually exclusive parameters.
- **z/VM systems:** If multiple TCP/IP stacks are used, all the sockets represented by a bit in the 3 bit lists must be created in the same TCP/IP stack.
- **z/VSE systems:** The SELECT and SELECTX functions are not supported.
- The following table lists the options that can be specified. The EQUate symbol is generated by the #SOCKET TCPIPDEF macro call and the field names are associated with the SOCKET-MISC-DEFINITIONS.

EQUate Symbol	Field Name	Description
SEL@BBKW	SOCKET-SELECT-BITBACKWARD	Specifies that the bits in the fullwords are in the backward order. This is the default value if the parameter is assigned to NULL.
SEL@BFRW	SOCKET-SELECT-BITFORWARD	Specifies that the bits in each fullword are in the forward order
SEL@BYTV	SOCKET-SELECT-BYTELIST	Specifies that the *read-list*, *write-list*, and *exception-list* are byte lists instead of bit lists.

PL/I programs: The SOCKET_MISC_DEFINITIONS is used and the dashes are replaced by underscores.

SEND

SEND sends data on a connected socket.

Assembler

```
label    #SOCKET SEND,
                 RETCODE=return-code,
                 ERRNO=errno,
                 RSNCODE=reason-code,
                 SOCK=socket-descriptor,
                 BUFFER=buffer,
                 BUFFERL=buffer-length,
                 FLAGS=flags,
                 RETLEN=sent-length,
                 PLIST=parameter-list-area,
                 RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-SEND,
return-code,
errno,
reason-code,
socket-descriptor,
buffer,
buffer-length,
flags,
sent-length
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor on which to do the send.
buffer	Specifies the name of the area containing the data to be sent.
buffer-length	Specifies the name of a fullword field containing the length of the *buffer. buffer-length* can be specified as an absolute expression.
flags	Specifies the name of a fullword field containing information on how the data is to be sent. The list of the different flags supported can be found in the MSGFLAGS DSECT generated by the #SOCKET TCPIPDEF macro call and in the SOCKET-MISC-DEFINITIONS record for other languages. For an explanation of flags that can be specified, see RECV function description. **z/VSE systems:** No flag values are supported and an error is returned if a value is specified.
sent-length	Specifies the name of a fullword field in which the actual length of the data sent is returned.

Notes

When the time-out value associated with the socket expires, the socket function terminates with the ETIMEDOUT errno code. For more information about socket time-outs, see Associating Time-outs to Sockets (see page 77).

SENDTO

SENDTO sends data on a datagram socket.

Assembler

```
label    #SOCKET SENDTO,
                 RETCODE=return-code,
                 ERRNO=errno,
                 RSNCODE=reason-code,
                 SOCK=socket-descriptor,
                 BUFFER=buffer,
                 BUFFERL=buffer-length,
                 FLAGS=flags,
                 SOCKADDR=sockaddr,
                 SOCKADDL=sockaddr-length,
                 RETLEN=sent-length,
                 PLIST=parameter-list-area,
                 RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-SENDTO,
return-code,
errno,
reason-code,
socket-descriptor,
buffer,
buffer-length,
flags,
sockaddr,
sockaddr-length,
sent-length
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor on which to do the send.
buffer	Specifies the name of the area containing the data to be sent.
buffer-length	Specifies the name of a fullword field containing the length of the *buffer*. *buffer-length* can be specified as an absolute expression.

Parameter	Description
flags	Specifies the name of a fullword field containing information on how the data is to be sent. The list of the different flags supported can be found in the MSGFLAGS DSECT generated by the #SOCKET TCPIPDEF macro call and in the SOCKET-MISC-DEFINITIONS record for other languages. For an explanation of flags that can be specified, see RECV function description.
sockaddr	Specifies the name of an area containing the sockaddr structure describing where data is to be sent. The format of this structure depends on the domain of the corresponding socket.
sockaddr-length	Specifies the name of a fullword field containing the length of *sockaddr*. *Sockaddr-length* can be specified as an absolute expression. The maximum value for this parameter is domain dependent. If the domain is AF_INET, it is the length of the SOCKET-SOCKADDR-IN record (SIN#LEN for Assembler). If the domain is AF_INET6, it is the length of the SOCKET-SOCKADDR-IN6 record (SIN6#LEN for Assembler).
sent-length	Specifies the name of a fullword field in which the actual length of the data sent is returned.

Notes

- When the time-out value associated with the socket expires, the socket function terminates with the ETIMEDOUT errno code. For more information about socket time-outs, see Associating Time-outs to Sockets (see page 77).
- **z/VSE systems:** The SENDTO function is not supported.

SETSOCKOPT

SETSOCKOPT sets options associated with a socket.

Assembler

```
label    #SOCKET SETSOCKOPT,
                 RETCODE=return-code,
                 ERRNO=errno,
                 RSNCODE=reason-code,
                 SOCK=socket-descriptor,
                 LEVEL=level,
                 OPTNAME=option-name,
                 OPTVAL=option-value,
                 OPTLEN=option-value-length,
                 PLIST=parameter-list-area,
                 RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-SETSOCKOPT,
return-code,
errno,
reason-code,
socket-descriptor,
level,
option-name,
option-value,
option-value-length
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor for which the service is to be performed.
level	Specifies the name of a fullword field containing the level for the *option*. *level* can be specified as an absolute expression.
option-name	Specifies the name of a fullword field indicating the *option* to set. *option-name* can be specified as an absolute expression.
option-value	Specifies the name of an area containing the data to associate with the socket.
option-value-length	Specifies the name of a fullword field containing the length of *option-value*. *option-value-length* can be specified as an absolute expression. The maximum value for this parameter is 16.

Notes

- The list of level and options currently supported are listed by the #SOCKET TCPIPDEF macro call for Assembler and in the SOCKET-MISC-DEFINITIONS record for other languages. For a description of the options that can be specified, see GETSOCKOPT.
- **z/VSE systems:** Only the SO@REUSE option is supported.

SETSTACK

SETSTACK sets the requested TCP/IP stack affinity for the current executing CA IDMS task.

Assembler

```
label    #SOCKET SETSTACK,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               NAME=stack-name,
               NAMEL=stack-name-length,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-SETSTACK,
return-code,
errno,
reason-code,
stack-name,
stack-name-length
```

Parameters

Parameter	Description
stack-name	Specifies the area containing the name of the TCP/IP stack to set. This name can be the JOBNAME of the corresponding TCPIP stack, a *hostname* or an IP-address in binary or string format.
stack-name-length	Specifies the name of a fullword field containing the length of *stack-name*. *stack-name-length* can be specified as an absolute expression. The maximum value for this parameter is 256.

Notes

- To clear TCP/IP stack affinity for the current task, call the SETSTACK function using *stack-name* value equal to '*ALL'.
- To restore the default TCP/IP stack affinity for the current task, call the SETSTACK function using *stack-name* value equal to '*DEFAULT'.
- For more information, see Using Multiple TCP/IP Stacks (see page 74).

SHUTDOWN

SHUTDOWN shuts down all or part of a duplex socket connection.

Assembler

```
label    #SOCKET SHUTDOWN,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               SOCK=socket-descriptor,
               HOW=how-condition,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-SHUTDOWN,
return-code,
errno,
reason-code,
socket-descriptor,
how-condition
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor to shut down.
how-condition	Specifies the name of a fullword field indicating the effect of the shutdown on read and write operations. *how-condition* can be specified as an absolute expression.

Notes

The following table lists the conditions that can be specified. The EQUate symbol is generated by the #SOCKET TCPIPDEF macro call and the field names are located in the SOCKET-MISC-DEFINITIONS record.

EQUate Symbol	Field Name	Description
SHUT_R	SOCKET-SHUTDOWN-READ	Terminates read communication (from the socket)

EQUate Symbol	Field Name	Description
SHUT_W	SOCKET-SHUTDOWN-WRITE	Terminates write communication (to the socket)
SHUT_RW	SOCKET-SHUTDOWN-READ-WRITE	Terminates both read and write communication

PL/I programs: The SOCKET_MISC_DEFINITIONS is used and the dashes are replaced by underscores.

SOCKET

SOCKET creates a socket in a communications domain.

Assembler

```
label    #SOCKET SOCKET,
               RETCODE=return-code,
               ERRNO=errno,
               RSNCODE=reason-code,
               DOMAIN=domain,
               TYPE=type,
               PROTNUM=protocol-number,
               NEWSOCK=new-socket-descriptor,
               PLIST=parameter-list-area,
               RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-SOCKET,
return-code,
errno,
reason-code,
domain,
type,
protocol-number,
new-socket-descriptor
```

Parameters

Parameter	Description
domain	Specifies the name of a fullword field containing the domain or address family of the socket. For a list of the domains that can be specified, see Notes (see page 133).

Parameter	Description
type	Specifies the name of a fullword field containing the type of the socket. *type* can be specified as an absolute expression. For a list of socket types that can be specified, see Notes (see page 133).
protocol-number	Specifies the name of a fullword field containing the protocol. *protocol-number* can be specified as an absolute expression. For a list of supported protocols, see Notes (see page 133).
new-socket-descriptor	Specifies the name of a fullword field where the newly created socket descriptor is returned.

Notes

- The maximum number of sockets that can be created globally in the DC/UCF system, and the maximum number of sockets that can be created by a single task in the DC/UCF system can be controlled:
  - At startup, through the MAXIMUM NUMBER OF SOCKETS and the MAXIMUM NUMBER OF SOCKETS PER TASK clause from the TCP/IP system generation statement.
  - Dynamically, through the DCMT VARY TCP/IP command.

  For more information:
  - About the TCP/IP system generation statement, see the *CA IDMS System Generation Quick Reference Guide*.
  - About the DCMT VARY TCP/IP command, see the *CA IDMS System Tasks and Operator Commands Guide*.
- The following table lists the domains that can be specified. The EQUate symbol is generated by the are located in the SOCKET-MISC-DEFINITIONS record.

EQUate Symbol	Field Name	Description
AF@INET*	SOCKET-FAMILY-AFINET	AF_INET address family
AF@INET6	SOCKET-FAMILY-AFINET6	AF_INET6 address family
z/VSE systems: * —Only supports DOMAIN=*AF@INET*		

- The following table lists the socket types that can be specified. The EQUate symbol is generated by the in the SOCKET-MISC-DEFINITIONS record.

EQUate Symbol	Field Name	Description
STREAM*	SOCKET-TYPE-STREAM	Stream—Connection oriented and reliable

EQUate Symbol	Field Name	Description
DATAGRAM	SOCKET-TYPE-DATAGRAM	Datagram—Connectionless and unreliable
z/VSE systems: * —Only supports TYPE=STREAM.		

- The following table lists the protocols that can be specified. The EQUate symbol is generated by the SOCKET-MISC-DEFINITIONS record.

EQUate Symbol	Field Name	Description
PROTIP	SOCKET-PROTOCOL-IP	Default protocol
PROTTCP*	SOCKET-PROTOCOL-TCP	TCP protocol
PROTUDP	SOCKET-PROTOCOL-UDP	UDP protocol
PROTIPV6	SOCKET-PROTOCOL-IPV6	IPv6 protocol
z/VSE systems: * —Only supports PROTNUM=PROTTCP		

PL/I programs: The SOCKET_MISC_DEFINITIONS is used and the dashes are replaced by underscores.

WRITE

WRITE sends data on a socket.

Assembler

```
label    #SOCKET WRITE,
                 RETCODE=return-code,
                 ERRNO=errno,
                 RSNCODE=reason-code,
                 SOCK=socket-descriptor,
                 BUFFER=buffer,
                 BUFFERL=buffer-length,
                 RETLEN=sent-length,
                 PLIST=parameter-list-area,
                 RGSV=(rgsv)
```

List of USING Parameters

```
SOCKET-FUNCTION-WRITE,
return-code,
errno,
reason-code,
socket-descriptor,
buffer,
buffer-length,
sent-length
```

Parameters

Parameter	Description
socket-descriptor	Specifies the name of a fullword field containing the socket descriptor on which to send.
buffer	Specifies the name of the area containing the data to be sent.
buffer-length	Specifies the name of a fullword field containing the length of the *buffer*. *buffer-length* can be specified as an absolute expression.
sent-length	Specifies the name of a fullword field in which the actual length of the data sent is returned.

Notes

When the time-out value associated with the socket expires, the socket function terminates with the ETIMEDOUT errno code. For more information about socket time-outs, see Associating Time-outs to Sockets (see page 77).

Chapter 5: Invoking System Tasks from Programs

This section contains the following topics:

Invoking Command List Modules from Programs

You can invoke the CLIST task from application programs. A program invokes a CLIST task by linking to the program invoked by the CLIST task. This program is RHDCCLST.

RHDCCLST sets up the logical terminal for command list processing. DC/UCF executes the specified command list module only when the issuing program returns control to DC/UCF.

Linking to RHDCCLST

The calling program links to program RHDCCLST, passing four mandatory parameters and optionally a fifth parameter:

```
#LINK PGM=RHDCCLST,PARMS=(PARM1,PARM2,PARM3,PARM4,PARM5)
```

Parameters

Mandatory Parameters

The link statement to RHDCCLST must include the following parameter list:

Parameter 1 (32 bytes)

Specifies the name of the module in the data dictionary. The name is left-justified and padded on the right with blanks.

Parameter 2 (halfword)

Specifies the version number.

Parameter 3 (halfword)

Specifies the PROMPT/NOPROMPT status as follows:

0

Specifies a PROMPT status.

1

Specifies a NOPROMPT status.

Parameter 4 (halfword)

Specifies the return code. On return from RHDCCLST, the return code can contain one of the following values:

0

Specifies command list processing has been set up successfully.

8

Specifies that the specified module was not found in the dictionary.

16

Specifies that the specified module has no text.

Optional Parameter

In addition, the link statement to RHDCCLST can include the following optional parameter:

Parameter 5 (16 bytes)

(DDS users only) Specifies the dictionary node and dictionary name.

Bytes 1-8

Identifies the DDS dictionary node that controls the data dictionary specified by the dictionary name.

Bytes 9-16

Identifies the data dictionary included in the database name table defined either by the current system or for the system identified in the dictionary node.

Note: Both the dictionary node and dictionary name are left-justified and padded with blanks.

Example

The following example invokes RHDCCLST to execute command list MYCLIST:

```
#LINK PGM='RHDCCLST',PARMS=(CLISTNAM,VERSION,PROMPT,RETCODE)
 .
 .
CLISTNAM  DC  CL32'MYCLIST'       Command List name
VERSION   DC  H'1'                Version 1
PROMPT    DC  H'1'                Don't prompt
RETCODE   DC  H'0'                Return code
```

More Information

For more information about the CLIST command, see the *CA IDMS System Tasks and Operator Commands Guide*.

Invoking DCMT and DCUF Commands from Programs

You can invoke DCMT and DCUF commands from application programs. The procedures for invoking these commands are similar: your program invokes the DCUF or DCMT command by linking to the same program that is invoked when the command is entered from a terminal. Unless the programs are changed on site by the database administrator, their names are as follows:

- RHDCMT00—Invokes all DISPLAY and VARY DCMT commands.
- RHDCUF00—Invokes all SET and SHOW DCUF commands.

Note: The program names end with two zeroes.

Linking to RHDCMT00 and RHDCUF00

RHDCMT00/RHDCUF00 handles all output. Therefore, you must specify the NONOVERLAYABLE option for RHDCMT00/RHDCUF00 on the system generation PROGRAM statement.

RHDCMT00 Link Statement

The calling program links to the DCMT program RHDCMT00, passing the addresses of INREC and OUTREC as parameters:

```
#LINK PGM='RHDCMT00',PARMS=(INREC,OUTREC)
```

RHDCUF00 Link Statement

The calling program then links to the DCUF program RHDCUF00, passing the addresses of INREC and OUTREC as parameters:

```
#LINK PGM='RHDCUF00',PARMS=(INREC,OUTREC)
```

Parameters

The application program must allocate storage for two parameters, INREC and OUTREC, before calling RHDCMT00 or RHDCUF00.

- INREC—Contains the command, prefixed by a halfword with the length of the command.
- OUTREC—Contains control information such as return code and output handling and where text output may be returned.

INREC Format

The format is identical for RHDCUF00 and RHDCMT00.

Field	Length	Type
Length of DCMT/DCUF command	2 bytes	Binary
DCMT/DCUF command, left-justified	Any number of bytes	Character

OUTREC Format

The format is identical for RHDCUF00 and RHDCMT00.

Field	Length	Type
*return-area-length:** Length of the *returned-text-area* allocated by the calling program at the end of OUTREC. The calling program must specify this value. If *return-area-length* and *output-code* are set to zero, all DCMT/DCUF text output is discarded. DCMT VARY and DCUF SET commands are still performed.	4 bytes	Binary

Field	Length	Type
*return-code:***	2 bytes	Binary
■ 0—Request accepted and processed ■ 4—Invalid syntax ■ 8—Invalid request (for example, SHUTDOWN, ABORT) ■ 12—Security violation ■ 16—Processing error ■ 20—The *output-code* is 0 or 2 and the *return-area-length* is less than *output-length*. All complete lines whose total length (including a one-byte line-length indicator for each line) is less than or equal to *return-area-length* are stored in *returned-text-area*. The value of *output-code* determines how any remaining output lines are handled. If *output-code* = 0, the lines are discarded. If *output-code* = 2, the lines are written to scratch.		
*output-code:**	2 bytes	Binary
Halfword code indicating DCMT output target as follows: ■ 0—The *returned-text-area* at the end of OUTREC. ■ 1—The scratch area with a scratch ID of "DCMT" or "DCUF" depending on the called program. Each line is written as a separate scratch record. ■ 2—Any complete lines of output whose total length (including a one-byte line-length indicator per line) is less than or equal to *return-area-length* are written to *returned-text-area*. Remaining output is written to the scratch area with a scratch ID of "DCMT."		
*output-length:***	4 bytes	Binary
Total length required for DCMT/DCUF output. It includes the total length all text lines plus one byte (line-length indicator) for each line. **Note:** Each line-length indicator byte is counted as part of the *output-length* regardless of whether the record is written to the *returned-text-area*, written to scratch, or discarded. The line-length indicator byte is inserted before each text line written to the *returned-text-area*. The line-length indicator is not written into scratch records. Using the #GETSCR or GET SCRATCH command with the appropriate parameters, you can determine the length of a individual scratch record.		
*returned-output-length:***	4 bytes	Binary
Total length of text lines inserted into the *returned-text-area*. The length includes a one-byte line-length indicator for each text line.		

Field	Length	Type
*returned-text-area:***	Variable	Character
Area where the DCMT/DCUF is to return text output. Each text line is preceded by a one-byte field which contains the length of the text line (excluding the line-length indicator) as a hexadecimal value. Only lines whose total length is less than *return-area-length* are written into the *returned-text-area*. Other lines are discarded or written to the scratch area depending on the value of *output-code*. DCMT/DCUF prefills this field with blanks up to the lesser value of the length specified in *return-area-length* or 256 bytes. If *return-area-length* is greater than 256 bytes, and *returned-output-length* is less than *return-area-length* any remaining storage in *returned-text-area* is not updated and remains as it was when RHDCMT00/RHDCUF00 was called. Note: To avoid storage overlays, the number of bytes allocated for the *returned-text-area* must be greater than or equal to the value assigned to *return-area-length*.		

Notes:

*—Information supplied by the calling program.

**—Information supplied by RHDCMT00/RHDCUF00.

Usage

Queued Requests

RHDCMT00 can process commands such as VARY AREA, but it may queue the action. Thus, a return code of zero does not always indicate that a VARY has occurred. The text returned to the output area denotes the status of the request. You can recheck the status of the request by including a SET TIMER POST or SET TIMER WAIT (#SETIME TYPE=POST or #SETTIME TYPE=WAIT in Assembler) statement in your program. After the interval has expired, you can reissue the VARY or issue a DISPLAY statement and check the returned text to determine the status of the request.

Using Scratch Area for Output

If you do not know the exact size of the output, you can specify to send it to the scratch area. This is also a useful method for dealing with possible future changes in output.

Examples

DCMT Example

The following example invokes RHDCMT00 to execute the DCMT VARY ACTIVE TASK command specified in INREC and to handle an invalid command:

```
DCMT     TITLE 'SAMPLE DRIVER TO CALL DCMT'
CALLDCMT CSECT
         LR    R12,R15
         USING CALLDCMT,R12
         B     BEGIN
         #MOPT CSECT=CALLDCMT,ENV=USER
BEGIN    DS    0H
         #GETSTG LEN=WORKDSL,PLIST=*,ADDR=(R2),TYPE=(USER,LONG),       X
               INIT=X'00'     GET WORK AREA FOR REENTRANCY.
        USING WORKDS,R2
*  FOLLOWING CODE ISSUES A VALID COMMAND TO CHANGE MAX TASKS.
*  RESULTING STATUS CODE, RETURNED LENGTH, AND RETURNED DATA
*  ARE DISPLAYED.
         MVC   INRECLN,=AL2(L'INRECTXT)  SET COMMAND LENGTH.
         MVC   INRECTXT,GOODCOMM         SET COMMAND.
         MVC   OUTRECLN,=AL4(L'OUTRECTX) SET MAXIMUM OUTPUT LENGTH.
         SR    R1,R1                     SET OUTPUT TYPE TO 0.
         STH   R1,OUTRECOD               REQUEST OUTPUT TO STORAGE.
         ST    R1,OUTRECTL               INITIALIZE RETURNED LENGTH.
         ST    R1,OUTRECAL               INITIALIZE RETURNED LENGTH.
         LA    R1,99                     INITIALIZE RETURN CODE.
         STH   R1,OUTRECRC
         BAL   R8,CALLDISP               CALL DCMT AND DISPLAY RESULTS.
*  FOLLOWING CODE ISSUES A COMMAND WITH INVALID SYNTAX.
*  RESULTING STATUS CODE, RETURNED LENGTH, AND RETURNED DATA
*  (ERROR MESSAGES) ARE DISPLAYED.
         MVC   INRECLN,=AL2(L'INRECTXT)  SET COMMAND LENGTH.
         MVC   INRECTXT,BADCOMM          SET COMMAND.
         MVC   OUTRECLN,=AL4(L'OUTRECTX) SET MAXIMUM OUTPUT LENGTH.
         SR    R1,R1                     SET OUTPUT TYPE TO 0.
         STH   R1,OUTRECOD               REQUEST OUTPUT TO STORAGE.
         ST    R1,OUTRECTL               INITIALIZE RETURNED LENGTH.
         ST    R1,OUTRECAL               INITIALIZE RETURNED LENGTH.
         LA    R1,99                     INITIALIZE RETURN CODE.
         STH   R1,OUTRECRC
         BAL   R8,CALLDISP               CALL DCMT AND DISPLAY RESULTS.
*  DONE WITH DISPLAYS.  TERMINATE PROGRAM.
         #RETURN
         EJECT
*  SUBROUTINE TO CALL DCMT AND DISPLAY RESULT.
*
```

```
CALLDISP DS    0H
         #LINK PGM='RHDCMT00',PARMS=(INREC,OUTREC)
         MVC   WORKRCLN,STATLIT
         LH    R15,OUTRECRC              GET RETURN CODE.
         CVD   R15,WORKDBL               MAKE IT PRINTABLE.
         UNPK  WORKRC,WORKDBL
         OI    WORKRC+L'WORKRC-1,X'F0'
         L     R5,OUTRECAL               GET LENGTH OF OUTPUT.
         CVD   R5,WORKDBL                MAKE IT PRINTABLE.
         UNPK  WORKLEN,WORKDBL
         OI    WORKLEN+L'WORKLEN-1,X'F0'
         #LINEOUT OUTLEN=80,OUTAREA=WORKRCLN
         LTR   R5,R5                     ANY RETURNED DATA?
         BZ    RETURN                    NO.
         LA    R4,OUTRECTX               GET A(OUTPUT).
PUTLINE  DS    0H                        PUT OUT ONE LINE.
         SR    R3,R3                     GET LENGTH OF ONE LINE.
         IC    R3,0(,R4)
         LA    R4,1(,R4)                 GET A(LINE TEXT)
         #LINEOUT OUTLEN=(R3),OUTAREA=(R4)
         AR    R4,R3                     POINT TO NEXT LENGTH BYTE.
         SR    R5,R3                     REDUCE OUTPUT LENGTH.
         BCTR  R5,0                      ACCOUNT FOR LENGTH BYTE.
         LTR   R5,R5                     STILL MORE OUTPUT?
         BP    PUTLINE                   GET LENGTH OF OUTPUT.
RETURN   DS    0H
         BR    R8
GOODCOMM DC    CL80'VARY ACTIVE TASK MAX TASK 43'
BADCOMM  DC    CL80'BAD SYNTAX IN THIS COMMAND'
STATLIT  DS    CL80' '                   TEMPLATE FOR STATUS LINE
         ORG   STATLIT                   REDEFINE STATUS LINE
         DC    CL13'RETURN CODE: '       RETURN CODE LITERAL
         DC    CL2' '                    SPACE FOR RETURN CODE
         DC    CL20'.  RETURNED LENGTH: ' LENGTH LITERAL
         DC    CL4' '                    SPACE FOR RETURN CODE
         DC    CL1'.'                    SPACE FOR RETURN CODE
         ORG   STATLIT+80                DONE WITH REDEFINE
         LTORG
         EJECT
WORKDS   DSECT
```

```
SYSPLIST DS    10F                    PLIST AREA
WORKDBL  DS    D                      TEMP WORK AREA
WORKRCLN DS    CL80                   OUTPUT LINE FOR RETURN CODE
         ORG   WORKRCLN               REDEFINE STATUS LINE
         DS    CL13                   RETURN CODE LITERAL
WORKRC   DS    CL2                    SPACE FOR RETURN CODE
         DS    CL20                    LENGTH LITERAL
WORKLEN  DS    CL4                    LENGTH OF RETURNED DATA
         DS    CL1                    SPACE FOR ENDING PERIOD
         ORG   WORKRCLN+80            DONE WITH STATUS LINE
INREC    DS    0F                     INPUT TO DCMT
INRECLN  DS    H                      INPUT LENGTH
INRECTXT DS    CL80                   INPUT COMMAND
         DS    H                      FILLER
OUTREC   DS    0F                     OUTPUT FROM DCMT
OUTRECLN DS    F                      MAXIMUM ALLOWED OUTPUT LENGTH
OUTRECRC DS    H                      RETURN CODE
OUTRECOD DS    H                      OUTPUT TYPE
OUTRECTL DS    F                      TOTAL LENGTH OF DCMT OUTPUT
OUTRECAL DS    F                      ACTUAL LENGTH RETURNED
OUTRECTX DS    CL132                  TEXT OUTPUT AREA
WORKDSL  EQU   *-WORKDS
         SPACE 2
         END
```

DCUF Example

The following example invokes RHDCUF00 to execute the DCUF SET PRINT CLASS command specified in INREC:

```
       #LINK PGM='RHDCUF00',PARMS=(INREC,OUTREC)
 .
 .
 .
INREC    DS    0F                  Input to DCUF
INRECLN  DC    Y(L'INRECTXT)          - length
INRECTXT DC    C'SET PRINT CLASS 01'  - command
*
OUTREC   DS    0F                  Output from DCUF
OUTRECLN DC    A(L'OUTRECTX)          - maximum allowed length
OUTRECRC DC    H'0'                   - return code
OUTRECOD DC    H'0'                   - output type
OUTRECTL DC    F'0'                   - total length
OUTRECAL DC    F'0'                   - actual length
OUTRECTX DS    CL132                  - actual output text
```

More Information

- For more information about the SET TIMER and GET SCRATCH statements, see the *CA IDMS DML Reference Guide for COBOL*.
- For more information about the #SETIME and #GETSCR statements, see the *CA IDMS DML Reference Guide for Assembler*.
- For more information about RHDCMT00 or RHDCUF00, see the *CA IDMS System Generation Guide*.
- For more information about DCMT or DCUF commands, see the *CA IDMS System Tasks and Operator Commands Guide*.

Invoking SDEL Command from Programs

You can invoke the SDEL command from application programs by linking to program RHDCSDEL.

Note: Securing the SDEL task code does not secure usage of the RHDCSDEL program. If you want to limit the use of RHDCSDEL, that program must be secured.

Linking to RHDCSDEL

The calling program links to program RHDCSDEL, passing the addresses DICTNAME, RETCODE, and OUTAREA as parameters:

```
#LINK PGM='RHDCSDEL',PARMS=(DICTNAME,RETCODE,OUTAREA)
```

Parameters

DICTNAME

Specifies the dictionary name of the updatable DDLDML and DDLCAT areas to be scanned for security definitions associated with logically deleted users.

This is an 8-character field, left-justified, and padded with blanks. If DICTNAME is set to blanks, DC/UCF processes the updatable DDLDML and DDLCAT areas of the default dictionary for the system. If DICTNAME is set to CL8'*ALL', all DDLDML and DDLCAT areas in the DMCL are processed.

RETCODE

Specifies a fullword in which RHDCSDEL provides a return code. The possible return codes are as follows:

00

Specifies processing was successful. The OUTAREA contains informational messages DC048004 and DC048008.

04

Specifies processing was successful but contains warnings. The possible causes are as follows:

- There were no logically deleted users to process. The OUTAREA contains informational message DC048002.
- The OUTAREA is too small to contain all output messages.

08

Specifies a processing error. The possible causes are as follows:

- The DICTNAME is invalid. The outarea contains error message DC048001.
- An unexpected database error was encountered. The OUTAREA contains error message DC048003.
- A BIND failed. The OUTAREA contains error message DC048004 or DC048006.

12

Specifies the fatal error, the DMCL module is invalid. The OUTAREA contains error message DC048007.

OUTAREA

Specifies an area where RHDCSDEL puts messages. The first fullword of the area must be initialized to the area length, which also includes the first fullword. Upon return, the first fullword contains the size of the messages. Each message is in the following format:

```
AL1(L'message),C'message'
```

Note: RETCODE is set to 04 if the output area is too small, unless a more severe error occurred.

Example

The following example invokes RHDCSDEL to clean up the security definitions for logically deleted users in the default dictionary:

```
#LINK PGM='RHDCSDEL,PARMS=(DICTNAME,RETCODE,OUTAREA)
.
.
RETCODE  DC    F'0'
DICTNAME DC    CL8'        '
*
OUTAREA  DS    0F
OUTAREAL DC    F'256'
OUTRECD  DC    XL252'00'
```

More Information

- For more information about the LINK statement, see the *CA IDMS DML Reference Guide* for the language of the calling program.
- For more information about securing a program, see the *CA IDMS Security Administration Guide*.

Invoking the SIGNON Task from Programs

You can invoke the SIGNON task from application programs. A program invokes the SIGNON task by linking the program invoked by the SIGNON task. This program is RHDCSNON.

Linking to RHDCSNON

The calling program links to program RHDCSNON, passing three mandatory parameters:

```
#LINK PGM=RHDCSNON,PARMS=(PARM1,PARM2,PARM3)
```

Parameters

The #LINK statement to RHDCSNON must include the following parameter list:

Parameter 1 (18-bytes)

Specifies the user ID. The user ID is left-justified and padded on the right with blanks.

Parameter 2 (8-bytes)

Specifies the password. The password is left-justified and padded on the right with blanks. For externally secured signons, this value can alternatively be a PassTicket. PassTickets are short-term substitutes for passwords which are targeted to a specific application.

Note: For more information on PassTickets, see the *CA IDMS Security Administration Guide*.

Parameter 3 (aligned halfword)

Specifies the return code. On return from RHDCSNON, the return code parameter can contain one of the following values:

0

Specifies the signon was successful.

4

Specifies the user is already signed on to another terminal, and multiple signons are disallowed.

8

Specifies the user ID was not authorized.

12

Specifies the password is invalid.

16

Specifies the user ID is blank (format error).

20

Specifies an error occurred when processing the dictionary.

24

Specifies the signon was stopped by the signon user exit.

Example

The following sample #LINK command invokes RHDCSNON, passing the ID and password stored in WKUSRID and WKPSWD:

```
          #LINK PGM='RHDCSNON', PARMS=(WKUSRID, WKPSWD, WKRCODE)
          .
          .
          .
WKUSRID   DS    CL32
WKPSWD    DS    CL8
WKRCODE   DS    H
```

More Information

- For more information about the SIGNON command, see the *CA IDMS System Tasks and Operator Commands Guide*.
- For more information about security, see the *CA IDMS Security Administration Guide*.

Chapter 6: Two-Phase Commit Support with RRS

This section contains the following topics:

Overview

RRS is IBM's resource recovery platform for z/OS. CA IDMS can exploit RRS services in the following ways:

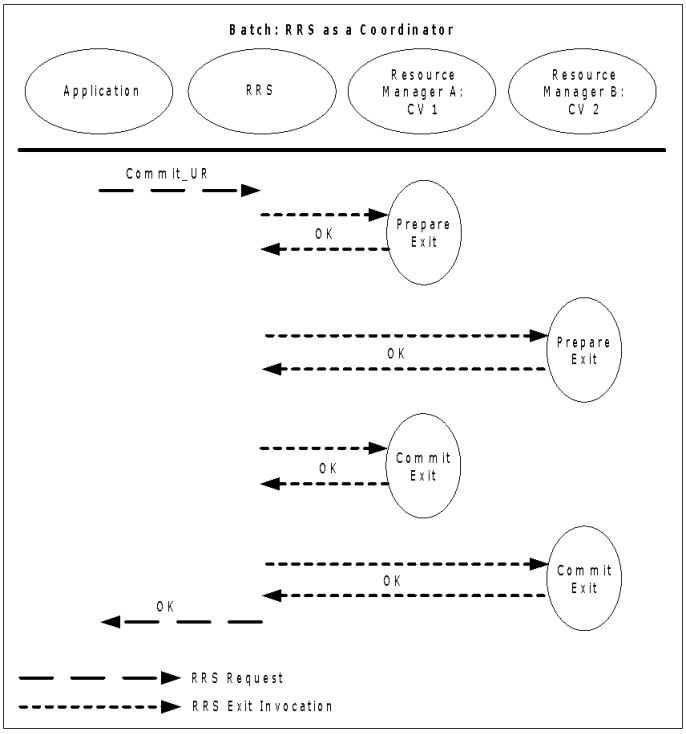
- A batch application can use RRS as a coordinator to ensure that the updates made through one or more central versions are coordinated with those of other resource managers such as MQSeries.
- An online application can update external resources through an RRS-enabled interface to ensure that those updates are coordinated with those made to CA IDMS resources.

This section discusses how RRS support is enabled and describes considerations associated with its use.

RRS Support for Batch Applications

A batch application updating resources controlled by multiple resource managers can make use of RRS services to guarantee atomicity of the updates. CA IDMS supports RRS for batch applications that make their database updates through one or more central versions running on the same operating system image as the batch job.

When RRS is used as the coordinator, each resource manager (RM), such as a CA IDMS central version that is accessed, expresses an interest in the unit of recovery (UR) controlled by RRS. To commit all changes as a unit, the application issues a Commit_UR (or an HLL Application_Commit_UR) request to RRS. The following diagram illustrates the flow of control that occurs:



Example

Consider a batch application that accesses CA IDMS and MQSeries and wants to coordinate the work done on each. To do this, the central version must be accessed through an RRS-enabled batch interface. The interface passes a context token to the central version so that it can express an interest in the UR associated with the context. At commit time, RRS invokes the central version's prepare and commit exits so that its work is coordinated with that of MQSeries.

Enabling RRS for Batch Applications

A batch application notifies CA IDMS that it wants to use RRS as a coordinator by specifying the following SYSIDMS parameter:

```
ENABLE_RRS=ON
```

CA IDMS then extracts the current context token and passes it on to the central version, which expresses interest in it.

If ENABLE_RRS=ON is established as a default in a SYSIDMS load module, it can be overridden at runtime by specifying the following parameter:

```
ENABLE_RRS=OFF
```

Note:

- The central version(s) to which the batch application's database sessions are directed must be started with RRS support and must be running on the same operating system image.
- It is not possible to access a pre-Release 16.0 central version if the batch job runs with RRS enabled. Local access is supported but is not part of the RRS UR.
- The 10.2 services batch interface (also known as IDML) does not support RRS.

Batch RRS Transaction Boundaries and Application Design Considerations

Batch applications that use RRS as a coordinator have to be carefully designed. The usage of RRS implies the following rules:

- The application verbs that mark a transaction boundary are the RRS verbs: Commit_UR or Backout_UR.
- Prior to issuing a Commit_UR, all database sessions whose transaction is under the control of RRS must be completed. This can be accomplished by performing the following tasks:
  - Issuing a FINISH TASK DML command
  - Explicitly finishing all active database sessions by issuing a FINISH or COMMIT RELEASE DML command for each one

Note: A FINISH TASK must be issued if a BIND TASK was issued.

Finishing a database session does not terminate its associated transaction when it is under the control of RRS; instead, the database session is closed and currency locks are released, but the transaction remains active and update locks are maintained until the RRS UR is committed or backed out.

It is possible to serially create and finish database sessions within a single RRS UR; however, unless transaction sharing is in effect, a deadlock may occur if a later session attempts to access a record that was updated by a previous session.

- When a ROLLBACK [TASK] DML command is issued, it results in the back out of the entire RRS UR, even if the application subsequently issues a Commit_UR request. At the time the ROLLBACK command is issued, the changes made to the CA IDMS database are backed out and the associated locks are released. However, the RRS UR is not backed out until an RRS commit or backout operation is initiated. If necessary, CA IDMS will initiate a BACKOUT operation during the first phase of commit processing to cause the RRS UR to be backed out.
- When an application program ends (normally or abnormally), the associated RRS context is terminated by the operating system. RRS default actions are to commit on normal context termination and backout on abnormal context termination.

Example of a COBOL Batch Program

The following extracts from a COBOL program show how to invoke the RRS Commit_UR and Backout_UR services. The COBOL program is a subroutine that is called to perform a certain action as defined in ACTION-CD. Only the CA IDMS task level and RRS actions are shown.

```
*RETRIEVAL
*NO-ACTIVITY-LOG
*DMLIST
 IDENTIFICATION DIVISION.
 PROGRAM-ID.           MBINDSUB.
*****************************************************************
*  SUBSCHEMA CONTROL IS PASSED FROM MAINLINE PROGRAM.
*****************************************************************
 ENVIRONMENT DIVISION.
 IDMS-CONTROL SECTION.
 PROTOCOL.               MODE IS BATCH DEBUG
                         IDMS-RECORDS MANUAL.
```

```
       DATA DIVISION.
       SCHEMA SECTION.
           DB EMPSS01 WITHIN EMPSCHM VERSION 100.
       WORKING-STORAGE SECTION.
       01  WK-DATA.
           02  I            PIC S9(4) COMP.
       01  COPY IDMS SUBSCHEMA-NAMES.
       01  COPY IDMS SUBSCHEMA-RECORDS.
       LINKAGE SECTION.
       01  DB-PARM.
           02  DBNAME-IN    PIC X(8).
           02  FILLER       PIC X.
           02  DBNODE-IN    PIC X(8).
           02  FILLER       PIC X.
           02  ACTION-CD    PIC X.
               88    ACT-BIND   VALUE 'R'.
               88    ACT-BINDU  VALUE 'U'.
               88    ACT-DML1   VALUE '1'.
               88    ACT-DML2   VALUE '2'.
               88    ACT-DML3   VALUE '3'.
               88    ACT-UPDT   VALUE '4'.
               88    ACT-FIN    VALUE 'F'.
               88    ACT-TCOM   VALUE 'C'.
               88    ACT-RCOM   VALUE 'D'.
               88    ACT-TFIN   VALUE 'X'.
               88    ACT-TBAK   VALUE 'B'.
               88    ACT-RBAK   VALUE 'Y'.
           02  RETURN-CD    PIC S9(8) COMP.
           ...
01  COPY IDMS SUBSCHEMA-CTRL.
       PROCEDURE DIVISION USING DB-PARM, SUBSCHEMA-CTRL.
       MAINLN SECTION.
           MOVE 0 TO RETURN-CD.
           ...
           IF ACT-BINDU
              PERFORM BIND-IT
           ELSE IF ACT-RCOM
              PERFORM RCOM-IT
           ELSE IF ACT-TFIN
              PERFORM TFIN-IT
           ELSE IF ACT-TBAK
              PERFORM TBAK-IT
           ELSE IF ACT-RBAK
              PERFORM RBAK-IT
           ELSE IF ...
           ...
           ELSE
              MOVE 32 TO RETURN-CD.
           GOBACK.
```

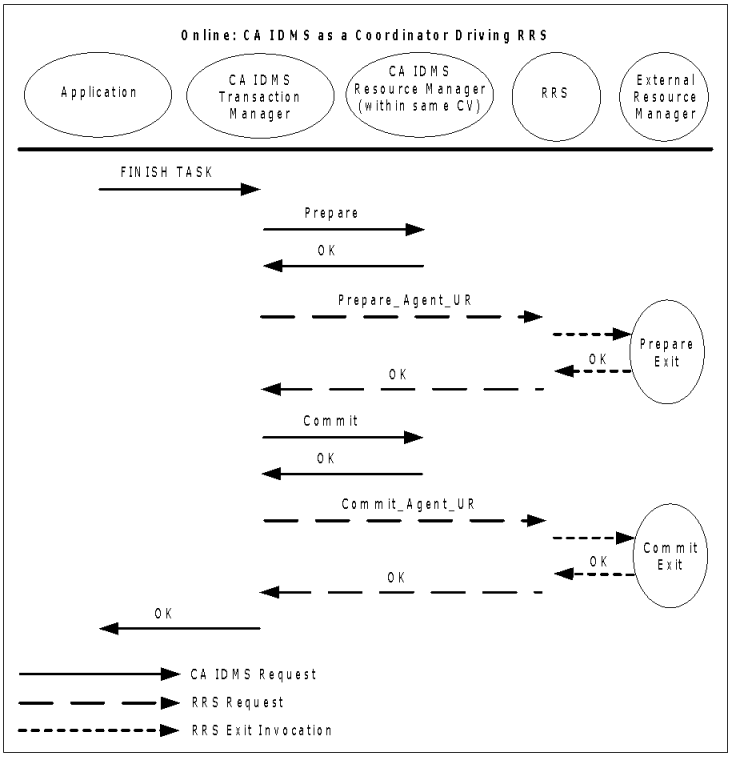
```
 BIND-IT SECTION.
     MOVE SPACES TO SUBSCHEMA-CTRL.
     MOVE 'MBINDSUB' TO PROGRAM-NAME.
     BIND RUN-UNIT DBNODE DBNODE-IN
                   DBNAME DBNAME-IN.
     READY USAGE-MODE UPDATE.
     PERFORM CHECK-STAT.
     BIND EMPLOYEE.
     PERFORM CHECK-STAT.
     BIND DEPARTMENT.
     PERFORM CHECK-STAT.
 ...
 TCOM-IT SECTION.
     COMMIT TASK.
     PERFORM CHECK-STAT.
 RCOM-IT SECTION.
*  Issue RRS Commit_UR
     CALL 'SRRCMIT' USING RETURN-CD.
     PERFORM CHECK-RRS.
 TFIN-IT SECTION.
     FINISH TASK.
     PERFORM CHECK-STAT.
 RBAK-IT SECTION.
*  Issue RRS Backout_UR
     CALL 'SRRBACK' USING RETURN-CD.
     PERFORM CHECK-RRS.
 ...
```

RRS Support for Online Applications

RRS can be used by an online application to ensure that updates made through external resource managers such as MQSeries are coordinated with those of CA IDMS. In order to exploit this functionality, the external resource manager must be accessed through its RRS-enabled interface.

Before accessing the external resource manager, the online task must establish a private RRS context. This context can then be passed to any external resource manager that wants to participate in the CA IDMS controlled transaction. Typically, online support for accessing external resources is provided by a third party vendor and, consequently, it is the vendor's responsibility to establish the private context and ensure that it is available to the external resource manager's RRS-enabled interface. The RRS-enabled interface passes the context to its resource manager so that it can register an interest in the context's UR.

To initiate a commit operation involving all interested resource managers, the online application issues a CA IDMS commit DML command (such as a FINISH TASK or a COMMIT WORK). The local transaction manager then uses RRS as an agent to coordinate its updates with those of the external resource managers.



Example

Consider an online application that accesses CA IDMS and MQSeries and wants to coordinate the work done on each. To do this, the application program invokes a third party interface to access MQSeries. The interface creates a private context (referred to as CTXPRIV) by calling IDMSIN01. Next, the interface accesses MQSeries through its RRS-enabled interface, specifying CTXPRIV.

The application program initiates a commit operation by issuing a DML command such as FINISH TASK. When this happens, the CA IDMS transaction manager becomes the coordinator and drives RRS as a participant. RRS in turn directs the actions of MQSeries in support of the commit operation.

Programming Interface

The RRSCTX IDMSIN01 function allows private context manipulation. It is designed for third party vendors who want to exploit the two-phase commit functionality. For a description of IDMSIN01 function RRSCTX, see Chapter 2: (see page 13).

Termination of a private context

If the private context is created by an SQL routine or database procedure that called IDMSIN01, the private context is terminated when the encompassing transaction is ended. Otherwise, the private context ends when the DC task ends.

Application Design Considerations

If an external resource manager, such as MQSeries, is invoked from within an SQL routine or database procedure, its work is committed or backed out when the encompassing transaction is committed or backed out. Otherwise, the work done by the external resource manager is committed when one of the following situations occur:

- A COMMIT TASK is executed
- A FINISH TASK is executed
- The online task ends normally

The work is backed out when one of the following situations occur:

- A ROLLBACK TASK is executed
- The online task ends abnormally

Appendix A: TCP/IP Error Codes

This section contains the following topics:

Return, Errno, and Reason Codes

The return code value returned by a call to the socket program interface can be a binary 0 (call successfully executed) or non-zero (an error occurred). In the latter case, the errno field explains why the function call failed. The situations that occur are described as follows:

- CA IDMS generates the error. Errno is set to a value in the range 12000-12999 as documented in the following table. The reason code is not used and is 0.
- The error is generated by operating system services. Errno and (where applicable) reason-code are documented in the appropriate operating system services documentation.
  - z/OS
    - *§ UNIX System Services - Messages and Codes*
    - *§ z/OS Communications Server - IP and SNA codes*
  - z/VM—See the *z/VM TCP/IP Programmer's Reference*
  - z/VSE
    - *§ Connectivity Systems TCP/IP for z/VSE: Programmer's Guide*
    - *§ Barnard Systems TCP/IP Tools*
    - *§ TCP/IP for z/VSE - IBM Program Setup and Supplementary Information*
- **z/VM systems:** For some errno codes returned by CA IDMS, the variable assigned to the RSNCODE parameter may contain the IPRCODE extracted from the corresponding IUCV parameter list. For the complete list of values, see the IPARML DSECT.
- **z/VSE and z/VM systems:** The value of some errno codes can differ from the equivalent standard POSIX value that is returned on z/OS. For example, the standard value for ETIMEDOUT errno code (connection timed out) is 1127, but is 60 on z/VM. The standard errno code is returned to the variable assigned to the ERRNO parameter. Applications must check the variable for common errno codes that are handled programatically. The errno code value returned by the operating system is saved in a variable assigned to the RSNCODE parameter.

ERRNO Numbers Set by the Socket Program Interface

The name shown in the following table is the EQUate symbol generated by the #SOCKET macro. The equivalent condition name in the SOCKET-CALL-INTERFACE record is prefixed with the following:

- SOCKET-ERRNO- for COBOL and CA ADS
- SOCKET_ERRNO_ for PL/I

Name	Value	Description
	1 - 11999	The ERRNO is generated by the operating system. See the appropriate operating system documentation.
RNOINPL	12000	Invalid #SOCKET parameter list
RNOINAEC	12001	Invalid ASYNCECB parameter
RNOINAII	12002	Invalid AINFOIN parameter
RNOINAIO	12003	Invalid AINFOOUT parameter
RNOINBF	12004	Invalid BUFFER parameter
RNOINBFL	12005	Invalid BUFFERL parameter
RNOINBKL	12006	Invalid BACKLOG parameter
RNOINCAL	12007	Invalid CANONAML parameter
RNOINCMD	12008	Invalid COMMAND parameter
RNOINDOM	12009	Invalid DOMAIN parameter
RNOINEL	12010	Invalid EXCELST parameter
RNOINELL	12011	Invalid EXCELSTL parameter
RNOINFLG	12012	Invalid FLAGS parameter
RNOINFMT	12013	Invalid FORMAT parameter
RNOINFLT	12014	Invalid FROMLTE parameter
RNOINHDL	12015	Invalid HANDLE parameter
RNOINHNA	12016	Invalid HOSTNAME parameter
RNOINHNL	12017	Invalid HOSTNAML parameter
RNOINHNT	12018	Invalid HOSTENTP parameter
RNOINHOW	12019	Invalid HOW parameter
RNOINIL	12020	Invalid IPADDRL parameter

Name	Value	Description
RNOINIP	12021	Invalid IPADDR parameter
RNOINIPS	12022	Invalid IPADDRS parameter
RNOINISL	12023	Invalid IPADDRSL parameter
RNOINLEV	12024	Invalid LEVEL parameter
RNOINMXP	12025	Invalid MAXPTERM parameter
RNOINMXT	12026	Invalid MAXTASK parameter
RNOINNA	12027	Invalid NAME parameter
RNOINNAL	12028	Invalid NAMEL parameter
RNOINNS	12029	Invalid NEWSOCK parameter
RNOINNSD	12030	Invalid NFDS parameter
RNOINONA	12031	Invalid OPTNAME parameter
RNOINOVA	12032	Invalid OPTVAL parameter
RNOINOVL	12033	Invalid OPTLEN parameter
RNOINPNA	12034	Invalid PROTNAME parameter
RNOINPNL	12035	Invalid PROTNAML parameter
RNOINPNT	12036	Invalid PROTENTP parameter
RNOINPNU	12037	Invalid PROTNUM parameter
RNOINPOR	12038	Invalid PORT parameter
RNOINRHL	12039	Invalid RETHNAML parameter
RNOINRIL	12040	Invalid RETIPASL parameter
RNOINRL	12041	Invalid READLST parameter
RNOINRLL	12042	Invalid READLSTL parameter
RNOINRLN	12043	Invalid RETLEN parameter
RNOINRND	12044	Invalid RETNFDS parameter
RNOINRNS	12045	Invalid RETNSTKS parameter
RNOINSA	12046	Invalid SOCKADDR parameter
RNOINSAL	12047	Invalid SOCKADDL parameter
RNOINSNA	12048	Invalid SERVNAME parameter
RNOINSNL	12049	Invalid SERVNAML parameter
RNOINSNT	12050	Invalid SERVENTP parameter

Name	Value	Description
RNOINSOC	12051	Invalid SOCK parameter
RNOINTLT	12052	Invalid TOLTE parameter
RNOINTYP	12053	Invalid TYPE parameter
RNOINWL	12054	Invalid WRITLST parameter
RNOINWLL	12055	Invalid WRITLSTL parameter
RNOINOPT	12056	Invalid OPTION parameter
RNOINTIM	12057	Invalid TIMEOUT parameter
RNOINARG	12058	Invalid ARGUMENT parameter
RNOINRV	12059	Invalid RETVAL parameter
RNOINECB	12060	Invalid ECB parameter
RNOINECL	12061	Invalid ECBLIST parameter
RNOINRSL	12062	Invalid RETSNAML parameter
RNOINBL	12063	Invalid BITLIST parameter
RNOINBLL	12064	Invalid BITLISTL parameter
RNOINBOR	12065	Invalid BITORDER parameter
RNOINNNA	12066	Invalid NODENAME parameter
RNOINSTA	12067	Invalid STATUS parameter
RNOINARL	12068	Invalid ARGUMENL parameter
RNO2BUFF	12100	Specify BUFFER and BUFFERL, or none of them
RNO2HNAM	12101	Specify HOSTNAME and HOSTNAML, or none of them
RNO2NAME	12102	Specify NAME and NAMEL, or none of them
RNO2PNAM	12103	Specify PROTNAME and PROTNAML, or none of them
RNO2SNAM	12104	Specify SERVNAME and SERVNAML, or none of them
RNO3HNAM	12105	Specify HOSTNAME/HOSTNAML/ RETHNAML, or none
RNO3SNAM	12106	Specify SERVNAME/SERVNAML/ RETSNAML, or none
RNORQHS	12107	HOSTNAME or SERVNAME (or both) is required
RNORQECB	12108	ECB or ECBLIST is required

Name	Value	Description
RNOXCECB	12109	ECB and ECBLIST are mutually exclusive
RNOIECBL	12110	Invalid ECB in ECBLIST
RNOINARQ	12111	Invalid asynchronous command request
RNOINAIS	12112	Invalid ADDRINFO structure
RNOSYSP1	12113	ASYNCECB and HANDLE are system parms
RNOINHDA	12114	Invalid area pointed to by HANDLE
RNOIIPA	12115	Invalid format for IP-address
RNOIIPA6	12116	Invalid format for IP-address (V6)
RNOFNS	12200	Function not supported by interface
RNOFRSVD	12201	Function reserved for the system
RNOCAAIO	12202	Cannot allocate an AIO parameter list
RNOCANSU	12203	Cannot assign new socket to user
RNOCRSFU	12204	Cannot remove socket from user table
RNOCSHNT	12205	Cannot save HOSTENT structure info
RNOCSAIO	12206	Cannot save ADDRINFO structure info
RNONAINF	12207	Cannot find ADDRINFO to free
RNONOLTE	12208	No LTE available from current TCE
RNOSLIND	12209	SOCKET line not defined
RNOSLINO	12210	SOCKET line not opened
RNOSLRCY	12211	TCP/IP has been recycled
RNOPINL	12212	Plug-in module not loaded
RNODRTCE	12213	Driver's TCE does not point to the PLE
RNOINEPI	12214	Invalid environment when entering the plug-in
RNOSENA	12215	Socket environment not active
RNOUSTCA	12216	User's socket table cannot be allocated
RNOUSTNE	12217	User's socket table does not exist
RNOSSTCA	12218	System's socket table cannot be allocated
RNOSSTNE	12219	System's socket table does not exist
RNOSTKNF	12220	Requested stack not found
RNOSTKNA	12221	Requested stack not active

Name	Value	Description
RNOSDTCE	12222	Socket Descriptor table cannot be extended
RNOCASWA	12223	Cannot allocate SELECT work area
RNOINSWA	12224	Inconsistent fields in SELECT work area
RNOSBLEM	12225	All SELECT bit lists are empty
RNOSNCSS	12226	All sockets not created under same stack
RNOCASBL	12227	Cannot allocate socket's bit list
RNOMAXSO	12228	Maximum number of sockets reached
RNOMAXST	12229	Maximum number of sockets per task reached
RNOCADNS	12230	Cannot allocate DNS work area
RNOINDNS	12231	Invalid response from DNS server
RNOPITNL	12232	(z/VSE only) Plugin table module not loaded
RNODDSNA	12233	No active DDSTCPIP PTE found for IDMS nodename
RNODDSNC	12234	Cannot build a DDS connection to IDMS nodename
RNODDSNF	12235	No free port found in PORT-RANGE
RNODDSMC	12236	Maximum number of connections reached
RNODDSRE	12237	Error during release of a DDS connection
RNONASTK	12238	No active stack found in the system
RNOCEXSI	12239	Connection on excluded stack ignored (internal)
RNOCSSNT	12240	Cannot save SERVENT structure info
RNOSRVNF	12241	Name/alias + protocol service not found
RNOPORNF	12242	Port number + protocol service not found
RNOSGNER	12243	SYSGEN internal error - wrong records counter
RNOSTKAA	12244	Requested stack already active
RNOSTKAI	12245	Requested stack already inactive
RNOCDSTK	12246	Cannot exclude the default stack
RNOSTKEX	12247	Owning stack has been dynamically excluded
RNOSENQ	12248	Socket environment is quiescing
RNOSFND	12249	Service file not defined in the CA IDMS system
RNOSTKCA	12250	System's Stack Table cannot be allocated

Name	Value	Description
RNOSTKNI	12251	System's Stack Table not initialized yet
RNODNSNA	12252	Internal DNS Resolver not available
RNOHIUCV	12300	HNDIUCV error
RNOCIUCV	12301	CMSIUCV error
RNOIUCVS	12302	IUCV error for a socket function
RNOSEVER	12303	IUCV connection severed by TCP/IP
RNOTOIUC	12304	Time-out during IUCV connection
	>12999	The ERRNO is generated by the operation system. See the appropriate operating system documentation.

HOSTENT Structure

The HOSTENT structure is returned by the GETHOSTBYADDR and GETHOSTBYNAME function calls.

Field	Description
Hostname	Address of hostname (null-terminated string)
Aliases	Address of a zero-terminated array of pointers to aliases, which are null-terminated strings
Address type	Address family of returned IP addresses (AF_INET or AF_INET6)
Address length	Length of returned IP addresses
Addresses	Address of a zero-terminated array of pointers to IP addresses

SERVENT Structure

The SERVENT structure is returned by the GETSERVBYNAME and GETSERVBYPORT function calls.

Field	Description
Service name	Address of a service name (null-terminated string).

Field	Description
Aliases	Address of a zero-terminated array of pointers to aliases, which are null-terminated strings.
Port	Port number associated with a service.
Protocol	Address of the protocol associated with a service (null-terminated string).

Socket Structure Descriptions

ADDRINFO Structure

The ADDRINFO structure is input and output to the GETADDRINFO function call.

Field	Description
Flags	A set of flags
Family	Address family (AF_INET or AF_INET6)
Socket type	Type of socket (STREAM or DATAGRAM)
Protocol	Protocol in use for the socket
SOCKADDR length	Length of SOCKADDR structure
Canonical name	Address of canonical name associated with input node name
SOCKADDR structure	Address of the SOCKADDR structure
New ADDRINFO	Address of next ADDRINFO structure

SOCKADDR Structure

The SOCKADDR structure describes the address of a socket. There are two versions of this structure: IPv4 and IPv6.

SOCKADDR for IPv4

Field	Description
Family	A 2-byte field describing the socket address family type: AF_INET
Port number	The port number for this socket

Field	Description
Address	The 4-byte IP address of the TCP/IP stack
Zeros	Eight bytes of binary zeros

SOCKADDR for IPv6

Field	Description
Family	A 2-byte field describing the socket address family type: AF_INET6
Port number	The port number for this socket
Flow	Flow information
Address	The 16-byte IP address of the TCP/IP stack
Scope ID	Scope identifier

TIMEVAL Structure

The TIMEVAL structure may be passed as input to the SELECT and SELECTX function calls in order to specify a wait interval.

Field	Description
Seconds	Number of seconds to wait
Microseconds	Number of microseconds to wait

Appendix B: TCP/IP Programming Examples

This section contains the following topics:

PL/I Examples

This section contains sample TCP/IP client and generic listener server programs written in PL/I.

PL/I TCP/IP Client Program

```
/*RETRIEVAL*/
/*DMLIST*/

/*****************************************************************/
/* The following program is an example of a TCP/IP client        */
/* program written in PL1.                                       */
/* The processing is the following:                              */
/*  - Create a socket for the client program.                    */
/*  - Convert the known dotted string format IPA to binary.      */
/*  - Find host information for connection.                      */
/*  - Establish a connection to the host listener.               */
/*  - Send message 1 to the listener (first 4 bytes = data length)*/
/*  - Read message 1 from  listener (first 4 bytes = data length) */
/*  - Send message 2 to the listener (first 4 bytes = data length)*/
/*  - Read message 2 from  listener (first 4 bytes = data length) */
/*  - Close socket and exit.                                     */
/*****************************************************************/
/* Notes for the PL/I compiler on VSE.                           */
/*  - in order to allow arithmetic operations on POINTER type    */
/*    variables, specify the LANGLVL(OS,SPROG) compiler option   */
/*  - there is no option to allow external names on 8 characters, */
/*    so replace all CALL IDMSOCKI by CALL IDMSOCK, as described  */
/*    in the Callable Services manual.                           */
/*****************************************************************/
```

```
PLICLI : PROC OPTIONS (REENTRANT,FETCHABLE);

 DCL MODE(IDMS_DC) DEBUG;
 DCL ADDR BUILTIN;
 DCL IDMSPLI  ENTRY OPTIONS(INTER,ASSEMBLER);
 DCL IDMSOCKI ENTRY OPTIONS(INTER,ASSEMBLER);
 DCL IDMSP    ENTRY;

 INCLUDE IDMS (SUBSCHEMA_CTRL);
 INCLUDE IDMS (SOCKET_CALL_INTERFACE);
 INCLUDE IDMS (SOCKET_MISC_DEFINITIONS);
 DCL 1 SOCKADDR1,
       3 INCLUDE IDMS (SOCKET-SOCKADDR-IN);
 DCL 1 AINFO1,
       3 INCLUDE IDMS (SOCKET_ADDRINFO);

 DCL 1 MSG01  CHAR (20) INIT (' Parameter string  :');
 DCL 1 MSG02  CHAR (20) INIT (' Socket descriptor :');
 DCL 1 MSG03  CHAR (20) INIT (' Resume count      :');
 DCL 1 MSG04  CHAR (15) INIT (' Starting read.');
 DCL 1 MSG05  CHAR (16) INIT (' Starting write.');
 DCL 1 MSG06  CHAR (16) INIT (' Closing socket.');
 DCL 1 MSG07  CHAR (20) INIT (' Socket return code:');
 DCL 1 MSG08  CHAR (20) INIT (' Socket reason code:');
 DCL 1 MSG09  CHAR (20) INIT (' Socket errno      :');
 DCL 1 MSG10  CHAR (20) INIT (' Buffer length     :');
 DCL 1 MSG11  CHAR (08) INIT (' Buffer:');
 DCL 1 MSG12  CHAR (22) INIT (' Data length too long.');

 DCL 1 MSG20  CHAR (19) INIT (' Calling GETHOSTID.');
 DCL 1 MSG21  CHAR (23) INIT (' Calling GETHOSTBYADDR.');
 DCL 1 MSG22  CHAR (21) INIT (' Calling GETADDRINFO.');
 DCL 1 MSG23  CHAR (22) INIT (' Calling FREEADDRINFO.');
 DCL 1 MSG24  CHAR (21) INIT (' Calling GETNAMEINFO.');
 DCL 1 MSG25  CHAR (16) INIT (' Calling SOCKET.');
 DCL 1 MSG26  CHAR (17) INIT (' Calling CONNECT.');
 DCL 1 MSG27  CHAR (20) INIT (' Calling GETSOCKOPT.');
 DCL 1 MSG28  CHAR (20) INIT (' Calling SETSOCKOPT.');
 DCL 1 MSG29  CHAR (19) INIT (' Calling GETSTACKS.');
 DCL 1 MSG30  CHAR (18) INIT (' Calling SETSTACK.');
 DCL 1 MSG31  CHAR (19) INIT (' Calling INET_PTON.');


 DCL 1 MSG97  CHAR (24) INIT (' Socket call successful.');
 DCL 1 MSG98  CHAR (19) INIT (' Socket call error.');
 DCL 1 MSG99,
       3 MSG99_1 CHAR (29) INIT (' Program PLICLI terminated.'),
       3 MSG99_2 CHAR (15) INIT (' Error count = '),
       3 MSG99_3 PIC '(4)9';
```

```
 /*****************************************************************/
 /* Modify DEST-PORT and IPA-HOST to connect to desired listener   */
 /* If the port number is greater than 32767 use the following     */
 /* DEST_PORT = (port number - 65536).                             */
 /*DCL 1 DEST_PORT           FIXED BINARY(15) INIT( (12345-65536) )*/
 /*****************************************************************/
DCL 1 DEST_PORT             FIXED BINARY(15) INIT(12345);
DCL 1 IPAHOST_REC,
      3 IPA_HOST CHAR (12) INIT ('255.255.25.2'),
      3 FILLER   CHAR (12) INIT (' '),
      3 IPA_HOSTL FIXED BINARY(31) INIT(16);
DCL 1 SOCKDESC     FIXED BINARY(31);
DCL 1 NIFLAGS      FIXED BINARY(31) INIT(0);
DCL 1 SNAPLEN      FIXED BINARY(31);
DCL 1 WK1          FIXED BINARY(31);
DCL 1 WK2          FIXED BINARY(31);
DCL 1 WK3          FIXED BINARY(31);
DCL 1 RETLEN       FIXED BINARY(31) INIT(0);
DCL 1 WK_LENGTH    FIXED BINARY(31);
DCL 1 WK_SUBSCRIPT FIXED BINARY(31);
DCL 1 WK_PTR       POINTER;
DCL 1 TEXT         CHAR(80) BASED(WK_PTR);
DCL 1 TERM_FLAG    FIXED BINARY(31) INIT(0);
DCL 1 ERROR_COUNT  FIXED BINARY(31) INIT(0);

DCL 1 RETURN_CODES,
      3 RETCD             FIXED BINARY(31),
      3 ERRNO             FIXED BINARY(31),
      3 RSNCD             FIXED BINARY(31);

DCL 1 IPADDR_REC,
      3 IPADDRBUFL        FIXED BINARY(31) INIT(16),
      3 IPADDRRETL        FIXED BINARY(31),
      3 IPADDRBUF         CHAR(16);


DCL 1 BUFFER,
      3 BUFLEN       FIXED BINARY(31),
      3 BUFTXT80     CHAR(80);

DCL 1 WORKW,
      3 WORK_WCC     CHAR(1),
      3 WORK         CHAR(80);

DCL 1 HOSTENTP POINTER;
DCL 1 HOSTENT1 BASED(HOSTENTP),
      3 INCLUDE IDMS (SOCKET_HOSTENT);
```

```
 DCL 1 HOSTENT_NAME   CHAR(64) BASED(HOSTENT_NAME_PTR);

 DCL 1 AINFOINP  POINTER;
 DCL 1 AINFOOUTP POINTER;
 DCL 1 AINFO2    BASED(AINFOOUTP),
       3 INCLUDE IDMS (SOCKET_ADDRINFO);

 DCL 1 HOST_IPA  FIXED BINARY(31) INIT(0);
/*****************************************************************/
/* Include also all the structures that we deliver in DLODPROT,   */
/* but that are not used by this test program.                    */
/*****************************************************************/
 INCLUDE IDMS (SOCKET_LISTENER_PARMS);
 INCLUDE IDMS (SOCKET_SOCKADDR_IN6);
 INCLUDE IDMS (SOCKET_TIMEVAL);


/*****************************************************************/
/* Create a socket in the communications domain                  */
/*****************************************************************/
WRITE LOG MESSAGE ID (9060300) PARMS FROM (MSG25) LENGTH (16);
CALL IDMSOCKI ( SOCKET_FUNCTION_SOCKET,
                SOCKET_RETCD,
                SOCKET_ERRNO,
                SOCKET_RSNCD,
                SOCKET_FAMILY_AFINET,
                SOCKET_TYPE_STREAM,
                SOCKET_PROTOCOL_TCP,
                SOCKDESC);
CALL TCP_CHECKRC;
IF (TERM_FLAG = 1) THEN GOTO TCP_EXIT;

/*****************************************************************/
/* Convert the IP address from dotted string format to binary.   */
/*****************************************************************/
WRITE LOG MESSAGE ID (9060300) PARMS FROM (MSG31) LENGTH (19);
CALL IDMSOCKI ( SOCKET_FUNCTION_INETPTON,
                SOCKET_RETCD,
                SOCKET_ERRNO,
                SOCKET_RSNCD,
                SOCKET_FAMILY_AFINET,
                IPA_HOST,
                IPA_HOSTL,
                HOST_IPA );
CALL TCP_CHECKRC;

/*****************************************************************/
/* Take the IP address and domain and resolve it through a name   */
/* server. If successful, return the information in a HOSTENT     */
```

```
/* structure.                                                        */
/*******************************************************************/
WRITE LOG MESSAGE ID (9060300) PARMS FROM (MSG21) LENGTH (23);
CALL IDMSOCKI ( SOCKET_FUNCTION_GETHOSTBYADDR,
                SOCKET_RETCD,
                SOCKET_ERRNO,
                SOCKET_RSNCD,
                HOST_IPA,
                SOCKET_IPADDR4L,
                SOCKET_FAMILY_AFINET,
                HOSTENTP);
CALL TCP_CHECKRC;

/*******************************************************************/
/* Connect  DEST_PORT                                              */
/*******************************************************************/
SOCKADDR1.SIN_FAMILY      = SOCKET_FAMILY_AFINET;
SOCKADDR1.SIN_PORT_NUMBER = DEST_PORT;
SOCKADDR1.SIN_ADDRESS     = HOST_IPA;
WRITE LOG MESSAGE ID (9060300) PARMS FROM (MSG26) LENGTH (17);
CALL IDMSOCKI ( SOCKET_FUNCTION_CONNECT,
                SOCKET_RETCD,
                SOCKET_ERRNO,
                SOCKET_RSNCD,
                SOCKDESC,
                SOCKADDR1,
                SOCKADDR_IN_LENGTH);
CALL TCP_CHECKRC;
IF (TERM_FLAG = 1) THEN DO;
  CALL TCP_CLOSE;
  GOTO TCP_EXIT;
  END;


/*******************************************************************/
/* Build two messages and send them to DEST_PORT                   */
/*******************************************************************/
BUFTXT80     = 'PLICLI TCP/IP test message number 00001 ';
BUFLEN       = 41;
WK_LENGTH    = 45;
WK_PTR       = ADDR(BUFLEN);
CALL TCP_WRITE;
IF (TERM_FLAG = 1) THEN GOTO TCP_EXIT;


/*******************************************************************/
/* Read the response from DEST_PORT                                */
/*******************************************************************/
WK_LENGTH = 4;
```

```
BUFLEN    = 0;
WK_PTR    = ADDR(BUFLEN);
CALL TCP_READ;
IF (TERM_FLAG = 1) THEN RETURN;
IF (BUFLEN > 80) THEN DO;
  WRITE LOG MESSAGE ID (9060300) PARMS FROM (MSG12) LENGTH (22);
  CALL TCP_CLOSE;
  RETURN;
  END;
WK_LENGTH = BUFLEN;
WK_PTR = ADDR(BUFTXT80);
CALL TCP_READ;
IF (TERM_FLAG = 1) THEN GOTO TCP_EXIT;
WORK = BUFTXT80;
WK1  = BUFLEN + 1;
WRITE LOG MESSAGE ID (9060300) PARMS FROM (MSG11) LENGTH (8)
                                     FROM (WORKW) LENGTH (WK1);

BUFTXT80     = 'PLICLI TCP/IP test message number 00002 ';
BUFLEN       = 41;
WK_LENGTH    = 45;
WK_PTR       = ADDR(BUFLEN);
CALL TCP_WRITE;
IF (TERM_FLAG = 1) THEN GOTO TCP_EXIT;

WK_LENGTH = 4;
BUFLEN    = 0;
WK_PTR    = ADDR(BUFLEN);
CALL TCP_READ;
IF (TERM_FLAG = 1) THEN GOTO TCP_EXIT;
IF (BUFLEN > 80) THEN DO;
  WRITE LOG MESSAGE ID (9060300) PARMS FROM (MSG12) LENGTH (22);
  CALL TCP_CLOSE;
  RETURN;
  END;
WK_LENGTH = BUFLEN;
WK_PTR = ADDR(BUFTXT80);
CALL TCP_READ;
IF (TERM_FLAG = 1) THEN RETURN;
WORK = BUFTXT80;
WK1  = BUFLEN + 1;
WRITE LOG MESSAGE ID (9060300) PARMS FROM (MSG11) LENGTH (8)
                                     FROM (WORKW) LENGTH (WK1);

/*****************************************************************/
/* Close the  socket and exit                                    */
/*****************************************************************/
CALL TCP_CLOSE;
GOTO TCP_EXIT;
```

```
TCP_EXIT:
   MSG99_3 = ERROR_COUNT;
   WRITE LINE TO TERMINAL FROM (MSG99) LENGTH (48);
   WRITE LOG MESSAGE ID (9060300) PARMS FROM (MSG99) LENGTH (48);
   RETURN;

/*********************************************************************/
/* Procedure to read a message from DEST_PORT                        */
/*********************************************************************/
TCP_READ: PROC;
  WRITE LOG MESSAGE ID (9060300) PARMS FROM (MSG04) LENGTH (15);
  DO WHILE (WK_LENGTH > 0);
    CALL IDMSOCKI ( SOCKET_FUNCTION_READ,
                    SOCKET_RETCD,
                    SOCKET_ERRNO,
                    SOCKET_RSNCD,
                    SOCKDESC,
                    WK_PTR->TEXT,
                    WK_LENGTH,
                    RETLEN);
    CALL TCP_CHECKRC;
    IF ((TERM_FLAG = 1) | (RETLEN = 0)) THEN DO;
      CALL TCP_CLOSE;
      RETURN;
      END;
    WK_PTR = WK_PTR + RETLEN;
    WK_LENGTH = WK_LENGTH - RETLEN;
    END;
END TCP_READ;

/*********************************************************************/
/* Procedure to send a message DEST_PORT                             */
/*********************************************************************/
TCP_WRITE: PROC;
  WRITE LOG MESSAGE ID (9060300) PARMS FROM (MSG05) LENGTH (16);
  DO WHILE (WK_LENGTH > 0);
    CALL IDMSOCKI ( SOCKET_FUNCTION_WRITE,
                    SOCKET_RETCD,
                    SOCKET_ERRNO,
                    SOCKET_RSNCD,
                    SOCKDESC,
                    WK_PTR->TEXT,
                    WK_LENGTH,
                    RETLEN);
    CALL TCP_CHECKRC;
    IF ((TERM_FLAG = 1) | (RETLEN = 0)) THEN DO;
      CALL TCP_CLOSE;
      RETURN;
```

```
      END;
    WK_PTR = WK_PTR + RETLEN;
    WK_LENGTH = WK_LENGTH - RETLEN;
    END;
END TCP_WRITE;

/*********************************************************************/
/* Procedure to close the socket                                     */
/*********************************************************************/
TCP_CLOSE: PROC;
  WRITE LOG MESSAGE ID (9060300) PARMS FROM (MSG06) LENGTH (16);
  CALL IDMSOCKI ( SOCKET_FUNCTION_CLOSE,
                  SOCKET_RETCD,
                  SOCKET_ERRNO,
                  SOCKET_RSNCD,
                  SOCKDESC);
  CALL TCP_CHECKRC;
END TCP_CLOSE;

/*********************************************************************/
/* Procedure to check the return codes                               */
/*********************************************************************/
TCP_CHECKRC: PROC;
  RETCD = SOCKET_RETCD;
  ERRNO = SOCKET_ERRNO;
  RSNCD = SOCKET_RSNCD;
  IF (RETCD ,= 0) THEN DO;
    TERM_FLAG = 1;
    ERROR_COUNT = ERROR_COUNT + 1;
    WRITE LOG MESSAGE ID (9060300) PARMS FROM (MSG98) LENGTH (19);
    SNAP FROM (RETURN_CODES) LENGTH (12);
    END;
  ELSE DO;
    TERM_FLAG = 0;
    WRITE LOG MESSAGE ID (9060300) PARMS FROM (MSG97) LENGTH (24);
    END;
END TCP_CHECKRC;

END PLICLI ;
```

PL/I TCP/IP Generic Listener Server Program

```
/*RETRIEVAL*/
/*DMLIST*/

/*****************************************************************/
/* The following program is an example of a TCP/IP generic       */
/* listener server program written in PL/I.                      */
/* The processing is the following:                              */
/*  - read a message from the client (first 4 bytes = data length)*/
/*  - send the message back to the client program                */
/*  - if the message text is equal to "STOP" or if the connection */
/*    is closed, then it closes its socket and return to the     */
/*    generic listener service.                                  */
/*  - if the message text is not equal to "STOP", then it returns */
/*    to the generic listener service without closing its socket. */
/*                                                               */
/* Notes for the PL/I compiler on VSE.                           */
/*  - in order to allow arithmetic operations on POINTER type    */
/*    variables, specify the LANGLVL(OS,SPROG) compiler option   */
/*  - there is no option to allow external names on 8 characters, */
/*    so replace all CALL IDMSOCKI by CALL IDMSOCK, as described  */
/*    in the Callable Services manual.                           */
/*****************************************************************/

PLILIS: PROC (P1, P2, P3)
         OPTIONS (REENTRANT,FETCHABLE);

/*****************************************************************/
/* Parameter list with which a listener program receives control  */
/*****************************************************************/
 DCL (P1,P2,P3)          POINTER;
 DCL SOCKET_PARMS        CHAR(80)      BASED (ADDR(P1));
 DCL SOCKET_DESCRIPTOR   FIXED BIN(31) BASED (ADDR(P2));
 DCL SOCKET_RESUME_COUNT FIXED BIN(31) BASED (ADDR(P3));

 DCL MODE(IDMS_DC) DEBUG;
 DCL ADDR BUILTIN;
 DCL IDMSPLI  ENTRY OPTIONS(INTER,ASSEMBLER);
 DCL IDMSOCKI ENTRY OPTIONS(INTER,ASSEMBLER);
 DCL IDMSP    ENTRY;
```

```
 INCLUDE IDMS (SUBSCHEMA_CTRL);
 INCLUDE IDMS (SOCKET_CALL_INTERFACE);

 DCL 1 MSG01  CHAR (20) INIT (' Parameter string  :');
 DCL 1 MSG02  CHAR (20) INIT (' Socket descriptor :');
 DCL 1 MSG03  CHAR (20) INIT (' Resume count      :');
 DCL 1 MSG04  CHAR (15) INIT (' Starting read.');
 DCL 1 MSG05  CHAR (16) INIT (' Starting write.');
 DCL 1 MSG06  CHAR (16) INIT (' Closing socket.');
 DCL 1 MSG07  CHAR (20) INIT (' Socket return code:');
 DCL 1 MSG08  CHAR (20) INIT (' Socket reason code:');
 DCL 1 MSG09  CHAR (20) INIT (' Socket errno      :');
 DCL 1 MSG10  CHAR (20) INIT (' Buffer length     :');
 DCL 1 MSG11  CHAR (08) INIT (' Buffer:');
 DCL 1 MSG12  CHAR (22) INIT (' Data length too long.');

 DCL 1 RETLEN         FIXED BINARY(31);
 DCL 1 WK_LENGTH      FIXED BINARY(31);
 DCL 1 WK_PTR         POINTER;
 DCL 1 TEXT           CHAR(80) BASED(WK_PTR);
 DCL 1 TERM_FLAG      FIXED BINARY(31) INITIAL(0);

 DCL 1 BUFFER,
       3 BUFLEN       FIXED BINARY(31),
       3 BUFTXT80     CHAR(80);

 DCL 1 WORKW,
       3 WORK_WCC     CHAR(1),
       3 WORK         CHAR(80);

/*********************************************************************/
/* Display the 3 input parameters                                    */
/*********************************************************************/
/*********************************************************************/
/* Read the first 4 bytes: will contain the remaining length         */
/*********************************************************************/
WK_LENGTH = 4;
BUFLEN = 0;
WK_PTR = ADDR(BUFLEN);
CALL TCP_READ;
IF (TERM_FLAG = 1) THEN RETURN;

/*********************************************************************/
/* Read the remaining data (maximum 80 characters are allowed)       */
/*********************************************************************/
IF (BUFLEN > 80)
  THEN DO;
    WRITE LOG MESSAGE ID (9060300)
      PARMS FROM (MSG12) LENGTH (22);
```

```
    CALL TCP_CLOSE;
    RETURN;
    END;

WK_LENGTH = BUFLEN;
WK_PTR = ADDR(BUFTXT80);
CALL TCP_READ;
IF (TERM_FLAG = 1) THEN RETURN;

WORK = BUFLEN;
WRITE LOG MESSAGE ID (9060300)
  PARMS FROM (MSG10) LENGTH (20)
        FROM (WORKW) LENGTH (15);
WORK = BUFTXT80;
WK_LENGTH = BUFLEN + 1;
WRITE LOG MESSAGE ID (9060300)
  PARMS FROM (MSG11) LENGTH (8)
        FROM (WORKW) LENGTH (WK_LENGTH);

/*********************************************************************/
/* Send the message back to the client                               */
/*********************************************************************/
WK_LENGTH = BUFLEN + 4;
WK_PTR = ADDR(BUFLEN);
CALL TCP_WRITE;

IF ((BUFLEN = 4) & (SUBSTR(BUFTXT80,1,4) = 'STOP'))
  THEN CALL TCP_CLOSE;

RETURN;

/*********************************************************************/
/* Procedure to read a message from the client                       */
/*********************************************************************/
TCP_READ: PROC;
  WRITE LOG MESSAGE ID (9060300)
    PARMS FROM (MSG04) LENGTH (15);
  DO WHILE (WK_LENGTH > 0);
    CALL IDMSOCKI ( SOCKET_FUNCTION_READ,
                    SOCKET_RETCD,
                    SOCKET_ERRNO,
                    SOCKET_RSNCD,
                    SOCKET_DESCRIPTOR,
                    WK_PTR->TEXT,
                    WK_LENGTH,
                    RETLEN);
    WORK = SOCKET_RETCD;
    WRITE LOG MESSAGE ID (9060300)
      PARMS FROM (MSG07) LENGTH (20)
```

```
            FROM (WORKW) LENGTH (15);
    IF ((SOCKET_RETCD ,= 0) | (RETLEN = 0))
      THEN DO;
        CALL TCP_ERROR;
        RETURN;
        END;
    WK_PTR = WK_PTR + RETLEN;
    WK_LENGTH = WK_LENGTH - RETLEN;
    END;
END TCP_READ;

/*********************************************************************/
/* Procedure to send a message to the client                         */
/*********************************************************************/
TCP_WRITE: PROC;
  WRITE LOG MESSAGE ID (9060300)
    PARMS FROM (MSG05) LENGTH (16);
  DO WHILE (WK_LENGTH > 0);
    CALL IDMSOCKI ( SOCKET_FUNCTION_WRITE,
                    SOCKET_RETCD,
                    SOCKET_ERRNO,
                    SOCKET_RSNCD,
                    SOCKET_DESCRIPTOR,
                    WK_PTR->TEXT,
                    WK_LENGTH,
                    RETLEN);
    WORK = SOCKET_RETCD;
    WRITE LOG MESSAGE ID (9060300)
      PARMS FROM (MSG07) LENGTH (20)
            FROM (WORKW) LENGTH (15);
    IF ((SOCKET_RETCD ,= 0) | (RETLEN = 0))
      THEN DO;
        CALL TCP_ERROR;
        RETURN;
        END;
    WK_PTR = WK_PTR + RETLEN;
    WK_LENGTH = WK_LENGTH - RETLEN;
    END;
END TCP_WRITE;

/*********************************************************************/
/* Procedure to close the socket                                     */
/*********************************************************************/
TCP_CLOSE: PROC;
  WRITE LOG MESSAGE ID (9060300)
    PARMS FROM (MSG06) LENGTH (16);
  CALL IDMSOCKI ( SOCKET_FUNCTION_CLOSE,
                  SOCKET_RETCD,
                  SOCKET_ERRNO,
```

```
                SOCKET_RSNCD,
                SOCKET_DESCRIPTOR);
  WORK = SOCKET_RETCD;
  WRITE LOG MESSAGE ID (9060300)
    PARMS FROM (MSG07) LENGTH (20)
          FROM (WORKW) LENGTH (15);
END TCP_CLOSE;

/*********************************************************************/
/* Procedure to process the socket call errors                       */
/*********************************************************************/
TCP_ERROR: PROC;
  WORK = SOCKET_RSNCD;
  WRITE LOG MESSAGE ID (9060300)
    PARMS FROM (MSG08) LENGTH (20)
          FROM (WORKW) LENGTH (15);
  WORK = SOCKET_ERRNO;
  WRITE LOG MESSAGE ID (9060300)
    PARMS FROM (MSG09) LENGTH (20)
          FROM (WORKW) LENGTH (15);
  WORK = RETLEN;
  WRITE LOG MESSAGE ID (9060300)
    PARMS FROM (MSG10) LENGTH (20)
          FROM (WORKW) LENGTH (15);
  CALL TCP_CLOSE;
  TERM_FLAG = 1;
END TCP_ERROR;

END PLILIS;
```

COBOL Examples

This section contains sample TCP/IP client and generic listener server programs written in COBOL.

COBOL TCP/IP Client Program

```
RETRIEVAL
NO-ACTIVITY-LOG
DMLIST

*****************************************************************
 The following program is an example of a TCP/IP client         *
 program written in COBOL.                                      *
 The processing is the following:                               *
  - Create a socket for the client program.                     *
  - Convert the known dotted string format IPA to binary.       *
  - Find host information for connection.                       *
  - Establish a connection to the host listener.                *
  - Send message 1 to the listener (first 4 bytes = data length)*
  - Read message 1 from  listener (first 4 bytes = data length) *
  - Send message 2 to the listener (first 4 bytes = data length)*
  - Read message 2 from  listener (first 4 bytes = data length) *
  - Close socket and exit.                                      *
*****************************************************************

IDENTIFICATION DIVISION.
PROGRAM-ID.              COBCLI.
ENVIRONMENT DIVISION.
IDMS-CONTROL SECTION.
PROTOCOL. MODE IS IDMS-DC DEBUG
          IDMS-RECORDS MANUAL.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  COPY IDMS SUBSCHEMA-CTRL.
01  COPY IDMS SOCKET-LISTENER-PARMS.
01  COPY IDMS SOCKET-SOCKADDR-IN6.
01  COPY IDMS SOCKET-TIMEVAL.
01  COPY IDMS RECORD SOCKET-CALL-INTERFACE.
01  COPY IDMS RECORD SOCKET-MISC-DEFINITIONS.
01  SOCKADDR1.
    02 COPY IDMS RECORD SOCKET-SOCKADDR-IN.
01  SOCKET-DESCRIPTOR     PIC S9(8) COMP.
*****************************************************************
 Modify DEST-PORT and IPA-HOST to connect to desired server      *
*****************************************************************
01  DEST-PORT        PIC  9(8) VALUE 12345.
01  IPAHOST-REC.
  02  IPA-HOST       PIC X(12) VALUE '255.255.25.2'.
  02  FILLER         PIC X(4) VALUE SPACES.
  02  IPA-HOSTL      PIC S9(8) COMP VALUE 16.
```

```
01  HOSTENTP  USAGE IS POINTER.
01  WK1              PIC S9(8) COMP.
01  WK2              PIC S9(8) COMP.
01  WK3              PIC S9(8) COMP.
01  WK-SUBSCRIPT     PIC S9(4) COMP.
01  WK-LENGTH        PIC S9(8) COMP.
01  RETLEN           PIC S9(8) COMP VALUE 0.
01  TERM-FLAG        PIC S9(8) COMP VALUE 0.
01  ERROR-COUNT      PIC S9(8) COMP VALUE 0.

01  BUFFER.
  03  BUFFER-ARRAY    PIC X(1) OCCURS 84 TIMES.
01  BUFFER-REDEF      REDEFINES BUFFER.
  03  BUFLEN          PIC 9(8) COMP.
  03  BUFTXT80        PIC X(80).
  03  BUFTXT80-REDEF1 REDEFINES BUFTXT80.
    05  BUFTXT04      PIC X(4).
    05  BUFTXT76      PIC X(76).
  03  BUFTXT80-REDEF2 REDEFINES BUFTXT80.
    05  BUFTXT-MSG    PIC X(41).
    05  BUFTXT-BLANK  PIC X(1).
    05  BUFTXT-FILLER PIC X(38).

01  WORKW.
  03  WORK-WCC       PIC X.
  03  WORK           PIC X(80).
  03  WORK-REDEF1    REDEFINES WORK.
    04  WORK-ARRAY   PIC X(1) OCCURS 80 TIMES.
  03  WORK-REDEF2    REDEFINES WORK.
    04  WORKNUM      PIC 9(8) DISPLAY.
    04  WORK-FILLER1 PIC X(72).

01  IPADDR-REC.
  02  IPADDRBUFL     PIC S9(8) COMP VALUE 16.
  02  IPADDRRETL     PIC S9(8) COMP.
  02  IPADDRBUF      PIC X(16).

01  RETURN-CODES.
  02  RETCD          PIC S9(8) COMP.
  02  ERRNO          PIC S9(8) COMP.
  02  RSNCD          PIC S9(8) COMP.

01  MSG01 PIC X(18) VALUE ' Creating Socket.'.
01  MSG02 PIC X(13) VALUE ' Connecting: '.
01  MSG03 PIC X(19) VALUE ' Socket return code'.
01  MSG04 PIC X(16) VALUE ' Starting read.'.
01  MSG05 PIC X(16) VALUE ' Starting write.'.
01  MSG06 PIC X(16) VALUE ' Closing Socket.'.
01  MSG07 PIC X(19) VALUE ' Socket reason code'.
```

```
01  MSG08 PIC X(19) VALUE ' Socket errno      '.
01  MSG10 PIC X(08) VALUE ' Buffer:'.
01  MSG11 PIC X(22) VALUE ' Data length too long.'.
01  MSG12 PIC X(19) VALUE ' Calling INET_PTON.'.
01  MSG13 PIC X(23) VALUE ' Calling GETHOSTBYADDR.'.
01  MSG97 PIC X(24) VALUE ' Socket call successful.'.
01  MSG98 PIC X(19) VALUE ' Socket call error.'.
01  MSG99.
  02  MSG99-1 PIC X(28) VALUE ' Program COBCLI terminated.'.
  02  MSG99-2 PIC X(15) VALUE ' Error count = '.
  02  MSG99-3 PIC  9(4) DISPLAY.

01  HOSTIPA              PIC  9(8) COMP.
LINKAGE SECTION.
01  COPY IDMS RECORD SOCKET-HOSTENT.
01  AINFO1.
    05 COPY IDMS RECORD SOCKET-ADDRINFO.

PROCEDURE DIVISION.

*****************************************************************
 Create a socket in the communications domain                   *
*****************************************************************
*
TCP-CLIENT-SOCKET.
*
    WRITE LOG MESSAGE ID 9060300
      PARMS FROM MSG01  LENGTH 18.
    CALL 'IDMSOCKI' USING SOCKET-FUNCTION-SOCKET,
                          SOCKET-RETCD,
                          SOCKET-ERRNO,
                          SOCKET-RSNCD,
                          SOCKET-FAMILY-AFINET,
                          SOCKET-TYPE-STREAM,
                          SOCKET-PROTOCOL-TCP,
                          SOCKET-DESCRIPTOR.
      PERFORM TCP-CLIENT-CHECKRC THRU TCP-CLIENT-CHECKRC-EXIT.
      IF TERM-FLAG = 1
        PERFORM TCP-CLIENT-CLOSE THRU TCP-CLIENT-CLOSE-EXIT
        GO TO TCP-CLIENT-EXIT.

*****************************************************************
 Convert the IP address from dotted string format to binary.    *
*****************************************************************
TCP-CLIENT-INETPTON.
      WRITE LOG MESSAGE ID 9060300 PARMS FROM MSG12 LENGTH 19.
      CALL 'IDMSOCKI' USING SOCKET-FUNCTION-INETPTON,
                            SOCKET-RETCD,
                            SOCKET-ERRNO,
```

```
                              SOCKET-RSNCD,
                              SOCKET-FAMILY-AFINET,
                              IPA-HOST,
                              IPA-HOSTL,
                              HOSTIPA.
      PERFORM TCP-CLIENT-CHECKRC THRU TCP-CLIENT-CHECKRC-EXIT.
*****************************************************************
 Take the IP address and domain and resolve it through a name   *
 server. If successful, return the information in a HOSTENT     *
 structure.                                                     *
*****************************************************************
TCP-CLIENT-GETHOST.
      WRITE LOG MESSAGE ID 9060300 PARMS FROM MSG13 LENGTH 23.
      CALL 'IDMSOCKI' USING SOCKET-FUNCTION-GETHOSTBYADDR,
                              SOCKET-RETCD,
                              SOCKET-ERRNO,
                              SOCKET-RSNCD,
                              HOSTIPA,
                              SOCKET-IPADDR4L,
                              SOCKET-FAMILY-AFINET,
                              HOSTENTP.
         PERFORM TCP-CLIENT-CHECKRC THRU TCP-CLIENT-CHECKRC-EXIT.

         SET ADDRESS OF SOCKET-HOSTENT TO HOSTENTP.

 TCP-CLIENT-CONNECT.
     SET ADDRESS OF SOCKET-HOSTENT TO HOSTENTP.
     WRITE LOG MESSAGE ID 9060300 PARMS FROM MSG02 LENGTH 13.
     MOVE SOCKET-FAMILY-AFINET TO SIN-FAMILY      OF SOCKADDR1.
     MOVE DEST-PORT            TO SIN-PORT-NUMBER OF SOCKADDR1.
     MOVE HOSTIPA              TO SIN-ADDRESS     OF SOCKADDR1.
     MOVE LOW-VALUES           TO SIN-ZEROS       OF SOCKADDR1.
     CALL 'IDMSOCKI' USING SOCKET-FUNCTION-CONNECT,
                           SOCKET-RETCD,
                           SOCKET-ERRNO,
                           SOCKET-RSNCD,
                           SOCKET-DESCRIPTOR,
                           SOCKADDR1,
                           SOCKADDR-IN-LENGTH.
     PERFORM TCP-CLIENT-CHECKRC THRU TCP-CLIENT-CHECKRC-EXIT.
     IF TERM-FLAG = 1
       PERFORM TCP-CLIENT-CLOSE THRU TCP-CLIENT-CLOSE-EXIT
       GO TO TCP-CLIENT-EXIT.

TCP-CLIENT-BUILD.
*
*****************************************************************
 Build and send first message to DEST-PORT                      *
*****************************************************************
```

```
    MOVE 'COBCLI - TCP/IP test message number 00001'
                                    TO BUFTXT-MSG.
    MOVE ' '                        TO BUFTXT-BLANK.
    MOVE 41 TO BUFLEN.
    MOVE 45 TO WK-LENGTH.
    MOVE 1  TO WK-SUBSCRIPT.
    PERFORM TCP-CLIENT-WRITE THRU TCP-CLIENT-WRITE-EXIT.
    IF TERM-FLAG = 1 GO TO TCP-CLIENT-EXIT.


****************************************************************
 Read the response from DEST-PORT                               *
****************************************************************
    MOVE 4 TO WK-LENGTH.
    MOVE 0 TO BUFLEN.
    MOVE 1 TO WK-SUBSCRIPT.
    PERFORM TCP-CLIENT-READ THRU TCP-CLIENT-READ-EXIT.
    IF TERM-FLAG = 1 GO TO TCP-CLIENT-EXIT.
    IF BUFLEN GREATER THAN 80
      WRITE LOG MESSAGE ID 9060300 PARMS FROM MSG11 LENGTH 22
      PERFORM TCP-CLIENT-CLOSE THRU TCP-CLIENT-CLOSE-EXIT
      GO TO TCP-CLIENT-EXIT.
    MOVE BUFLEN TO WK-LENGTH.
    MOVE 5      TO WK-SUBSCRIPT.
    PERFORM TCP-CLIENT-READ THRU TCP-CLIENT-READ-EXIT.
    IF TERM-FLAG = 1 GO TO TCP-CLIENT-EXIT.
    MOVE BUFTXT80 TO WORK.
    MOVE BUFLEN TO WK1.
    ADD 1 TO WK1.
    WRITE LOG MESSAGE ID 9060300 PARMS FROM MSG10 LENGTH 8
                                       FROM WORKW LENGTH WK1.
****************************************************************
 Build and send second message to DEST-PORT                     *
****************************************************************
    MOVE 'COBCLI - TCP/IP test message number 00002'
                                    TO BUFTXT-MSG.
    MOVE ' '                        TO BUFTXT-BLANK.
    MOVE 41 TO BUFLEN.
    MOVE 45 TO WK-LENGTH.
    MOVE 1  TO WK-SUBSCRIPT.
    PERFORM TCP-CLIENT-WRITE THRU TCP-CLIENT-WRITE-EXIT.
    IF TERM-FLAG = 1 GO TO TCP-CLIENT-EXIT.

****************************************************************
 Read the response from DEST-PORT                               *
****************************************************************
    MOVE 4 TO WK-LENGTH.
    MOVE 0 TO BUFLEN.
```

```
    MOVE 1 TO WK-SUBSCRIPT.
    PERFORM TCP-CLIENT-READ THRU TCP-CLIENT-READ-EXIT.
    IF TERM-FLAG = 1 GO TO TCP-CLIENT-EXIT.
    IF BUFLEN GREATER THAN 80
      WRITE LOG MESSAGE ID 9060300 PARMS FROM MSG11 LENGTH 22
      PERFORM TCP-CLIENT-CLOSE THRU TCP-CLIENT-CLOSE-EXIT
      GO TO TCP-CLIENT-EXIT.
    MOVE BUFLEN TO WK-LENGTH.
    MOVE 5      TO WK-SUBSCRIPT.
    PERFORM TCP-CLIENT-READ THRU TCP-CLIENT-READ-EXIT.
    IF TERM-FLAG = 1 GO TO TCP-CLIENT-EXIT.
    MOVE BUFTXT80 TO WORK.
    MOVE BUFLEN TO WK1.
    ADD 1 TO WK1.
    WRITE LOG MESSAGE ID 9060300 PARMS FROM MSG10 LENGTH 8
                                       FROM WORKW LENGTH WK1.


TCP-CLIENT-CLOSE-IT.
*****************************************************************
 Close the socket and exit                                      *
*****************************************************************
    PERFORM TCP-CLIENT-CLOSE THRU TCP-CLIENT-CLOSE-EXIT.
    GO TO TCP-CLIENT-EXIT.

TCP-CLIENT-EXIT.
    MOVE ERROR-COUNT TO MSG99-3.
    WRITE LINE TO TERMINAL FROM MSG99 LENGTH 48.
    WRITE LOG MESSAGE ID 9060300 PARMS FROM MSG99 LENGTH 48.
    GOBACK.
*

*****************************************************************
 Procedure to read a message from DEST-PORT                     *
*****************************************************************
TCP-CLIENT-READ.
    WRITE LOG MESSAGE ID 9060300 PARMS FROM MSG04  LENGTH 15.
    PERFORM UNTIL WK-LENGTH = 0
      CALL 'IDMSOCKI' USING SOCKET-FUNCTION-READ,
                            SOCKET-RETCD,
                            SOCKET-ERRNO,
                            SOCKET-RSNCD,
                            SOCKET-DESCRIPTOR,
                            BUFFER-ARRAY(WK-SUBSCRIPT),
                            WK-LENGTH,
                            RETLEN
      PERFORM TCP-CLIENT-CHECKRC THRU TCP-CLIENT-CHECKRC-EXIT
      IF TERM-FLAG = 1 OR RETLEN = 0
        PERFORM TCP-CLIENT-CLOSE THRU TCP-CLIENT-CLOSE-EXIT
```

```
        GO TO TCP-CLIENT-READ-EXIT
        END-IF
      ADD RETLEN TO WK-SUBSCRIPT
      SUBTRACT RETLEN FROM WK-LENGTH
    END-PERFORM.
TCP-CLIENT-READ-EXIT.
    EXIT.

*****************************************************************
 Procedure to send a message to DEST_PORT                       *
*****************************************************************
TCP-CLIENT-WRITE.
    WRITE LOG MESSAGE ID 9060300 PARMS FROM MSG05 LENGTH 16.
    PERFORM UNTIL WK-LENGTH = 0
      CALL 'IDMSOCKI' USING SOCKET-FUNCTION-WRITE,
                            SOCKET-RETCD,
                            SOCKET-ERRNO,
                            SOCKET-RSNCD,
                            SOCKET-DESCRIPTOR,
                            BUFFER-ARRAY(WK-SUBSCRIPT),
                            WK-LENGTH,
                            RETLEN
      PERFORM TCP-CLIENT-CHECKRC THRU TCP-CLIENT-CHECKRC-EXIT
      IF TERM-FLAG = 1 OR RETLEN = 0
        PERFORM TCP-CLIENT-CLOSE THRU TCP-CLIENT-CLOSE-EXIT
        GO TO TCP-CLIENT-WRITE-EXIT
        END-IF
      ADD RETLEN TO WK-SUBSCRIPT
      SUBTRACT RETLEN FROM WK-LENGTH
    END-PERFORM.
TCP-CLIENT-WRITE-EXIT.
    EXIT.

*****************************************************************
 Procedure to close the socket                                  *
*****************************************************************
TCP-CLIENT-CLOSE.
    WRITE LOG MESSAGE ID 9060300 PARMS FROM MSG06 LENGTH 16.
    CALL 'IDMSOCKI' USING SOCKET-FUNCTION-CLOSE,
                          SOCKET-RETCD,
                          SOCKET-ERRNO,
                          SOCKET-RSNCD,
                          SOCKET-DESCRIPTOR.
    PERFORM TCP-CLIENT-CHECKRC THRU TCP-CLIENT-CHECKRC-EXIT.

TCP-CLIENT-CLOSE-EXIT.
    EXIT.

*****************************************************************
```

```
 Procedure to check the return codes                          *
*****************************************************************
TCP-CLIENT-CHECKRC.
    MOVE SOCKET-RETCD TO RETCD.
    MOVE SOCKET-ERRNO TO ERRNO.
    MOVE SOCKET-RSNCD TO RSNCD.
    IF RETCD NOT = 0
      MOVE 1 TO TERM-FLAG
      ADD  1 TO ERROR-COUNT
      WRITE LOG MESSAGE ID 9060300 PARMS FROM MSG98 LENGTH 19
      SNAP FROM RETURN-CODES LENGTH 12
    ELSE
      MOVE 0 TO TERM-FLAG
      WRITE LOG MESSAGE ID 9060300 PARMS FROM MSG97 LENGTH 24
    END-IF.
TCP-CLIENT-CHECKRC-EXIT.
    EXIT.

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

    COPY IDMS IDMS-STATUS.
IDMS-ABORT SECTION.
IDMS-ABORT-EXIT.
    EXIT.
```

COBOL TCP/IP Generic Listener Server Program

```
RETRIEVAL
NO-ACTIVITY-LOG
DMLIST

*****************************************************************
 The following program is an example of a TCP/IP generic       *
 listener server program written in COBOL.                     *
 The processing is the following:                              *
  - read a message from the client (first 4 bytes = data length)*
  - send the message back to the client program                 *
  - if the message text is equal to "STOP" or if the connection *
    is closed, then it closes its socket and return to the     *
    generic listener service.                                  *
  - if the message text is not equal to "STOP", then it returns *
    to the generic listener service without closing its socket. *
*****************************************************************
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID.              COBLIS.
ENVIRONMENT DIVISION.
IDMS-CONTROL SECTION.
PROTOCOL. MODE IS IDMS-DC DEBUG
          IDMS-RECORDS MANUAL.

DATA DIVISION.

WORKING-STORAGE SECTION.

01  COPY IDMS SUBSCHEMA-CTRL.
01  COPY IDMS RECORD SOCKET-CALL-INTERFACE.

01  MSG01  PIC X(20) VALUE ' Parameter string  :'.
01  MSG02  PIC X(20) VALUE ' Socket descriptor :'.
01  MSG03  PIC X(20) VALUE ' Resume count      :'.
01  MSG04  PIC X(15) VALUE ' Starting read.'.
01  MSG05  PIC X(16) VALUE ' Starting write.'.
01  MSG06  PIC X(16) VALUE ' Closing socket.'.
01  MSG07  PIC X(20) VALUE ' Socket return code:'.
01  MSG08  PIC X(20) VALUE ' Socket reason code:'.
01  MSG09  PIC X(20) VALUE ' Socket errno      :'.
01  MSG10  PIC X(20) VALUE ' Buffer length     :'.
01  MSG11  PIC X(08) VALUE ' Buffer:'.
01  MSG12  PIC X(22) VALUE ' Data length too long.'.

01  RETLEN           PIC S9(8) COMP.
01  WK-LENGTH        PIC S9(8) COMP.
01  WK-SUBSCRIPT     PIC S9(4) COMP.
01  TERM-FLAG        PIC S9(8) COMP VALUE 0.

01  BUFFER.
  03  BUFFER-ARRAY   PIC X(1) OCCURS 84 TIMES.
01  BUFFER-REDEF     REDEFINES BUFFER.
  03  BUFLEN         PIC 9(8) COMP.
  03  BUFTXT80       PIC X(80).
  03  BUFTXT80-REDEF REDEFINES BUFTXT80.
    05  BUFTXT04     PIC X(4).
    05  BUFTXT76     PIC X(76).

01  WORKW.
  03  WORK-WCC       PIC X.
  03  WORK           PIC X(80).

LINKAGE SECTION.
*****************************************************************
 Parameter list with which a listener program receives control  *
*****************************************************************
```

```
01  SOCKET-PARMS         PIC X(80).
01  SOCKET-DESCRIPTOR    PIC S9(8) COMP.
01  SOCKET-RESUME-COUNT  PIC S9(8) COMP.

PROCEDURE DIVISION USING SOCKET-PARMS,
                         SOCKET-DESCRIPTOR,
                         SOCKET-RESUME-COUNT.

*****************************************************************
 Display the 3 input parameters                                 *
*****************************************************************
TCP-START.

*****************************************************************
 Read the first 4 bytes: will contain the remaining length      *
*****************************************************************
    MOVE 4 TO WK-LENGTH.
    MOVE 0 TO BUFLEN.
    MOVE 1 TO WK-SUBSCRIPT.
    PERFORM TCP-READ THRU TCP-READ-EXIT.
    IF TERM-FLAG = 1 GO TO TCP-EXIT.

*****************************************************************
 Read the remaining data (maximum 80 characters are allowed)    *
*****************************************************************
    IF BUFLEN GREATER THAN 80
      WRITE LOG MESSAGE ID 9060300
        PARMS FROM MSG12 LENGTH 22
      PERFORM TCP-CLOSE THRU TCP-CLOSE-EXIT
      GO TO TCP-EXIT.

    MOVE BUFLEN TO WK-LENGTH.
    MOVE 5      TO WK-SUBSCRIPT.
    PERFORM TCP-READ THRU TCP-READ-EXIT.
    IF TERM-FLAG = 1 GO TO TCP-EXIT.

    MOVE BUFLEN TO WORK.
    WRITE LOG MESSAGE ID 9060300
      PARMS FROM MSG10 LENGTH 20
            FROM WORKW LENGTH 9.
    MOVE BUFTXT80 TO WORK.
    MOVE BUFLEN TO WK-LENGTH.
    ADD 1 TO WK-LENGTH.
    WRITE LOG MESSAGE ID 9060300
      PARMS FROM MSG11 LENGTH 8
            FROM WORKW LENGTH WK-LENGTH.

*****************************************************************
 Send the message back to the client                            *
```

```
*****************************************************************
    MOVE BUFLEN TO WK-LENGTH.
    ADD 4 TO WK-LENGTH.
    MOVE 1 TO WK-SUBSCRIPT.
    PERFORM TCP-WRITE THRU TCP-WRITE-EXIT.

    IF BUFLEN = 4 AND BUFTXT04 = 'STOP'
      PERFORM TCP-CLOSE THRU TCP-CLOSE-EXIT.

TCP-EXIT.
    GOBACK.

*****************************************************************
 Procedure to read a message from the client                    *
*****************************************************************
TCP-READ.
    WRITE LOG MESSAGE ID 9060300
      PARMS FROM MSG04  LENGTH 15.
    PERFORM UNTIL WK-LENGTH = 0
      CALL 'IDMSOCKI' USING SOCKET-FUNCTION-READ,
                            SOCKET-RETCD,
                            SOCKET-ERRNO,
                            SOCKET-RSNCD,
                            SOCKET-DESCRIPTOR,
                            BUFFER-ARRAY(WK-SUBSCRIPT),
                            WK-LENGTH,
                            RETLEN
      MOVE SOCKET-RETCD TO WORK
      WRITE LOG MESSAGE ID 9060300
        PARMS FROM MSG07 LENGTH 20
              FROM WORKW LENGTH 9
      IF SOCKET-RETCD NOT = 0 OR RETLEN = 0
        PERFORM TCP-ERROR THRU TCP-ERROR-EXIT
        GO TO TCP-READ-EXIT
        END-IF
      ADD RETLEN TO WK-SUBSCRIPT
      SUBTRACT RETLEN FROM WK-LENGTH
    END-PERFORM.
TCP-READ-EXIT.
    EXIT.

*****************************************************************
 Procedure to send a message to the client                      *
*****************************************************************
TCP-WRITE.
    WRITE LOG MESSAGE ID 9060300
      PARMS FROM MSG05 LENGTH 16.
    PERFORM UNTIL WK-LENGTH = 0
      CALL 'IDMSOCKI' USING SOCKET-FUNCTION-WRITE,
```

```
                              SOCKET-RETCD,
                              SOCKET-ERRNO,
                              SOCKET-RSNCD,
                              SOCKET-DESCRIPTOR,
                              BUFFER-ARRAY(WK-SUBSCRIPT),
                              WK-LENGTH,
                              RETLEN
      MOVE SOCKET-RETCD TO WORK
      WRITE LOG MESSAGE ID 9060300
        PARMS FROM MSG07 LENGTH 20
              FROM WORKW LENGTH 9
      IF SOCKET-RETCD NOT = 0 OR RETLEN = 0
        PERFORM TCP-ERROR THRU TCP-ERROR-EXIT
        GO TO TCP-WRITE-EXIT
        END-IF
      ADD RETLEN TO WK-SUBSCRIPT
      SUBTRACT RETLEN FROM WK-LENGTH
    END-PERFORM.
TCP-WRITE-EXIT.
    EXIT.

*****************************************************************
 Procedure to close the socket                                  *
*****************************************************************
TCP-CLOSE.
    WRITE LOG MESSAGE ID 9060300
      PARMS FROM MSG06 LENGTH 16.
    CALL 'IDMSOCKI' USING SOCKET-FUNCTION-CLOSE,
                          SOCKET-RETCD,
                          SOCKET-ERRNO,
                          SOCKET-RSNCD,
                          SOCKET-DESCRIPTOR.
    MOVE SOCKET-RETCD TO WORK.
    WRITE LOG MESSAGE ID 9060300
      PARMS FROM MSG07 LENGTH 20
            FROM WORKW LENGTH 9.
TCP-CLOSE-EXIT.
    EXIT.

*****************************************************************
 Procedure to process the socket call errors                    *
*****************************************************************
TCP-ERROR.
    MOVE SOCKET-RSNCD TO WORK.
    WRITE LOG MESSAGE ID 9060300
       PARMS FROM MSG08 LENGTH 20
             FROM WORKW LENGTH 9.
    MOVE SOCKET-ERRNO TO WORK.
    WRITE LOG MESSAGE ID 9060300
```

```
       PARMS FROM MSG09 LENGTH 20
             FROM WORKW LENGTH 9.
    MOVE RETLEN  TO WORK.
    WRITE LOG MESSAGE ID 9060300
       PARMS FROM MSG10 LENGTH 20
             FROM WORKW LENGTH 9.
    PERFORM TCP-CLOSE THRU TCP-CLOSE-EXIT.
    MOVE 1 TO TERM-FLAG.
TCP-ERROR-EXIT.
    EXIT.

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

    COPY IDMS IDMS-STATUS.
IDMS-ABORT SECTION.
IDMS-ABORT-EXIT.
    EXIT.
```

Assembler Examples

This section contains sample TCP/IP client and generic listener server programs written in Assembler.

Assembler TCP/IP Client Program

```
TITLE 'Sample ASSEMBLER client TCP/IP program'
* ASMCLI   RENT EP=ASMCLIEP
*****************************************************************
* The following program is an example of a TCP/IP client        *
* client program written in Assembler.                          *
* The processing is the following:                              *
*  - Create a socket for the client program.                    *
*  - Convert the known dotted string format IPA to binary.      *
*  -   (Host IPA is defined in IPADOT, see below.)              *
*  - Find host information for connection.                      *
*  -   (Host port is defined in DESTPORT, see below.)           *
*  - Establish a connection to the host listener.               *
*  - Send message 1 to the listener (first 4 bytes = data length)*
*  - Read message 1 from  listener (first 4 bytes = data length) *
*  - Send message 2 to the listener (first 4 bytes = data length)*
*  - Read message 2 from  listener (first 4 bytes = data length) *
*  - Close socket and exit.                                     *
*****************************************************************
```

```
*--------------------------------------------------------------------*
         MACRO
&LABEL.   #SAVEREG
&LABEL.   ST    R12,0(,R13)         Save R12
         ST    R14,4(,R13)         Save R14
         STM   R2,R8,8(R13)        Save R2-R8
         LA    R13,9*4(,R13)
         MEND
*--------------------------------------------------------------------*
         MACRO
&LABEL.   #RESTREG
&LABEL.   LA    R12,9*4             Get register stack entry length
         SR    R13,R12             Get A(previous register stack entry)
         L     R12,0(,R13)         Restore R12
         L     R14,4(,R13)         Restore R14
         LM    R2,R8,8(R13)        Restore R2-R8
         MEND
*--------------------------------------------------------------------*
         MACRO
&LABEL.   MSGTXT &TXT.
         LCLC  &TMP.
&TMP.     SETC  '&SYSNDX'.
&LABEL.   DC    AL1(L2&TMP).
L1&TMP.   DC    C&TXT.
L2&TMP.   EQU   *-L1&TMP.
         MEND
*--------------------------------------------------------------------*
ASMCLI   CSECT
         @MODE MODE=IDMSDC
         #MOPT CSECT=ASMCLI,ENV=USER,RMODE=ANY,AMODE=ANY
         ENTRY ASMCLIEP
ASMCLIEP DS    0H
         BALR  R12,0
         BCTR  R12,0
         BCTR  R12,0
         USING ASMCLIEP,R12
         B     TCPSTART            Branch around constants
*--------------------------------------------------------------------*
* Modify the following connection parameters before compiling        -*
*--------------------------------------------------------------------*
         DS    0F
DESTPORT DC    F'12345'             Known host port for connection
IPADOT   DC    CL13'255.255.00.01'  Known IP address of host
         DC    CL4'   '             filler
IPADOTL  DC    F'17'                Total length of dotted string IPA
MAX_LOOP DC    F'2'                 Maximum message count to send
*
TCPSTART DS    0H
*
```

```
         #GETSTG TYPE=(USER,SHORT),ADDR=(R1),PLIST=*,INIT=X'0',        X
               LEN=WORKAREL
         LR    R11,R1
         USING WORKAREA,R11
         LA    R13,REGSTACK          R13 -> Register stack
*  Initialize some WORKAREA fields
         XR    R5,R5
         ST    R5,ERRCOUNT
         MVI   OUTAREA,L'OUTAREAT
         MVC   OVRLOG,=X'8000000000'
         MVC   OVRLOGCO,=X'C000000000'
         MVC   WKCLEAR,=XL8'0F0F0F0F0F0F0F0F'
         MVC   TRTAB,=CL16'0123456789ABCDEF'
         MVC   TRTABX,=XL6'FAFBFCFDFEFF'
*****************************************************************
* Create a socket in the communications domain                  *
*****************************************************************
         LA    R1,MSG01            Display socket function.
         L     R15,=A(DISLINE)
         BALR  R14,R15
         #SOCKET SOCKET,                                         X
               DOMAIN=AF@INET,                                   X
               TYPE=STREAM,                                      X
               PROTNUM=6,                                        X
               NEWSOCK=S_NEWSOC,                                 X
               RETCODE=RETCODE,ERRNO=ERRNO,RSNCODE=RSNCODE
         LA    R1,MSG02            Display results of socket function.
         L     R15,=A(DISRC)
         BALR  R14,R15             Display the 3 return codes.
         CLC   RETCODE,=F'0'       Socket function successful?
         BNE   TCPCLOSE             N. Close socket and exit.
*****************************************************************
* Convert the IP address from dotted string format to binary.    *
*****************************************************************
         LA    R1,MSG12            Display INET_PTON socket function.
         L     R15,=A(DISLINE)
         BALR  R14,R15
         #SOCKET INET_PTON,DOMAIN=AF@INET,                        X
               IPADDRS=IPADOT,IPADDRSL=IPADOTL,IPADDR=HOSTIPA,   X
               RETCODE=RETCODE,ERRNO=ERRNO,RSNCODE=RSNCODE
         LA    R1,MSG13            Display INET_PTON results.
         L     R15,=A(DISRC)
         BALR  R14,R15             Display the 3 return codes.
*****************************************************************
* Take the IP address and domain and resolve it through a name   *
* server. If successful, return the information in a HOSTENT     *
* structure.                                                     *
*****************************************************************
         LA    R1,MSG14             Display GETHOSTBYADDR function.
```

```
         L     R15,=A(DISLINE)
         BALR  R14,R15
         LA    R2,BUFFER            R2 -> buffer for host information
         #SOCKET GETHOSTBYADDR,IPADDR=HOSTIPA,IPADDRL=4,               X
               DOMAIN=AF@INET,HOSTENTP=(R2),                          X
               RETCODE=RETCODE,ERRNO=ERRNO,RSNCODE=RSNCODE
         LA    R1,MSG15             Display GETHOSTBYADDR results
         L     R15,=A(DISRC)
         BALR  R14,R15              Display the 3 return codes
         CLC   RETCODE,=F'0'        GETHOSTBYADDR successful?
         BNE   TCPCLOSE              N. Close socket and exit.
*********************************************************************
* Prepare to connect to DESTPORT                                     *
*********************************************************************
         LA    R5,SOCKADDC          R5 -> Socket address structure.
         USING SOCK@IN,R5
         MVI   SIN@FAM,AF@INET       Get the family,
         MVC   SIN@ADDR(4),HOSTIPA         binary IPA, and
         MVC   SIN@PORT(2),DESTPORT+2       port number
         LA    R1,MSG03                       for the Host connect.
         L     R15,=A(DISLINE)       Display socket connect function.
         BALR  R14,R15
*********************************************************************
* Connect to DESTPORT                                                *
*********************************************************************
         #SOCKET CONNECT,                                             X
               SOCK=S_NEWSOC,                                         X
               SOCKADDR=(R5),                                         X
               SOCKADDL=SIN#LEN,                                      X
               RETCODE=RETCODE,ERRNO=ERRNO,RSNCODE=RSNCODE
         LA    R1,MSG04
         L     R15,=A(DISRC)
         BALR  R14,R15              Display the 3 return codes.
         CLC   RETCODE,=F'0'        Connect successful?
         BNE   TCPCLOSE              N. Close socket and exit.
*********************************************************************
* Write and read two messages to/from DESTPORT                       *
*********************************************************************
WRLOOP   DS    0H
         L     R5,LOOP_CNT              Add 1
         LA    R5,1(R5)                    to current
         ST    R5,LOOP_CNT                     message count
         ST    R5,WORK1                             and
         UNPK  WORK2(9),WORK1(5)                      make
         NC    WORK2(8),WKCLEAR                        it
         TR    WORK2(8),TRTAB                      displayable.
         MVC   BUFTXT_S(1),WORK2+7
         LA    R7,BUFFER_A              R7 -> buffer array.
*                                       Build client message.
```

```
*****************************************************************
* Send a message to the listener                                *
*****************************************************************
         MVC   BUFTXT_M(33),=C'ASMCLI TCP/IP test message number'
         MVC   BUFTXT_B(1),=C' '        Blank character.
         MVC   BUFLEN(4),=F'43'         Length of message text.
         MVC   WK_LEN(4),=F'47'         Total length of message.
         BAL   R4,TCPWRITE              Send message to listener.
*
*****************************************************************
* Read the first 4 bytes of reply: contains the remaining length *
*****************************************************************
         LA    R5,4
         ST    R5,WK_LEN
         XR    R5,R5
         ST    R5,BUFLEN
         LA    R7,BUFFER_A              R7 -> buffer array.
         BAL   R4,TCPREAD               Perform socket read function.
*
         CLC   BUFLEN(4),=F'80'          Incoming buffer less than 80?
         BL    READLIS                    Y. Read listener reply.
         LA    R1,MSG11                   N. Reply too long,issue
         L     R15,=A(DISLINE)               error message
         BALR  R14,R15                     and
         B     TCPCLOSE                  close socket.
READLIS  DS    0H
         MVC   WK_LEN,BUFLEN              Reply msg length=read length.
         LA    R7,BUFFER_A+4              R7 -> reply from listener.
         BAL   R4,TCPREAD                 Read listener reply.
*
         CLC   MAX_LOOP,LOOP_CNT          All messages sent?
         BH    WRLOOP                      N. loop until done.
         B     TCPCLOSE                    Y. Close socket and exit.
*****************************************************************
* Routine to read a message from DESTPORT                       *
*****************************************************************
TCPREAD  DS    0H
         LA    R1,MSG05                   Display socket read function.
         L     R15,=A(DISLINE)
         BALR  R14,R15
        #SOCKET  READ,SOCK=S_NEWSOC,BUFFER=(R7),                      X
               RETLEN=RETLEN,BUFFERL=WK_LEN,                          X
               RETCODE=RETCODE,ERRNO=ERRNO,RSNCODE=RSNCODE
*
         LA    R1,MSG06      Display results of read.
         L     R15,=A(DISRC)
         BALR  R14,R15       Display the 3 return codes.codes
         CLC   RETCODE,=F'0' Successful read?
         BNE   TCPCLOSE        N.  Error close socket.
```

```
         A     R7,RETLEN     Adjust buffer array pointer
         L     R5,WK_LEN        and
         S     R5,RETLEN         read length
         ST    R5,WK_LEN        with reply length.
         CLI   WK_LEN,0        Read done?
         BNE   TCPREAD          N. Read some remainder.
         BR    R4               Y. Return.
*********************************************************************
* Routine to send a message to DESTPORT                             *
*********************************************************************
TCPWRITE DS    0H              Display socket write function.
         LA    R1,MSG07
         L     R15,=A(DISLINE)
         BALR  R14,R15
        #SOCKET WRITE,SOCK=S_NEWSOC,BUFFER=(R7),                      X
               RETLEN=RETLEN,BUFFERL=WK_LEN,                          X
               RETCODE=RETCODE,ERRNO=ERRNO,RSNCODE=RSNCODE
*
         LA    R1,MSG08               Display results of socket write.
         L     R15,=A(DISRC)
         BALR  R14,R15                Display the 3 return codes.
*
         CLC   RETCODE,=F'0'          Write successful?
         BNE   TCPCLOSE                 N. Close socket and exit.
         MVC   OUTAREA,WK_LEN+3       Message text length.
         MVC   OUTAREAT,BUFTXT80      Message text.
         LA    R1,OUTAREA             Display buffer contents
         L     R15,=A(DISLINE)
         BALR  R14,R15
*
         A     R7,RETLEN               Adjust buffer array
         L     R5,WK_LEN                 and
         S     R5,RETLEN                  write length
         ST    R5,WK_LEN                   with message length.
         CLI   WK_LEN,0                Anything left to write?
         BNE   TCPWRITE                  Y. Loop back.
         BR    R4                        N. Return.
*********************************************************************
* Close the socket and exit                                         *
*********************************************************************
TCPCLOSE DS    0H
         LA    R1,MSG09           Display closing socket.
         L     R15,=A(DISLINE)
         BALR  R14,R15            Display the 3 return codes.
         #SOCKET CLOSE,SOCK=S_NEWSOC,                                 X
               RETCODE=RETCODE,ERRNO=ERRNO,RSNCODE=RSNCODE
*
         LA    R1,MSG10           Display socket close function.
         L     R15,=A(DISRC)
```

```
         BALR  R14,R15             Display the 3 return codes.
*
         MVC   OUTAREAT(41),MSG99+1  Display ASMCLI temrimation msg.
         L     R5,ERRCOUNT           Get number of socket errors
         ST    R5,WORK1                 and
         UNPK  WORK2(9),WORK1(5)           make
         NC    WORK2(8),WKCLEAR               it
         TR    WORK2(8),TRTAB                  displayable
         MVC   ERRINDEC(4),WORK2+4                on the
         MVC   OUTAREAT+41(4),ERRINDEC             terminal.
         IC    R3,MSG99
         LA    R3,4(R3)
         STC   R3,OUTAREA
         LR    R1,R3
         LA    R3,OUTAREAT
         #LINEOUT OUTLEN=(R1),OUTAREA=(R3),OPTNS=(NOWAIT,TRLATIN)
*
         LA    R3,OUTAREA
         #WTL  MSGID=M#999043,MSGDICT=NO,OVRIDES=OVRLOG,               X
               PARMS=((R3)),RGSV=(R2-R8)
*
         #RETURN
         #BALI
*
         DROP  R12
         LTORG
         TITLE 'ASMCLI   - DISRC : DISPLAY THE RETURN CODES'
*------------------------------------------------------------------------*
*-  Routine to display the 3 return values from any TCP/IP calls.    -*
*-  Input : RETCODE, ERRNO and RSNCODE from the WORKAREA             -*
*-          R1 -> String to start with (e.g. MSGTXT format)          -*
*------------------------------------------------------------------------*
DISRC    DS    0H
         #SAVEREG                 Save the caller's registers
*
         LR    R12,R15
         USING DISRC,R12
*
         MVI   OUTAREA,L'OUTAREAT
         MVI   OUTAREAT,C' '
         MVC   OUTAREAT+1(L'OUTAREAT-1),OUTAREAT
*
         XR    R2,R2
         IC    R2,0(,R1)              Get message length
         BCTR  R2,0                   -1 for EX
         EX    R2,DISRCEX             Copy text
         LA    R2,1+1+OUTAREAT(R2)    Point to next free space
*
         CLC   RETCODE,=F'0'
```

```
         BE    DISRC00
         L     R3,ERRCOUNT
         LA    R3,1(R3)
         ST    R3,ERRCOUNT
DISRC00  DS    0H
         MVC   WORK1(4),RETCODE
         UNPK  WORK2(9),WORK1(5)
         NC    WORK2(8),WKCLEAR
         TR    WORK2(8),TRTAB
         MVC   0(8,R2),=CL8'RETCODE='
         MVC   8(8,R2),WORK2
*
         L     R15,ERRNO
         CVD   R15,WORK1
         OI    WORK1+7,X'0F'
         UNPK  WORK2(10),WORK1+2(6)
         LA    R14,WORK2
         LA    R15,9
DISRC01  DS    0H
         CLI   0(R14),C'0'
         BNE   DISRC02
         MVI   0(R14),C' '
         LA    R14,1(,R14)
         BCT   R15,DISRC01
DISRC02  DS    0H
         MVC   17(8,R2),=CL6'ERRNO='
         MVC   23(8,R2),WORK2+2
*
         MVC   WORK1(4),RSNCODE
         UNPK  WORK2(9),WORK1(5)
         NC    WORK2(8),WKCLEAR
         TR    WORK2(8),TRTAB
         MVC   32(8,R2),=CL8'RSNCODE='
         MVC   40(8,R2),WORK2
*
         LA    R1,OUTAREA
         L     R15,=A(DISLINE)
         BALR  R14,R15
*
         #RESTREG                  Restore the caller's registers
         BR     R14
*
         DROP  R12
*
DISRCEX  MVC   OUTAREAT(0),1(R1)       COPY TEXT
         LTORG
         TITLE 'ASMCLI - DISLINE : Write message line to log'
*---------------------------------------------------------------------*
*-  Subroutine to write a message line to the log.                   -*
```

```
*-  Input : R1 = A(output message) (first byte = message length)     -*
*--------------------------------------------------------------------
DISLINE  DS    0H
         #SAVEREG                  Save the caller's registers
*
         LR    R12,R15
         USING DISLINE,R12
         LR    R3,R1                   Get parm in R3
*
         #WTL  MSGID=M#999043,MSGDICT=NO,OVRIDES=OVRLOG,              X
               PARMS=((R3)),RGSV=(R2-R8)
*
         #RESTREG                  Restore the caller's registers
*
         BR    R14
*
         DROP  R12
*
M#999043 DC  PL4'9990430'
*
MSG01    MSGTXT 'Creating Socket.'
MSG02    MSGTXT 'SOCKET call:'
MSG03    MSGTXT 'Connecting: '
MSG04    MSGTXT 'CONNECT call:'
MSG05    MSGTXT 'Starting read.'
MSG06    MSGTXT 'READ call:'
MSG07    MSGTXT 'Starting write.'
MSG08    MSGTXT 'WRITE call:'
MSG09    MSGTXT 'Closing Socket.'
MSG10    MSGTXT 'CLOSE call:'
MSG11    MSGTXT 'Data length too long.'
MSG12    MSGTXT 'Calling INET_PTON.'
MSG13    MSGTXT 'INET_PTON call:'
MSG14    MSGTXT 'Calling GETHOSTBYADDR.'
MSG15    MSGTXT 'GETHOSTBYADDR call:'
MSG99    MSGTXT 'Program ASMCLI terminated. Error count = '
*
         LTORG
*
*
         TITLE 'ASMCLI - WORK AREA'
WORKAREA DSECT
*---------------------------------*
*-  DYNAMIC DATA                 -*
*---------------------------------*
SYSPLIST DS    16F
REGSTACK DS    32F
*
         DS    0D
```

```
WORK1    DC    XL10'00'
         DS    0D
WORK2    DC    XL10'00'            AT LEAST 10 BYTES DOUBLEWORD ALIGNED
*
*
HOSTIPA  DS    F             Binary IPA address of host
HOSTENTP DS    F             Pointer to HOSTENT structure
LOOP_CNT DS    F             Message counter
WK_LEN   DS    F             Length for read or write
ERRCOUNT DS    F             Number of socket errors
ERRINDEC DS    CL4           Number of socket errors decimal
*
*
SOCKADDC DS    (SIN#LEN)X          SOCKADDR for the client
SOCKDESN DS    F
S_NEWSOC DS    F
RETLEN   DS    F
RETCODE  DS    F
ERRNO    DS    F
RSNCODE  DS    F
*
*
OUTAREA  DS    X
OUTAREAT DC    CL80' '             OUTPUT AREA
         DS    0F
BUFFER   DS    CL84
         ORG   BUFFER
BUFLEN   DS    F
BUFTXT80 DS    CL80
         ORG   BUFTXT80
BUFTXT_M DS    CL33                 Message text
BUFTXT_B DS    CL1                  Blank character
BUFTXT_S DS    CL5                  Message sequence number
BUFTXT_F DS    CL41                 Filler
         ORG   BUFFER
BUFFER_A DS    CL84
*
*
*
*----------------------------------*
*-  STATIC DATA                   -*
*----------------------------------*
*
OVRLOG   DS    X'8000000000'       #WTL TO LOG ONLY
OVRLOGCO DS    X'C000000000'       #WTL TO LOG + CONSOLE
*
*
WKCLEAR  DS    XL8'0F0F0F0F0F0F0F0F'
TRTAB    DS    CL16'0123456789ABCDEF'
```

```
TRTABX   DS    XL6'FAFBFCFDFEFF'
*
WORKAREL EQU   *-WORKAREA
*
*-------------------------------------------------------------------*
*-  TCP/IP TABLES                                                  -*
*-------------------------------------------------------------------*
*
         #SOCKET TCPIPDEF
         #SOCKET ERRNOS
*
         END   ASMCLIEP
```

Assembler TCP/IP Generic Listener Server Program

```
TITLE 'Sample ASSEMBLER listener TCP/IP'
* ASMLIS   RENT EP=ASMLISEP
*********************************************************************
* The following program is an example of a TCP/IP generic        *
* listener server program written in Assembler.                  *
* The processing is the following:                               *
*  - read a message from the client (first 4 bytes = data length)*
*  - send the message back to the client program                  *
*  - if the message text is equal to "STOP" or if the connection *
*    is closed, then it closes its socket and return to the      *
*    generic listener service.                                   *
*  - if the message text is not equal to "STOP", then it returns *
*    to the generic listener service without closing its socket. *
*********************************************************************
*-------------------------------------------------------------------*
         MACRO
&LABEL.   #SAVEREG
&LABEL.   ST    R12,0(,R13)         Save R12
         ST    R14,4(,R13)         Save R14
         STM   R2,R8,8(R13)        Save R2-R8
         LA    R13,9*4(,R13)
         MEND
*-------------------------------------------------------------------*
         MACRO
&LABEL.   #RESTREG
&LABEL.   LA    R12,9*4             Get register stack entry length
         SR    R13,R12             Get A(previous register stack entry)
         L     R12,0(,R13)         Restore R12
         L     R14,4(,R13)         Restore R14
         LM    R2,R8,8(R13)        Restore R2-R8
         MEND
```

```
*--------------------------------------------------------------------*
         MACRO
&LABEL.   MSGTXT &TXT.
         LCLC  &TMP.
&TMP.     SETC  '&SYSNDX'.
&LABEL.   DC    AL1(L2&TMP).
L1&TMP.   DC    C&TXT.
L2&TMP.   EQU   *-L1&TMP.
         MEND
```

```
*-------------------------------------------------------------------*
ASMLIS CSECT
         @MODE MODE=IDMSDC
         #MOPT CSECT=ASMLIS,ENV=USER,RMODE=ANY,AMODE=ANY
         ENTRY ASMLISEP
ASMLISEP DS    0H
         BALR  R12,0
         BCTR  R12,0
         BCTR  R12,0
         USING ASMLISEP,R12
GETWORK  #GETSTG TYPE=(USER,SHORT),ADDR=(R1),PLIST=*,INIT=X'0',        X
               LEN=WORKAREL
         LR    R11,R1
         USING WORKAREA,R11
         LA    R13,REGSTACK
         MVI   OUTAREA,L'OUTAREAT
         MVC   OVRLOG,=X'8000000000'
         MVC   OVRLOGCO,=X'C000000000'
         MVC   WKCLEAR,=XL8'0F0F0F0F0F0F0F0F'
         MVC   TRTAB,=CL16'0123456789ABCDEF'
         MVC   TRTABX,=XL6'FAFBFCFDFEFF'
*
TCPSTART DS    0H
*********************************************************************
* Read the first 4 bytes: will contain the remaining length      *
*********************************************************************
*
         LA    R5,4
         ST    R5,WK_LEN               Set read length to 4.
         XR    R5,R5
         ST    R5,BUFLEN               Set buffer length to 0.
         LA    R7,BUFFER_A             R7 -> Buffer array.
         BAL   R4,TCPREAD              Perform the read.
*
         CLC   BUFLEN(4),=F'80'         Incoming buffer less than 80?
         BL    READCLI                   Y. Read client message.
         LA    R1,MSG04                  N. Message too long,issue
         L     R15,=A(DISLINE)              error message
         BALR  R14,R15                            and
         B     TCPCLOSE                       close socket.
READCLI  DS    0H
         MVC   WK_LEN,BUFLEN            client msg length=read length.
         LA    R7,BUFFER_A+4            R7 -> client message to read.
         BAL   R4,TCPREAD               Read client message.
*
         XC    OUTAREAT(L'OUTAREAT),OUTAREAT  Clear out message area
         MVC   OUTAREAT(15),=C'Buffer Length: '   Build
         L     R1,BUFLEN                            buffer length
         CVD   R1,WORK1                               display
```

```
         OI    WORK1+7,X'0F'
         UNPK  WORK2(9),WORK1+3(5)
         MVC   WORK2(8),WORK2+1    Shift value 1 byte to the left.
         MVI   WORK2+8,C' '
         MVI   WORK2+9,C' '        Clear last 2 bytes from WORK2 field.
         MVC   OUTAREAT+15(2),WORK2+6
         LA    R1,OUTAREA            Display read buffer length.
         L     R15,=A(DISLINE)
         BALR  R14,R15
         XC    OUTAREAT(L'OUTAREAT),OUTAREAT        Build
         MVC   OUTAREAT(8),=C'Buffer: '                buffer
         MVC   OUTAREAT+8,BUFTXT80                       display.
         LA    R1,OUTAREA            Display read buffer text.
         L     R15,=A(DISLINE)
         BALR  R14,R15
*****************************************************************
* Send the message back to the client                           *
*****************************************************************
         MVC   WK_LEN,BUFLEN
         L     R5,WK_LEN
         LA    R5,4(R5)               Include 1st 4 bytes
         ST    R5,WK_LEN                 in message length.
         LA    R7,BUFFER_A             R7 -> Buffer array.
         BAL   R4,TCPWRITE             Perform the write.
         CLC   BUFLEN,=F'4'            Incoming buffer length = 4?
         BNE   LISEXIT                   N. Return.
         CLC   BUFTXT04,=C'STOP'         Y. Stop listener?
         BNE   LISEXIT                      N. Return.
         B     TCPCLOSE                     Y. Close socket.
*****************************************************************
* Routine to read a message from the client                     *
*****************************************************************
TCPREAD  DS    0H
         LA    R1,MSG01                  Display socket read function.
         L     R15,=A(DISLINE)
         BALR  R14,R15
        #SOCKET  READ,SOCK=S_NEWSOC,BUFFER=(R7),                      X
               RETLEN=RETLEN,BUFFERL=WK_LEN,                          X
               RETCODE=RETCODE,ERRNO=ERRNO,RSNCODE=RSNCODE
*
         LA    R1,MSG06                  Display results of read.
         L     R15,=A(DISRC)
         BALR  R14,R15                   Display the 3 return codes.
         CLC   RETCODE,=F'0'             Successful read?
         BNE   TCPCLOSE                    N.  Error close socket.
         CLC   RETLEN,=F'0'              Anything left to read?
         BE    TCPCLOSE                   N.Close socket.
         A     R7,RETLEN                 Adjust buffer array pointer
         L     R5,WK_LEN                  and
```

```
         S     R5,RETLEN                    read length
         ST    R5,WK_LEN                       with message length.
         CLI   WK_LEN,0
         BNE   TCPREAD
         BR    R4
*********************************************************************
* Routine to send a message to the client                           *
*********************************************************************
TCPWRITE DS    0H
         LA    R1,MSG02               Display socket write function.
         L     R15,=A(DISLINE)
         BALR  R14,R15
        #SOCKET WRITE,SOCK=S_NEWSOC,BUFFER=(R7),                      X
               RETLEN=RETLEN,BUFFERL=WK_LEN,                          X
               RETCODE=RETCODE,ERRNO=ERRNO,RSNCODE=RSNCODE
*
         LA    R1,MSG05               Display results of socket write.
         L     R15,=A(DISRC)
         BALR  R14,R15                Display the 3 return codes.
*
         CLC   RETCODE,=F'0'          Write successful?
         BNE   TCPCLOSE                N. Close socket.
         A     R7,RETLEN              Adjust buffer array
         L     R5,WK_LEN               and
         S     R5,RETLEN                write length
         ST    R5,WK_LEN                  with message length.
         CLI   WK_LEN,0               Anything left to write?
         BNE   TCPWRITE                Y. Loop back.
         BR    R4                      N. Return.
*********************************************************************
* Close the socket and exit                                         *
*********************************************************************
TCPCLOSE DS    0H
         #SOCKET CLOSE,SOCK=S_NEWSOC,                                 X
               RETCODE=RETCODE,ERRNO=ERRNO,RSNCODE=RSNCODE
*
         LA    R1,MSG07               Display socket close function.
         L     R15,=A(DISRC)
         BALR  R14,R15                Display the 3 return codes.
*
LISEXIT  DS    0H
*
         #RETURN                           Return to caller
         #BALI
*
         DROP  R12
         LTORG
         TITLE 'ASMLIS01 - DISRC : DISPLAY THE RETURN CODES'
*--------------------------------------------------------------------*
```

```
*-  Rotuine to display the 3 return values from any TCP/IP calls.    -*
*-  Input : RETCODE, ERRNO and RSNCODE from the workarea             -*
*-          R1 -> String to start with (e.g.MSGTXT FORMAT)           -*
*-----------------------------------------------------------------------*
DISRC    DS    0H
*
         #SAVEREG
*
         LR    R12,R15
         USING DISRC,R12
*
         MVI   OUTAREAT,C' '
         MVC   OUTAREAT+1(L'OUTAREAT-1),OUTAREAT
*
         XR    R2,R2
         IC    R2,0(,R1)               Get message length.
         BCTR  R2,0                    -1 FOR EX.
         EX    R2,DISRCEX              Copy text.
         LA    R2,1+1+OUTAREAT(R2)     Point to next free space.
*
         MVC   WORK1(4),RETCODE
         UNPK  WORK2(9),WORK1(5)
         NC    WORK2(8),WKCLEAR
         TR    WORK2(8),TRTAB
         MVC   0(8,R2),=CL8'RETCODE='
         MVC   8(8,R2),WORK2
*
         L     R15,ERRNO
         CVD   R15,WORK1
         OI    WORK1+7,X'0F'
         UNPK  WORK2(10),WORK1+2(6)
         LA    R14,WORK2
         LA    R15,9
DISRC01  DS    0H
         CLI   0(R14),C'0'
         BNE   DISRC02
         MVI   0(R14),C' '
         LA    R14,1(,R14)
         BCT   R15,DISRC01
DISRC02  DS    0H
         MVC   17(8,R2),=CL6'ERRNO='
         MVC   23(8,R2),WORK2+2
*
         MVC   WORK1(4),RSNCODE
         UNPK  WORK2(9),WORK1(5)
         NC    WORK2(8),WKCLEAR
         TR    WORK2(8),TRTAB
         MVC   32(8,R2),=CL8'RSNCODE='
         MVC   40(8,R2),WORK2
```

```
*
         LA    R1,OUTAREA
         L     R15,=A(DISLINE)
         BALR  R14,R15
*
         #RESTREG
*
         BR    R14
*
         DROP  R12
*
DISRCEX  MVC   OUTAREAT(0),1(R1)       Copy text
         LTORG
         TITLE 'ASMLIS - DISLINE : WRITE MESSAGE LINE TO LOG'
*---------------------------------------------------------------------*
*-  Subroutine to write a message line to the log.                   -*
*-  Input : R1 = A(output message) (first byte = message length)     -*
*---------------------------------------------------------------------
DISLINE  DS    0H
         #SAVEREG
*
         LR    R12,R15
         USING DISLINE,R12
         LR    R3,R1                   Get parm in R3.
*
         #WTL  MSGID=M#999043,MSGDICT=NO,OVRIDES=OVRLOG,              X
               PARMS=((R3)),RGSV=(R2-R8)
*
         #RESTREG
*
         BR    R14
*
         DROP  R12
M#999043 DC  PL4'9990430'
         LTORG
*
*
MSG01 MSGTXT 'Starting read.'
MSG02 MSGTXT 'Starting write.'
MSG03 MSGTXT 'Closing Socket.'
MSG04 MSGTXT 'Data length too long.'
MSG05 MSGTXT 'WRITE call:'
MSG06 MSGTXT 'READ call:'
MSG07 MSGTXT 'CLOSE call:'
         DS    0F
         TITLE 'ASMLIS - WORK AREA'
WORKAREA DSECT
*---------------------------------*
*-  DYNAMIC DATA                 -*
```

```
*---------------------------------*
SYSPLIST DS    16F
REGSTACK DS    32F
*
         DS    0D
WORK1    DC    XL10'00'
         DS    0D
WORK2    DC    XL10'00'            AT LEAST 10 BYTES DOUBLEWORD ALIGNED
*
*
WK_LEN   DS    F
*
*
SOCKADDC DS    (SIN#LEN)X          SOCKADDR for the LISTENER
S_NEWSOC DS    F
RETLEN   DS    F
RETCODE  DS    F
ERRNO    DS    F
RSNCODE  DS    F
*
*
         DS    0F
BUFFER   DS    CL84
         ORG   BUFFER
BUFLEN   DS    F
BUFTXT80 DS    CL80
         ORG   BUFTXT80
BUFTXT04 DS    CL4
BUFTXT76 DS    CL76
         ORG   BUFFER
BUFFER_A DS    CL84
*
*
OUTAREA  DS    X
OUTAREAT DC    CL80' '             OUTPUT AREA
         DS    0D
*
*
*---------------------------------*
*-  STATIC DATA                  -*
*---------------------------------*
*
OVRLOG   DS    X'8000000000'       #WTL TO LOG ONLY
OVRLOGCO DS    X'C000000000'       #WTL TO LOG + CONSOLE
*
WKCLEAR  DS    XL8'0F0F0F0F0F0F0F0F'
TRTAB    DS    CL16'0123456789ABCDEF'
TRTABX   DS    XL6'FAFBFCFDFEFF'
*
```

```
WORKAREL EQU   *-WORKAREA
*
*-------------------------------------------------------------------*
*-  TCP/IP TABLES                                                  -*
*-------------------------------------------------------------------*
*
         #SOCKET TCPIPDEF
         #SOCKET ERRNOS
*
         END   ASMLISEP
```

CA ADS Examples

This section contains sample TCP/IP client and generic listener server programs written in CA ADS.

CA ADS TCP/IP Client Program

```
*****************************************************************
* The following program is an example of a TCP/IP client        *
* program written in ADS.                                       *
* The processing is the following:                              *
*  - Create a socket for the client program.                    *
*  - Convert the known dotted string format IPA to binary.      *
*  -   (Host IPA is defined in IPADOT, see below.)              *
*  - Find host information for connection.                      *
*  -   (Host port is defined in DESTPORT, see below.)           *
*  - Establish a connection to the host listener.               *
*  - Send message 1 to the listener (first 4 bytes = data length)*
*  - Read message 1 from  listener (first 4 bytes = data length) *
*  - Send message 2 to the listener (first 4 bytes = data length)*
*  - Read message 2 from  listener (first 4 bytes = data length) *
*  - Close socket and exit.                                     *
*****************************************************************
***              A D S C L I                                  ***
*** IDD input                                                  ***
*****************************************************************
       SET OPTIONS DEFAULT IS ON INPUT 1 THRU 80.
```

```
ADD RECORD ADSCLI-WORK-RECORD.
    02 WK-RETCD                PIC S9(8) COMP.
    02 WK-ERRNO                PIC S9(8) COMP.
    02 WK-RSNCD                PIC S9(8) COMP.
    02 SOCKDESC                PIC S9(8) COMP.
    02 LOOP-COUNT              PIC S9(8) COMP.
    02 RETLEN                  PIC S9(8) COMP.
    02 WK-LENGTH               PIC S9(8) COMP.
    02 WK-SUBSCRIPT            PIC S9(8) COMP.
    02 HOSTIPA                 PIC  9(8) COMP.
    02 HOSTENTP                PIC  9(8) COMP.
    02 MAX-LOOP                PIC  9(4) COMP VALUE 2.
    02 DEST-PORT               PIC  9(8) COMP VALUE 12345.
    02 IPA-HOST                PIC X(12) VALUE '255.255.25.2'.
    02 FILLER                  PIC X(4) VALUE SPACES.
    02 IPA-HOSTL               PIC S9(8) COMP VALUE 16.
ADD RECORD ADSCLI-BUFFER-RECORD.
    02 BUFFER                  PIC X(84).
    02 BUFFER-REDEF1      REDEFINES BUFFER.
       03 BUFLEN               PIC 9(8) COMP.
       03 BUFTXT80             PIC X(80).
       03 BUFTXT80-REDEF  REDEFINES BUFTXT80.
          04 BUFTXT-MSG        PIC X(29).
          04 BUFTXT-SEQ        PIC X(5).
          04 BUFTXT-BLANK      PIC X(1).
          04 BUFTXT-FILLER     PIC X(45).
    02 BUFFER-REDEF2      REDEFINES BUFFER.
       03 BUFFER-ARRAY         PIC X(1) OCCURS 84.

ADD PROCESS ADSCLI-PM MODULE SOURCE FOLLOWS
    WRITE TO LOG MESSAGE TEXT = 'ADSCLI: Starting dialog.'.

    !*********************************************************************
    ! Create a socket                                                    *
    !*********************************************************************
    IF (SOCKET(SOCKET-FUNCTION-SOCKET,
               SOCKET-RETCD,
               SOCKET-ERRNO,
               SOCKET-RSNCD,
               SOCKET-FAMILY-AFINET,
               SOCKET-TYPE-STREAM,
               SOCKET-PROTOCOL-TCP,
               SOCKDESC) EQ 0)
    THEN DO.
       WRITE TO LOG MESSAGE TEXT = 'ADSCLI: SOCKET successful.'.
       END.
    ELSE DO.
       WRITE TO LOG MESSAGE TEXT = 'ADSCLI: SOCKET error'.
       CALL TCPERROR.
```

```
   LEAVE ADS.
   END.

!*********************************************************************
! Convert the IP address from dotted string format to binary.      *
!*********************************************************************
IF (SOCKET(SOCKET-FUNCTION-INETPTON,
           SOCKET-RETCD,
           SOCKET-ERRNO,
           SOCKET-RSNCD,
           SOCKET-FAMILY-AFINET,
           IPA-HOST,
           IPA-HOSTL,
           HOSTIPA) EQ 0)
THEN DO.
   WRITE TO LOG MESSAGE TEXT = 'ADSCLI: INETPTON successful.'.
   END.
ELSE DO.
   WRITE TO LOG MESSAGE TEXT = 'ADSCLI: INETPTON error.'.
   CALL TCPERROR.
   CALL TCPCLOSE.
   LEAVE ADS.
   END.
!*********************************************************************
! Take the IP address and domain and resolve it through a name     *
! server. If successful, return the information in a HOSTENT       *
! structure.                                                       *
!*********************************************************************
IF (SOCKET(SOCKET-FUNCTION-GETHOSTBYADDR,
           SOCKET-RETCD,
           SOCKET-ERRNO,
           SOCKET-RSNCD,
           HOSTIPA,
           SOCKET-IPADDR4L,
           SOCKET-FAMILY-AFINET,
           HOSTENTP) EQ 0)
THEN DO.
   WRITE TO LOG MESSAGE TEXT = 'ADSCLI:GETHOSTBYADDR successful.'.
   END.
ELSE DO.
   WRITE TO LOG MESSAGE TEXT = 'ADSCLI: GETHOSTBYADDR error'.
   CALL TCPERROR.
   CALL TCPCLOSE.
   LEAVE ADS.
   END.

MOVE SOCKET-FAMILY-AFINET TO SIN-FAMILY.
MOVE DEST-PORT            TO SIN-PORT-NUMBER.
MOVE HOSTIPA              TO SIN-ADDRESS.
```

```
MOVE LOW-VALUES           TO SIN-ZEROS.
IF (SOCKET(SOCKET-FUNCTION-CONNECT,
           SOCKET-RETCD,
           SOCKET-ERRNO,
           SOCKET-RSNCD,
           SOCKDESC,
           SOCKADDR-IN,
           SOCKADDR-IN-LENGTH) EQ 0)
THEN DO.
   WRITE TO LOG MESSAGE TEXT = 'ADSCLI: CONNECT successful.'.
   END.
ELSE DO.
   WRITE TO LOG MESSAGE TEXT = 'ADSCLI: CONNECT error.'.
   CALL TCPERROR.
   CALL TCPCLOSE.
   LEAVE ADS.
   END.

!*********************************************************************
! Loop of write and read of messages with the server               *
!*********************************************************************
MOVE 1 TO LOOP-COUNT.
WHILE LOOP-COUNT LE MAX-LOOP
REPEAT.

   MOVE 'ADSCLI test message number ' TO BUFTXT-MSG.
   MOVE LOOP-COUNT             TO BUFTXT-SEQ.
   MOVE ' '                    TO BUFTXT-BLANK.
   MOVE 37 TO BUFLEN.
   MOVE 41 TO WK-LENGTH.
   MOVE 1  TO WK-SUBSCRIPT.
   CALL TCPWRITE.

 ! Read the first 4 bytes: will contain the remaining length
   MOVE 4  TO WK-LENGTH.
   MOVE 0  TO BUFLEN.
   MOVE 1  TO WK-SUBSCRIPT.
   CALL TCPREAD.

 ! Read the remaining data (maximum 80 characters are allowed)
   IF (BUFLEN GT 80)
   THEN DO.
      WRITE TO LOG MESSAGE TEXT = 'ADSCLI: Data length too long.'.
      CALL TCPCLOSE.
      LEAVE ADS.
      END.

   MOVE BUFLEN TO WK-LENGTH.
   MOVE 5      TO WK-SUBSCRIPT.
```

```
   CALL TCPREAD.

   SNAP RECORD (ADSCLI-BUFFER-RECORD).

   ADD 1 TO LOOP-COUNT.
   END. ! WHILE LOOP-COUNT

!*********************************************************************
! Loop completed. Close the socket and exit the program.           *
!*********************************************************************
WRITE TO LOG MESSAGE TEXT = 'ADSCLI: READ/WRITE loop completed.'.

CALL TCPCLOSE.

WRITE TO LOG MESSAGE TEXT = 'ADSCLI: Dialog ended successfully.'.
LEAVE ADS.

!*********************************************************************
! Subroutine to read a message from the client                     *
!*********************************************************************
DEFINE SUBROUTINE TCPREAD.
   WHILE WK-LENGTH GT 0 REPEAT.
      IF (SOCKET(SOCKET-FUNCTION-READ,
                 SOCKET-RETCD,
                 SOCKET-ERRNO,
                 SOCKET-RSNCD,
                 SOCKDESC,
                 BUFFER-ARRAY(WK-SUBSCRIPT),
                 WK-LENGTH,
                 RETLEN) EQ 0)
      THEN DO.
         WRITE TO LOG MESSAGE TEXT = 'ADSCLI: READ successful.'.
         END.
      ELSE DO.
         WRITE TO LOG MESSAGE TEXT = 'ADSCLI: READ error.'.
         CALL TCPERROR.
         CALL TCPCLOSE.
         LEAVE ADS.
         END.
      IF (RETLEN = 0)
      THEN DO.
         WRITE TO LOG MESSAGE TEXT = 'ADSCLI: READ 0 bytes'.
         CALL TCPCLOSE.
         LEAVE ADS.
         END.
      ADD RETLEN TO WK-SUBSCRIPT.
      SUBTRACT RETLEN FROM WK-LENGTH.
      END. ! READ LOOP
   GOBACK.
```

```
!*********************************************************************
! Subroutine to send a message to the client                         *
!*********************************************************************
DEFINE SUBROUTINE TCPWRITE.
   WHILE WK-LENGTH GT 0 REPEAT.
      IF (SOCKET(SOCKET-FUNCTION-WRITE,
                 SOCKET-RETCD,
                 SOCKET-ERRNO,
                 SOCKET-RSNCD,
                 SOCKDESC,
                 BUFFER-ARRAY(WK-SUBSCRIPT),
                 WK-LENGTH,
                 RETLEN) EQ 0)
      THEN DO.
         WRITE TO LOG MESSAGE TEXT = 'ADSCLI: WRITE successful.'.
         END.
      ELSE DO.
         WRITE TO LOG MESSAGE TEXT = 'ADSCLI: WRITE error.'.
         CALL TCPERROR.
         CALL TCPCLOSE.
         LEAVE ADS.
         END.
      IF (RETLEN = 0)
      THEN DO.
         WRITE TO LOG MESSAGE TEXT = 'ADSCLI: WRITE 0 bytes.'.
         CALL TCPCLOSE.
         LEAVE ADS.
         END.
      ADD RETLEN TO WK-SUBSCRIPT.          SUBTRACT RETLEN FROM WK-LENGTH.
      END. ! WRITE LOOP
   GOBACK.

!*********************************************************************
! Subroutine to close the socket                                     *
!*********************************************************************
DEFINE SUBROUTINE TCPCLOSE.
   IF (SOCKET(SOCKET-FUNCTION-CLOSE,
              SOCKET-RETCD,
              SOCKET-ERRNO,
              SOCKET-RSNCD,
              SOCKDESC) EQ 0)
   THEN DO.
      WRITE TO LOG MESSAGE TEXT = 'ADSCLI: CLOSE successful.'.
      END.
   ELSE DO.
      WRITE TO LOG MESSAGE TEXT = 'ADSCLI: CLOSE error.'.
      CALL TCPERROR.
      LEAVE ADS.
```

```
        END.
      GOBACK.

    !*********************************************************************
    ! Subroutine to process the socket calls errors                      *
    !*********************************************************************
    DEFINE SUBROUTINE TCPERROR.
       MOVE SOCKET-RETCD TO WK-RETCD.
       MOVE SOCKET-ERRNO TO WK-ERRNO.
       MOVE SOCKET-RSNCD TO WK-RSNCD.
       SNAP RECORD (ADSCLI-WORK-RECORD).
       GOBACK.

MSEND.
```

CA ADS TCP/IP Generic Listener Server Program

```
*********************************************************************
** The following program is an example of a TCP/IP generic         *
** listener server program written in ADS.                         *
** The processing is the following:                                *
** - read a message from the client (first 4 bytes = data length)*
** - send the message back to the client program                   *
** - if the message text is equal to "STOP" or if the connection *
**   is closed, then it closes its socket and return to the       *
**   generic listener service.                                    *
** - if the message text is not equal to "STOP", then it returns *
**   to the generic listener service without closing its socket. *
```

```
************************************************************************
***              A D S L I S                                         ***
*** IDD input                                                        ***
*** Use also the following work records defined for ADSCLI:          ***
***     ADSCLI-WORK-RECORD                                           ***
***     ADSCLI-BUFFER-RECORD                                         ***
************************************************************************
       SET OPTIONS DEFAULT IS ON INPUT 1 THRU 80.

ADD PROCESS ADSLIS-PM MODULE SOURCE FOLLOWS
   WRITE TO LOG MESSAGE TEXT = 'ADSLIS: STARTING DIALOG'.
   SNAP RECORD (SOCKET-LISTENER-PARMS).
   ! Read the first 4 bytes: will contain the remaining length
   MOVE 4  TO WK-LENGTH.
   MOVE 0  TO BUFLEN.
   MOVE 1  TO WK-SUBSCRIPT.
   CALL TCPREAD.

   MOVE BUFLEN TO WK-LENGTH.         ! Read data
   MOVE 5      TO WK-SUBSCRIPT.
   CALL TCPREAD.

   IF ((BUFLEN = 4) AND (BUFTXT80 = 'STOP'))
   THEN DO.
     WRITE TO LOG MESSAGE TEXT = 'ADSLIS: STOP MESSAGE RECEIVED'.
     CALL TCPCLOSE.
     LEAVE ADS.
     END.

   MOVE BUFLEN TO WK-LENGTH.         ! Echo the message
   ADD  4    TO WK-LENGTH.           ! Include header
   MOVE 1    TO WK-SUBSCRIPT.
   WHILE WK-LENGTH GT 0 REPEAT.
     IF ( SOCKET(
             SOCKET-FUNCTION-WRITE,
             SOCKET-RETCD,
             SOCKET-ERRNO,
             SOCKET-RSNCD,
             SOCKET-LISTENER-SOCKDESC,
             BUFFER-ARRAY(WK-SUBSCRIPT),
             WK-LENGTH,
             RETLEN) NE 0)
     THEN DO.
        WRITE TO LOG MESSAGE TEXT = 'ADSLIS: WRITE ERROR'.
        CALL TCPERROR.
        CALL TCPCLOSE.
        LEAVE ADS.
        END.
     ADD RETLEN TO WK-SUBSCRIPT.
```

```
  SUBTRACT RETLEN FROM WK-LENGTH.
END.

WRITE TO LOG MESSAGE TEXT = 'ADSLIS: One message processed'.
LEAVE ADS.

 !*********************************************************************
 ! Subroutine to read a message                                       *
 !*********************************************************************
DEFINE SUBROUTINE TCPREAD.
    WHILE WK-LENGTH GT 0 REPEAT.
       IF (SOCKET(SOCKET-FUNCTION-READ,
                  SOCKET-RETCD,
                  SOCKET-ERRNO,
                  SOCKET-RSNCD,
                  SOCKDESC,
                  BUFFER-ARRAY(WK-SUBSCRIPT),
                  WK-LENGTH,
                  RETLEN) EQ 0)
       THEN DO.
          WRITE TO LOG MESSAGE TEXT = 'ADSLIS: READ successful.'.
          END.
       ELSE DO.
          WRITE TO LOG MESSAGE TEXT = 'ADSLIS: READ error.'.
          CALL TCPERROR.
          CALL TCPCLOSE.
          LEAVE ADS.
          END.
       IF (RETLEN = 0)
       THEN DO.
          WRITE TO LOG MESSAGE TEXT = 'ADSLIS: READ 0 bytes'.
          CALL TCPCLOSE.
          LEAVE ADS.
          END.
       ADD RETLEN TO WK-SUBSCRIPT.
       SUBTRACT RETLEN FROM WK-LENGTH.
       END. ! READ LOOP
    GOBACK.

 !*********************************************************************
 ! Subroutine to close the socket                                     *
 !*********************************************************************
 DEFINE SUBROUTINE TCPCLOSE.
    IF (SOCKET(SOCKET-FUNCTION-CLOSE,
               SOCKET-RETCD,
               SOCKET-ERRNO,
               SOCKET-RSNCD,
               SOCKDESC) EQ 0)
    THEN DO.
```

```
            WRITE TO LOG MESSAGE TEXT = 'ADSLIS: CLOSE successful.'.
            END.
         ELSE DO.
            WRITE TO LOG MESSAGE TEXT = 'ADSLIS: CLOSE error.'.
            CALL TCPERROR.
            LEAVE ADS.
            END.
         GOBACK.

      !*********************************************************************
      ! Subroutine to process the socket calls errors                      *
      !*********************************************************************
      DEFINE SUBROUTINE TCPERROR.
         MOVE SOCKET-RETCD TO WK-RETCD.
         MOVE SOCKET-ERRNO TO WK-ERRNO.
         MOVE SOCKET-RSNCD TO WK-RSNCD.
         SNAP RECORD (ADSCLI-WORK-RECORD).
         GOBACK.

MSEND.
```

Index

L

N

O

P

R

S

T