This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the "Documentation") is for your informational purposes only and is subject to change or withdrawal by CA at any time. This Documentation is proprietary information of CA and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2017 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.
# Table of Contents

**IDMSCALC** ........................................................................................................................................................................... 13

  Calling the IDMSCALC Routine ................................................................................................................................. 13

  The IDMSCALC Argument .............................................................................................................................................. 13

**IDMSIN01** ........................................................................................................................................................................... 15

  IDMSIN01 Timestamp Display ................................................................................................................................. 16

  Calling IDMSIN01 from an Assembler Program ........................................................................................................... 17

    Considerations .............................................................................................................................................................. 17

    IDMSIN01 Assembler Macro Syntax .......................................................................................................................... 17

      Syntax ........................................................................................................................................................................ 17

      Parameters ................................................................................................................................................................. 18

    Parameters ................................................................................................................................................................. 20

    Assembler Program Calling IDMSIN01 .......................................................................................................................... 24

  Calling IDMSIN01 from a CA ADS Dialog ................................................................................................................... 30

  Calling IDMSIN01 from a COBOL Program .................................................................................................................. 30

    COBOL Program Calling IDMSIN01 ............................................................................................................................... 30

  Calling IDMSIN01 from a PL/I Program ......................................................................................................................... 37

**TCP/IP API Support** ......................................................................................................................................................... 46

  TCP/IP Programming Support for Online Applications ................................................................................................. 47

  Socket Macro Interface for Assembler Programs ......................................................................................................... 47

    Notes ........................................................................................................................................................................ 48

    Assembler Structure Description .................................................................................................................................. 49

  CA ADS Socket Interface ............................................................................................................................................... 49

    Comparing IDMSOCKI and SOCKET .......................................................................................................................... 50

    Considerations .............................................................................................................................................................. 51

    CA ADS Structure Description .................................................................................................................................... 51

  Socket Call Interface for COBOL .................................................................................................................................... 52

    Considerations .............................................................................................................................................................. 53

    COBOL Structure Description ...................................................................................................................................... 54

  Socket Call Interface for PL/I .......................................................................................................................................... 54

    Notes ........................................................................................................................................................................ 55

    PL/I Structure Description ........................................................................................................................................ 56
Generic Listener Service .......................................................................................................................... 57
  Implementation ......................................................................................................................................... 57
Application Design Considerations .............................................................................................................. 58
  Using Stream Sockets .............................................................................................................................. 59
    Receiving Data ...................................................................................................................................... 59
    Sending Data ........................................................................................................................................ 59
  TCP/IP Coding Samples ............................................................................................................................ 59
Miscellaneous TCP/IP Considerations ......................................................................................................... 60
  Using the TCP/IP Trace Facility ............................................................................................................... 60
  Using Multiple TCP/IP Stacks .................................................................................................................. 60
  Associating Time-outs to Sockets ............................................................................................................. 63
Function Descriptions ................................................................................................................................. 63
  ACCEPT ................................................................................................................................................ 67
    Parameters ........................................................................................................................................... 67
    Notes .................................................................................................................................................. 67
  BIND .................................................................................................................................................... 68
    Parameters ........................................................................................................................................... 68
  CLOSE ................................................................................................................................................... 68
    Parameters ........................................................................................................................................... 68
  CONNECT ............................................................................................................................................... 69
    Parameters ........................................................................................................................................... 69
    Notes .................................................................................................................................................. 69
  FCNTL .................................................................................................................................................. 70
    Parameters ........................................................................................................................................... 70
    Notes .................................................................................................................................................. 70
  FD_CLR ................................................................................................................................................ 71
    Parameters ........................................................................................................................................... 71
    Notes .................................................................................................................................................. 72
  FD_ISSET ............................................................................................................................................ 72
    Parameters ........................................................................................................................................... 72
    Notes .................................................................................................................................................. 72
  FD_SET ................................................................................................................................................ 73
    Parameters ........................................................................................................................................... 73
    Notes .................................................................................................................................................. 73
  FD_ZERO ............................................................................................................................................. 73
    Parameters ........................................................................................................................................... 74
    Notes .................................................................................................................................................. 74
  FREEADDRINFO ................................................................................................................................. 74
    Parameters ........................................................................................................................................... 74
    Notes .................................................................................................................................................. 74
  GETADDRINFO ............................................................................................................................... 75
    Parameters ........................................................................................................................................... 75
<table>
<thead>
<tr>
<th>Callable Services Reference</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notes</td>
<td>76</td>
</tr>
<tr>
<td>GETHOSTBYADDR</td>
<td>76</td>
</tr>
<tr>
<td>Parameters</td>
<td>77</td>
</tr>
<tr>
<td>Notes</td>
<td>77</td>
</tr>
<tr>
<td>GETHOSTBYNAME</td>
<td>78</td>
</tr>
<tr>
<td>Parameters</td>
<td>78</td>
</tr>
<tr>
<td>Notes</td>
<td>78</td>
</tr>
<tr>
<td>GETHOSTID</td>
<td>79</td>
</tr>
<tr>
<td>Parameters</td>
<td>79</td>
</tr>
<tr>
<td>Notes</td>
<td>79</td>
</tr>
<tr>
<td>GETHOSTNAME</td>
<td>79</td>
</tr>
<tr>
<td>Parameters</td>
<td>79</td>
</tr>
<tr>
<td>GETNAMEINFO</td>
<td>80</td>
</tr>
<tr>
<td>Parameters</td>
<td>80</td>
</tr>
<tr>
<td>Notes</td>
<td>81</td>
</tr>
<tr>
<td>GETPEERNAME</td>
<td>81</td>
</tr>
<tr>
<td>Parameters</td>
<td>82</td>
</tr>
<tr>
<td>GETSERVBYNAME</td>
<td>82</td>
</tr>
<tr>
<td>Parameters</td>
<td>83</td>
</tr>
<tr>
<td>Notes</td>
<td>83</td>
</tr>
<tr>
<td>GETSERVBYPORT</td>
<td>83</td>
</tr>
<tr>
<td>Parameters</td>
<td>84</td>
</tr>
<tr>
<td>Notes</td>
<td>84</td>
</tr>
<tr>
<td>GETSOCKNAME</td>
<td>84</td>
</tr>
<tr>
<td>Parameters</td>
<td>85</td>
</tr>
<tr>
<td>GETSOCKOPT</td>
<td>85</td>
</tr>
<tr>
<td>Parameters</td>
<td>86</td>
</tr>
<tr>
<td>Notes</td>
<td>86</td>
</tr>
<tr>
<td>GETSTACKS</td>
<td>86</td>
</tr>
<tr>
<td>Parameters</td>
<td>87</td>
</tr>
<tr>
<td>Notes</td>
<td>87</td>
</tr>
<tr>
<td>HTONL</td>
<td>87</td>
</tr>
<tr>
<td>Parameters</td>
<td>88</td>
</tr>
<tr>
<td>HTONS</td>
<td>88</td>
</tr>
<tr>
<td>Parameters</td>
<td>88</td>
</tr>
<tr>
<td>INET_ADDR</td>
<td>88</td>
</tr>
<tr>
<td>Parameters</td>
<td>89</td>
</tr>
<tr>
<td>INET_NTOA</td>
<td>89</td>
</tr>
<tr>
<td>Parameters</td>
<td>89</td>
</tr>
<tr>
<td>INET_NTOP</td>
<td>90</td>
</tr>
<tr>
<td>Parameters</td>
<td>90</td>
</tr>
<tr>
<td>INET_PTON</td>
<td>91</td>
</tr>
<tr>
<td>Parameters</td>
<td>91</td>
</tr>
<tr>
<td>Callable Services</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
</tr>
<tr>
<td>IOCTL</td>
<td>91</td>
</tr>
<tr>
<td>Parameters</td>
<td>92</td>
</tr>
<tr>
<td>Notes</td>
<td>92</td>
</tr>
<tr>
<td>LISTEN</td>
<td>92</td>
</tr>
<tr>
<td>Parameters</td>
<td>93</td>
</tr>
<tr>
<td>NTOHL</td>
<td>93</td>
</tr>
<tr>
<td>Parameters</td>
<td>93</td>
</tr>
<tr>
<td>NTOHS</td>
<td>94</td>
</tr>
<tr>
<td>Parameters</td>
<td>94</td>
</tr>
<tr>
<td>READ</td>
<td>94</td>
</tr>
<tr>
<td>Parameters</td>
<td>94</td>
</tr>
<tr>
<td>Notes</td>
<td>95</td>
</tr>
<tr>
<td>RECV</td>
<td>95</td>
</tr>
<tr>
<td>Parameters</td>
<td>95</td>
</tr>
<tr>
<td>Notes</td>
<td>96</td>
</tr>
<tr>
<td>RECVFROM</td>
<td>96</td>
</tr>
<tr>
<td>Parameters</td>
<td>96</td>
</tr>
<tr>
<td>Notes</td>
<td>97</td>
</tr>
<tr>
<td>SELECT and SELECTX</td>
<td>97</td>
</tr>
<tr>
<td>Parameters</td>
<td>99</td>
</tr>
<tr>
<td>Notes</td>
<td>100</td>
</tr>
<tr>
<td>SEND</td>
<td>101</td>
</tr>
<tr>
<td>Parameters</td>
<td>101</td>
</tr>
<tr>
<td>Notes</td>
<td>101</td>
</tr>
<tr>
<td>SENDTO</td>
<td>101</td>
</tr>
<tr>
<td>Parameters</td>
<td>102</td>
</tr>
<tr>
<td>Notes</td>
<td>102</td>
</tr>
<tr>
<td>SETSOCKOPT</td>
<td>103</td>
</tr>
<tr>
<td>Parameters</td>
<td>103</td>
</tr>
<tr>
<td>Notes</td>
<td>103</td>
</tr>
<tr>
<td>SETSTACK</td>
<td>103</td>
</tr>
<tr>
<td>Parameters</td>
<td>104</td>
</tr>
<tr>
<td>Notes</td>
<td>104</td>
</tr>
<tr>
<td>SHUTDOWN</td>
<td>104</td>
</tr>
<tr>
<td>Parameters</td>
<td>105</td>
</tr>
<tr>
<td>Notes</td>
<td>105</td>
</tr>
<tr>
<td>SOCKET</td>
<td>105</td>
</tr>
<tr>
<td>Parameters</td>
<td>106</td>
</tr>
<tr>
<td>Notes</td>
<td>106</td>
</tr>
<tr>
<td>WRITE</td>
<td>107</td>
</tr>
<tr>
<td>Parameters</td>
<td>107</td>
</tr>
<tr>
<td>Notes</td>
<td>108</td>
</tr>
</tbody>
</table>
Invoking System Tasks from Programs ............................................. 130
Linking to RHDCCCLST ................................................................. 130
Parameters .................................................................................. 130
  Mandatory Parameters ............................................................. 130
Optional Parameter .................................................................. 131
Example .................................................................................... 131
Invoking Command List Modules from Programs ...................... 132
Invoking DCMT and DCUF Commands from Programs ............... 132

Callable Services Reference 9
Linking to RHDCMT00 and RHDCUF00 ................................................................. 132
  RHDCMT00 Link Statement ............................................................................. 132
  RHDCUF00 Link Statement ............................................................................. 133
Parameters ......................................................................................................... 133
  INREC Format ............................................................................................... 133
  OUTREC Format ............................................................................................. 133
Usage .................................................................................................................... 135
  Queued Requests ........................................................................................... 135
  Using Scratch Area for Output ....................................................................... 135
Examples ............................................................................................................. 135
  DCMT Example .............................................................................................. 135
  DCUF Example .............................................................................................. 137
Invoking SDEL Command from Programs ......................................................... 137
  Linking to RHDCSDEL .................................................................................... 137
Parameters ......................................................................................................... 137
Example .............................................................................................................. 138
More Information ............................................................................................... 139
Invoking the SIGNON Task from Programs ....................................................... 139
  RHDCSNON .................................................................................................... 139
Parameters ......................................................................................................... 139
Example .............................................................................................................. 140

Two-Phase Commit Support with RRS ................................................................. 141
  RRS Support for Batch Applications ................................................................. 141
    Example ........................................................................................................ 142
    Enabling RRS for Batch Applications .............................................................. 143
    Batch RRS Transaction Boundaries and Application Design Considerations ...... 143
      Example of a COBOL Batch Program ............................................................ 144
  RRS Support for Online Applications ............................................................... 145
    Example ........................................................................................................ 147
    Programming Interface .................................................................................. 147
    Application Design Considerations ................................................................ 147

TCP/IP Error Codes ............................................................................................ 148
  Return, Errno, and Reason Codes .................................................................... 148
    ERRNO Numbers Set by the Socket Program Interface .................................. 149
  HOSTENT Structure ......................................................................................... 155
  SERVENT Structure ......................................................................................... 156
  Socket Structure Descriptions ....................................................................... 156
TCP/IP Programming Examples ................................................................. 158
PL/I Examples .......................................................................................... 158
  PL/I TCP/IP Client Program ................................................................. 158
  PL/I TCP/IP Generic Listener Server Program ...................................... 163
COBOL Examples ..................................................................................... 166
  COBOL TCP/IP Client Program ............................................................. 166
  COBOL TCP/IP Generic Listener Server Program .................................. 172
Assembler Examples .................................................................................. 175
  Assembler TCP/IP Client Program ....................................................... 175
  Assembler TCP/IP Generic Listener Server Program .............................. 182
CA ADS Examples ..................................................................................... 187
  CA ADS TCP/IP Client Program ........................................................... 187
  CA ADS TCP/IP Generic Listener Server Program ............................... 191
Callable Services Reference

For more information, see the following topics:

- IDMSCALC (see page 13)
- IDMSIN01 (see page 15)
- TCP/IP API Support (see page 46)
- Web Services API Support (see page 109)
- Invoking System Tasks from Programs (see page 130)
- Two-Phase Commit Support with RRS (see page 141)
- TCP/IP Error Codes (see page 148)
- TCP/IP Programming Examples (see page 158)
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.

The following topics are discussed on this page:

- Calling the IDMSCALC Routine (see page 13)
- The IDMSCALC Argument (see page 13)

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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Usage</th>
<th>Size (bytes)</th>
<th>COBOL Picture</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Binary</td>
<td>4</td>
<td>PIC S9(9) COMP</td>
<td>Specifies the target page number for storage of the record.</td>
</tr>
<tr>
<td>(Output)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### CA IDMS Reference - 19.0

<table>
<thead>
<tr>
<th>Field</th>
<th>Usage</th>
<th>Size (bytes)</th>
<th>COBOL Picture</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Input</td>
<td>Binary 4</td>
<td>PIC S9(9) COMP</td>
<td>Specifies the number of the highest page on which the record can be stored.</td>
</tr>
<tr>
<td>3</td>
<td>Input</td>
<td>Binary 4</td>
<td>PIC S9(9) COMP</td>
<td>Specifies the number of the lowest page on which the record can be stored.</td>
</tr>
<tr>
<td>4</td>
<td>Input</td>
<td>Binary 2</td>
<td>PIC S9(4) COMP</td>
<td>Specifies the length, in bytes, of the CALC key value.</td>
</tr>
<tr>
<td>5</td>
<td>Input</td>
<td>Char 1-256</td>
<td>PIC X(nnn)</td>
<td>Specifies the value of the CALC key.</td>
</tr>
</tbody>
</table>

**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.
IDMSIN01

The IDMS module contains an IDMSIN01 entry point that 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 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, 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 EVBLOClEK.
   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)
```

### IDMSIN01 Timestamp Display

The following table displays the local and external timestamps of the GETDATE function:

<table>
<thead>
<tr>
<th>Assembler Parameters</th>
<th>COBOL, ADS, PL/I Parameters</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERNAL</td>
<td>0</td>
<td>Internal 8 byte DATETIME</td>
<td>26 bytes (1994-07-18-12.01.18.458382)</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>1</td>
<td>Get current time 26 bytes (1994-07-18-12.01.18.458382)</td>
<td></td>
</tr>
<tr>
<td>EXTERNAL</td>
<td>2</td>
<td>26 bytes (1994-07-18-12.01.18.458382)</td>
<td>Internal 8 byte DATETIME</td>
</tr>
<tr>
<td>TIMEINT</td>
<td>3</td>
<td>Internal 8 byte DATETIME</td>
<td>External time 8 bytes (HH.MM.SS)</td>
</tr>
<tr>
<td>TIMEEXT</td>
<td>4</td>
<td>External time (HH.MM.SS)</td>
<td>Internal 8-Byte DATETIME</td>
</tr>
<tr>
<td>DATEINT</td>
<td>5</td>
<td>Internal 8 byte DATETIME</td>
<td>Displayable 10 bytes Date (YYYY-MM-DD)</td>
</tr>
<tr>
<td>DATEEXT</td>
<td>6</td>
<td>Displayable 10 bytes Date (YYYY-MM-DD)</td>
<td>Internal 8 byte DATETIME</td>
</tr>
<tr>
<td>INTERNAL-UTC</td>
<td>7</td>
<td>Internal 8 byte DATETIME</td>
<td>26 bytes (1994-07-18-12.01.18.458382) converted from UTC to local time.</td>
</tr>
</tbody>
</table>
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.

Considerations

- 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.

- After changing the SYSCTL DD name, to return to the default state, change the value back to 'SYSCTL' or blank spaces.

IDMSIN01 Assembler Macro Syntax

Syntax

```
IDMSIN01 IDMSIN01
  GETDATE date-parms
  GETMSG msg-parms
  GETUSER user-parms
  GETPROF profile-parms
  NOTRACE
  RRSCTX rrs-parms
  SETPROF profile-parms
  STRCONV string-parms
  SYSCTL sysctl-parms
  TRACE
  TXNSOFF
  TXNSON
  ENVINFO envinfo-parms
  FRMTDBK frmtdbk-parms

  ,PLIST= SYSPLIST
  ,RPB= SQLRPB

```
Parameters

IDMSIN01 indicates a request for the following IDMSIN01 functions, specified by the parameters, described below:

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

- **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.
  - **,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 an 8-byte internal DATETIME stamp is returned as a 26-character date-time display.
    - **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.
    - **INTERNAL-UTC**
      Specifies that an 8-byte internal DATETIME stamp is returned as a 26-character date-time display converted to local time (from UTC).

- **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 `xxxxxxx` 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. The descriptions can be found in the IBM section 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. The descriptions can be found in the IBM section 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.
  
  - `,RESULT=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 where 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.

  ,DBKFORM=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 SQLRBP. 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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSPLIST</td>
<td>DC 10F'0' Standard PLIST</td>
</tr>
<tr>
<td>SQLRPB</td>
<td>DC XL36 '00' RPB used by IDMSIN01 macro</td>
</tr>
<tr>
<td>SQLMSGB</td>
<td>DS 0F SQL error messages control block</td>
</tr>
<tr>
<td>SQLMAX</td>
<td>DC F'6' Max. number of SQL error lines</td>
</tr>
<tr>
<td>SQLSIZE</td>
<td>DC F'80' Error line size</td>
</tr>
<tr>
<td>SQLMCNT</td>
<td>DC F'0' Act. number of messages returned</td>
</tr>
<tr>
<td>SQLLINE</td>
<td>DC 6CL80' ' Allow for 6 error messages</td>
</tr>
<tr>
<td>XTRAPKEY</td>
<td>DS CL8 Key value for SETPROF + GETPROF</td>
</tr>
<tr>
<td>XTRAPVAL</td>
<td>DS CL32 Variable for SETPROF + GETPRO</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DS XL8 Internal date/time stamp</td>
</tr>
<tr>
<td>DATEFLD</td>
<td>DS CL26 Edited date/time used by GETDATE</td>
</tr>
<tr>
<td>TIMESMP</td>
<td>DS XL8 Internal TIME stamp</td>
</tr>
<tr>
<td>TIMESHOW</td>
<td>DS CL8 External TIME display</td>
</tr>
<tr>
<td>DATESTMP</td>
<td>DS XL8 Internal DATE stamp</td>
</tr>
<tr>
<td>DATESHOW</td>
<td>DS CL10 External DATE display</td>
</tr>
<tr>
<td>USERID</td>
<td>DS CL32 Current user id returned by USERID</td>
</tr>
<tr>
<td>DDYSYSCTL</td>
<td>DS CL8 DDNAME for SYSCTL</td>
</tr>
<tr>
<td>BLANKS</td>
<td>DC CL133' ' Blanks for all</td>
</tr>
<tr>
<td>RRSCTX</td>
<td>DC XL16'00' 16-byte context token</td>
</tr>
<tr>
<td>RRSFUNC</td>
<td>DS X RRS context function:</td>
</tr>
<tr>
<td>RRSFNGET</td>
<td>EQU X'01' - Get RRS context</td>
</tr>
<tr>
<td>RRSFNSET</td>
<td>EQU X'02' - Set RRS context</td>
</tr>
<tr>
<td>CONVFUNC</td>
<td>DS CL4 STRCONV function:</td>
</tr>
<tr>
<td>CONVF2A</td>
<td>EQU 'ETOA' - EBCDIC --&gt; ASCII</td>
</tr>
<tr>
<td>CONVF2E</td>
<td>EQU 'ATOE' - ASCII --&gt; EBCDIC</td>
</tr>
<tr>
<td>STRING</td>
<td>DC C'String to convert' String to convert</td>
</tr>
<tr>
<td>STRSTRINGL</td>
<td>DC A(L'STRING) String length</td>
</tr>
<tr>
<td>SPACE</td>
<td>DS 0F Align on a fullword boundary</td>
</tr>
<tr>
<td>EVBLOCK</td>
<td>DS XL(EV$DSLEN) Runtime environment return area</td>
</tr>
<tr>
<td>COPY</td>
<td>#ENVINFO Copy in runtime environment</td>
</tr>
<tr>
<td>DBKEYFLD</td>
<td>DS F Dbkey to be formatted</td>
</tr>
<tr>
<td>DBKEYFMT</td>
<td>DS H Associated database-key format</td>
</tr>
<tr>
<td>DBKCHAR</td>
<td>DS CL12 Output field for formatted dbkey</td>
</tr>
</tbody>
</table>

---

### Call IDMSIN01 to deactivate the DML trace or SQL trace

- Call IDMSIN01 to deactivate the DML trace or SQL trace which was originally activated by the corresponding SYSIDMS parm (DMLTRACE=ON or SQLTRACE=ON).

### Call IDMSIN01 to request a 'GETPROF' to get the user

- 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.
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.

Call IDMSIN01 to request a 'STRCONV' to convert a string from EBCDIC to ASCII.

CONVFUN - specifies which conversion:
  ETOA - EBCDIC to ASCII
  ATOE - ASCII to EBCDIC

BUFFER - SPECIFIES STRING TO CONVERT
BUFFERL - SPECIFIES LENGTH OF STRING

MVC CONVFUNC,=A(CONVFE2A)
Convert EBCDIC to ASCII

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

MVC CONVFUNC,=A(CONVF2EA)
Convert ASCII to EBCDIC

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,
   PRERESULT=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

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
$PRINT WORKLINE Print the date

*********************************************************
* Call IDMSIN01 to have an 8 byte internal DATETIME stamp
* converted from UTC to local time returned as a
* displayable 26 character DATE/TIME display.
* DATEIN is the address of the 8 byte internal
* UTC formatted DATETIME stamp.
* DATEOUT is the address of the 26 byte local DATE/TIME returned.
*********************************************************
IDMSIN01 GETDATE,
DATEIN=DATETIME,
DATEOUT=DATEFLD,
FORMAT=INTERNAL-UTC,
ERROR=ERROROUT
MVC WORKLINE.BLANKS Clear print work line
MVC WORKLINE+10(14),=C’LOCAL DATETIME --- -> ‘
MVC WORKLINE+30(26),DATEFLD Displayable date/time
$PRINT WORKLINE Print the date/time

*********************************************************
* 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 ‘
$PRINT WORKLINE Print the heading
$PRINT BLANKS Print 1 blank line
L R3,SQLMCNT Get number of message lines returned
LA R5,SQLLINE Point at first message line
MVC WORKLINE,BLANKS Clear print work line
MSGLOOP MVC WORKLINE+3(R8),8(R5) Move SQL error message to print line
$PRINT 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.


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-STOREAGE 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-TXNSON   VALUE 28.
      88 IN01-FN-TXNSOFF  VALUE 29.
      88 IN01-FN-RRSSCTX  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-RRSSCTX-GET VALUE 01.
      88 IN01-FN-RRSSCTX-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.
CA IDMS Reference - 19.0

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-TXNOFF 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 WK-KEYWD WK-VALUE.
IF REQUEST-RETURN NOT = 0
   DISPLAY 'SETPROF returned error ' REQUEST-RETURN.

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

*********************************************************
**SET IN01-FN-SYSCTL TO TRUE.**
MOVE 'SYSCTL73' TO WK-SYSCTL.
CALL 'IDMSIN01' USING RPB 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.
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.
*********************************************************
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.
*********************************************************
SET IN01-FN-GETDATE TO TRUE.
MOVE 1 TO WK-DTS-FORMAT
CALL 'IDMSIN01' USING RPB REQ-WK
       WK-DTS-FORMAT WK-DTS.
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.
*********************************************************
SET IN01-FN-GETDATE TO TRUE.
MOVE 2 TO WK-DTS-FORMAT
CALL 'IDMSIN01' USING RPB REQ-WK
       WK-DTS-FORMAT.
DISPLAY 'THE DATE AND TIME IS --> ' WK-CDTS.

******************************************************************
* Call IDMSIN01 to have an 8 byte internal DATETIME stamp
* returned as a displayable 26 character DATE/TIME display.
******************************************************************
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.
* 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 have an 8 byte internal DATETIME stamp converted from UTC to local 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 (0).
* Parm 4 is the address of the 8 byte internal UTC formatted DATETIME stamp.
* Parm 5 is the address of the 26 byte local DATE/TIME returned.

**SET IN01-FN-GETDATE TO TRUE.**

MOVE 7 TO WK-DTS-FORMAT
MOVE 'UNKNOWN' TO WK-CDTS
CALL 'IDMSIN01' USING RPB REQ-WK
WK-DTS-FORMAT WK-DTS WK-CDTS.
DISPLAY 'THE LOCAL DATE AND TIME IS --> ' WK-CDTS.

* Call IDMSIN01 to retrieve SQL error messages into a user buffer that will then be displayed back to the user.
* What's 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 IDMSIN1 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),
CA IDMS Reference - 19.0

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 (' PLIN01 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 SQLLINE (6) CHAR(80);

/* ****************************************************************
/* The following SQL include statement is needed only if
/* 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 ...
/* ****************************************************************

15-Jan-2018 38/193
/*
 * 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 :-) 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 :-) 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 :-) 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 :-) 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 DTS,
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;
/*
Stephen K. Bender
*************************************************************************
* Call IDMSIN01 to have an 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;
/*
Stephen K. Bender
*************************************************************************
* Call IDMSIN01 to have 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);
/*
Stephen K. Bender
*************************************************************************
* 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 have an 8 byte internal DATETIME stamp
* converted from UTC to local 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 (0).
* Parm 4 is the address of the 8 byte internal
* UTC formatted DATETIME stamp.
* Parm 5 is the address of the 26 byte local
* DATE/TIME returned.
*******************************************************************************/
REQUEST_CODE = IN01_FN_GETDATE;
WK_DTS_FORMAT = 7;
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 LOCAL DATE AND TIME IS --> ' || WK_CDTS);
/*
* Call IDMSIN01 to retrieve SQL error messages into a user
* buffer that will then be displayed back to the user.
* What is 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.
*******************************************************************************/
CA IDMS Reference - 19.0

/*
 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;
TCP/IP API Support

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 z/OS, z/VSE, and z/VM operating systems.

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.

For more information, see the following topics:

- TCP/IP Programming Support for Online Applications (see page 47)
- Socket Macro Interface for Assembler Programs (see page 47)
- CA ADS Socket Interface (see page 49)
- Socket Call Interface for COBOL (see page 52)
- Socket Call Interface for PL/I (see page 54)
- Generic Listener Service (see page 57)
- Application Design Considerations (see page 58)
- Miscellaneous TCP/IP Considerations (see page 60)
- Function Descriptions (see page 63)
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 .

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 CA IDMS System Tasks and Operator Commands (https://docops.ca.com/display/IDMSCU19/CA+IDMS+System+Tasks+and+Operator+Commands).
- 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 section IDMSIN01 (see page 15).

Socket Macro Interface for Assembler Programs

Contents

- Notes (see page 48)
- Assembler Structure Description (see page 49)

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>Specifies the optional Assembler label.</td>
</tr>
<tr>
<td>function</td>
<td>Specifies the name of the function to execute. A detailed description of the</td>
</tr>
<tr>
<td></td>
<td>supported functions can be found in Function Descriptions (see page 63).</td>
</tr>
<tr>
<td>return-code</td>
<td>Specifies the name of a fullword that receives the outcome of the operation.</td>
</tr>
<tr>
<td></td>
<td>The following are the possible values:</td>
</tr>
<tr>
<td></td>
<td>0 -- No error occurred.</td>
</tr>
<tr>
<td></td>
<td>-1 -- A socket error was encountered; the errno and reason-code fields</td>
</tr>
<tr>
<td></td>
<td>contain more detailed information about the error.</td>
</tr>
<tr>
<td>errno</td>
<td>Specifies the name of a fullword that receives the ERRNO value when return-</td>
</tr>
<tr>
<td></td>
<td>code is -1. For more information, see Return, Errno, and Reason Codes (see</td>
</tr>
<tr>
<td></td>
<td>page 148).</td>
</tr>
<tr>
<td>reason-code</td>
<td>Specifies the name of a fullword that receives the reason code value when</td>
</tr>
<tr>
<td></td>
<td>return-code is -1. For more information, see Return, Errno, and Reason Codes</td>
</tr>
<tr>
<td></td>
<td>(see page 148).</td>
</tr>
<tr>
<td>parameter-list-area</td>
<td>Specifies the name of an area or register pointing to the area that is</td>
</tr>
<tr>
<td></td>
<td>used to build the #SOCKET parameter list. The default is SYSPLIST. The</td>
</tr>
<tr>
<td></td>
<td>length of the parameter-list-area used by the macro depends on the #SOCKET</td>
</tr>
<tr>
<td></td>
<td>function that is called; the longest parameter-list currently needed for a</td>
</tr>
<tr>
<td></td>
<td>#SOCKET call is 16 fullwords.</td>
</tr>
<tr>
<td>rgsr</td>
<td>Specifies the registers to be saved. This parameter applies only to system</td>
</tr>
<tr>
<td></td>
<td>mode programs. The default is (R2-R8).</td>
</tr>
<tr>
<td>call-value</td>
<td>Indicates whether to generate the parameter list and/or execute the function.</td>
</tr>
<tr>
<td></td>
<td>The following are the possible values:</td>
</tr>
<tr>
<td></td>
<td>YES -- Generates the parameter list and executes the function. This is the</td>
</tr>
<tr>
<td></td>
<td>default.</td>
</tr>
<tr>
<td></td>
<td>NO -- Generates the parameter list, but does not execute the function.</td>
</tr>
<tr>
<td></td>
<td>ONLY -- Executes the function for which a parameter list is pre-built.</td>
</tr>
</tbody>
</table>

### 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 156).

**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:

```plaintext
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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>function</td>
<td>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 63).</td>
</tr>
<tr>
<td>return-code</td>
<td>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.</td>
</tr>
<tr>
<td>errno</td>
<td>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 148).</td>
</tr>
<tr>
<td>reason-code</td>
<td>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 148).</td>
</tr>
</tbody>
</table>

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.

Considerations

- 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:

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

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 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:

<table>
<thead>
<tr>
<th>Parameter Description</th>
</tr>
</thead>
</table>
| 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 63). |
| 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 148). |
| 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 148). |

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

- 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 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 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 156).

Socket Call Interface for PL/I

Contents

- Notes (see page 55)
- PL/I Structure Description (see page 56)

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:

<table>
<thead>
<tr>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>function</td>
</tr>
</tbody>
</table>

### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>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 63).</td>
</tr>
<tr>
<td>return_code</td>
<td>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);</td>
</tr>
<tr>
<td>errno</td>
<td>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 148).</td>
</tr>
<tr>
<td>reason_code</td>
<td>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 148).</td>
</tr>
</tbody>
</table>

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_RETCDE,
  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)  P0INTER;
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.
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.

Considerations

- 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.

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.
- 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/LTERM pair is signed off automatically.

Application Design Considerations

Contents

- Using Stream Sockets (see page 59)
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

**Contents**

- Using the TCP/IP Trace Facility (see page 60)
- Using Multiple TCP/IP Stacks (see page 60)
- Associating Time-outs to Sockets (see page 63)

---

#### 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 *Reference section*.

---

#### 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 Administrating section*.
- For more information about the SYSIDMS parameters, see the *CA IDMS Common Administrating section*.
- For more information about the DCMT VARY TCP/IP command, see the *Reference section*.
- For more information about the SYSTCPD file, see the *CA IDMS System Reference section*.

**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 Reference section.
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 70). For more information about the INACTIVE INTERVAL or EXTERNAL WAIT parameters of the TASK statement, see the CA IDMS Administrating section.

Function Descriptions
- Notes (see page 67)
- BIND (see page 68)
- Parameters (see page 68)
- CLOSE (see page 68)
- Parameters (see page 69)
- CONNECT (see page 69)
- Parameters (see page 69)
- Notes (see page 69)
- FCNTL (see page 70)
  - Parameters (see page 70)
  - Notes (see page 70)
- FD_CLR (see page 71)
  - Parameters (see page 71)
  - Notes (see page 72)
- FD_ISSET (see page 72)
  - Parameters (see page 72)
  - Notes (see page 72)
- FD_SET (see page 73)
  - Parameters (see page 73)
  - Notes (see page 73)
- FD_ZERO (see page 73)
  - Parameters (see page 74)
  - Notes (see page 74)
- FREEADDRINFO (see page 74)
  - Parameters (see page 74)
  - Notes (see page 74)
- GETADDRINFO (see page 75)
  - Parameters (see page 75)
  - Notes (see page 76)
- GETHOSTBYADDR (see page 76)
  - Parameters (see page 77)
  - Notes (see page 77)
- GETHOSTBYNAME (see page 78)
  - Parameters (see page 78)
  - Notes (see page 78)
- GETHOSTID (see page 79)
  - Parameters (see page 79)
  - Notes (see page 79)
- GETHOSTNAME (see page 79)
  - Parameters (see page 79)
- GETNAMEINFO (see page 80)
  - Parameters (see page 80)
  - Notes (see page 81)
- GETPEERNAME (see page 81)
  - Parameters (see page 82)
- GETSERVBNAME (see page 82)
  - Parameters (see page 83)
  - Notes (see page 83)
- GETSERVBPORT (see page 83)
  - Parameters (see page 84)
  - Notes (see page 84)
- GETSOCKNAME (see page 84)
  - Parameters (see page 85)
- GETSOCKOPT (see page 85)
  - Parameters (see page 86)
  - Notes (see page 86)
- GETSTACKS (see page 86)
  - Parameters (see page 87)
  - Notes (see page 87)
- HTONL (see page 87)
  - Parameters (see page 88)
- HTONS (see page 88)
  - Parameters (see page 88)
- INET_ADDR (see page 88)
  - Parameters (see page 89)
- INET_NTOA (see page 89)
  - Parameters (see page 89)
- INET_NTOP (see page 90)
  - Parameters (see page 90)
- INET_PTON (see page 91)
  - Parameters (see page 91)
- IOCTL (see page 91)
  - Parameters (see page 92)
  - Notes (see page 92)
- LISTEN (see page 92)
  - Parameters (see page 93)
- NTOHL (see page 93)
  - Parameters (see page 93)
- NTOHS (see page 94)
  - Parameters (see page 94)
- READ (see page 94)
  - Parameters (see page 94)
  - Notes (see page 95)
- RECV (see page 95)
  - Parameters (see page 95)
  - Notes (see page 96)
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>Specifies the name of a fullword field containing the socket descriptor that was used on the BIND and LISTEN functions.</td>
</tr>
<tr>
<td>sockaddr</td>
<td>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.</td>
</tr>
<tr>
<td>sockaddr-length</td>
<td>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.</td>
</tr>
<tr>
<td>new-socket-descriptor</td>
<td>Specifies the name of a fullword field where the socket descriptor of the new connection is returned.</td>
</tr>
</tbody>
</table>

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 63).
BIND

BIND assigns a local name to an unnamed socket.

Assembler

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

List of USING Parameters

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>Specifies the name of a fullword field containing the socket descriptor to bind.</td>
</tr>
<tr>
<td>sockaddr</td>
<td>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.</td>
</tr>
<tr>
<td>sockaddr-length</td>
<td>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).</td>
</tr>
</tbody>
</table>

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,
       RSCODE=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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>Specifies the name of a fullword field containing the socket descriptor to close.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>Specifies the name of a fullword containing the socket descriptor to which to connect.</td>
</tr>
</tbody>
</table>

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 63).
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.

**Assemble**

```
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>Specifies the name of a fullword field containing the socket descriptor to process.</td>
</tr>
<tr>
<td>command</td>
<td>Specifies the name of a fullword field containing the command to perform on the socket. command can be specified as an absolute expression.</td>
</tr>
<tr>
<td>argument</td>
<td>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.</td>
</tr>
<tr>
<td>returned-value</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>EQUate Symbol</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F@GETFL</td>
<td>SOCKET-FCNTL-GET</td>
<td>Gets file status command</td>
</tr>
</tbody>
</table>
### EQUate Symbol

<table>
<thead>
<tr>
<th>EQUate Symbol</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F@SETFL</td>
<td>SOCKET-FCNTL-SET</td>
<td>Sets file status command</td>
</tr>
<tr>
<td>F@GETIMO</td>
<td>SOCKET-FCNTL-GETIMO</td>
<td>Gets time-out value associated with a socket</td>
</tr>
<tr>
<td>F@SETIMO*</td>
<td>SOCKET-FCNTL-SETIMO</td>
<td>Associates a time-out value with a socket</td>
</tr>
<tr>
<td>NONBLOCK</td>
<td>SOCKET-FCNTL-NONBLOCK</td>
<td>Sets socket in non-blocking mode</td>
</tr>
</tbody>
</table>

* 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**

```assembly
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>Specifies the name of a fullword field containing the socket descriptor whose bit must be cleared (set to zero) in the bit list.</td>
</tr>
<tr>
<td>bit-list</td>
<td>Specifies the name of the area containing the bit list.</td>
</tr>
<tr>
<td>bit-list-length</td>
<td>Specifies the name of a fullword field containing the length of the bit-list in bytes. The value of <code>bit-list-length</code> can be specified as an absolute expression. The value of <code>bit-list-length</code> must be a multiple of 4.</td>
</tr>
<tr>
<td>bit-order</td>
<td>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 <code>option</code> parameter of the SELECT or SELECTX function. The value of <code>bit-order</code> can be specified as an absolute expression. The following are the accepted values: SEL@BBKW (default) SEL@BFWD</td>
</tr>
</tbody>
</table>
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,
      SCK=socket-descriptor,
      BITLIST=bit-list,
      BITLISTL=bit-list-length,
      BITORDER=bit-order,
      RETVAL=returned-bit-status,
      PLIST=parameter-list-area
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>Specifies the name of a fullword field containing the socket descriptor whose bit needs testing in the bit list.</td>
</tr>
<tr>
<td>bit-list</td>
<td>Specifies the name of the area containing the bit list.</td>
</tr>
<tr>
<td>bit-list-length</td>
<td>Specifies the name of a fullword field containing the length of the bit-list in bytes. Can be specified as an absolute expression. Must be a multiple of 4.</td>
</tr>
<tr>
<td>bit-order</td>
<td>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. Can be specified as an absolute expression. The following are the accepted values: SEL@BBKW (default) SEL@BFWD</td>
</tr>
<tr>
<td>returned-bit-status</td>
<td>Specifies the name of a fullword field that will contain the status of the tested bit: 0 -- OFF 1 -- ON</td>
</tr>
</tbody>
</table>

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**

```assembly
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>socket-descriptor</code></td>
<td>Specifies the name of a fullword field containing the socket descriptor whose bit must be set ON in the bit list.</td>
</tr>
<tr>
<td><code>bit-list</code></td>
<td>Specifies the name of the area containing the bit list.</td>
</tr>
<tr>
<td><code>bit-list-length</code></td>
<td>Specifies the name of a fullword field containing the length of the bit-list in bytes. <code>bit-list-length</code> can be specified as an absolute expression. <code>bit-list-length</code> must be a multiple of 4.</td>
</tr>
<tr>
<td><code>bit-order</code></td>
<td>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 <code>option</code> parameter of the SELECT or SELECTX function. <code>bit-order</code> can be specified as an absolute expression. The following are the accepted values: SEL@BBKW (default) SEL@BFWD</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit-list</td>
<td>Specifies the name of the area containing the bit list.</td>
</tr>
<tr>
<td>bit-list-length</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pointer-to-addrinfo-structure</td>
<td>Specifies the name of a fullword field containing the address of the ADDRINFO structure to release.</td>
</tr>
</tbody>
</table>

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**

```assembly
label
  #SOCKET GETADDRINFO,
  RETCODE=return-code,
  ERRNO=errno,
  RSNCODE=reason-code,
  HOSTNAME=hostname,
  HOSTNAML=hostname-length,
  SERVNAME=service-name,
  SERVNAMEL=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**

<table>
<thead>
<tr>
<th>Parameter Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>hostname</strong></td>
<td>Specifies the name of an area containing the name of the host to resolve.</td>
</tr>
<tr>
<td><strong>hostname-length</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>service-name</strong></td>
<td>Specifies the name of an area containing the name of the service.</td>
</tr>
<tr>
<td><strong>service-name-length</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
Parameter Description

**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.

**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 156).
- 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 Reference section.
- 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.

<table>
<thead>
<tr>
<th>EQUate Symbol</th>
<th>Field Name</th>
<th>TCP Protocol Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI@PASSV</td>
<td>SOCKET-AFLAGS-PASSIVE</td>
<td>AI_PASSIVE</td>
</tr>
<tr>
<td>AI@CANOK</td>
<td>SOCKET-AFLAGS-CANONNAMEOK</td>
<td>AI_CANONNAMEOK</td>
</tr>
<tr>
<td>AI@NHOST</td>
<td>SOCKET-AFLAGS-NUMERICHOST</td>
<td>AI_NUMERICHOST</td>
</tr>
<tr>
<td>AI@NSERV</td>
<td>SOCKET-AFLAGS-NUMERICSERV</td>
<td>AI_NUMERICSERV</td>
</tr>
<tr>
<td>AI@V4MAP</td>
<td>SOCKET-AFLAGS-V4MAPPED</td>
<td>AI_V4MAPPED</td>
</tr>
<tr>
<td>AI@ALL</td>
<td>SOCKET-AFLAGS-ALL</td>
<td>AI_ALL</td>
</tr>
<tr>
<td>AI@ADDRRC</td>
<td>SOCKET-AFLAGS-ADDRCONFIG</td>
<td>AI_ADDRCONFIG</td>
</tr>
</tbody>
</table>

**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

\[
\text{label} \quad \#\text{SOCKET GETHOSTBYADDR,}
\]
\[
\text{RETCODE=return-code,}
\]
\[
\text{ERRNO=errno,}
\]
\[
\text{RSNCODE=reason-code,}
\]
\[
\text{IPADDR=ip-address,}
\]
\[
\text{IPADDRL=ip-address-length,}
\]
\[
\text{DOMAIN=domain,}
\]
\[
\text{HOSTENTP=hostentp,}
\]
\[
\text{PLIST=parameter-list-area,}
\]
\[
\text{RGSV=(rgsv)}
\]

List of USING Parameters

\[
\text{SOCKET-FUNCTION-GETHOSTBYADDR, return-code, errno, reason-code, ip-address, ip-address-length, domain, hostentp}
\]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip-address</td>
<td>Specifies the name of a fullword field containing the binary format IP address to resolve.</td>
</tr>
<tr>
<td>ip-address-length</td>
<td>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.</td>
</tr>
<tr>
<td>domain</td>
<td>Specifies the name of a fullword field containing the domain. domain can be specified as an absolute expression. Currently, only AF_INET is supported.</td>
</tr>
<tr>
<td>hostentp</td>
<td>Specifies the name of a fullword field in which the system returns the address of a HOSTENT structure containing the information about the host.</td>
</tr>
</tbody>
</table>

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 156).

**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 Reference section.

**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 Reference section 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hostname</code></td>
<td>Specifies the name of an area containing the name of the host to resolve.</td>
</tr>
<tr>
<td><code>hostname-length</code></td>
<td>Specifies the name of a fullword field containing the length of <code>hostname</code>. <code>hostname-length</code> can be specified as an absolute expression.</td>
</tr>
<tr>
<td><code>hostentp</code></td>
<td>Specifies the name of a fullword field where the system returns the address of a HOSTENT structure containing the information about the host.</td>
</tr>
</tbody>
</table>

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 156).

**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 Reference section.

**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 Reference section 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip-address</td>
<td>Specifies the name of a fullword field in which the service returns the IP address in binary format.</td>
</tr>
</tbody>
</table>

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,ret
turned-hostname-length
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostname</td>
<td>Specifies the name of an area in which the service returns the host name.</td>
</tr>
</tbody>
</table>
### GETNAMEINFO

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

**Assembler**

```assembly
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

```assembly
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>hostname-length</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>returned-hostname-length</strong></td>
<td>Specifies the name of a fullword field in which the actual length of the host name is returned.</td>
</tr>
<tr>
<td><strong>sockaddr</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>sockaddr-length</strong></td>
<td>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).</td>
</tr>
<tr>
<td><strong>service-name</strong></td>
<td>Specifies the name of an area where the system returns the service name corresponding to the port number specified in the sockaddr structure.</td>
</tr>
<tr>
<td><strong>service-name-length</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>returned-service-name-length</td>
<td>Specifies the name of a fullword field into which the actual length of the service name is returned. <code>service-name</code>, <code>service-name-length</code> and <code>returned-service-name-length</code> are optional; specify all three parameters, or none of them. If none of these parameters are specified, <code>hostname</code>, <code>hostname-length</code>, and <code>returned-hostname-length</code> must be specified.</td>
</tr>
<tr>
<td>hostname</td>
<td>Specifies the name of an area where the system returns the <code>hostname</code> corresponding to the IP address specified in the sockaddr structure.</td>
</tr>
<tr>
<td>hostname-length</td>
<td>Specifies the name of a fullword field containing the length of <code>hostname</code>. <code>hostname-length</code> can be specified as an absolute expression. The maximum value for this parameter is 4096.</td>
</tr>
<tr>
<td>returned-hostname-length</td>
<td>Specifies the name of a fullword field into which the length of the host name is returned. <code>hostname</code>, <code>hostname-length</code> and <code>returned-hostname-length</code> are optional; specify all three parameters, or none of them. If none of these parameters are specified, <code>service-name</code>, <code>service-name-length</code>, and <code>returned-service-name-length</code> must be specified.</td>
</tr>
<tr>
<td>flags</td>
<td>Specifies the name of a fullword field containing flags to control the resolution of the socket address.</td>
</tr>
</tbody>
</table>

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 Reference section.
- 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`.

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

⚠️ **PL/I programs:** The `SOCKET_MISC_DEFINITION` is used and the dashes are replaced by underscores.

GETPEERNAME

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

Assembler
#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

<table>
<thead>
<tr>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
</tr>
<tr>
<td>sockaddr</td>
</tr>
<tr>
<td>sockaddr-length</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service-name</td>
<td>Specifies the name of an area containing the name of the service to resolve.</td>
</tr>
<tr>
<td>service-name-length</td>
<td>Specifies the name of a fullword field containing the length of service-name. service-name-length can be specified as an absolute expression.</td>
</tr>
<tr>
<td></td>
<td>Specifies the maximum value for this parameter is 256.</td>
</tr>
<tr>
<td>protocol-name</td>
<td>Specifies the name of an area containing the name of the protocol to use.</td>
</tr>
<tr>
<td>protocol-name-length</td>
<td>Specifies the name of a fullword field containing the length of protocol-name. protocol-name-length can be specified as an absolute expression.</td>
</tr>
<tr>
<td></td>
<td>Specifies the maximum value for this parameter is 256.</td>
</tr>
<tr>
<td>serventp</td>
<td>Specifies the name of a fullword field where the system returns the address of a SERVENT structure containing the information about the service.</td>
</tr>
</tbody>
</table>

Notes

- The services socket functions are supported by CA IDMS's internal services resolver. For more information, see the CA IDMS System Reference section.

- 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 156).

- 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port-number</td>
<td>Specifies the name of a fullword field containing the port-number to resolve.</td>
</tr>
<tr>
<td>protocol-name</td>
<td>Specifies the name of an area containing the name of the protocol to use.</td>
</tr>
<tr>
<td>protocol-name-length</td>
<td>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.</td>
</tr>
<tr>
<td>serventp</td>
<td>Specifies the name of a fullword field where the system returns the address of a SERVENT structure containing the information about the service.</td>
</tr>
</tbody>
</table>

Notes

- The services socket functions are supported by CA IDMS's internal services resolver. For more information, see the CA IDMS System Reference section.

- 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 156).

- 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
CA IDMS Reference - 19.0

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>Specifies the name of a fullword field containing the socket descriptor from which to retrieve the name.</td>
</tr>
<tr>
<td>sockaddr</td>
<td>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.</td>
</tr>
<tr>
<td>sockaddr-length</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>Specifies the name of a fullword field containing the socket descriptor for which the service is to be performed.</td>
</tr>
<tr>
<td>level</td>
<td>Specifies the name of a fullword field containing the level for the option. level can be specified as an absolute expression.</td>
</tr>
<tr>
<td>option-name</td>
<td>Specifies the name of a fullword field indicating the option to retrieve. option-name can be specified as an absolute expression.</td>
</tr>
<tr>
<td>option-value</td>
<td>Specifies the name of an area that will contain the requested data.</td>
</tr>
<tr>
<td>option-value-length</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

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

**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
List of USING Parameters

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer</td>
<td>Specifies the name of a buffer that receives the list of all the stacks. This parameter is optional.</td>
</tr>
<tr>
<td>buffer-length</td>
<td>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.</td>
</tr>
<tr>
<td>output-format</td>
<td>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.</td>
</tr>
<tr>
<td>output-length</td>
<td>Specifies the name of a fullword field containing the actual length required to hold all the output in the requested format.</td>
</tr>
<tr>
<td>stacks-count</td>
<td>Specifies the name of a fullword field containing the number of TCP/IP stacks currently defined (but not necessarily active) in the system.</td>
</tr>
</tbody>
</table>

**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 60).

**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.
CA IDMS Reference - 19.0

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input-field</td>
<td>Specifies the name of a fullword field containing the integer to convert.</td>
</tr>
<tr>
<td>output-field</td>
<td>Specifies the name of a fullword field that receives the converted integer.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input-field</td>
<td>Specifies the name of a halfword field containing the integer to convert.</td>
</tr>
<tr>
<td>output-field</td>
<td>Specifies the name of a halfword field that receives the converted integer.</td>
</tr>
</tbody>
</table>

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,
IPADDR=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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ip-address</strong></td>
<td>Specifies the name of an area containing the IP address in standard dotted string format.</td>
</tr>
<tr>
<td><strong>ip-address-string</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>ip-address-string-length</strong></td>
<td>Specifies the name of a fullword field that will contain the IP address in binary format.</td>
</tr>
</tbody>
</table>

**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,
       IPADDS=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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ip-address</strong></td>
<td>Specifies the name of a fullword field containing the IP address in binary format.</td>
</tr>
</tbody>
</table>
Parameter | Description
---|---
`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**

```assembly
label  
#SOCKET_INET_NTOP, 
RETCODE=return-code, 
ERRNO=errno, 
REASONCODE=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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>domain</code></td>
<td>Specifies the name of a fullword field containing the domain. <code>domain</code> can be specified as an absolute expression. Possible values are AF@INET and AF@INET6.</td>
</tr>
<tr>
<td><code>ip-address</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>ip-address-string</code></td>
<td>Specifies the name of an area in which to return the IP address in standard string format.</td>
</tr>
<tr>
<td><code>ip-address-string-length</code></td>
<td>Specifies the name of a fullword field containing the length of <code>ip-address-string</code>. <code>ip-address-string-length</code> can be specified as an absolute expression. The maximum value for this parameter is 4096.</td>
</tr>
<tr>
<td><code>returned-ip-address-string-length</code></td>
<td>Specifies the name of a fullword field in which the actual length of the IP address string is returned.</td>
</tr>
</tbody>
</table>
INET_PTON

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

Assembler

\[
\text{label} \quad \#\text{SOCKET INET_PTON,} \\
\quad \text{RETCODE=return-code,} \\
\quad \text{ERRNO=errno,} \\
\quad \text{RSNCODE=reason-code,} \\
\quad \text{DOMAIN=domain,} \\
\quad \text{IPADDR=ip-address\_area,} \\
\quad \text{IPADDRSL=ip-address\_string\_length,} \\
\quad \text{IPADDR=ip-address,} \\
\quad \text{PLIST=parameter\_list\_area,} \\
\quad \text{RGSV=(rgsv)}
\]

List of USING Parameters

\[
\text{SOCKET-FUNCTION-INETPTON,return-code,errno,reason-code,domain,ip-address\_string,return} \\
\quad \text{ed-ip-address\_string\_length,ip-address}
\]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain</td>
<td>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.</td>
</tr>
<tr>
<td>ip-address-string</td>
<td>Specifies the name of an area containing the IP address in standard string format.</td>
</tr>
<tr>
<td>ip-address-string-length</td>
<td>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</td>
</tr>
<tr>
<td>ip-address</td>
<td>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.</td>
</tr>
</tbody>
</table>

IOCTL

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

Assembler

\[
\text{Label} \quad \#\text{SOCKET IOCTL,} \\
\quad \text{RETCODE=return-code,} \\
\quad \text{ERRNO=errno,} \\
\quad \text{RSNCODE=reason-code,} \\
\quad \text{SOCK=socket\_descriptor,} \\
\quad \text{COMMAND=command,} \\
\quad \text{ARGUMENT=argument,} \\
\quad \text{ARGUMENTL=argument\_length,} \\
\quad \text{PLIST=parameter\_list\_area,} \\
\quad \text{RGSV=(rgsv)}
\]
List of USING Parameters

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>The name of a fullword field containing the socket-descriptor to process.</td>
</tr>
<tr>
<td>command</td>
<td>The name of a fullword field containing the command to perform on the socket. command can be specified as an absolute expression.</td>
</tr>
<tr>
<td>argument</td>
<td>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.</td>
</tr>
<tr>
<td>argument-length</td>
<td>The name of a fullword field that contains the length of the argument area.</td>
</tr>
</tbody>
</table>

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.

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

**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,`
CA IDMS Reference - 19.0

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

<table>
<thead>
<tr>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
</tr>
</tbody>
</table>

Specifies the name of a fullword field containing the socket descriptor on which to listen.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>backlog</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>input-field</td>
</tr>
</tbody>
</table>

Specifies the name of a fullword field containing the integer to convert.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>output-field</td>
</tr>
</tbody>
</table>

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

\[
\text{\textit{label}} ~ \#\text{SOCKET} \ NTOHS, \\
\text{FIELDIN} = \textit{input-field}, \\
\text{FIELDOUT} = \textit{output-field}, \\
\text{PLIST} = \textit{parameter-list-area}, \\
\text{RGSV} = (\text{rgsv})
\]

List of USING Parameters

\text{SOCKET-FUNCTION-NTOHS, input-field, output-field}

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{input-field}</td>
<td>Specifies the name of a halfword field containing the integer to convert.</td>
</tr>
<tr>
<td>\textit{output-field}</td>
<td>Specifies the name of a halfword field that receives the converted integer.</td>
</tr>
</tbody>
</table>

READ

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

Assembler

\[
\text{\textit{label}} ~ \#\text{SOCKET} \ READ, \\
\text{RETCODE} = \textit{return-code}, \\
\text{ERRNO} = \textit{errno}, \\
\text{RSNCODE} = \textit{reason-code}, \\
\text{SOCK} = \textit{socket-descriptor}, \\
\text{BUFFER} = \textit{buffer}, \\
\text{BUFFERL} = \textit{buffer-length}, \\
\text{RETLEN} = \textit{read-length}, \\
\text{PLIST} = \textit{parameter-list-area}, \\
\text{RGSV} = (\text{rgsv})
\]

List of USING Parameters

\text{SOCKET-FUNCTION-READ, return-code, errno, reason-code, socket-descriptor, buffer, buffer-length, read-length}

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{socket-descriptor}</td>
<td>Specifies the name of a fullword field containing the socket descriptor to read from.</td>
</tr>
<tr>
<td>\textit{buffer}</td>
<td>Specifies the name of the area where the data is to be placed.</td>
</tr>
</tbody>
</table>
Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer-length</td>
<td>Specifies the name of a fullword field containing the length of the buffer. buffer-length can be specified as an absolute expression.</td>
</tr>
<tr>
<td>read-length</td>
<td>Specifies the name of a fullword field in which the actual length of the data read is returned.</td>
</tr>
</tbody>
</table>

**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 63).

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>Specifies the name of a fullword field containing the socket descriptor from which to read.</td>
</tr>
<tr>
<td>buffer</td>
<td>Specifies the name of the area where the data is to be placed.</td>
</tr>
<tr>
<td>buffer-length</td>
<td>Specifies the name of a fullword field containing the length of the buffer. buffer-length can be specified as an absolute expression.</td>
</tr>
<tr>
<td>flags</td>
<td>Specifies the name of a fullword field containing information on how the data is to be received.</td>
</tr>
<tr>
<td>read-length</td>
<td>Specifies the name of a fullword field in which the actual length of the data read is returned.</td>
</tr>
</tbody>
</table>
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 63).

- 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.

<table>
<thead>
<tr>
<th>EQUate Symbol</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG@DROU</td>
<td>SOCKET-MSGFLAGS-DONTRoute</td>
<td>Send without network routing</td>
</tr>
<tr>
<td>MSG@OOB</td>
<td>SOCKET-MSGFLAGS-OOB</td>
<td>Send and receive out-of-band data</td>
</tr>
<tr>
<td>MSG@PEEK</td>
<td>SOCKET-MSGFLAGS-PEEK</td>
<td>Peek at incoming data</td>
</tr>
<tr>
<td>MSG@WALL</td>
<td>SOCKET-MSGFLAGS-WAITALL</td>
<td>Wait until all data returned</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Parameter Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>Specifies the name of a fullword field containing the socket descriptor from which to read.</td>
</tr>
</tbody>
</table>
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer</td>
<td>Specifies the name of the area where the data is to be placed.</td>
</tr>
<tr>
<td>buffer-length</td>
<td>Specifies the name of a fullword field containing the length of the buffer. buffer-length can be specified as an absolute expression.</td>
</tr>
<tr>
<td>flags</td>
<td>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.</td>
</tr>
<tr>
<td>sockaddr</td>
<td>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.</td>
</tr>
<tr>
<td>sockaddr-length</td>
<td>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.</td>
</tr>
<tr>
<td>read-length</td>
<td>Specifies the name of a fullword field in which the actual length of the data read is returned.</td>
</tr>
</tbody>
</table>

### 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 63).

- **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**

```assembly
label        #SOCKET SELECT,
RETCODE=return-code,
ERRNO=errno,
RSNCODE=reason-code,
```
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

<table>
<thead>
<tr>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number-of-socket-descriptors</td>
</tr>
<tr>
<td>read-list</td>
</tr>
<tr>
<td>read-list-length</td>
</tr>
<tr>
<td>write-list</td>
</tr>
<tr>
<td>write-list-length</td>
</tr>
<tr>
<td>exception-list</td>
</tr>
<tr>
<td>exception-list-length</td>
</tr>
<tr>
<td>option</td>
</tr>
<tr>
<td>timeval-structure</td>
</tr>
<tr>
<td>returned-number-of-descriptors</td>
</tr>
<tr>
<td>ecb</td>
</tr>
<tr>
<td>ecb-list</td>
</tr>
</tbody>
</table>
Parameter Description

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 nth 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.

<table>
<thead>
<tr>
<th>EQUate Symbol</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEL@BB KW</td>
<td>SOCKET-SELECT-BITBACKWARD</td>
<td>Specifies that the bits in the fullwords are in the backward order. This is the default value if the parameter is assigned to NULL.</td>
</tr>
<tr>
<td>SEL@BF RW</td>
<td>SOCKET-SELECT-BITFORWARD</td>
<td>Specifies that the bits in each fullword are in the forward order</td>
</tr>
<tr>
<td>SEL@BY TV</td>
<td>SOCKET-SELECT-BYTELIST</td>
<td>Specifies that the read-list, write-list, and exception-list are byte lists instead of bit lists.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>Specifies the name of a fullword field containing the socket descriptor on which to do the send.</td>
</tr>
<tr>
<td>buffer</td>
<td>Specifies the name of the area containing the data to be sent.</td>
</tr>
<tr>
<td>buffer-length</td>
<td>Specifies the name of a fullword field containing the length of the buffer. buffer-length can be specified as an absolute expression.</td>
</tr>
<tr>
<td>flags</td>
<td>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.</td>
</tr>
<tr>
<td>sent-length</td>
<td>Specifies the name of a fullword field in which the actual length of the data sent is returned.</td>
</tr>
</tbody>
</table>

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 63).

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>Specifies the name of a fullword field containing the socket descriptor on which to do the send.</td>
</tr>
<tr>
<td>buffer</td>
<td>Specifies the name of the area containing the data to be sent.</td>
</tr>
<tr>
<td>buffer-length</td>
<td>Specifies the name of a fullword field containing the length of the buffer. buffer-length can be specified as an absolute expression.</td>
</tr>
<tr>
<td>flags</td>
<td>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.</td>
</tr>
<tr>
<td>sockaddr</td>
<td>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.</td>
</tr>
<tr>
<td>sockaddr-length</td>
<td>Specifies the name of a fullword field containing the length of sockaddr. Socketaddr-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).</td>
</tr>
<tr>
<td>sent-length</td>
<td>Specifies the name of a fullword field in which the actual length of the data sent is returned.</td>
</tr>
</tbody>
</table>

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 63).
- 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,
    RSCODE=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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>Specifies the name of a fullword field containing the socket descriptor for which the service is to be performed.</td>
</tr>
<tr>
<td>level</td>
<td>Specifies the name of a fullword field containing the level for the option. level can be specified as an absolute expression.</td>
</tr>
<tr>
<td>option-name</td>
<td>Specifies the name of a fullword field indicating the option to set. option-name can be specified as an absolute expression.</td>
</tr>
<tr>
<td>option-value</td>
<td>Specifies the name of an area containing the data to associate with the socket.</td>
</tr>
<tr>
<td>option-value-length</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack-name</td>
<td>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.</td>
</tr>
<tr>
<td>stack-name-length</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 60).

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

<table>
<thead>
<tr>
<th>Parameter Description</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>Specifies the name of a fullword field containing the socket descriptor to shut down.</td>
</tr>
<tr>
<td>how-condition</td>
<td>Specifies the name of a fullword field indicating the effect of the shutdown on read and write operations. <em>how-condition</em> can be specified as an absolute expression.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>EQUate Symbol</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHUT_R</td>
<td>SOCKET-SHUTDOWN-READ</td>
<td>Terminates read communication (from the socket)</td>
</tr>
<tr>
<td>SHUT_W</td>
<td>SOCKET-SHUTDOWN-WRITE</td>
<td>Terminates write communication (to the socket)</td>
</tr>
<tr>
<td>SHUT_RW</td>
<td>SOCKET-SHUTDOWN-READ-WRITE</td>
<td>Terminates both read and write communication</td>
</tr>
</tbody>
</table>

⚠️ **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

```assembly
label    #SOCKET SOCKET,
        RETCODE=return-code,
        ERRNO=errno,
        RSNCOD=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

<table>
<thead>
<tr>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>domain</strong></td>
</tr>
<tr>
<td><strong>type</strong></td>
</tr>
<tr>
<td><strong>protocol-number</strong></td>
</tr>
<tr>
<td><strong>new-socket-descriptor</strong></td>
</tr>
</tbody>
</table>

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 section.
- About the DCMT VARY TCP/IP command, see the Reference section.

- 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.

<table>
<thead>
<tr>
<th>EQUate Symbol</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AF@INET*</td>
<td>SOCKET-FAMILY-AFINET</td>
<td>AF_INET address family</td>
</tr>
<tr>
<td>AF@INET6</td>
<td>SOCKET-FAMILY-AFINET6</td>
<td>AF_INET6 address family</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>EQUate Symbol</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STREAM*</td>
<td>SOCKET-TYPE-STREAM</td>
<td>Stream -- Connection oriented and reliable</td>
</tr>
</tbody>
</table>
### EQUate Symbol

<table>
<thead>
<tr>
<th>EQUate Symbol</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATAGRAM</td>
<td>SOCKET-TYPE-DATAGRAM</td>
<td>Datagram -- Connectionless and unreliable</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>EQUate Symbol</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROTI P</td>
<td>SOCKET-PROTOCOL-IP</td>
<td>Default protocol</td>
</tr>
<tr>
<td>PROTTCP*</td>
<td>SOCKET-PROTOCOL-TCP</td>
<td>TCP protocol</td>
</tr>
<tr>
<td>PROTU D P</td>
<td>SOCKET-PROTOCOL-UDP</td>
<td>UDP protocol</td>
</tr>
<tr>
<td>PROT IPV6</td>
<td>SOCKET-PROTOCOL-IPV6</td>
<td>IPv6 protocol</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket-descriptor</td>
<td>Specifies the name of a fullword field containing the socket descriptor on which to send.</td>
</tr>
<tr>
<td>buffer</td>
<td>Specifies the name of the area containing the data to be sent.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>buffer-length</td>
<td>Specifies the name of a fullword field containing the length of the buffer. buffer-length can be specified as an absolute expression.</td>
</tr>
<tr>
<td>sent-length</td>
<td>Specifies the name of a fullword field in which the actual length of the data sent is returned.</td>
</tr>
</tbody>
</table>

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 63).
Web Services API Support

The CA IDMS Web Services API is a call-level interface for the programmatic invocation and provision of Web services within CA IDMS. This section includes the following topics:

- Web Services API Functions (see page 109)
- Web Services API Records (see page 117)
- Supported Programming Languages (see page 126)

The following programming languages, supported within CA IDMS, can use the IDMS Web Services API:

- Assembler
- CA IDMS ADS/Online
- COBOL
- PL/I

The CA IDMS Web Services interface supports the 1.1 version of the SOAP (Simple Object Access Protocol) service protocol. SOAP is a highly-used, industry standard protocol for passing XML-based information between programs in a distributed environment. A general understanding of SOAP is strongly recommended before beginning your Web Services development efforts. For more information on SOAP, as well as Web Services in general, see the World Wide Web consortium (W3C) web site.

Note that the Web Services API is only available for IDMS regions running on z/OS.

Web Services API Functions

This section describes the following functions supported within the CA IDMS Web Services API, and in the COBOL, ADS and PL/I languages:

- GETOPTION (see page 110)
- INITIALIZE (see page 111)
- RECEIVE (see page 112)
- RELEASE (see page 113)
- REQUEST (see page 113)
- SEND (see page 114)
- SETOPTION (see page 115)
- Common Parameters (see page 117)
Note: Only parameters specific to the function are listed with the function. For a list of the parameters common to all these Web Services API functions see Common Parameters (see page 117).

GETOPTION

The GETOPTION function (code 12) is used to retrieve the settings for any of the CA IDMS Web services system-level options, for the current Web Services request.

The following options can be retrieved using GETOPTION:

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Option Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG-SERVICES</td>
<td>1</td>
<td>Turns on logging within the IDMS Web Services.</td>
</tr>
<tr>
<td>LOG-PROGRAM</td>
<td>2</td>
<td>Turns on logging within a specific user program.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Developers must add code within their service programs to act on this field.</td>
</tr>
<tr>
<td>REQUIRE-SIGNON</td>
<td>3</td>
<td>Dictates that provided Services must sign on to IDMS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pertains to Provider Services only.</td>
</tr>
<tr>
<td>CHECK-AUTH</td>
<td>4</td>
<td>Indicates that provided Services must participate in a security group.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pertains to Provider Services only.</td>
</tr>
<tr>
<td>CONNECT-TIMEOUT</td>
<td>5</td>
<td>Specifies the wait time for connecting to external services.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pertains to Consumer Services only.</td>
</tr>
<tr>
<td>READ-WRITE-TIMEOUT</td>
<td>6</td>
<td>Specifies the wait time for all other TCP/IP API calls.</td>
</tr>
<tr>
<td>XML-CODE-PAGE</td>
<td>7</td>
<td>Indicates the code page used for COBOL XML processing.</td>
</tr>
</tbody>
</table>

Parameters

- **option-number**
  (Input parameter) Specifies the name of a full-word pointer field to the `option-number`. A sample definition of the `option-number` field can be found in the WS-OPTION-NUMBER-RECORD record, as WS-OPTION-NUMBER. The layout for the WS-OPTION-NUMBER-RECORD record can be found in Web Services API Records (see page 117).

- **option-value**
  (Output parameter) Specifies the name of a full-word pointer field to the `option-value`. A sample definition of the `option-value` field can be found in the WS-OPTION-VALUE-RECORD record, as WS-OPTION-VALUE. The layout for the WS-OPTION-VALUE-RECORD record can be found in the Web Services API Records (see page 117) section.
Examples

The following examples show how to code the GETOPTION function in COBOL, ADS, and PL/I:

Example for COBOL, ADS and PL/I

```
WS-GETOPTION, return-code, error-info, request-handle, option-number, option-value.
```

Considerations for GETOPTION

- Developers can use the elements contained within the WS-CALL-INTERFACE record to simplify their use of the CA IDMS Web Services API.
- CA ADS/Online programs must pass all parameters as records. As a convenience, all single-element parameters have been added to their own records, using a prefix of 'WS-' and a suffix of '-RECORD'. For example, the `option-number` element can be found under record `WS-OPTION-NUMBER-RECORD`.

For more information on the Web services system-level options, refer to "Product Configuration" at Product Requirements, Configuration, and Components (https://docops.ca.com/display/IDMS/Product+Requirements%2C+Configuration%2C+and+Components).

INITIALIZE

The INITIALIZE function (code 4) allocates and initializes the data structures used in the Web services request. It is only needed when consuming an external Web service.

Parameters

- `interface-version`
  Specifies the name of a full-word pointer field to the WS-INTERFACE-VERSION-NUMBER record. The layout for the WS-INTERFACE-VERSION-NUMBER record can be found in Web Services API Records (see page 117).

Examples

The following examples show how to code the INITIALIZE function in COBOL, ADS, and PL/I:

Example for COBOL, ADS and PL/I

```
WS-INITIALIZE, return-code, error-info, request-handle, interface-version.
```

Considerations for INITIALIZE

Following is information you need to know about the INITIALIZE function:

- The supported interface version numbers are: Major version = 01, Minor version = 00
- Only one INITIALIZE function call can be made within a program.
For Consumer services, the INITIALIZE function must be called before any other Web Services API function call.

The function returns a REQUEST-HANDLE field, which must be passed on all subsequent Web Services function calls.

The function is not needed for Provider services. Services hosted within CA IDMS are automatically initialized by the IDMS Web services interface, before being called by the Soap Server program (IDMSWSSS). The request-handle is passed to the Provider implementation program on invocation.

RECEIVE

The RECEIVE function (code 24) is used to return the address and length of an incoming Web service Request buffer.

Parameters

- request-msg-ptr
  Specifies the name of a full-word pointer field that points to a pointer that points to the WS-REQUEST-MESSAGE-DATA record. The WS-REQUEST-MESSAGE-DATA record is populated on return from the RECEIVE function call. The layout for the WS-REQUEST-MESSAGE-DATA record can be found in Web Services API Records (see page 117).

- request-message-descriptor
  Specifies the name of a full-word pointer field that points to the WS-REQUEST-MESSAGE-DESCRIPTOR record. The WS-REQUEST-MESSAGE-DESCRIPTOR record is populated on return from the RECEIVE function call. The layout for the WS-REQUEST-MESSAGE-DESCRIPTOR record can be found in Web Services API Records (see page 117).

Examples

The following examples show how to code the RECEIVE function in COBOL, ADS, and PL/I:

Example: COBOL, ADS and PL/I

```cobol
```

Considerations for RECEIVE

Following is information you need to know about the RECEIVE function:

- After calling the RECEIVE function, set the address of your WS-REQUEST-MESSAGE-DATA record to the address contained in the REQUEST-MSG-PTR record.
- The RECEIVE function can be invoked by a Provider service only.
- The RECEIVE function can only be issued once within a Provider service.
The RECEIVE function must return the entire Request.

The RECEIVE function performs all security checks. This includes sign-on processing (if the 'REQUIRE SIGNON' configuration option is enabled) as well as security group validation (if the 'CHECK AUTHORIZATION' configuration option is enabled). Any security failure will produce an HTTP 401 response.

**RELEASE**

The RELEASE function (code 28) is used to terminate a Web Services request. It frees all structures allocated on behalf of the Web Services request.

*Note:* RELEASE function parameters are common to all these Web Services API functions. For a list of the RELEASE parameters see Common Parameters (see page 117).

### Examples

The following examples show how to code the RELEASE function in COBOL, ADS, and PL/I:

**Example: COBOL, ADS and PL/I**

```
WS-RELEASE, return-code, error-info, request-handle.
```

**REQUEST**

The REQUEST function (code 16) is a synchronous function used to invoke an external Web service. Control is returned to the calling program after the external Web services is invoked and the Response message is returned.

### Parameters

- **request-info**
  Specifies the name of a full-word pointer field to the WS-REQUEST-INFO record. This structure contains the attributes of the service to be invoked. The layout for the WS-REQUEST-INFO record can be found in Web Services API Records (see page 117).

- **request-message-data**
  Specifies the name of a full-word pointer field that points to a pointer (a pointer to a pointer) that points to the WS-REQUEST-MESSAGE-DATA record. The WS-REQUEST-MESSAGE-DATA record must be properly populated at the time of the call. The layout for the WS-REQUEST-MESSAGE-DATA record can be found in Web Services API Records (see page 117).

- **request-message-descriptor**
  Specifies the name of a full-word pointer field, which points to the WS-REQUEST-MESSAGE-DESCRIPTOR record. The layout for the WS-REQUEST-MESSAGE-DESCRIPTOR record can be found in Web Services API Records (see page 117). The WS-REQUEST-MESSAGE-DESCRIPTOR record must be properly populated at the time of the call. This means that:
The WS-REQ-MSG-BUFF-LEN field must be set to reflect the total size of the Request Buffer, as contained in the WS-REQUEST-MESSAGE-DATA record.

- Both the WS-REQ-MSG-SOURCE-FMT, as well as the WS-REQ-MSG-TARGET-FMT fields must be set to 'X' (for XML).
- The WS-REQ-MSG-BYTES-REM must be set to zero.

**response-message-data**
Specifies the name of a full-word pointer field that points to a pointer (a pointer to a pointer) that points to the WS-RESPONSE-MESSAGE-DATA record. The WS-RESPONSE-MESSAGE-DATA record will be populated on return from the REQUEST function call. The layout for the WS-RESPONSE-MESSAGE-DATA record can be found in Web Services API Records (see page 117).

**response-message-descriptor**
Specifies the name of a full-word pointer field, which points to the WS-RESPONSE-MESSAGE-DESCRIPTOR record. The WS-RESPONSE-MESSAGE-DESCRIPTOR record will be populated on return from the REQUEST function call. The layout for the WS-RESPONSE-MESSAGE-DESCRIPTOR record can be found in Web Services API Records (see page 117).

**Examples**

The following examples show how to code the REQUEST function in COBOL, ADS, and PL/I:

**Example: COBOL, ADS and PL/I**

```plaintext
```

**Considerations for REQUEST**

Following is information you need to know about the REQUEST function:

- The REQUEST function should not be used within Provider service programs.
- The REQUEST function must be preceded by an INITIALIZE function call.

**SEND**

The SEND function (code 20) is used to transmit a Response message to a service Consumer.

**Parameters**

- **response-msg-ptr**
  The name of a full-word pointer field (a pointer to a pointer) that points to a variable length buffer containing the service Response message.

- **response-message-descriptor**
  The name of a full-word pointer field that points to the output WS-MESSAGE-DESCRIPTOR record. The layout for the WS-MESSAGE-DESCRIPTOR record can be found in Web Services API Records (see page 117).
Examples

The following examples show how to code the SEND function in COBOL, ADS, and PL/I:

Example: COBOL, ADS and PL/I

\`
\`

Considerations for SEND

Following is information you need to know about the SEND function:

- Prior to calling the SEND function, set the RESPONSE-MSG-PTR pointer to the address of your WS-RESPONSE-MESSAGE-DATA.
- The SEND function can be invoked by a Provider service only.
- The SEND function can only be issued once.
- The SEND function must return the entire Response.

SETOPTION

The SETOPTION function (code 8) is used to override the default settings for any of the CA IDMS Web services system-level options, for the current Web Services request.

The following options are configurable, using SETOPTION:

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Option Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG-SERVICES</td>
<td>1</td>
<td>Turns on logging within the IDMS Web Services.</td>
</tr>
<tr>
<td>LOG-PROGRAM</td>
<td>2</td>
<td>Turns on logging within a specific user program.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Developers must add code within their service programs to act on this field.</td>
</tr>
<tr>
<td>REQUIRE-SIGNON</td>
<td>3</td>
<td>Dictates that provided Services must sign on to IDMS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pertains to Provider Services only. This option must be set before the RECEIVE function is invoked. See 'Considerations for SETOPTION' below.</td>
</tr>
<tr>
<td>CHECK-AUTH</td>
<td>4</td>
<td>Indicates that provided Services must participate in a security group.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pertains to Provider Services only. This option must be set before the RECEIVE function is invoked. See 'Considerations for SETOPTION' below.</td>
</tr>
<tr>
<td>CONNECT-TIMEOUT</td>
<td>5</td>
<td>Specifies the wait time for connecting to external services.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pertains to Consumer Services only.</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Specifies the wait time for all other TCP/IP API calls.</td>
</tr>
<tr>
<td>Option Name</td>
<td>Option Number</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>READ-WRITE-TIMEOUT</td>
<td></td>
<td>Indicates the code page used for COBOL XML processing.</td>
</tr>
<tr>
<td>XML-CODE-PAGE</td>
<td>7</td>
<td>Indicates the code page used for COBOL XML processing.</td>
</tr>
</tbody>
</table>

**Parameters**

- **option-number**
  Specifies the name of a full-word pointer field to the option-number. A sample definition of the option-number field can be found in the WS-OPTION-NUMBER-RECORD record, as WS-OPTION-NUMBER. The layout for the WS-OPTION-NUMBER-RECORD record can be found in Web Services API Records (see page 117). (https://docops.ca.com/display/IDMSCU/Web+Services+Interface+Records)

- **option-value**
  Specifies the name of a full-word pointer field to the option-value. A sample definition of the option-value field can be found in the WS-OPTION-VALUE-RECORD record, as WS-OPTION-VALUE. The layout for the WS-OPTION-VALUE-RECORD record can be found in the Web Services API Records (see page 117) section.

**Examples**

The following examples show how to code the SETOPTION function in COBOL, ADS, and PL/I:

**Example: COBOL, ADS and PL/I**

`WS-SETOPTION, return-code, error-info, request-handle, option-number, option-value.`

**Considerations for SETOPTION**

- Developers can use the elements contained within the WS-CALL-INTERFACE record to simplify their use of the CA IDMS Web Services API.

- Several options pertain to only one service type (Consumer or Provider). No attempt is made to validate the pertinence of the option, based on the service type setting (maintained in the Web Services Request Block).

- CA ADS/Online programs must pass all parameters as records. As a convenience, all single-element parameters are added to their own records, using a prefix of WS and a suffix of -RECORD. For example, the option-number element can be found under record WS-OPTION-NUMBER-RECORD.

- Any desired changes to the REQUIRE-SIGNON and CHECK-AUTH options must be made before the RECEIVE function is invoked, in order for them to be properly acted upon.

For more information on the Web services system-level options, refer to "Product Configuration" at Product Requirements, Configuration, and Components (https://docops.ca.com/display/IDMS/Product+Requirements%2C+Configuration%2C+and+Components).
Common Parameters

The following parameters are common to all the above CA IDMS Web Services API functions:

- **function-code**
  Specifies a 4-byte, full-word-aligned, integer field that the program sets to the desired Web Services function. Listed below is a sample definition of the `function-code` field: 01 WS-FUNCTION-CODE PIC S9(8) COMP.
  A detailed description of the supported functions can be found in Web Services API Records (see page 117).

- **return-code**
  Specifies a 4-byte, full-word-aligned, integer field that communicates the outcome of the operation. Listed below is a sample definition of the `return-code` field:
  01 WS-RETURN-CODE PIC S9(8) COMP.
  A detailed description of the supported functions can be found in Web Services API Records (see page 117).

- **error-info**
  Specifies a 4-byte, full-word-aligned, pointer to the WS-ERROR-INFO record structure. The WS-ERROR-INFO record contains detailed error information on the various warnings/errors returned on the Web Services API function calls.
  The layout for the WS-ERROR-INFO record can be found in Web Services API Records (see page 117).

- **request-handle**
  Specifies a 4-byte, full-word-aligned, integer field that identifies the Web Services REQUEST block. Listed below is a sample definition of the `request-handle` field:
  01 WS-REQUEST-HANDLE PIC S9(8) COMP.
  The `request-handle` field is required for all the CA IDMS Web Services API function calls.

Web Services API Records

This section describes the records used within the CA IDMS Web Services API and includes the following records:

- WS-FUNCTION-CODE-RECORD (see page 118)
- WS-INTERFACE-VERSION-NUMBER (see page 119)
- WS-REQUEST-INFO (see page 119)
- WS-REQUEST-HANDLE-RECORD (see page 120)
- WS-OPTION-NUMBER-RECORD (see page 120)
- WS-OPTION-VALUE-RECORD (see page 121)
- WS-REQUEST-MSG-DATA (Module) (see page 122)
- WS-REQUEST-MSG-PTR-RECORD (see page 122)
- WS-RESPONSE-MSG-DATA (Module) (see page 123)
- WS-RESPONSE-MSG-PTR-RECORD (see page 123)
- WS-REQUEST-MSG-DESCRIPTOR (see page 123)
- WS-RESPONSE-MSG-DESCRIPTOR (see page 124)
Each of these records is added to the Data Dictionary during product installation. The records are provided to simplify your use of the Web Services API during program development.

Since CA IDMS ADS/Online only supports the passing of records when linking to a program, record definitions are provided for each individual parameter within the Web Services API.

**WS-FUNCTION-CODE-RECORD**

```plaintext
ADD RECORD NAME IS WS-FUNCTION-CODE-RECORD VERSION IS 1.
  05 WS-FUNCTION-CODE PIC S9(8) COMP.
  88 WS-INITIALIZE VALUE 4.
     88 WS-SETOPTION VALUE 8.
     88 WS-GETOPTION VALUE 12.
     88 WS-REQUEST VALUE 16.
     88 WS-SEND VALUE 20.
     88 WS-RECEIVE VALUE 24.
     88 WS-RELEASE VALUE 28.
```

**Elements**

- **WS-FUNCTION-CODE**
  The code needed to invoke the desired Web Services function. Level 88 sub-elements include the following:

  - **WS-INITIALIZE**
    Directs the invocation of the Initialize function.

  - **WS-SETOPTION**
    Directs the invocation of the SetOption function.

  - **WS-GETOPTION**
    Directs the invocation of the GetOption function.

  - **WS-REQUEST**
    Directs the invocation of the Request function.

  - **WS-SEND**
    Directs the invocation of the Send function.

  - **WS-RECEIVE**
    Directs the invocation of the Receive function.

  - **WS-RELEASE**
    Directs the invocation of the Release function.
WS-INTERFACE-VERSION-NUMBER

ADD
RECORD NAME IS WS-INTERFACE-VERSION-NUMBER VERSION IS 1
  .
    05 WS-IVN-MAJOR-RELEASE
    PICTURE IS 99
    USAGE IS DISPLAY
    .
    05 WS-IVN-MINOR-RELEASE
    PICTURE IS 99
    USAGE IS DISPLAY
    .

Elements

- **WS-IVN-MAJOR-RELEASE**
  
The latest, installed major release of the Web Services API. Valid value is one.

- **WS-IVN-MINOR-RELEASE**
  
The latest, installed minor release of the Web Services API. Valid value is zero.

WS-REQUEST-INFO

ADD
RECORD NAME IS WS-REQUEST-INFO VERSION IS 1
  .
    05 WS-REQUEST-HOST
    PICTURE IS X(256)
    USAGE IS DISPLAY
    .
    05 WS-REQUEST-HOST-PORT
    PICTURE IS S9(08)
    USAGE IS COMP
    SYNCHRONIZED
    .
    05 WS-REQUEST-SERVICE
    PICTURE IS X(256)
    USAGE IS DISPLAY
    .
    05 WS-REQUEST-SOAPACTION
    PICTURE IS X(256)
    USAGE IS DISPLAY
    .
    05 WS-SERVICE-CREDENTIALS
    USAGE IS DISPLAY
    .
      10 WS-USERID
      PICTURE IS X(256)
      USAGE IS DISPLAY
      .
      10 WS-PASSWORD
      PICTURE IS X(256)
      USAGE IS DISPLAY
      .
    05 WS-SERVICE-AUTHORIZATION
    PICTURE IS X
    USAGE IS DISPLAY
    VALUE IS ( SPACE )
Elements

- **WS-REQUEST-HOST**
  The DNS Host-Name or IP-Address of the Service Host.

- **WS-REQUEST-HOST-PORT**
  The listening Port number, on the machine identified by WS-REQUEST-HOST.

- **WS-REQUEST-SERVICE**
  The name of the service to be invoked.

- **WS-REQUEST-SOAPACTION**
  Optional parameter. Value is dependent on the specific service. Identifies the action (or operation) to be taken for the service being invoked.

- **WS-USERID**
  Optional, based on the specific service. Identifies the User-ID for running the service. This field is required when WS-SERVICE-AUTHORIZATION is set to WS-BASIC-ACCESS.

- **WS-PASSWORD**
  Optional, based on the specific service. Identifies the password for running the service. This field is required when WS-SERVICE-AUTHORIZATION is set to WS-BASIC-ACCESS.

- **WS-SERVICE-AUTHORIZATION**
  The type of authorization required for the service being invoked. The only (non-space) value supported is "B," for Basic Access Authorization. When WS-BASIC-ACCESS is selected WS-USERID and WS-PASSWORD must be populated.

**WS-REQUEST-HANDLE-RECORD**

```
ADD
RECORD NAME IS WS-REQUEST-HANDLE-RECORD VERSION IS 1.
   05 WS-REQUEST-HANDLE PIC S9(8) COMP.
```

Elements

- **WS-REQUEST-HANDLE**
  The unique handle of your request. This parameter must be provided on all Web Services API function calls.

**WS-OPTION-NUMBER-RECORD**

```
ADD
RECORD NAME IS WS-OPTION-NUMBER-RECORD VERSION IS 1.
   05 WS-OPTION-NUMBER PIC 99.
      88 WS-OPT-LOG-SVCS VALUE 1.
      88 WS-OPT-LOG-PGM VALUE 2.
      88 WS-OPT-REQ-SIGNON VALUE 3.
```
Elements

- **WS-OPTION-NUMBER**
  The option number associated with the desired option. This field is used with the SETOPTION and GETOPTION functions. Level 88 sub-elements include the following:

  - **WS-OPT-LOG-SVCS**
    Directs the retrieval or modification of the Log Services option.

  - **WS-OPT-LOG-PGM**
    Directs the retrieval or modification of the Log Program option.

  - **WS-OPT-REQ-SIGNON**
    Directs the retrieval or modification of the Require Signon option.

  - **WS-OPT-CHECK-AUTH**
    Directs the retrieval or modification of the Check Authorization option.

  - **WS-OPT-CONN-TIMEOUT**
    Directs the retrieval or modification of the Connection Timeout option. Connection Timeout specifies the time, in seconds, that a TCP/IP Connect operation waits before the operation is aborted.

  - **WS-OPT-RW-TIMEOUT**
    Directs the retrieval or modification of the Read/Write Timeout option. Read/Write Timeout specifies the time, in seconds, that a TCP/IP Send or Receive operation waits before the operation is aborted.

  - **WS-OPT-XML-CPAGE**
    Directs the retrieval or modification of the XML CodePage option. XMLCodePage specifies the code page used on internal XML operations within the Web Services modules. The default is 1140.


**WS-OPTION-VALUE-RECORD**

```
ADD RECORD NAME IS WS-OPTION-VALUE-RECORD VERSION IS 1.
  05 WS-OPTION-VALUE.
    PICTURE IS X(8).
    USAGE IS DISPLAY.
```
Elements

- **WS-OPTION-VALUE**
  The option value for the SETOPTION and GETOPTION functions. Depending on the function used, this field contains either the current value of the option or a new value.

### WS-REQUEST-MSG-DATA (Module)

The WS-REQUEST-MSG-DATA record definition is in the form of an IDMS Module. This is a variable length record, whose size depends on the WS-REQ-MSG-BUFF-LEN element value, which exists in the WS-REQUEST-MSG-DESCRIPTOR record. This type of construct (compatible with COBOL) is not allowed for a record definition within CA IDMS.

```
ADD
MODULE NAME IS WS-REQUEST-MSG-DATA VERSION IS 1
    LANGUAGE IS COBOL
    MODULE SOURCE FOLLOWS
    01 WS-REQUEST-MSG-DATA.
    05 WS-REQUEST-MESSAGE.
    10 WS-REQUEST-MSG-BYTE PIC X(1)
    OCCURS 0 TO 1 TIMES
    DEPENDING ON WS-REQ-MSG-BUFF-LEN.
MSEND
```

Elements

- **WS-REQUEST-MESSAGE**
  Group-level element providing a reference to the entire Request message.

- **WS-REQUEST-MSG-BYTE**
  A variable length buffer, whose size depends on the value specified in the WS-REQ-MSG-BUFF-LEN element. The WS-REQ-MSG-BUFF-LEN element resides in the WS-REQUEST-MSG-DESCRIPTOR record.

### WS-REQUEST-MSG-PTR-RECORD

```
ADD
RECORD NAME IS WS-REQUEST-MSG-PTR-RECORD VERSION IS 1.
    05 WS-REQUEST-MSG-PTR POINTER.
```

Elements

- **WS-REQUEST-MSG-PTR**
  Pointer record to be used when calling the REQUEST function. On return from that call, set the address of WS-REQUEST-MESSAGE to the address contained in this pointer.
WS-RESPONSE-MSG-DATA (Module)

The WS-RESPONSE-MSG-DATA record definition is in the form of an IDMS Module. This is a variable length record, whose size depends on the WS-RSP-MSG-BUFF-LEN element value, which exists in the WS-RESPONSE-MSG-DESCRIPTOR record. This type of construct (compatible with COBOL) is not allowed for a record definition within CA IDMS.

ADD
MODULE NAME IS WS-RESPONSE-MSG-DATA VERSION IS 1
LANGUAGE IS COBOL
MODULE SOURCE FOLLOWS
01 WS-RESPONSE-MSG-DATA.
   05 WS-RESPONSE-MESSAGE.
   10 WS-RESPONSE-MSG-BYTE PIC X(1)
      OCCURS 0 TO 1 TIMES
      DEPENDING ON WS-RSP-MSG-BUFF-LEN.
MSEND

Elements

- **WS-RESPONSE-MESSAGE**
  Group-level element providing a reference to the entire Response message.

- **WS-RESPONSE-MSG-BYTE**
  The Response message is a variable length buffer, whose size depends on the value specified in the WS-RSP-MSG-BUFF-LEN element. The WS-RSP-MSG-BUFF-LEN element resides in the WS-RESPONSE-MSG-DESCRIPTOR record.

WS-RESPONSE-MSG-PTR-RECORD

ADD
RECORD NAME IS WS-RESPONSE-MSG-PTR-RECORD VERSION IS 1.
   05 WS-RESPONSE-MSG-PTR POINTER.

Elements

- **WS-RESPONSE-MSG-PTR**
  Pointer record to be used when calling the SEND function. Prior to making the call, set this pointer to the address of WS-RESPONSE-MESSAGE.

WS-REQUEST-MSG-DESCRIPTOR

ADD
RECORD NAME IS WS-REQUEST-MSG-DESCRIPTOR VERSION IS 1.
   05 WS-REQ-MSG-BUFF-LEN
      PICTURE IS S9(8)
      USAGE IS COMP
   .
   05 FILLER
      PICTURE IS X(64)
      USAGE IS DISPLAY
      VALUE IS ( 'FUTURE EXPANSION' )
   .
Elements

- **WS-REQ-MSG-BUFF-LEN**
The length of the Request message buffer.

**WS-RESPONSE-MSG-DESCRIPTOR**

```
ADD RECORD NAME IS WS-RESPONSE-MSG-DESCRIPTOR VERSION IS 1.
  05 WS-RSP-MSG-BUFF-LEN
    PICTURE IS S9(8)
    USAGE IS COMP
  .
  05 FILLER
    PICTURE IS X(64)
    USAGE IS DISPLAY
    VALUE IS ('FUTURE EXPANSION')
  .
```

Elements

- **WS-RSP-MSG-BUFF-LEN**
The length of the Response message buffer.

**WS-RETURN-CODE-RECORD**

```
ADD RECORD NAME IS WS-RETURN-CODE-RECORD VERSION IS 1.
  05 WS-RETURN-CODE PIC S9(8) COMP.
```

Elements

- **WS-RETURN-CODE**
The return code for the invoked function. Detailed information can be found in Web Services API Return Codes (see page 125), below.

**WS-ERROR-INFO**

```
ADD RECORD NAME IS WS-ERROR-INFO VERSION IS 1.
  05 WS-ERROR-TYPE PIC X(01) VALUE SPACES.
    88 WS-ERROR-INTERNAL VALUE 'I'.
    88 WS-ERROR-API VALUE 'A'.
    88 WS-ERROR-XML VALUE 'X'.
    88 WS-ERROR-HTTP VALUE 'H'.
    88 WS-ERROR-TCP/IP VALUE 'T'.
    88 WS-ERROR-SOAP VALUE 'S'.
    88 WS-ERROR-OTHER VALUE 'O'.
  05 WS-ERROR-CODE PIC S9(8) COMP VALUE 0.
  05 WS-ERROR-TEXT PIC X(256) VALUE SPACES.
```

Elements

- **WS-ERROR-TYPE**
The category of the error message, which gives a general idea of where the error occurred. Level 88 sub-elements include the following:
- **WS-ERROR-INTERNAL**
  An internal system error occurred. These errors are typically generated from failures in IDMS/DC type operations.

- **WS-ERROR-API**
  An API-level error has occurred. These errors are generated due to a failure to adhere to the Web Services API protocol.

- **WS-ERROR-XML**
  An XML-type error has occurred. These errors are generated during the Generation and Parsing of XML strings.

- **WS-ERROR-HTTP**
  An unexpected HTTP status code is received. Refer to the W3C web page on HTTP Status Codes for more information on this class of errors.

- **WS-ERROR-TCPIP**
  An unexpected TCP/IP return code is received. Refer to TCP/IP API Support (see page 46) for more information on these errors.

- **WS-ERROR-SOAP**
  An unexpected SOAP fault is received. Detailed information on these errors are typically communicated in the WS-ERROR-CODE and WS-ERROR-TEXT elements.

- **WS-ERROR-OTHER**
  An unclassified error occurred. Refer to the WS-ERROR-CODE and WS-ERROR-TEXT elements for more information.

- **WS-ERROR-CODE**
  The message error code, which can be used to look up additional information about where the error occurred, or how it can be resolved.

- **WS-ERROR-TEXT**
  The text of the error message, which gives additional content to the error.

### Web Services API Return Codes

<table>
<thead>
<tr>
<th>Value</th>
<th>Severity Level</th>
<th>Description/Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Warning</td>
<td>Requested function is processed. Warning message generated.</td>
</tr>
<tr>
<td>8</td>
<td>Error</td>
<td>Requested function failed. Failure message generated. Service request (if created) is retained.</td>
</tr>
<tr>
<td>12</td>
<td>Critical</td>
<td>Requested function failed. Services Request is automatically Terminated/Released.</td>
</tr>
<tr>
<td>16</td>
<td>Systemic</td>
<td>Requested function failed. Services Request is automatically Terminated/Released. Problem impacts all Web Services.</td>
</tr>
</tbody>
</table>
Supported Programming Languages

This section describes the call syntax for the Web Services API in each of the supported programming languages. For a complete description of all API records, refer to Web Services API Records (https://docops.ca.com/display/IDMSCU/Web+Services+Interface+Records).

- Call Interface for ADS/Online (see page 126)
- Call Interface for COBOL (see page 127)
- Call Interface for PL/I (see page 128)
- Common Parameters (see page 129)

Note: See Common Parameters (see page 129) for a description of the parameters common to the following supported programming languages.

Call Interface for ADS/Online

This section describes the ADS/Online call interface.

Syntax

ADS/Online Dialogs use the LINK statement to invoke the CA IDMS Web Services, as follows:

```
LINK TO PROGRAM 'IDMSWSI' USING
  (function-code,
   return-code,
   error-info, request-handle,
   function-dependent-parameter1,
   ...).
```

Considerations

Following is additional information about the ADS/Online call interface:

- To omit an optional parameter in the parameter list, code a space followed by a comma.
- The ADS "LINK TO PROGRAM" statement requires all parameters to be records. You should create a record with a single record element for parameters that represent a single element.
- The following pre-defined records are provided during installation to assist in writing Web Services applications:
  - WS-ERROR-INFO – Contains the error class, detailed error code and error text fields used to communicate warning/error conditions
WS-CALL-INTERFACE – Contains various individual fields used in the CA IDMS Web Services API

The WS-CALL-INTERFACE record contains the following fields:

- function-code
- return-code
- request-handle

Example

Each supported function is represented by a field, whose value is the function number. The following example illustrates how to issue the Web Services Initialize request using the fields within the WS-CALL-INTERFACE and WS-ERROR-INFO records:

```
LINK TO PROGRAM 'IDMSWSI' USING
(WS-INITIALIZE,
 WS-RETURN-CODE,
 WS-ERROR-INFO,
 WS-REQUEST-HANDLE).
```

Call Interface for COBOL

This section describes the COBOL call interface.

Syntax

Programs written in COBOL use the CALL statement to invoke the CA IDMS Web Services:

```
TRANSFER TO 'IDMSWSI' LINK USING
 function-code,
 return-code,
 error-info,
 request-handle,
 function-dependent-parameter1,
 function-dependent-parameter2.
```

Considerations

Following is additional information about the COBOL call interface:

- If an optional parameter is not specified in the parameter list, it should be replaced by a parameter that depends on the COBOL compiler:
  - COBOL for z/OS: Specify reserved keyword OMITTED.
  - 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 Web Services applications:
WS-ERROR-INFO: Contains the error class, detailed error code and error text fields used to communicate warning/error conditions

WS-CALL-INTERFACE: Contains various individual fields used in the CA IDMS Web Services API

The WS-CALL-INTERFACE record contains the following fields:

- function-code
- return-code
- request-handle

Example

Each supported function is represented by a field, whose value is the function number. The following example illustrates how to issue the Web Services Initialize request using the fields within the WS-CALL-INTERFACE and WS-ERROR-INFO records:

TRANSFER TO 'IDMSWSI' LINK USING
  WS-INITIALIZE,
  WS-RETURN-CODE,
  WS-ERROR-INFO,
  WS-REQUEST-HANDLE.

Call Interface for PL/I

This section describes the PL/I call interface.

Syntax

Programs written in PL/I use the CALL statement to invoke the CA IDMS Web Services:

CALL IDMSWSI USING
  (function-code,
   return-code,
   error-info,
   request-handle,
   function-dependent-parameter1,
   . . .);

Considerations

Following is additional information about the PL/I call interface:

- If an optional parameter is not specified in the parameter list, replace it by an asterisk (*).

- The following pre-defined records are provided during installation to assist in writing Web Services applications:
  - WS-ERROR-INFO: Contains the error class, detailed error code and error text fields used to communicate warning/error conditions
  - WS-CALL-INTERFACE: Contains various individual fields used in the CA IDMS Web Services API
- WS-INTERFACE-VERSION-NUMBER: Contains the Major and Minor interface version number fields

- The WS-CALL-INTERFACE record contains the following fields:
  - function-code
  - return-code
  - request-handle

**Example**

Each supported function is represented by a field, whose value is the function number. The following example illustrates how to issue the Web Services *Initialize* request using the fields within the WS-CALL-INTERFACE and WS-ERROR-INFO records:

```
CALL 'IDMSWSI' USING (WS-INITIALIZE,
  WS-RETURN-CODE,
  WS-ERROR-INFO,
  WS-REQUEST-HANDLE);
```

**Common Parameters**

All calls to the #IDMSWSI macro must pass the following four parameters. Depending on the function, additional parameters may be available:

- **function**
  Specifies a full-word field that the program sets to the desired Web Services function.

- **return-code**
  Specifies a full-word field that communicates the outcome of the operation.

- **error-info**
  Specifies a full-word pointer to the WS-ERROR-INFO record structure. The WS-ERROR-INFO record contains detailed error information on the various warnings/errors returned on the Web Services API function calls.

- **request-handle**
  Specifies a full-word field that identifies the Web Services "Request" block.

**Note:** This field is required for all CA IDMS Web Services API function calls.
Invoking System Tasks 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 (see page 130)
- Parameters (see page 130)
  - Mandatory Parameters (see page 130)
  - Optional Parameter (see page 131)
- Example (see page 131)

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.

For more information on the CLIST command, see CA IDMS System Tasks and Operator Commands (https://docops.ca.com/display/IDMSCU19/CA+IDMS+System+Tasks+and+Operator+Commands).

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
```

For more information, see the following topics:
Invoking Command List Modules from Programs

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.

This article describes the following information:

- Linking to RHDCMT00 and RHDCUF00 (see page 132)
- Parameters (see page 133)
- Usage (see page 135)
- Examples (see page 135)

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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of DCMT/DCUF command</td>
<td>2 bytes</td>
<td>Binary</td>
</tr>
<tr>
<td>DCMT/DCUF command, left-justified</td>
<td>Any number of bytes</td>
<td>Character</td>
</tr>
</tbody>
</table>

OUTREC Format

The format is identical for RHDCUF00 and RHDCMT00.

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>return-area-length:*</td>
<td>4 bytes</td>
<td>Binary</td>
</tr>
<tr>
<td>The 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.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>return-code:**</td>
<td>2 bytes</td>
<td>Binary</td>
</tr>
<tr>
<td>0 -- Request accepted and processed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 -- Invalid syntax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 -- Invalid request (for example, SHUTDOWN, ABORT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 -- Security violation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 -- Processing error</td>
<td></td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>output-code:</strong></td>
<td>2</td>
<td>Binary</td>
</tr>
</tbody>
</table>
| 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." |
|                      | 2      | Bytes  |

**output-length:**
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.

<table>
<thead>
<tr>
<th>returned-output-length:</th>
<th>4</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total length of text lines inserted into the returned-text-area. The length includes a one-byte line-length indicator for each text line.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>returned-text-area:</th>
<th>Variable Character</th>
<th></th>
</tr>
</thead>
</table>
| 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 #SETIME 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:

```assembly
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.
ST R1,OUTRECOD REQUEST OUTPUT TO STORAGE.
ST R1,OUTRECTL 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
STH R1,OUTRECRC
BAL R8,CALLDISP
* 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
CVD R15,WORKDBL
UNPK WORKRC,WORKDBL
OI WORKRC+L'WORKRC-1,X'F0'
L R5,OUTRECAL
CVD R5,WORKDBL
UNPK WORKLEN,WORKDBL
OI WORKLEN+L'WORKLEN-1,X'F0'
#LINEOUT OUTLEN=80,OUTAREA=WORKRCLN
LTR R5,R5
RETURN NO.
LA R4,OUTRECTX
PUTLINE DS 0H
SR R3,R3
IC R3,0(,R4)
LA R4,1(,R4)
#LINEOUT OUTLEN=(R3),OUTAREA=(R4)
AR R4,R3
P R5,R5
BCTR R5,0
LTR R5,R5
BP PUTLINE
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
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
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
OUTRECRCC 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
### DCUF Example

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

```plaintext
#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
OUTRECCRC DC H'0' - return code
OUTRECCOD DC H'0' - output type
OUTRECCTL DC F'0' - total length
OUTRECCAL DC F'0' - actual length
OUTRECTX DS CL132 - actual output text
```

### Invoking SDEL Command from Programs

#### Contents
- Linking to RHDCSDEL (see page 137)
- Parameters (see page 137)
- Example (see page 138)
- More Information (see page 139)

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:

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

#### Parameters

- **DICTNAME**
  
  Specifies the dictionary name of the updatable DDLDM 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:
More Information

- For more information about the LINK statement, see the CA IDMS DML Reference section 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.

This article describes the following information:

- RHDCSNON (see page 139)
- Parameters (see page 139)
- Example (see page 140)

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.
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
```
Two-Phase Commit Support with RRS

RRS is an IBM 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.

For more information, see the following topics:
- RRS Support for Batch Applications (see page 141)
- RRS Support for Online Applications (see page 145)

RRS Support for Batch Applications

Contents

- Example (see page 142)
- Enabling RRS for Batch Applications (see page 143)
- Batch RRS Transaction Boundaries and Application Design Considerations (see page 143)
- Example of a COBOL Batch Program (see page 144)

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.
```
RRS Support for Online Applications

Contents

- Example (see page 147)
- Programming Interface (see page 147)
- Application Design Considerations (see page 147)

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.

RRS Support for Online Applications
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
TCP/IP Error Codes

For more information, see the following topics:

- Return, Errno, and Reason Codes (see page 148)
- HOSTENT Structure (see page 155)
- SERVENT Structure (see page 156)
- Socket Structure Descriptions (see page 156)

Return, Errno, and Reason Codes

Contents

- ERRNO Numbers Set by the Socket Program Interface (see page 149)

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**
  - § 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>The ERRNO is generated by the operating system. See the appropriate operating system documentation.</td>
</tr>
<tr>
<td>RNOINPL</td>
<td>12000</td>
<td>Invalid #SOCKET parameter list</td>
</tr>
<tr>
<td>RNOINAEC</td>
<td>12001</td>
<td>Invalid ASYNCECB parameter</td>
</tr>
<tr>
<td>RNOINAIL</td>
<td>12002</td>
<td>Invalid AINFOIN parameter</td>
</tr>
<tr>
<td>RNOINAI</td>
<td>12003</td>
<td>Invalid AINFOOUT parameter</td>
</tr>
<tr>
<td>RNOINBF</td>
<td>12004</td>
<td>Invalid BUFFER parameter</td>
</tr>
<tr>
<td>RNOINBL</td>
<td>12005</td>
<td>Invalid BUFFERL parameter</td>
</tr>
<tr>
<td>RNOINBK</td>
<td>12006</td>
<td>Invalid BACKLOG parameter</td>
</tr>
<tr>
<td>RNOINCA</td>
<td>12007</td>
<td>Invalid CANONAML parameter</td>
</tr>
<tr>
<td>RNOINC</td>
<td>12008</td>
<td>Invalid COMMAND parameter</td>
</tr>
<tr>
<td>RNOIND</td>
<td>12009</td>
<td>Invalid DOMAIN parameter</td>
</tr>
<tr>
<td>RNOINEL</td>
<td>12010</td>
<td>Invalid EXCELST parameter</td>
</tr>
<tr>
<td>RNOINEL</td>
<td>12011</td>
<td>Invalid EXCELSTL parameter</td>
</tr>
<tr>
<td>RNOINFL</td>
<td>12012</td>
<td>Invalid FLAGS parameter</td>
</tr>
<tr>
<td>RNOINF</td>
<td>12013</td>
<td>Invalid FORMAT parameter</td>
</tr>
<tr>
<td>RNOINFL</td>
<td>12014</td>
<td>Invalid FROMLITE parameter</td>
</tr>
<tr>
<td>RNOINH</td>
<td>12015</td>
<td>Invalid HANDLE parameter</td>
</tr>
<tr>
<td>RNOINH</td>
<td>12016</td>
<td>Invalid HOSTNAME parameter</td>
</tr>
<tr>
<td>RNOINH</td>
<td>12017</td>
<td>Invalid HOSTNAML parameter</td>
</tr>
<tr>
<td>Name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>RNOINHNT</td>
<td>12018</td>
<td>Invalid HOSTENTP parameter</td>
</tr>
<tr>
<td>RNOINH OW</td>
<td>12019</td>
<td>Invalid HOW parameter</td>
</tr>
<tr>
<td>RNOINIL</td>
<td>12020</td>
<td>Invalid IPADDRL parameter</td>
</tr>
<tr>
<td>RNOINIP</td>
<td>12021</td>
<td>Invalid IPADDR parameter</td>
</tr>
<tr>
<td>RNOINIP S</td>
<td>12022</td>
<td>Invalid IPADDRS parameter</td>
</tr>
<tr>
<td>RNOINIS L</td>
<td>12023</td>
<td>Invalid IPADDRSL parameter</td>
</tr>
<tr>
<td>RNOINLE V</td>
<td>12024</td>
<td>Invalid LEVEL parameter</td>
</tr>
<tr>
<td>RNOINM XP</td>
<td>12025</td>
<td>Invalid MAXPTERM parameter</td>
</tr>
<tr>
<td>RNOINM XT</td>
<td>12026</td>
<td>Invalid MAXTASK parameter</td>
</tr>
<tr>
<td>RNOINN A</td>
<td>12027</td>
<td>Invalid NAME parameter</td>
</tr>
<tr>
<td>RNOINN AL</td>
<td>12028</td>
<td>Invalid NAMEL parameter</td>
</tr>
<tr>
<td>RNOINNS D</td>
<td>12029</td>
<td>Invalid NEWSOCK parameter</td>
</tr>
<tr>
<td>RNOINNS D</td>
<td>12030</td>
<td>Invalid NFDS parameter</td>
</tr>
<tr>
<td>RNOINO NA</td>
<td>12031</td>
<td>Invalid OPTNAME parameter</td>
</tr>
<tr>
<td>RNOINO VA</td>
<td>12032</td>
<td>Invalid OPTVAL parameter</td>
</tr>
<tr>
<td>RNOINO VL</td>
<td>12033</td>
<td>Invalid OPTLEN parameter</td>
</tr>
<tr>
<td>RNOINP NA</td>
<td>12034</td>
<td>Invalid PROTNAME parameter</td>
</tr>
<tr>
<td>RNOINP NL</td>
<td>12035</td>
<td>Invalid PROTNAML parameter</td>
</tr>
<tr>
<td>RNOINP NT</td>
<td>12036</td>
<td>Invalid PROTENTP parameter</td>
</tr>
<tr>
<td>RNOINP NU</td>
<td>12037</td>
<td>Invalid PROTNUM parameter</td>
</tr>
<tr>
<td>RNOINP OR</td>
<td>12038</td>
<td>Invalid PORT parameter</td>
</tr>
<tr>
<td>RNOINR HL</td>
<td>12039</td>
<td>Invalid RETHNAML parameter</td>
</tr>
<tr>
<td></td>
<td>12040</td>
<td>Invalid RETIPASL parameter</td>
</tr>
<tr>
<td>Name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>RNOINRI</td>
<td>L</td>
<td>Invalid READLST parameter</td>
</tr>
<tr>
<td>RNOINRL</td>
<td>12041</td>
<td>Invalid READLST parameter</td>
</tr>
<tr>
<td>RNOINRL</td>
<td>12042</td>
<td>Invalid READLST parameter</td>
</tr>
<tr>
<td>RNOINRL</td>
<td>12043</td>
<td>Invalid RETLEN parameter</td>
</tr>
<tr>
<td>RNOINRD</td>
<td>12044</td>
<td>Invalid RETNFDS parameter</td>
</tr>
<tr>
<td>RNOINRN</td>
<td>12045</td>
<td>Invalid RETNSTKS parameter</td>
</tr>
<tr>
<td>RNOINSA</td>
<td>12046</td>
<td>Invalid SOCKADDR parameter</td>
</tr>
<tr>
<td>RNOINSA</td>
<td>12047</td>
<td>Invalid SOCKADDL parameter</td>
</tr>
<tr>
<td>RNOINSN</td>
<td>12048</td>
<td>Invalid SERVNAME parameter</td>
</tr>
<tr>
<td>RNOINSN</td>
<td>12049</td>
<td>Invalid SERVNAML parameter</td>
</tr>
<tr>
<td>RNOINSN</td>
<td>12050</td>
<td>Invalid SERVENTP parameter</td>
</tr>
<tr>
<td>RNOINSO</td>
<td>12051</td>
<td>Invalid SOCK parameter</td>
</tr>
<tr>
<td>RNOINTL</td>
<td>12052</td>
<td>Invalid TOLTE parameter</td>
</tr>
<tr>
<td>RNOINTP</td>
<td>12053</td>
<td>Invalid TYPE parameter</td>
</tr>
<tr>
<td>RNOINW</td>
<td>12054</td>
<td>Invalid WRITLST parameter</td>
</tr>
<tr>
<td>RNOINWL</td>
<td>12055</td>
<td>Invalid WRITLST parameter</td>
</tr>
<tr>
<td>RNOINWO</td>
<td>12056</td>
<td>Invalid OPTION parameter</td>
</tr>
<tr>
<td>RNOINTI</td>
<td>12057</td>
<td>Invalid TIMEOUT parameter</td>
</tr>
<tr>
<td>RNOINAR</td>
<td>12058</td>
<td>Invalid ARGUMENT parameter</td>
</tr>
<tr>
<td>RNOINRV</td>
<td>12059</td>
<td>Invalid RETVAL parameter</td>
</tr>
<tr>
<td>RNOINEC</td>
<td>12060</td>
<td>Invalid ECB parameter</td>
</tr>
<tr>
<td>RNOINEC</td>
<td>12061</td>
<td>Invalid ECBLIST parameter</td>
</tr>
<tr>
<td></td>
<td>12062</td>
<td>Invalid RETSNAML parameter</td>
</tr>
<tr>
<td>Name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>RNOINRS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNOINBL</td>
<td>12063</td>
<td>Invalid BITLIST parameter</td>
</tr>
<tr>
<td>RNOINBL</td>
<td>12064</td>
<td>Invalid BITLISTL parameter</td>
</tr>
<tr>
<td>RNOINB</td>
<td>12065</td>
<td>Invalid BITORDER parameter</td>
</tr>
<tr>
<td>RNOINN</td>
<td>12066</td>
<td>Invalid NODENAME parameter</td>
</tr>
<tr>
<td>RNOINST</td>
<td>12067</td>
<td>Invalid STATUS parameter</td>
</tr>
<tr>
<td>RNOINAR</td>
<td>12068</td>
<td>Invalid ARGUMENL parameter</td>
</tr>
<tr>
<td>RNO2BU</td>
<td>12100</td>
<td>Specify BUFFER and BUFFERL, or none of them</td>
</tr>
<tr>
<td>RNO2HN</td>
<td>12101</td>
<td>Specify HOSTNAME and HOSTNAML, or none of them</td>
</tr>
<tr>
<td>RNO2NA</td>
<td>12102</td>
<td>Specify NAME and NAMEL, or none of them</td>
</tr>
<tr>
<td>RNO2PN</td>
<td>12103</td>
<td>Specify PROTNAME and PROTNAML, or none of them</td>
</tr>
<tr>
<td>RNO2SN</td>
<td>12104</td>
<td>Specify SERVNAME and SERVNAML, or none of them</td>
</tr>
<tr>
<td>RNO3HN</td>
<td>12105</td>
<td>Specify HOSTNAME/HOSTNAML/RETHNAML, or none</td>
</tr>
<tr>
<td>RNO3SN</td>
<td>12106</td>
<td>Specify SERVNAME/SERVNAML/RETSNAML, or none</td>
</tr>
<tr>
<td>RNORQH</td>
<td>12107</td>
<td>HOSTNAME or SERVNAME (or both) is required</td>
</tr>
<tr>
<td>RNORQE</td>
<td>12108</td>
<td>ECB or ECBLIST is required</td>
</tr>
<tr>
<td>RNOXCE</td>
<td>12109</td>
<td>ECB and ECBLIST are mutually exclusive</td>
</tr>
<tr>
<td>RNOIECB</td>
<td>12110</td>
<td>Invalid ECB in ECBLIST</td>
</tr>
<tr>
<td>RNOINAR</td>
<td>12111</td>
<td>Invalid asynchronous command request</td>
</tr>
<tr>
<td>RNOINAI</td>
<td>12112</td>
<td>Invalid ADDRINFO structure</td>
</tr>
<tr>
<td>RNOSYSP</td>
<td>12113</td>
<td>ASYNCECB and HANDLE are system parms</td>
</tr>
<tr>
<td>RNOINH</td>
<td>12114</td>
<td>Invalid area pointed to by HANDLE</td>
</tr>
<tr>
<td>Name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>RNOIIPA</td>
<td>12115</td>
<td>Invalid format for IP-address</td>
</tr>
<tr>
<td>RNOIIPA</td>
<td>12116</td>
<td>Invalid format for IP-address (V6)</td>
</tr>
<tr>
<td>RNOFNS</td>
<td>12200</td>
<td>Function not supported by interface</td>
</tr>
<tr>
<td>RNOFRSV</td>
<td>12201</td>
<td>Function reserved for the system</td>
</tr>
<tr>
<td>RNOCAAI</td>
<td>12202</td>
<td>Cannot allocate an AIO parameter list</td>
</tr>
<tr>
<td>RNOCAN</td>
<td>12203</td>
<td>Cannot assign new socket to user</td>
</tr>
<tr>
<td>RNOCRSF</td>
<td>12204</td>
<td>Cannot remove socket from user table</td>
</tr>
<tr>
<td>RNOCSH</td>
<td>12205</td>
<td>Cannot save HOSTENT structure info</td>
</tr>
<tr>
<td>RNOCSAI</td>
<td>12206</td>
<td>Cannot save ADDRINFO structure info</td>
</tr>
<tr>
<td>RNONAI</td>
<td>12207</td>
<td>Cannot find ADDRINFO to free</td>
</tr>
<tr>
<td>RNONOL</td>
<td>12208</td>
<td>No LTE available from current TCE</td>
</tr>
<tr>
<td>RNOSLIN</td>
<td>12209</td>
<td>SOCKET line not defined</td>
</tr>
<tr>
<td>RNOSLR</td>
<td>12210</td>
<td>SOCKET line not opened</td>
</tr>
<tr>
<td>RNOSLRC</td>
<td>12211</td>
<td>TCP/IP has been recycled</td>
</tr>
<tr>
<td>RNOPINL</td>
<td>12212</td>
<td>Plug-in module not loaded</td>
</tr>
<tr>
<td>RNOINES</td>
<td>12213</td>
<td>Driver's TCE does not point to the PLE</td>
</tr>
<tr>
<td>RNOINESP</td>
<td>12214</td>
<td>Invalid environment when entering the plug-in</td>
</tr>
<tr>
<td>RNOSEN</td>
<td>12215</td>
<td>Socket environment not active</td>
</tr>
<tr>
<td>RNOUST</td>
<td>12216</td>
<td>User's socket table cannot be allocated</td>
</tr>
<tr>
<td>RNOUST</td>
<td>12217</td>
<td>User's socket table does not exist</td>
</tr>
<tr>
<td>RNOSSTC</td>
<td>12218</td>
<td>System's socket table cannot be allocated</td>
</tr>
<tr>
<td>RNOSST</td>
<td>12219</td>
<td>System's socket table does not exist</td>
</tr>
<tr>
<td></td>
<td>12220</td>
<td>Requested stack not found</td>
</tr>
<tr>
<td>Name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>RNOSTK</td>
<td>NF</td>
<td>Requested stack not active</td>
</tr>
<tr>
<td>RNOSTK</td>
<td>12221</td>
<td>Requested stack not active</td>
</tr>
<tr>
<td>RNOSTK</td>
<td>NA</td>
<td>Requested stack not active</td>
</tr>
<tr>
<td>RNOSDT</td>
<td>CE</td>
<td>Socket Descriptor table cannot be extended</td>
</tr>
<tr>
<td>RNOSDT</td>
<td>12222</td>
<td>Socket Descriptor table cannot be extended</td>
</tr>
<tr>
<td>RNOCAS</td>
<td>WA</td>
<td>Cannot allocate SELECT work area</td>
</tr>
<tr>
<td>RNOCAS</td>
<td>12223</td>
<td>Cannot allocate SELECT work area</td>
</tr>
<tr>
<td>RNOINS</td>
<td>WA</td>
<td>Inconsistent fields in SELECT work area</td>
</tr>
<tr>
<td>RNOINS</td>
<td>12224</td>
<td>Inconsistent fields in SELECT work area</td>
</tr>
<tr>
<td>RNOSBLE</td>
<td>M</td>
<td>All SELECT bit lists are empty</td>
</tr>
<tr>
<td>RNOSBLE</td>
<td>12225</td>
<td>All SELECT bit lists are empty</td>
</tr>
<tr>
<td>RNOSNC</td>
<td>SS</td>
<td>All sockets not created under same stack</td>
</tr>
<tr>
<td>RNOSNC</td>
<td>12226</td>
<td>All sockets not created under same stack</td>
</tr>
<tr>
<td>RNOCAS</td>
<td>BL</td>
<td>Cannot allocate socket's bit list</td>
</tr>
<tr>
<td>RNOCAS</td>
<td>12227</td>
<td>Cannot allocate socket's bit list</td>
</tr>
<tr>
<td>RNOMAX</td>
<td>12228</td>
<td>Maximum number of sockets reached</td>
</tr>
<tr>
<td>RNOMAX</td>
<td>SO</td>
<td>Maximum number of sockets reached</td>
</tr>
<tr>
<td>RNOMAX</td>
<td>12229</td>
<td>Maximum number of sockets per task reached</td>
</tr>
<tr>
<td>RNOMAX</td>
<td>ST</td>
<td>Maximum number of sockets per task reached</td>
</tr>
<tr>
<td>RNOCAD</td>
<td>NS</td>
<td>Cannot allocate DNS work area</td>
</tr>
<tr>
<td>RNOCAD</td>
<td>12230</td>
<td>Cannot allocate DNS work area</td>
</tr>
<tr>
<td>RNOIND</td>
<td>NS</td>
<td>Invalid response from DNS server</td>
</tr>
<tr>
<td>RNOIND</td>
<td>12231</td>
<td>Invalid response from DNS server</td>
</tr>
<tr>
<td>RNOPITN</td>
<td>L</td>
<td>(z/VSE only) Plugin table module not loaded</td>
</tr>
<tr>
<td>RNODDS</td>
<td>NA</td>
<td>No active DDSTCPIP PTE found for IDMS nodename</td>
</tr>
<tr>
<td>RNODDS</td>
<td>12233</td>
<td>No active DDSTCPIP PTE found for IDMS nodename</td>
</tr>
<tr>
<td>RNODDS</td>
<td>NC</td>
<td>Cannot build a DDS connection to IDMS nodename</td>
</tr>
<tr>
<td>RNODDS</td>
<td>12234</td>
<td>Cannot build a DDS connection to IDMS nodename</td>
</tr>
<tr>
<td>RNODDS</td>
<td>NF</td>
<td>No free port found in PORT-RANGE</td>
</tr>
<tr>
<td>RNODDS</td>
<td>12235</td>
<td>No free port found in PORT-RANGE</td>
</tr>
<tr>
<td>RNODDS</td>
<td>MC</td>
<td>Maximum number of connections reached</td>
</tr>
<tr>
<td>RNODDS</td>
<td>12236</td>
<td>Maximum number of connections reached</td>
</tr>
<tr>
<td>RNODDS</td>
<td>RE</td>
<td>Error during release of a DDS connection</td>
</tr>
<tr>
<td>RNODDS</td>
<td>12237</td>
<td>Error during release of a DDS connection</td>
</tr>
<tr>
<td>RNONAS</td>
<td>TK</td>
<td>No active stack found in the system</td>
</tr>
<tr>
<td>RNONAS</td>
<td>12238</td>
<td>No active stack found in the system</td>
</tr>
<tr>
<td>RNOCEXS</td>
<td>12239</td>
<td>Connection on excluded stack ignored (internal)</td>
</tr>
<tr>
<td>RNOCEXS</td>
<td>I</td>
<td>Connection on excluded stack ignored (internal)</td>
</tr>
<tr>
<td>RNOCSS</td>
<td>NT</td>
<td>Cannot save SERVENT structure info</td>
</tr>
<tr>
<td>RNOCSS</td>
<td>12240</td>
<td>Cannot save SERVENT structure info</td>
</tr>
<tr>
<td></td>
<td>12241</td>
<td>Name/alias + protocol service not found</td>
</tr>
</tbody>
</table>
## Name | Value | Description |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RNOSRV</td>
<td>NF</td>
<td></td>
</tr>
<tr>
<td>RNOPOR</td>
<td>12242</td>
<td>Port number + protocol service not found NF</td>
</tr>
<tr>
<td>RNOSGN</td>
<td>12243</td>
<td>SYSGEN internal error - wrong records counter ER</td>
</tr>
<tr>
<td>RNOSTK</td>
<td>12244</td>
<td>Requested stack already active AA</td>
</tr>
<tr>
<td>RNOSTK</td>
<td>12245</td>
<td>Requested stack already inactive AI</td>
</tr>
<tr>
<td>RNOCDS</td>
<td>12246</td>
<td>Cannot exclude the default stack TK</td>
</tr>
<tr>
<td>RNOSTKE</td>
<td>12247</td>
<td>Owning stack has been dynamically excluded X</td>
</tr>
<tr>
<td>RNOSEN</td>
<td>12248</td>
<td>Socket environment is quiescing Q</td>
</tr>
<tr>
<td>RNOSFN</td>
<td>12249</td>
<td>Service file not defined in the CA IDMS system D</td>
</tr>
<tr>
<td>RNOSTKC</td>
<td>12250</td>
<td>System's Stack Table cannot be allocated A</td>
</tr>
<tr>
<td>RNOSTK</td>
<td>12251</td>
<td>System's Stack Table not initialized yet NI</td>
</tr>
<tr>
<td>RNODNS</td>
<td>12252</td>
<td>Internal DNS Resolver not available NA</td>
</tr>
<tr>
<td>RNOHIUCV</td>
<td>12300</td>
<td>HNDIUCV error CV</td>
</tr>
<tr>
<td>RNOCIUC</td>
<td>12301</td>
<td>CMSIUCV error V</td>
</tr>
<tr>
<td>RNOIUCV</td>
<td>12302</td>
<td>IUCV error for a socket function S</td>
</tr>
<tr>
<td>RNOSEVE</td>
<td>12303</td>
<td>IUCV connection severed by TCP/IP R</td>
</tr>
<tr>
<td>RNOTOIUC</td>
<td>12304</td>
<td>Time-out during IUCV connection UC</td>
</tr>
</tbody>
</table>

>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.
### SERVENT Structure

The SERVENT structure is returned by the GETSERVbyname and GETSERVbyPORT function calls.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service name</td>
<td>Address of a service name (null-terminated string).</td>
</tr>
<tr>
<td>Aliases</td>
<td>Address of a zero-terminated array of pointers to aliases, which are null-terminated strings.</td>
</tr>
<tr>
<td>Port</td>
<td>Port number associated with a service.</td>
</tr>
<tr>
<td>Protocol</td>
<td>Address of the protocol associated with a service (null-terminated string).</td>
</tr>
</tbody>
</table>

### Socket Structure Descriptions

**Contents**
- ADDRINFO Structure (see page 156)
- SOCKADDR Structure (see page 157)
  - SOCKADDR for IPv4 (see page 157)
  - SOCKADDR for IPv6 (see page 157)
- TIMEVAL Structure (see page 157)

### ADDRINFO Structure

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

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

**SOCKADDR Structure**

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

**SOCKADDR for IPv4**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Family</td>
<td>A 2-byte field describing the socket address family type: AF_INET</td>
</tr>
<tr>
<td>Port number</td>
<td>The port number for this socket</td>
</tr>
<tr>
<td>Address</td>
<td>The 4-byte IP address of the TCP/IP stack</td>
</tr>
<tr>
<td>Zeros</td>
<td>Eight bytes of binary zeros</td>
</tr>
</tbody>
</table>

**SOCKADDR for IPv6**

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

**TIMEVAL Structure**

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seconds</td>
<td>Number of seconds to wait</td>
</tr>
<tr>
<td>Microseconds</td>
<td>Number of microseconds to wait</td>
</tr>
</tbody>
</table>
TCP/IP Programming Examples

For more information, see the following topics:

- PL/I Examples (see page 158)
- COBOL Examples (see page 166)
- Assembler Examples (see page 175)
- CA ADS Examples (see page 187)

PL/I Examples

Contents

- PL/I TCP/IP Client Program (see page 158)
- PL/I TCP/IP Generic Listener Server Program (see page 163)

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

PL/I TCP/IP Client Program

```pli
/*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 (20) 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 (');  
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 (21) INIT (' Calling FREEADDRINFO.');  
DCL 1 MSG24 CHAR (16) 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 ('255.255.255.2'),  
    3 FILLER CHAR (12) INIT (''),  
    3 IPA HOSTL FIXED BINARY(31) INIT(16);  
DCL 1 SOCKDESC FIXED BINARY(31);  
DCL 1 NFLAGS 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 BUFSIZE 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 DLODPRT, */
/* but that are not used by this test program. */
******************************************************************************
INCLUDE IDMS (SOCKET_LISTENER_PARM);
INCLUDE IDMS (SOCKET_SOCKADDR_INET);
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_RSPCD,
  SOCKET_FAMILY_INET,
  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_RSPCD,
  SOCKET_FAMILY_INET,
  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,
CALL TCP_CHECKRC;

BUFFTXT80 = 'PLICLI TCP/IP test message number 00001 ';
BUFLEN = 41;
WK_LENGTH = BUFLEN;
WK_PTR = ADDR(BUFLEN);
CALL TCP_WRITE;
IF (TERM_FLAG = 1) THEN GOTO TCP_EXIT;

BUFFTXT80 = 'PLICLI TCP/IP test message number 00002 ';
BUFLEN = 41;
WK_LENGTH = BUFLEN;
WK_PTR = ADDR(BUFLEN);
CALL TCP_WRITE;
IF (TERM_FLAG = 1) THEN GOTO TCP_EXIT;
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;
END TCP_WRITE;
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;
RETC= SOCKET RETCD;
ERRNO = SOCKET ERRNO;
RSNCD = SOCKET RSNCD;
IF (RETC = 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 (' Closing socket.');
DCL 1 MSG06 CHAR (16) INIT (' Starting write.');
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) & (STR(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) OR (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) OR (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

Contents
- COBOL TCP/IP Client Program (see page 166)
- COBOL TCP/IP Generic Listener Server Program (see page 172)

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 HOSTENYP 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 ARRAY PIC X(1) OCCURS 84 TIMES.
  01 BUFFER-REDEF REDEFINES BUFFER.
  01 BUFLEN PIC 9(8) COMP.
  01 BUFTXT80 PIC X(80).
  03 BUFTXT80-REDEF PIC X(4).
  05 BUFTXT04 PIC X(4).
  05 BUFTXT76 PIC X(76).
  03 BUFTXT80-REDEF2 PIC X(80).
  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).
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-RETC, SOCKET-ERRNO,
SOCKET-RSNCD,
SOCKET-DESRIPTOR,
BUFFER-ARRAY(WK-SUBSCRIPT),
WK-LENGTH,
RETLN
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-RETCM
   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-RETCM
   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-RETCM TO RETCM.
MOVE SOCKET-ERRNO TO ERRNO.
MOVE SOCKET-RSNCD TO RSNCD.
IF RETCM 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(88) 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 S9(8) COMP.
  03 BUFTXT80 PIC X(80).
  03 BUFTXT80-REDEF REDEFINES BUFTXT80.
  05 BUFTXT84 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,
RETLLEN
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,
RETLLEN
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
*****************************************************************
Assembler Examples

Contents

- Assembler TCP/IP Client Program (see page 175)
- Assembler TCP/IP Generic Listener Server Program (see page 182)

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'
* ASMLI RENT EP=ASMLIEP
******************************************************************************
* 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.
* - 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.
******************************************************************************
*---------------------------------------------------------------------*
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)
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   R14,0(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', 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 OVRLOGC0,=X'C000000000'
MVC WKCLEAR,=X'L8'0F0F0F0F0F0F0F0F'
MVC TRTAB,=CL16'0123456789ABCDEF'
MVC TRTABX,=XL6'FAFBFCFDFFEF'

* 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?
CA IDMS Reference - 19.0

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,
    IPADDRS=IPADOT,IPADDRSL=IPADOTL,IPADDR=HOSTIPA,
    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,
    DOMAIN=AF_INET,HOSTENTP=(R2),
    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)
BALR R14,R15 Display socket connect function.

*******************************************************************************
* Connect to DESTPORT *
*******************************************************************************
#SOCKET CONNECT,
    SOCK=S_NEWSOC,
    SOCKADDR=(R5),
    SOCKADDL=SIN#LEN,
    RETCODE=RETCODE,ERRNO=ERRNO,RSNCODE=RSNCODE
LA R1,MSG04 Connect successful?
L R15,=A(DISLINE)
BALR R14,R15 Display the 3 return codes.

*******************************************************************************
* 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 R7 -> buffer array.
LA R7,BUFFER_A 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' '  
MVC BUFTXT B(4)=F'43'  
MVC WK LEN(4),=F'47'  
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,BUFTXT  
LA R7,BUFFER A  
  R7 -> buffer array.  
BAL R4,TCPREAD  
  Perform socket read function.  

CLC BUFTXT(4),=F'80'  
  Incoming buffer less than 80?  
BL READLIS  
  Y. Read listener reply.  
LA R1,MSG11  
  N. Reply too long, issue error message and  
  BALR R14,R15  
  close socket.  
READLIS DS 0H  
MVC WK_LEN,BUFTXT  
  Reply msg length=read length.  
LA R7,BUFFER A+4  
  R7 -> reply from listener.  
BAL R4,TCPREAD  
  Read listener reply.  

******************************************************************  
* 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),  
  RETLEN=RETLEN,BUFFERL=WK_LEN,  
  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.  
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  
LA R1,MSG07  
  Display socket write function.  
L R15,=A(DISLINE)  
BALR R14,R15  
#SOCKET WRITE,SOCK=S_NEWSOC,BUFFER=(R7),  
  RETLEN=RETLEN,BUFFERL=WK_LEN,  
  RETCODE=RETCODE,ERRNO=ERRNO,RSNCODE=RSNCODE  

LA R1,MSG08  
  Display results of socket write.  
L R15,=A(DISRC)  
BALR R14,R15  
*  
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.
X
#SOCKET CLOSE,SOCK=S_NEWSOC,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 ASMCCLI temrination 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,OUTAREAX
#WTL MSGID=M#999043,MSGDICT=NO,OVRIDES=OVRLOG,

PARMS=((R3)),RGSV=(R2-R8)
*

#RETURN
#BALI
*
DROP R12
LTORG
TITLE 'ASMCCLI    - 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'OUTAREAX
MVI OUTAREAX,C'
MVC OUTAREAT+1(L'OUTAREAT-1),OUTAREAX
*
XR R2,R2
IC R2,0(1,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  8(R2),='CL8'RETCODE='  
MVC  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(R2),='CL6'ERRNO='  
MVC  23(R2),WORK2+2  
  
MVC  WORK1(4),RSNCODE  
UNPK  WORK2(9),WORK1(5)  
NC  WORK2(8),WKCLEAR  
TR  WORK2(8),TRTAB  
MVC  32(R2),='CL8'RSNCODE='  
MVC  40(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  
  'CA IDMS Reference - 19.0'  
  *---------------------------------------------------------------------*  
  *- 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,  
       PARMS=((R3)),RGSV=(R2-R8)  
  
#RESTREG  Restore the caller's registers  
  
BR   R14  
  
DROP R12  
  
M#999043 DC PL4'9990430'  
  
#WTL  MSGID=M#999043,MSGDICT=NO,OVRIDES=OVRLOG,  
       PARMS=((R3)),RGSV=(R2-R8)  
  
#RESTREG  Restore the caller's registers  
  
BR   R14  
  
DROP R12  
  
M#999043 DC PL4'9990430'  
  
#WTL  MSGID=M#999043,MSGDICT=NO,OVRIDES=OVRLOG,  
       PARMS=((R3)),RGSV=(R2-R8)  
  
#RESTREG  Restore the caller's registers  
  
BR   R14  
  
DROP R12  
  
M#999043 DC PL4'9990430'  
  
#WTL  MSGID=M#999043,MSGDICT=NO,OVRIDES=OVRLOG,  
       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:'

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
*DS   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
*DS   SOCKADDR DS   (SIN#LEN)X   SOCKADDR for the client
SOCKDESN DS   F
S NEWSOC DS   F
RETLLEN DS   F
RETCODE DS   F
ERRNO DS   F
RSNCODE DS   F
*DS   OUTAREA DS   X      OUTPUT AREA
OUTAREAT DC  CL80' '     OUTPUT AREA
BS    0F
BUFFER DS   CL84
OR     BUFFER
BUFLEN DS   F
BUFTXT80 DS   CL80
OR     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
OR     BUFFER
BUFFER_A DS   CL84
*DS   INTERFACE DSECT

*----------------------------------*
*- STATIC DATA                     *
*----------------------------------*
LTORG

*
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 returns 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,(R13)         Save R12
ST    R14,(R13)         Save R14
STM   R2,R8,(R13)       Save R2-R8
LA    R13,9*4,(R13)     Get register stack entry length
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,(R13)         Restore R12
L    R14,(R13)         Restore R14
LM   R2,R8,(R13)       Restore R2-R8
MEND

MACRO
&LABEL.   MSGTXT "TXT.
LCLC "TMP.
SETC 'SYSNDX'.
&LABEL.   DC   AL1(L2&TMP).
L1&TMP.   DC   C"TXT.
L2&TMP.   DC   C"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', LEN=WKAREL
LR R11,R1
USING WORKAREA,R11
LA R13,REGSTACK
MVI OUTAREA,L'OUTAREAT
MVC OVRLOG,=X'0000000000'
MVC OVRLOGCO,=X'C000000000'
MVC WKCLEAR,=XL'B0F0F0F0F0F0F0F0'  
MVC TRTAB,=CL16'0123456789ABCDEF'
MVC TRTABX,=XL6'FAFBFCFDFFE'

TCPSTART DS 0H
******************************************************************
* Read the first 4 bytes: will contain the remaining length 
******************************************************************
*                                                                 
LA R5,4
ST R5,WK_LEN
XR R5,R5
ST R5,BUFLEN
LA R7,BUFFER_A
BAL R4,TCPREAD

CLC BUFLEN(4),=F'80'  
Y. Read client message.
LA R1,MSG04
L R15,=A(DISLINE)
BALR R14,R15
B TCPCLOSE

READCLI DS 0H
MVC WK_LEN,BUFLEN
LA R7,BUFFER_A+4
BAL R4,TCPREAD

XC OUTAREAT(L'OUTAREAT),OUTAREAT Clear out message area
MVC OUTAREAT(15),=C'Buffer Length: ' Build
L R1,BUFLEN
CVD R1,WORK1
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,R5(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.
CA IDMS Reference - 19.0

******************************************************************
* 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),
   RETLEN=RETLEN,BUFFERL=WK LEN,
   RETCODE=RETCODE,ERRNO=ERRNO,RSNCODE=RSNCODE
*  
LA R1,MSG06           Display results of read.
L R15,=A(DISRC)
BALR R14,R15
CLC RETCODE,=F'0'
BNE TCPCLOSE
CLC RETLEN,=F'0'
BE TCPCLOSE
A R7,RETLEN
L R5,WK LEN
S R5,RETLEN
ST R5,WK LEN
ST R5,RETLEN
CLC 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),
   RETLEN=RETLEN,BUFFERL=WK LEN,
   RETCODE=RETCODE,ERRNO=ERRNO,RSNCODE=RSNCODE
*  
LA R1,MSG05           Display results of socket write.
L R15,=A(DISRC)
BALR R14,R15
*  
CLC RETCODE,=F'0'
BNE TCPCLOSE
A R7,RETLEN
L R5,WK LEN
S R5,RETLEN
ST R5,WK LEN
ST R5,RETLEN
CLC WK LEN,0
BNE TCPWRITE
BR R4

******************************************************************
* Close the socket and exit                                       *
******************************************************************
TCPCLOSE DS 0H
#SOCKET CLOSE,SOCK=S NEWSOC,
   RETCODE=RETCODE,ERRNO=ERRNO,RSNCODE=RSNCODE
*  
LA R1,MSG07           Display socket close function.
L R15,=A(DISRC)
BALR R14,R15
*  
LISEXIT DS 0H
*  
#RETURN
#_BALI

*  
DROP R12
LTORG
TITLE 'ASMLIS01 - 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                              *
*                  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    0(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
RETLN 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'800000000000' #WTL TO LOG ONLY
OVRLOGCO DS X'C00000000000' #WTL TO LOG + CONSOLE
* WKCLEAR DS X'0F0F0F0F0F0F0F0F'
CA ADS Examples

Contents
- CA ADS TCP/IP Client Program (see page 187)
- CA ADS TCP/IP Generic Listener Server Program (see page 191)

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 ***

ADD RECORD ADSCLI-WORK-RECORD.
02 WK-RETCD PIC S9(8) COMP.
02 WK-ERRCO PIC S9(8) COMP.
02 WK-RSNCDO 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-LANGTH 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 '255.255.255.255'.
02 FILLER PIC X(4) VALUE SPACES.
02 IPA-HOSTL PIC S9(8) COMP VALUE 16.
ADD RECORD ADSCLI-BUFFER-RECORD.
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 TCPErrroR.
  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 TCPErrroR.
  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.
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 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 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.
*****************************************************************************
*** ADSLIS ***
*** IDD input ***
*** Use also the following work records defined for ADSCLI: ***
*** ADSCLl-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 BUFSIZE 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 TCPPERERROR.
    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 TCPPERERROR.
    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.