Examples: APIs

This article contains example programs that use APIs. It also contains examples of two exit programs. For additional information concerning API examples, see the System API Programming manual on the V5R1 Supplemental Manuals Web site.

See Code disclaimer information for information pertaining to code examples.

The examples included are:

- Deleting Old Spooled Files
- Changing an Active Job
- Changing a Job Schedule Entry
- Creating Your Own Telephone Directory
- Creating and Manipulating a User Index
- Creating a Batch Machine
- Defining Queries
- Generating and Sending an Alert
- Diagnostic Reporting
- Listing Directories
- Listing Subdirectories
- Saving to Multiple Devices
- Scanning String Patterns
- Using COBOL Program to Call APIs
- Using the User-Defined Communications Programs for File Transfer
- Using the Control Device (QTACTLDV) API
- Using a Data Queue
- Using Environment Variables
- Saving and Restoring System-Level Environment Variables
- Using ILE Common Execution Environment Data APIs
- Using the Generic Terminal APIs
- Using Profile Handles
- Using Registration Facility APIs
- Using Semaphores and Shared Memory
- Using SNA/Management Services Transport APIs
- Using Source Debugger APIs
- Using the Spawn Process and Wait for Child Process APIs
- Working with Stream Files

The exit program examples are:

- Creating a Program Temporary Fix Exit Program
- Using the Operational Assistant Exit Program for Operational Assistant Backup

Deleting Old Spooled Files

The following application program runs using the Delete Old Spooled Files (DLTOLDSPLF) command. This example has three major parts:

1. The DLTOLDSPLF command calls the delete old spooled files (DLTOLDSPLF) program in one of the following languages:
    - OPM RPG
    - OPM COBOL
    - ILE C
2. The DLTOLDSPLF program is supplied in OPM RPG, OPM COBOL, and ILE C. It does the following:
    a. Creates a user space (QUSCRTUS API).
    b. Generates a list of spooled files (QUSLSPL API).
    c. Retrieves information from a user space using one of the following:
        - QUSRTVUS API
        - QUSPTRUS API
    d. Retrieves more spooled file attribute information received from the user space (QUSRSPLA API).
    e. Calls the CLDLT program to delete the spooled files.
    f. Sends a message to the user (QMHSNDM API).
    g. Deletes the user space (QUSDLTUS API).
3. The CL delete (CLDLT) program does the following:
    a. Deletes the specified spooled files (DLTSPLF command).
    b. Sends a message if the spooled file was deleted (SNDPGMMSG command).

Note: The programs and source code used as examples in the spooled file portion of this article exist only in printed form. They are not stored electronically on the iSeries server.

DLTOLDSPLF Command Source

The command source for the DLTOLDSPLF command follows:

```
/*************************************************************/
/*                                                           */
/* CMD:  DLTOLDSPLF                                          */
/*                                                           */
/* LANGUAGE:  CL COMMAND SOURCE                              */
/*                                                           */
/* DESCRIPTION:  COMMAND SOURCE FOR THE DLTOLDSPLF COMMAND WHICH*/
/*               INVOKES THE DLTOLDSPLF PROGRAM.             */
/*                                                           */
/*************************************************************/
      CMD PROMPT('DELETE OLD SPOOLED FILES')
/* PARAMETERS FOR LIST OF SPOOLED FILES (QUSLSPL)            */
      PARM KWD(USRPRFNME)  +
           TYPE(*SNAME)    +
           LEN(10)         +
           MIN(1)          +
           SPCVAL(*ALL)    +
           PROMPT('User Profile Name:')
      PARM KWD(OUTQUEUE)   +
           TYPE(QUAL1)     +
```

```
             MIN(1)          +
             PROMPT('Output Queue:')
   /* INFORMATION NEEDED FOR PROGRAM                                  */
         PARM KWD(DELETEDATE) +
              TYPE(*DATE)     +
              PROMPT('Last Deletion Date:')
QUAL1:   QUAL TYPE(*NAME) LEN(10)  SPCVAL(*ALL)
         QUAL TYPE(*NAME) LEN(10) SPCVAL(*LIBL *CURLIB '          ')  +
              PROMPT('Library Name:')
```

To create the CL command, specify the following:

```
CRTCMD CMD(QGPL/DLTOLDSPLF) PGM(QGPL/DLTOLDSPLF) +
    SRCFILE(QGPL/QCMDSRC) ALLOW(*IPGM *BPGM)
```

To delete old spooled files, you can use one of the application programs provided in the following languages:

- RPG
- COBOL
- ILE C

RPG DLTOLDSPLF Program

To delete old spooled files, use the following RPG program:

```
     H* ****************************************************************
     H* ****************************************************************
     H*                                                                *
     H* MODULE:    DLTOLDSPLF                                          *
     H*                                                                *
     H* LANGUAGE:  RPG                                                 *
     H*                                                                *
     H* FUNCTION:  THIS APPLICATION WILL DELETE OLD SPOOLED FILES      *
     H*            FROM THE SYSTEM, BASED ON THE INPUT PARAMETERS.     *
     H*                                                                *
     H* APIs USED:                                                     *
     H*            QUSCRTUS -- Create User Space                       *
     H*            QUSLSPLF -- List Spooled Files                      *
     H*            QUSRTVUS -- Retrieve User Space                     *
     H*            QUSRSPLA -- Retrieve Spooled File Attributes        *
     H*            QMHSNDPM -- Send Program Message                    *
     H*            QUSDLTUS -- Delete User Space                       *
     H*                                                                *
     H* ****************************************************************
     H* ****************************************************************
     E/COPY QRPGSRC,EUSRSPLA
     I              'NUMBER OF SPOOLED -  C         MSGTXT
     I              'FILES DELETED:  '
     IMSGDTA      DS
     I                                        1  35 MSGDT1
     I                                       36  400DLTCNT
     ISTRUCT      DS
     I                                    B   1   40USSIZE
     I                                    B   5   80GENLEN
     I                                    B   9  120RTVLEN
     I                                    B  13  160STRPOS
     I                                    B  17  200RCVLEN
     I                                    B  21  240SPLF#
     I                                    B  25  280MSGDLN
     I                                    B  29  320MSGQ#
     I                                       33  38 FIL#
     I                                       39  42 MSGKEY
     I I            'DLTOLDSPLFQTEMP     '   43  62 USRSPC
     I I            '*REQUESTER          '   63  82 MSGQ
```

```
ITGTDAT      DS
I                                       1   1 TGTCEN
I                                       2   3 TGTYR
I                                       4   5 TGTMTH
I                                       6   7 TGTDAY
I/COPY QRPGSRC,QUSGEN
I/COPY QRPGSRC,QUSLSPL
I/COPY QRPGSRC,QUSRSPLA
I*****************************************************************
I* The following is copied from QSYSINC/QRPGSRC member QUSEC
I* so that the variable length field QUSBNG can be defined
I* as 100 bytes for exception data.  The defined field is
I* named EXCDTA.
I*****************************************************************
IQUSBN       DS
I*                                                Qus EC
I                                     B   1   40QUSBNB
I*                                                Bytes Provided
I                                     B   5   80QUSBNC
I*                                                Bytes Available
I                                         9  15 QUSBND
I*                                                Exception Id
I                                        16  16 QUSBNF
I*                                                Reserved
I*                                       17  17 QUSBNG
I*                                       Varying length
I                                        17 116 EXCDTA
IDATSTR      DS
I                                       1   1 DATCEN
I                                     202 203 DATYR
I                                     204 205 DATMTH
I                                     206 207 DATDAY
C* ***************************************************************
C* ***************************************************************
C*                                                                 *
C*               EXECUTABLE CODE STARTS HERE                       *
C*                                                                 *
C* ***************************************************************
C* ***************************************************************
C*                                                                 *
C           *ENTRY    PLIST
C                     PARM           USRNAM 10
C                     PARM           OUTQ   20
C                     PARM           DLTDAT  7
C                     MOVE DLTDAT    TGTDAT
C                     Z-ADD0         DLTCNT
C                     MOVE *BLANKS   QUSBN
C                     Z-ADD0         QUSBNB
C*                                                                 *
C* CREATE A USER SPACE TO STORE THE LIST OF SPOOLED FILES.         *
C*                                                                 *
C                     CALL 'QUSCRTUS'
C                     PARM           USRSPC
C                     PARM *BLANKS   USEXAT 10
C                     PARM 1024      USSIZE
C                     PARM ' '       USINIT  1
C                     PARM '*CHANGE 'USAUTH 10
C                     PARM *BLANKS   USTEXT 50
C                     PARM '*YES    'USREPL 10
C                     PARM           QUSBN
C*                                                                 *
C* FILL THE USER SPACE JUST CREATED WITH SPOOLED FILES AS          *
C* DEFINED IN THE CL COMMAND.                                      *
C*                                                                 *
C                     CALL 'QUSLSPL'
C                     PARM           USRSPC
C                     PARM 'SPLF0100'FMTNM1  8
```

```
C                   PARM           USRNAM
C                   PARM           OUTQ
C                   PARM '*ALL    'FRMTYP 10
C                   PARM '*ALL    'USRDTA 10
C                   PARM           QUSBN
C*                                                                 *
C* THE USER SPACE IS NOW FILLED WITH THE LIST OF SPOOLED FILES.    *
C* NOW USE THE QUSRTVUS API TO FIND THE NUMBER OF ENTRIES AND      *
C* THE OFFSET AND SIZE OF EACH ENTRY IN THE USER SPACE.            *
C*                                                                 *
C                   Z-ADD140       GENLEN
C                   Z-ADD1         STRPOS
C*                                                                 *
C                   CALL 'QUSRTVUS'
C                   PARM           USRSPC
C                   PARM           STRPOS
C                   PARM           GENLEN
C                   PARM           QUSBP
C                   PARM           QUSBN
C*                                                                 *
C* CHECK THE GENERIC HEADER DATA STRUCTURE FOR NUMBER OF LIST      *
C* ENTRIES, OFFSET TO LIST ENTRIES, AND SIZE OF EACH LIST ENTRY.   *
C*                                                                 *
C                   Z-ADDQUSBPQ    STRPOS
C                   ADD  1         STRPOS
C                   Z-ADDQUSBPT    RTVLEN
C                   Z-ADD209       RCVLEN
C                   Z-ADD1         COUNT  150
C*                                                                 *
C* ****************************************************************
C* ****************************************************************
C*                                                                 *
C*     BEGINNING OF LOOP (DO WHILE COUNT <= QUSBPS)                *
C*                                                                 *
C* ****************************************************************
C*                                                                 *
C           COUNT     DOWLEQUSBPS
C*                                                                 *
C* RETRIEVE THE INTERNAL JOB IDENTIFIER AND INTERNAL SPOOLED FILE  *
C* IDENTIFIER FROM THE ENTRY IN THE USER SPACE.  THIS INFORMATION  *
C* WILL BE USED TO RETRIEVE THE ATTRIBUTES OF THE SPOOLED FILE.    *
C* THIS WILL BE DONE FOR EACH ENTRY IN THE USER SPACE.             *
C*                                                                 *
C                   CALL 'QUSRTVUS'
C                   PARM           USRSPC
C                   PARM           STRPOS
C                   PARM           RTVLEN
C                   PARM           QUSFT
C                   PARM           QUSBN
C*                                                                 *
C* NOW RETRIEVE THE SPOOLED FILE ATTRIBUTES USING THE QUSRSPLA     *
C* API.                                                            *
C*                                                                 *
C                   MOVE *BLANKS   JOBINF
C                   MOVEL'*INT'    JOBINF 26
C                   MOVE QUSFTH    QUSFXD
C                   MOVE QUSFTJ    QUSFXF
C                   MOVEL'*INT'    SPLFNM 10
C                   MOVE *BLANKS   SPLF#
C*                                                                 *
C                   CALL 'QUSRSPLA'
C                   PARM           QUSFX
C                   PARM           RCVLEN
C                   PARM 'SPLA0100'FMTNM2  8
C                   PARM           JOBINF
C                   PARM           QUSFXD
C                   PARM           QUSFXF
```

```
C                     PARM           SPLFNM
C                     PARM           SPLF#
C                     PARM           QUSBN
C*                                                                     *
C* CHECK QUSFX DATA STRUCTURE FOR DATE FILE OPENED.                    *
C* DELETE SPOOLED FILES THAT ARE OLDER THAN THE TARGET DATE           *
C* SPECIFIED ON THE COMMAND.  A MESSAGE IS SENT FOR EACH SPOOLED      *
C* FILE DELETED.                                                      *
C*                                                                    *
C*                                                                    *
C                     MOVE QUSFX7    DATSTR

C           DATCEN    IFLT TGTCEN
C                     EXSR CLDLT
C                     ELSE
C           DATCEN    IFEQ TGTCEN

C           DATYR     IFLT TGTYR
C                     EXSR CLDLT
C                     ELSE
C           DATYR     IFEQ TGTYR
C           DATMTH    IFLT TGTMTH
C                     EXSR CLDLT
C                     ELSE                              NOT LT MTH
C           DATMTH    IFEQ TGTMTH
C           DATDAY    IFLE TGTDAY
C                     EXSR CLDLT
C                     END                               FOR LE DAY
C                     END                               FOR EQ MTH
C                     END                               FOR ELSE MTH
C                     END                               FOR EQ YR
C                     END                               FOR ELSE YR

C                     END                               FOR EQ CEN
C                     END                               FOR ELSE CEN

C*                                                                     *
C* GO  BACK AND PROCESS THE REST OF THE ENTRIES IN THE USER           *
C* SPACE.                                                             *
C           QUSBPT    ADD  STRPOS    STRPOS
C           1         ADD  COUNT     COUNT
C                     END
C* ********************************************************** *
C* ********************************************************** *
C*                                                                    *
C*              END OF LOOP                                           *
C*                                                                    *
C* ********************************************************** *
C* ********************************************************** *
C*                                                                    *
C* AFTER ALL SPOOLED FILES HAVE BEEN DELETED THAT MET THE             *
C* REQUIREMENTS, SEND A FINAL MESSAGE TO THE USER.                    *
C* DELETE THE USER SPACE OBJECT THAT WAS CREATED.                     *
C*                                                                    *
C                     MOVELMSGTXT    MSGDT1
C                     CALL 'QMHSNDM'
C                     PARM *BLANKS   MSGID   7
C                     PARM *BLANKS   MSGFIL 20
C                     PARM           MSGDTA
C                     PARM 40        MSGDLN
C                     PARM '*INFO   'MSGTYP 10
C                     PARM           MSGQ
C                     PARM 1         MSGQ#
C                     PARM *BLANKS   RPYMQ  10
C                     PARM           MSGKEY
C                     PARM           QUSBN
C*                                                                     *
```

```
C* DELETE THE USER SPACE OBJECT THAT WAS CREATED.                *
C*                                                               *
C                     CALL 'QUSDLTUS'
C                     PARM           USRSPC
C                     PARM           QUSBN
C*                                                              *
C*                                                              *
C* ************************************************************ *
C* ************************************************************ *
C*                                                              *
C*                       END OF PROGRAM                         *
C*                                                              *
C* ************************************************************ *
C                     RETRN
C*
C* ************************************************************ *
C*                                                              *
C*                   CLDLT  SUBROUTINE                          *
C*                                                              *
C* THIS SUBROUTINE CALLS A CL PROGRAM THAT WILL DELETE A SPOOLED *
C* FILE AND SEND A MESSAGE THAT THE SPOOLED FILE WAS DELETED.    *
C*                                                              *
C* ************************************************************ *
C*                                                              *
C           CLDLT     BEGSR
C*                                                              *
C* KEEP A COUNTER OF HOW MANY SPOOLED FILES ARE DELETED.        *
C*                                                              *
C                     ADD  1         DLTCNT
C                     MOVE QUSFXL    FIL#
C                     CALL 'CLDLT'
C                     PARM           QUSFXK
C                     PARM           QUSFXJ
C                     PARM           QUSFXH
C                     PARM           QUSFXG
C                     PARM           FIL#
C                     PARM           QUSFXM
C                     PARM           QUSFXN
C                     ENDSR
```

To create the RPG program, specify the following:

```
CRTRPGPGM PGM(QGPL/DLTOLDSPLF) SRCFILE(QGPL/QRPGSRC)
```

COBOL DLTOLDSPLF Program

To delete spooled files, you can use this COBOL DLTOLDSPLF program:

```
      *************************************************************
      *                                                           *
      * PROGRAM:  DLTOLDSPLF                                      *
      *                                                           *
      * LANGUAGE:  COBOL                                          *
      *                                                           *
      * DESCRIPTION:  DELETE OLD SPOOLED FILES                    *
      *                                                           *
      * APIs USED:  QUSCRTUS, QUSLSPL, QUSRTVUS, QUSRSPLA, QUSDLTUS,*
      *             AND QMHSNDM.                                  *
      *************************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  DLTOLDSPLF.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-AS400.
```

```
 OBJECT-COMPUTER. IBM-AS400.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 DATA DIVISION.
 FILE SECTION.
 WORKING-STORAGE SECTION.
 COPY QUSGEN OF QSYSINC-QLBLSRC.
 COPY QUSLSPL OF QSYSINC-QLBLSRC.
 COPY QUSRSPLA OF QSYSINC-QLBLSRC.
****************************************************************
* VALUES USED FOR ERROR CODE                                   *
****************************************************************
* The following is copied from QSYSINC/QLBLSRC member QUSEC
* so that the variable length field EXCEPTION-DATA can be defined
* as 100 bytes for exception data.
****************************************************************
 01  QUS-EC.
     05  BYTES-PROVIDED                    PIC S9(00009) BINARY.
     05  BYTES-AVAILABLE                   PIC S9(00009) BINARY.
     05  EXCEPTION-ID                      PIC  X(00007).
     05  RESERVED                          PIC  X(00001).
*    05  EXCEPTION-DATA                    PIC  X(00001).
*
*                                    Varying length
     05 EXCEPTION-DATA          PIC X(100).

****************************************************************
* VALUES USED FOR THE QUSCRTUS PROGRAM                         *
****************************************************************
 01  CRTUS-INFO.
     05  CRT-SPCNAME            PIC X(20)
                VALUE "DLTOLDSPLFQTEMP     ".
     05  CRT-EXTATTR            PIC X(10) VALUE SPACE.
     05  CRT-SPCSIZE            PIC S9(9) BINARY VALUE 1024.
     05  CRT-INITSPACE          PIC X VALUE " ".
     05  CRT-AUTHORITY          PIC X(10) VALUE "*CHANGE   ".
     05  CRT-DESCRIPTION        PIC X(50) VALUE SPACE.
     05  CRT-USRRPL             PIC X(10) VALUE "*YES      ".

****************************************************************
* VALUES USED FOR THE QUSRTVUS PROGRAM                         *
****************************************************************
 01  RTV-START-POS              PIC S9(9) BINARY VALUE 1.
 01  RTV-LENGTH                 PIC S9(9) BINARY VALUE 140.

 01  RTVSPLA-JOB-ID             PIC X(26) VALUE "*INT".

****************************************************************
* VALUES USED FOR THE QUSLSPL AND QUSRSPLA PROGRAM             *
****************************************************************
 01  RSPLA-DATE.
     05 R-CENTURY               PIC X.
     05 R-YEAR                  PIC X(2).
     05 R-MONTH                 PIC X(2).
     05 R-DAY                   PIC X(2).
 01  LSPLA-FORMAT               PIC X(8)  VALUE "SPLF0100".
 01  LSPLA-USERDATA             PIC X(10) VALUE "*ALL      ".
 01  LSPLA-FORMTYPE             PIC X(10) VALUE "*ALL      ".
 01  RSPLA-JOB-NAME             PIC X(26) VALUE "*INT".
 01  RSPLA-NAME                 PIC X(10) VALUE "*INT".
 01  RSPLA-NUMBER               PIC S9(9) BINARY VALUE -1.
 01  RSPLA-FORMAT               PIC X(10) VALUE "SPLA0100  ".
 01  SPLA-VAR-LENGTH            PIC S9(9) BINARY VALUE 800.
 01  DLT-COUNT                  PIC 9(15) VALUE 0.
 01  DLT-SPL-NUMBER             PIC 9(6).

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

```
* VALUES USED FOR THE QMHSNDM PROGRAM                          *
*****************************************************************
  01  MSG-ID                     PIC X(7)  VALUE SPACE.
  01  MSG-FL-NAME                PIC X(20) VALUE SPACE.
  01  MSG-DATA.
      05  DATA-MD                PIC X(34)
          VALUE "NUMBER OF SPOOLED FILES DELETED : ".
      05  DLT-NUM-MD             PIC X(20) VALUE SPACE.
  01  MSG-DATA-LEN               PIC S9(9) BINARY VALUE 54.
  01  MSG-TYPE                   PIC X(10) VALUE "*INFO     ".
  01  MSG-QUEUE                  PIC X(20)
          VALUE "*REQUESTER          ".
  01  MSG-Q-NUM                  PIC S9(9) BINARY VALUE 1.
  01  RPY-MSG                    PIC X(10) VALUE SPACE.
  01  MSG-KEY                    PIC X(4)  VALUE SPACE.

*****************************************************************
* PARAMETERS THAT ARE PASSED TO THIS PROGRAM FROM THE COMMAND *
*****************************************************************
 LINKAGE SECTION.
 01 PARM-USERNAME               PIC X(10).
 01 PARM-OUTQ                   PIC X(20).
 01 PARM-DATE.
     05 P-CENTURY               PIC X.
     05 P-YEAR                  PIC X(2).
     05 P-MONTH                 PIC X(2).
     05 P-DAY                   PIC X(2).

*****************************************************************
* BEGINNING OF EXECUTABLE CODE.                                 *
*****************************************************************
 PROCEDURE DIVISION USING PARM-USERNAME,
                          PARM-OUTQ,
                          PARM-DATE.

 MAIN-PROGRAM.

*    ************************************************************
*    * INITIALIZE ERROR CODE STRUCTURE.                         *
*    ************************************************************

      MOVE 116 TO BYTES-PROVIDED.
      MOVE 0 TO BYTES-AVAILABLE.
      MOVE SPACES TO EXCEPTION-ID.
      MOVE SPACES TO RESERVED OF QUS-EC.
      MOVE SPACES TO EXCEPTION-DATA.

*    ************************************************************
*    * CREATE THE USER SPACE USING INPUT PARMS FOR THE CALL     *
*    ************************************************************

     CALL "QUSCRTUS" USING CRT-SPCNAME,
                           CRT-EXTATTR,
                           CRT-SPCSIZE,
                           CRT-INITSPACE,
                           CRT-AUTHORITY,
                           CRT-DESCRIPTION,
                           CRT-USRRPL,
                           QUS-EC.

*    ************************************************************
*    * LIST THE SPOOLED FILES TO THE USER SPACE OBJECT.          *
*    ************************************************************

     CALL "QUSLSPL"  USING CRT-SPCNAME,
                           LSPLA-FORMAT,
                           PARM-USERNAME,
```

```
                          PARM-OUTQ,
                          LSPLA-FORMTYPE,
                          LSPLA-USERDATA,
                          QUS-EC.

*    ************************************************************
*    * RETRIEVE ENTRY INFORMATION FROM THE USER SPACE.         *
*    ************************************************************

     CALL "QUSRTVUS" USING CRT-SPCNAME,
                          RTV-START-POS,
                          RTV-LENGTH,
                          QUS-GENERIC-HEADER-0100,
                          QUS-EC.

*    ************************************************************
*    * IF ANY SPOOLED FILES WERE FOUND MATCHING THE SEARCH     *
*    * CRITERIA, RETRIEVE DETAILED INFORMATION AND DECIDE      *
*    * WHETHER TO DELETE THE FILE OR NOT.                      *
*    ************************************************************

     IF NUMBER-LIST-ENTRIES OF QUS-GENERIC-HEADER-0100
                           GREATER THAN ZERO THEN
         ADD 1 TO OFFSET-LIST-DATA OF QUS-GENERIC-HEADER-0100
                           GIVING RTV-START-POS.
         PERFORM CHECK-AND-DELETE THROUGH
             CHECK-AND-DELETE-END NUMBER-LIST-ENTRIES
                           OF QUS-GENERIC-HEADER-0100 TIMES.

*    ************************************************************
*    * CALL THE QUSDLTUS API TO DELETE THE USER SPACE           *
*    * WE CREATED, AND TO SEND A MESSAGE TELLING HOW MANY      *
*    * SPOOLED FILES WERE DELETED.                             *
*    ************************************************************

     CALL "QUSDLTUS" USING CRT-SPCNAME,
                          QUS-EC.

     MOVE DLT-COUNT TO DLT-NUM-MD.
     CALL "QMHSNDM" USING MSG-ID,
                          MSG-FL-NAME,
                          MSG-DATA,
                          MSG-DATA-LEN,
                          MSG-TYPE,
                          MSG-QUEUE,
                          MSG-Q-NUM,
                          RPY-MSG,
                          MSG-KEY,
                          QUS-EC.

     STOP RUN.

*    ************************************************************
*    * CHECK THE DATE OF THE SPOOLED FILE.  IF IT IS OLDER      *
*    * OR EQUAL TO THE DATE PASSED IN, CALL THE PROCEDURE      *
*    * TO DELETE THE SPOOLED FILE.                             *
*    ************************************************************

 CHECK-AND-DELETE.
     CALL "QUSRTVUS" USING CRT-SPCNAME,
                          RTV-START-POS,
                          SIZE-EACH-ENTRY OF
                                    QUS-GENERIC-HEADER-0100,
                          QUS-SPLF0100,
                          QUS-EC.

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

```
*    * ADVANCE TO NEXT SPOOLED FILE FOR PROCESSING THE CHECK  *
*    * AND DELETE.                                            *
*    ***********************************************************

     ADD SIZE-EACH-ENTRY OF QUS-GENERIC-HEADER-0100 TO
                            RTV-START-POS GIVING RTV-START-POS.

*    ***********************************************************
*    * RETRIEVE THE ATTRIBUTES FOR THE SPOOLED FILE TO GET    *
*    * THE CREATE DATE FOR THE SPOOLED FILE.                  *
*    ***********************************************************

     CALL "QUSRSPLA" USING QUS-SPLA0100,
                           SPLA-VAR-LENGTH,
                           RSPLA-FORMAT,
                           RSPLA-JOB-NAME,
                           INT-JOB-ID OF QUS-SPLF0100,
                           INT-SPLF-ID OF QUS-SPLF0100,
                           RSPLA-NAME,
                           RSPLA-NUMBER,
                           QUS-EC.

     MOVE DATE-FILE-OPEN OF QUS-SPLA0100 TO RSPLA-DATE.

*    ***********************************************************
*    * COMPARE THE CREATE DATE WITH THE DATE THAT WAS PASSED  *
*    * IN AS PARAMETER.                                       *
*    ***********************************************************

     IF R-CENTURY IS LESS THAN P-CENTURY THEN
       PERFORM DLT-SPLF THROUGH DLT-SPLF-END
     ELSE
       IF R-CENTURY IS EQUAL TO P-CENTURY THEN

     IF R-YEAR IS LESS THAN P-YEAR THEN
       PERFORM DLT-SPLF THROUGH DLT-SPLF-END
     ELSE
       IF R-YEAR IS EQUAL TO P-YEAR THEN
         IF R-MONTH IS LESS THAN P-MONTH THEN
           PERFORM DLT-SPLF THROUGH DLT-SPLF-END
         ELSE
           IF R-MONTH IS EQUAL TO P-MONTH THEN
             IF R-DAY IS LESS THAN OR EQUAL TO P-DAY THEN
               PERFORM DLT-SPLF THROUGH DLT-SPLF-END.

 CHECK-AND-DELETE-END.

*    ***********************************************************
*    * THIS IS THE PROCEDURE TO DELETE THE SPOOLED FILE.      *
*    * ALL OF THE SPOOLED FILES WITH CREATE DATE OLDER OR     *
*    * EQUAL TO THE DATE PASSED IN AS PARAMETER WILL BE       *
*    * DELETED.                                               *
*    ***********************************************************

 DLT-SPLF.
     ADD 1 TO DLT-COUNT.
     MOVE SPLF-NUMBER OF QUS-SPLA0100 TO DLT-SPL-NUMBER.

     CALL "CLDLT" USING SPLF-NAME OF QUS-SPLA0100,
                        JOB-NUMBER OF QUS-SPLA0100,
                        USR-NAME OF QUS-SPLA0100,
                        JOB-NAME OF QUS-SPLA0100,
                        DLT-SPL-NUMBER,
                        FORM-TYPE OF QUS-SPLA0100,
                        USR-DATA OF QUS-SPLA0100.

 DLT-SPLF-END.
```

To create the COBOL program, specify the following:

```
CRTCBLPGM PGM(QGPL/DLTOLDSPLF) SRCFILE(QGPL/QCBLSRC)
```

ILE C DLTOLDSPLF Program

To delete spooled files, you can use this ILE C DLTOLDSPLF program:

```
/*****************************************************************/
/*  PROGRAM:  DLTOLDSPLF                                         */
/*                                                               */
/*  LANGUAGE: ILE C for OS/400                                   */
/*                                                               */
/*  DESCRIPTION:  THIS IS AN EXAMPLE PROGRAM FOR THE USE OF      */
/*                USER SPACES WRITTEN IN ILE C for OS/400.       */
/*                THE FLOW OF THIS PROGRAM IS AS FOLLOWS:        */
/*               (1) CREATE A USER SPACE USING QUSCRTUS          */
/*               (2) GET LIST OF SPOOLED FILES IN THE USER SPACE */
/*                   USING QUSLSPL                               */
/*               (3) KEEP POINTER TO ENTRY LIST IN THE USER SPACE*/
/*               (4) ENTER LOOP                                  */
/*                    RETRIEVE LIST ENTRY                        */
/*                    RETRIEVE MORE INFORMATION USING QUSRSPLA   */
/*                    IF SPOOLED FILE IS TOO OLD                 */
/*                       DELETE SPOOLED FILE                     */
/*                       INCREMENT DELETE COUNTER                */
/*                   END LOOP                                    */
/*               (5) DELETE USER SPACE                           */
/*                                                               */
/*  APIs USED:  QUSCRTUS, QUSLSPL, QUSRSPLA, QUSPTRUS, QUSDLTUS, */
/*              QMHSNDPM, AND QMHSNDM.                           */
/*                                                               */
/*****************************************************************/
#include <string.h>     /*strcpy, strncpy, strcmp               */
#include <stdio.h>
#include <qusec.h>      /*Error code structures                 */
#include <qusgen.h>     /*General user space structures         */
#include <quscrtus.h>   /*Linkage info, structures for QUSCRTUS  */
#include <quslspl.h>    /*Linkage info, structures for QUSLSPL   */
#include <qusptrus.h>   /*Linkage info, structures for QUSPTRUS  */
#include <qusrspla.h>   /*Linkage info, structures for QUSRSPLA  */
#include <qusdltus.h>   /*Linkage info, structures for QUSDLTUS  */
#include <qmhsndm.h>    /*Linkage info, structures for QMHSNDM   */
#include <qmhsndpm.h>   /*Linkage info, structures for QMHSNDPM  */


#pragma linkage(CLDLT,OS)
  void CLDLT (char file_name[10],
              char job_number[6],
              char usr_name[10],
              char job_name[10],
              char file_number[6],
              char form_type[10],
              char usr_data[10]);

  void error_check (void);

Qus_Generic_Header_0100_t *space;
char *list_section;
Qus_SPLF0100_t *entry_list;
Qus_SPLA0100_t *Rcv_Spl_Var;
/*****************************************************************/
/* PARMS FOR CLDLT                                               */
/*****************************************************************/
```

```
char job_nmbr[6];
char usr_nm[10];
char job_nm[10];
char sp_job_name[10];
char sp_spl_number[6];
char File_Number[] = "*LAST ";
/*************************************************************/
/* PARMS FOR QUSLSPL                                         */
/*************************************************************/
char frmt[8];
char usr[10];
char OutQ_Nm[20];
char ls_frm_typ[10];
char Usr_dat[10];
/*************************************************************/
/* PARMS FOR QUSRSPLA                                        */
/*************************************************************/
char Rcv_Var[724];
int  Rcv_lgth = 724;
char Rtv_Fmt[8];
char Qal_Jb_Nam[] = "*INT                      ";
char Splf_Name[] = "*INT      ";
int  Splf_Number = -1;
/*************************************************************/
/* PARMS FOR QUSCRTUS                                        */
/*************************************************************/
char spc_name[20];
char ext_atr[10];
int  initial_size;
char initial_value[1];
char auth[10];
char desc[50];
char replace[10];
/*************************************************************/
/* PARMS FOR QMHSNDPM AND QMHSNDM                            */
/*************************************************************/
char msg_id[7];
char msg_fl_name[20];
char msg_data[50];
int  msg_data_len;
char msg_type[10];
char pgm_queue[10];
int  pgm_stk_cnt;
char msg_key[4];
/*************************************************************/
/* PARMS FOR QMHSNDM                                         */
/*************************************************************/
int  msg_q_num;
char msg_queue[20];
char rpy_mq[10];
/*************************************************************/
/* MISCELLANEOUS VARIABLES                                   */
/*************************************************************/
char pack_dlt_count[15];
int  dlt_cnt;
int count;
char tmp_spl_number[7];
char dlt_date[7];
char spc_date[7];
int  api_code;
Qus_EC_t err_code;

/*************************************************************/
/* PROCEDURE TO CHECK THE ERRCODE RETURNED FROM CALLS TO APIs    */
/*************************************************************/
void error_check(void)
{
```

```
if (err_code.Bytes_Available != 0){
     strncpy(msg_id,"CPF9898",7);
     strncpy(msg_fl_name,"QCPFMSG   *LIBL     ",20);
     strncpy(msg_data,"An error has occurred calling ",29);
     switch (api_code){
        case  1  : strncat(msg_data,"QUSCRTUS.",9);
        case  2  : strncat(msg_data,"QUSLSPL. ",9);
        case  3  : strncat(msg_data,"QUSPTRUS.",9);
        case  4  : strncat(msg_data,"QUSRSPLA.",9);
        case  5  : strncat(msg_data,"QUSDLTUS.",9);
        case  6  : strncat(msg_data,"QMHSNDM. ",9);
        default  : strncat(msg_data,"UNKNOWN. ",9);
        }
     msg_data_len = 38;
     strncpy(msg_type,"*ESCAPE   ",10);
     strncpy(pgm_queue,"*         ",10);
     pgm_stk_cnt = 1;

     QMHSNDPM(msg_id,msg_fl_name,msg_data,msg_data_len,msg_type,
              pgm_queue,pgm_stk_cnt,msg_key,&err_code);
     }
}

/*****************************************************************/
/* START OF MAINLINE                                             */
/*****************************************************************/

main(argc,argv)
int argc;
char *argv[];
{

/*****************************************************************/
/*  Read in and assign the command-line arguments to respective  */
/*  variables                                                    */
/*****************************************************************/
strncpy(usr,argv[1],10);
strncpy(OutQ_Nm,argv[2],20);
strncpy(dlt_date,argv[3],7);

/*****************************************************************/
/*  Assign value to specific variables in the program            */
/*****************************************************************/
strcpy(spc_name,"DLTOLDSPLFQTEMP     ");
memset(ext_atr,' ',10);
initial_size = 1024;
strcpy(initial_value," ");
strcpy(auth,"*CHANGE   ");
memset(desc,' ',50);
strcpy(frmt,"SPLF0100");
strcpy(replace,"*YES      ");
strcpy(ls_frm_typ,"*ALL       ");
strcpy(Usr_dat,"*ALL      ");
strcpy(Rtv_Fmt,"SPLA0100");

/*****************************************************************/
/*  Call external program to create a user space                 */
/*****************************************************************/
err_code.Bytes_Provided = 0;
api_code = 1;
QUSCRTUS(spc_name,ext_atr,initial_size,initial_value,auth,desc,replace,
         &err_code);
/*****************************************************************/
/*  Call external program to list spooled files into user space  */
/*****************************************************************/
api_code = 2;
QUSLSPL(spc_name,frmt,usr,OutQ_Nm,ls_frm_typ,Usr_dat,&err_code);
```

```
/*****************************************************************/
/*  Call external program to get a pointer to the user space     */
/*  and get addressability to the list data section.             */
/*****************************************************************/
api_code = 3;
QUSPTRUS(spc_name,&space,&err_code);
list_section = (char *)space;
list_section = list_section + space->Offset_List_Data;
entry_list = (Qus_SPLF0100_t *) list_section;
dlt_cnt = 0;
count = 1;

/*****************************************************************/
/*  Loop through the entry list and delete old spooled files     */
/*****************************************************************/
while (count <= space->Number_List_Entries) {
/*****************************************************************/
/*  Call external program to retrieve more spool information     */
/*****************************************************************/
 api_code = 4;
 QUSRSPLA(Rcv_Var,Rcv_lgth,Rtv_Fmt,Qal_Jb_Nam,
          entry_list->Int_Job_ID,entry_list->Int_Splf_ID,
          Splf_Name,Splf_Number,&err_code);
 Rcv_Spl_Var = (Qus_SPLA0100_t *)Rcv_Var;
 strncpy(spc_date,Rcv_Spl_Var->Date_File_Open,7);
/*****************************************************************/
/*  If spooled file is too old delete it                         */
/*****************************************************************/
 if (strncmp(spc_date,dlt_date,7) <= 0 ) {
   strncpy(job_nm,Rcv_Spl_Var->Job_Name,10);
   strncpy(job_nmbr,Rcv_Spl_Var->Job_Number,6);
   strncpy(usr_nm,Rcv_Spl_Var->Usr_Name,10);
   strncpy(sp_job_name,Rcv_Spl_Var->Splf_Name,10);
/*****************************************************************/
/*  Convert the spooled file number to character.                */
/*****************************************************************/
   memcpy (sp_spl_number,"      ",6);
   sprintf(tmp_spl_number,"%d",Rcv_Spl_Var->Splf_Number);
   memcpy(sp_spl_number,tmp_spl_number,strlen(tmp_spl_number));
/*****************************************************************/
/*  Delete the spooled file.                                     */
/*****************************************************************/
   CLDLT(sp_job_name,job_nmbr,usr_nm,
         job_nm,sp_spl_number,ls_frm_typ,Usr_dat);
   dlt_cnt++;
  } /*IF*/
 strcpy(spc_date," ");
 count++;
 entry_list++;
} /*WHILE*/
/*****************************************************************/
/*  Remove the user space                                        */
/*****************************************************************/
api_code = 5;
QUSDLTUS(spc_name, &err_code);

/*****************************************************************/
/*  Send final message to user indicating number of spooled files*/
/*  deleted.                                                     */
/*****************************************************************/
api_code = 6;
strncpy(msg_id,"       ",7);
strncpy(msg_fl_name,"                    ",20);
sprintf(msg_data,"Number of spooled files deleted:  %d", dlt_cnt);
msg_data_len = strlen(msg_data);
strncpy(msg_type,"*INFO     ",10);
strncpy(msg_queue,"*REQUESTER          ",20);
```

```
msg_q_num = 1;
strncpy(rpy_mq,"          ",10);
QMHSNDM(msg_id,msg_fl_name,msg_data,msg_data_len,msg_type,
        msg_queue,msg_q_num,rpy_mq,msg_key, &err_code);

}
```

To create an ILE C program, specify the following:

```
CRTBNDC PGM(QGPL/DLTOLDSPLF) SRCFILE(QGPL/QCSRC)
```

CL Delete (CLDLT) Program

The DLTOLDSPLF program, written in OPM RPG, OPM COBOL, or ILE C, calls a CL program named CLDLT. The CLDLT program deletes the spooled files and the user space. The following is the CL source for the CLDLT program.

```
/*********************************************************************/
/*                                                                   */
/*  PROGRAM:  CLDLT                                                  */
/*                                                                   */
/*  LANGUAGE:  CL                                                    */
/*                                                                   */
/*  DESCRIPTION:  THIS PROGRAM WILL DELETE A SPECIFIC SPOOLED FILE   */
/*                USING THE DLTSPLF COMMAND AND SEND A MESSAGE WHEN  */
/*                THE FILE IS DELETED.                               */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
/*                                                                   */
PGM (&FILNAM &JOBNUM &USRNAM &JOBNAM &FILNUM &FRMTYP &USRDTA)
/*                                                                   */
/* ***************************************************************** */
/*                                                                   */
/*  DECLARE SECTION                                                  */
/*                                                                   */
/*********************************************************************/
/*                                                                   */
 DCL &FILNAM *CHAR 10
 DCL &JOBNUM *CHAR 6
 DCL &USRNAM *CHAR 10
 DCL &JOBNAM *CHAR 10
 DCL &FILNUM *CHAR 6
 DCL &FRMTYP *CHAR 10
 DCL &USRDTA *CHAR 10
 MONMSG CPF0000
/*                                                                   */
/*********************************************************************/
/*                                                                   */
/* EXECUTABLE CODE                                                   */
/*                                                                   */
/*********************************************************************/
/*                                                                   */

          DLTSPLF    FILE(&FILNAM)                              +
                     JOB(&JOBNUM/&USRNAM/&JOBNAM)               +
                     SPLNBR(&FILNUM)                            +
                     SELECT(&USRNAM *ALL &FRMTYP &USRDTA)
          SNDPGMMSG  MSG('Spooled file ' *CAT &FILNAM *CAT      +
                          ' number ' *CAT &FILNUM *CAT ' job '  +
                          *CAT &JOBNUM *CAT  '/'                +
                          *CAT &USRNAM *CAT '/' *CAT &JOBNAM *CAT +
                          ' deleted.')                          +
                     TOUSR(*REQUESTER)
ENDPGM
```

To create the CL program, specify the following:

```
CRTCLPGM PGM(QGPL/CLDLT) SRCFILE(QGPL/QCLSRC)
```

Changing an Active Job

This command interface to the Change Active Jobs (CHGACTJOB) program can reduce the run priority of active jobs with the same name. You can also reduce the run priority of jobs using a specified user name. You may:

- Specify a job name or the *ALL value.
- Specify the user name as the *ALL value.
- Use the default run priority of 99.

The CHGACTJOB command ensures that one of the following is true:

- Not all jobs were specified.
- The *ALL value was not specified for the JOB parameter.
- The *ALL value was not specified for the USER parameter.

This example uses the following APIs:

- Create User Space (QUSCRTUS)
- List Job (QUSLJOB)
- Retrieve User Space (QUSRTVUS)
- Retrieve Job Information (QUSRJOBI)

The following is the message description needed for the Change Active Jobs (CHGACTJOB) command:

```
ADDMSGD MSGID(USR3C01) MSGF(QCPFMSG) +
MSG('JOB(*ALL) is not valid with USER(*ALL)') SEV(30)
```

The following is the command definition for the CHGACTJOB command:

```
    CMD         PROMPT('Change Active Jobs')
                /* CPP CHGACTJOB */
    PARM        KWD(JOB) TYPE(*NAME) LEN(10) +
                  SPCVAL((*ALL)) MIN(1) +
                  PROMPT('Job name:')
    PARM        KWD(USER) TYPE(*NAME) LEN(10) DFT(*ALL) +
                  SPCVAL((*ALL) (*CURRENT)) PROMPT('User +
                  name:')
    PARM        KWD(RUNPTY) TYPE(*DEC) LEN(5 0) DFT(99) +
                  RANGE(00 99) PROMPT('Run priority:')
    DEP         CTL(&USER *EQ *ALL) PARM((&JOB *NE *ALL)) +
                  NBRTRUE(*EQ 1) MSGID(USR3C01)
```

To create the command, specify the following:

```
CRTCMD CMD(QGPL/CHGACTJOB) PGM(QGPL/CHGACTJOB) +
  SRCFILE(QGPL/CMDSRC)
```

The following is the command-processing program that is written in CL to list the active jobs and reduce the run priority if necessary:

```
 /* ************************************************************ */
 /* PROGRAM:  CHGACTJOB                                          */
```

```
/*                                                                   */
/* LANGUAGE:  CL                                                     */
/*                                                                   */
/* DESCRIPTION:  THIS PROGRAM WILL REDUCE THE RUN PRIORITY OF ACTIVE */
/*               JOBS WITH THE SAME NAME.                            */
/*                                                                   */
/* APIs USED:  QUSCRTUS, QUSLJOB, QUSRTVUS, QUSRJOBI                 */
/*                                                                   */
/* ***************************************************************** */
            PGM        PARM(&JOB &USER &RUNPTY)

/*                                                                   */
/* Input parameters                                                  */
/*                                                                   */

            DCL        VAR(&JOB) TYPE(*CHAR) LEN(10) +
                         /* Input job name */
            DCL        VAR(&USER) TYPE(*CHAR) LEN(10) +
                         /* Input user name */
            DCL        VAR(&RUNPTY) TYPE(*DEC) LEN(5 0) +
                         /* Input run priority */

/*                                                                   */
/* Local variables                                                   */
/*                                                                   */

            DCL        VAR(&RJOB) TYPE(*CHAR) LEN(10) +
                         /* Retrieve job name */
            DCL        VAR(&RUSER) TYPE(*CHAR) LEN(10) +
                         /* Retrieve user name */
            DCL        VAR(&RNBR) TYPE(*CHAR) LEN(6) +
                         /* Retrieve job number */
            DCL        VAR(&RUNPTYC) TYPE(*CHAR) LEN(5) +
                         /* Input run priority in character form */
            DCL        VAR(&RUNPTY8) TYPE(*DEC) LEN(8 0) +
                         /* Retrieve run priority after convert from +
                            binary 4 */
            DCL        VAR(&RUNPTY5) TYPE(*DEC) LEN(5 0) +
                         /* Retrieve run priority in decimal 5,0 +
                            form */
            DCL        VAR(&RUNPTY5C) TYPE(*CHAR) LEN(5) +
                         /* Retrieve run priority in character form */
            DCL        VAR(&RUNPTY4) TYPE(*CHAR) LEN(4) +
                         /* Retrieve run priority in binary 4 form */
            DCL        VAR(&NUMBER) TYPE(*CHAR) LEN(6) +
                         /* Current job number */
            DCL        VAR(&USRSPC) TYPE(*CHAR) LEN(20) +
                         VALUE('CHGA      QTEMP     ') +
                         /* User space name for APIs */
            DCL        VAR(&EUSRSPC) TYPE(*CHAR) LEN(10) +
                         /* User space name for commands */
            DCL        VAR(&JOBNAME) TYPE(*CHAR) LEN(26) +
                         VALUE('                    *ALL  ') +
                         /* Full job name for list job  */
            DCL        VAR(&BIN4) TYPE(*CHAR) LEN(4) +
                         /* Number of jobs for list job and +
                            User space offset in binary 4 form */
            DCL        VAR(&LOOP) TYPE(*DEC) LEN(8 0) +
                         /* Number of jobs from list job */
            DCL        VAR(&DEC8) TYPE(*DEC) LEN(8 0) +
                         /* User space offset in decimal 8,0 form */
            DCL        VAR(&ELEN) TYPE(*DEC) LEN(8 0) +
                         /* List job entry length in decimal 8,0 +
                            form */
            DCL        VAR(&ELENB) TYPE(*CHAR) LEN(4) +
                         /* List job entry length in binary 4 +
                            form */
```

```
            DCL        VAR(&LJOBE) TYPE(*CHAR) LEN(52) +
                         /* Retrieve area for list job entry */
            DCL        VAR(&INTJOB) TYPE(*CHAR) LEN(16) +
                         /* Retrieve area for internal job id */
            DCL        VAR(&JOBI) TYPE(*CHAR) LEN(104) +
                         /* Retrieve area for job information */
            DCL        VAR(&JOBTYPE) TYPE(*CHAR) LEN(1) +
                         /* Job type */

/*                                                                  */
/* Start of executable code                                         */
/*                                                                  */

/*                                                                  */
/* Retrieve job number to use for local user space name             */
/*                                                                  */

            RTVJOBA    NBR(&NUMBER)
            CHGVAR     VAR(%SST(&USRSPC 5 6)) VALUE(&NUMBER)
            CHGVAR     VAR(&EUSRSPC) VALUE(%SST(&USRSPC 1 10))

/*                                                                  */
/* Delete user space if it already exists                           */
/*                                                                  */

            DLTUSRSPC  USRSPC(QTEMP/&EUSRSPC)
            MONMSG CPF0000

/*                                                                  */
/* Create user space                                                */
/*                                                                  */

            CALL QUSCRTUS (&USRSPC 'CHGACTJOB ' X'00000100' ' ' +
                           '*ALL      ' +
                           'CHGACTJOB TEMPORARY USER SPACE-
                  ')

/*                                                                  */
/* Set up job name for list jobs                                    */
/*                                                                  */

            CHGVAR     VAR(%SST(&JOBNAME 1 10)) VALUE(&JOB)
            CHGVAR     VAR(%SST(&JOBNAME 11 10)) VALUE(&USER)

/*                                                                  */
/* List active jobs with job name specified                         */
/*                                                                  */

            CALL QUSLJOB  (&USRSPC 'JOBL0100' &JOBNAME +
                           '*ACTIVE   ')

/*                                                                  */
/* Retrieve number of entries returned. Convert to decimal and      */
/* if zero go to NOJOBS label to send out 'No jobs' message.        */
/*                                                                  */

            CALL QUSRTVUS (&USRSPC X'00000085' X'00000004' +
                           &BIN4)
            CHGVAR     &LOOP      %BINARY(&BIN4)
            IF         COND(&LOOP = 0) THEN(GOTO CMDLBL(NOJOBS))

/*                                                                  */
/* Retrieve list entry length, convert to decimal.                  */
/* Retrieve list entry offset, convert to decimal, and add one      */
/* to set the position.                                             */
/*                                                                  */
```

```
            CALL QUSRTVUS (&USRSPC X'00000089' X'00000004' +
                           &ELENB)
            CHGVAR     &ELEN      %BINARY(&ELENB)
            CALL QUSRTVUS (&USRSPC X'0000007D' X'00000004' +
                           &BIN4)
            CHGVAR     &DEC8      %BINARY(&BIN4)
            CHGVAR     VAR(&DEC8) VALUE(&DEC8 + 1)

/*                                                                  */
/* Loop for the number of jobs until no more jobs then go to        */
/* ALLDONE label                                                    */
/*                                                                  */

STARTLOOP:  IF (&LOOP = 0) THEN(GOTO ALLDONE)

/*                                                                  */
/* Convert decimal position to binary 4 and retrieve list job entry */
/*                                                                  */

            CHGVAR     %BINARY(&BIN4)   &DEC8
            CALL QUSRTVUS (&USRSPC &BIN4 &ELENB +
                           &LJOBE)

/*                                                                  */
/* Copy internal job identifier and retrieve job information for    */
/* basic performance information.                                   */
/*                                                                  */

            CHGVAR     VAR(&INTJOB) VALUE(%SST(&LJOBE 27 16))
            CALL QUSRJOBI (&JOBI X'00000068' 'JOBI0100' +
                           '*INT                      ' +
                           &INTJOB)

/*                                                                  */
/* Copy job type and if subsystem monitor, spool reader, system job, */
/* spool writer, or SCPF system job then loop to next job           */
/*                                                                  */

            CHGVAR     VAR(&JOBTYPE) VALUE(%SST(&JOBI 61 1))
            IF         COND((&JOBTYPE = 'M') *OR (&JOBTYPE = 'R')  +
                        *OR (&JOBTYPE = 'S') *OR (&JOBTYPE = 'W')  +
                        *OR (&JOBTYPE = 'X')) +
                         THEN(GOTO CMDLBL(ENDLOOP))

/*                                                                  */
/* Copy run priority, convert to decimal, convert to decimal 5,0,   */
/* and if request run priority is less than or equal to the current */
/* run priority then loop to next job.                              */
/*                                                                  */

            CHGVAR     VAR(&RUNPTY4) VALUE(%SST(&JOBI 65 4))
            CHGVAR     &RUNPTY8      %BINARY(&RUNPTY4)
            CHGVAR     VAR(&RUNPTY5) VALUE(&RUNPTY8)
            IF         COND(&RUNPTY5 *GE &RUNPTY) THEN(GOTO +
                         CMDLBL(ENDLOOP))

/*                                                                  */
/* Retrieve job name, convert to run priority to character,  change */
/* the job run priority and seen message stating the run priority   */
/* was changed.                                                     */
/*                                                                  */

            CHGVAR     VAR(&RJOB) VALUE(%SST(&JOBI 9 10))
            CHGVAR     VAR(&RUSER) VALUE(%SST(&JOBI 19 10))
            CHGVAR     VAR(&RNBR) VALUE(%SST(&JOBI 29 6))
            CHGVAR     VAR(&RUNPTYC) VALUE(&RUNPTY)
            CHGVAR     VAR(&RUNPTY5C) VALUE(&RUNPTY5)
```

```
            CHGJOB     JOB(&RNBR/&RUSER/&RJOB) RUNPTY(&RUNPTYC)
            MONMSG     MSGID(CPF1343) EXEC(GOTO CMDLBL(ENDLOOP))
            SNDPGMMSG  MSG('Job' *BCAT &RNBR *TCAT '/' *TCAT +
                         &RUSER *TCAT '/' *TCAT &RJOB *BCAT 'run +
                         priority was change from' *BCAT &RUNPTY5C +
                         *BCAT 'to' *BCAT &RUNPTYC *TCAT '.')

 /*                                                                 */
 /* At end of loop set new decimal position to next entry and       */
 /* decrement loop counter by one.                                  */
 /*                                                                 */

 ENDLOOP:    CHGVAR     VAR(&DEC8) VALUE(&DEC8 + &ELEN)
             CHGVAR     VAR(&LOOP) VALUE(&LOOP - 1)
             GOTO       CMDLBL(STARTLOOP)

 /*                                                                 */
 /* Send message that no jobs were found.                           */
 /*                                                                 */

 NOJOBS:     SNDPGMMSG  MSG('No jobs found.')

 /*                                                                 */
 /* All done. Now delete temporary user space that we created.      */
 /*                                                                 */

 ALLDONE:    DLTUSRSPC  USRSPC(QTEMP/&EUSRSPC)
             MONMSG CPF0000
             ENDPGM
```

The program can be changed to change the run priority by removing the IF statement to compare the current and requested run priority.

To create the CL program, specify the following:

```
CRTCLPGM PGM(QGPL/CHGACTJOB) SRCFILE(QGPL/QCLSRC)
```

You can change the command to:

- Specify a different printer device.
- Specify a different output queue.
- Specify different job attributes that the Change Job (CHGJOB) command can change.
- List only jobs on an output queue and remove the spooled files.
- Provide a menu to select jobs to be changed.

Changing a Job Schedule Entry

This command interface to the Change Job Schedule Entry User (CHGSCDEUSR) program can change the USER parameter in the job schedule entry. You may:

- Specify a job schedule entry name
- Specify a generic job schedule entry name
- Specify the *ALL value

This example uses the following APIs:

- Create User Space (QUSCRTUS)

- List Job Schedule Entries (QWCLSCDE)
- Retrieve User Space (QUSRTVUS)

The following is the command definition for the CHGSCDEUSR command:

```
    CMD         PROMPT('Change Job Schedule Entry User')
                /* CPP CHGSCDEUSR */
    PARM        KWD(JOB) TYPE(*GENERIC) LEN(10) +
                  SPCVAL((*ALL)) +
                  MIN(1) PROMPT('Job name:')
    PARM        KWD(OLDUSER) TYPE(*NAME) LEN(10) +
                  MIN(1) PROMPT('Old user name:')
    PARM        KWD(NEWUSER) TYPE(*NAME) LEN(10) +
                  MIN(1) PROMPT('New user name:')
```

To create the command, specify the following:

```
CRTCMD CMD(QGPL/CHGSCDEUSR) PGM(QGPL/CHGSCDEUSR) +
  SRCFILE(QGPL/QCMDSRC)
```

The following is the command-processing program that is written in CL to list the job schedule entries and change the user if necessary:

```
 /* ************************************************************ */
 /* PROGRAM:  CHGSCDEUSR                                         */
 /*                                                              */
 /* LANGUAGE:  CL                                                */
 /*                                                              */
 /* DESCRIPTION:  THIS PROGRAM WILL CHANGE THE USER FOR A LIST OF    */
 /*               JOB SCHEDULE ENTRIES.                          */
 /*                                                              */
 /* APIs USED:  QUSCRTUS, QWCLSCDE, QUSRTVUS                     */
 /*                                                              */
 /* ************************************************************ */
             PGM         PARM(&JOBNAME &OLDUSER &NEWUSER)

 /*                                                              */
 /* Input parameters are as follows:                             */
 /*                                                              */

             DCL         VAR(&JOBNAME) TYPE(*CHAR) LEN(10) /* Input +
                           job name */
             DCL         VAR(&OLDUSER) TYPE(*CHAR) LEN(10) /* Input +
                           old user name */
             DCL         VAR(&NEWUSER) TYPE(*CHAR) LEN(10) /* Input +
                           new user name */

 /*                                                              */
 /* Local variables are as follows:                              */
 /*                                                              */

             DCL         VAR(&USRSPC) TYPE(*CHAR) LEN(20) +
                           VALUE('CHGSCDEUSRQTEMP     ') /* User +
                           space name for APIs */
             DCL         VAR(&CNTHDL) TYPE(*CHAR) LEN(16) +
                           VALUE('                ') /* Continuation +
                           handle */
             DCL         VAR(&NUMENTB) TYPE(*CHAR) LEN(4) /* Number +
                           of entries from list job schedule entries +
                           in binary form */
             DCL         VAR(&NUMENT) TYPE(*DEC) LEN(8 0) /* Number +
                           of entries from list job schedule entries +
                           in decimal form */
             DCL         VAR(&HDROFFB) TYPE(*CHAR) LEN(4) /* Offset +
                           to the header portion of the user space in +
```

```
                  binary form */
            DCL       VAR(&HDRLENB) TYPE(*CHAR) LEN(4) /* Length +
                  to the header portion of the user space in +
                  binary form */
            DCL       VAR(&GENHDR) TYPE(*CHAR) LEN(140) /* Generic +
                  header information from the user space */
            DCL       VAR(&HDRINFO) TYPE(*CHAR) LEN(26) /* Header +
                  information from the user space */
            DCL       VAR(&LSTSTS) TYPE(*CHAR) LEN(1) /* Status +
                  of the list in the user space */
            DCL       VAR(&OFFSETB) TYPE(*CHAR) LEN(4) /* Offset +
                  to the list portion of the user space in +
                  binary form */
            DCL       VAR(&STRPOSB) TYPE(*CHAR) LEN(4) /* Starting +
                  position in the user space  in binary form */
            DCL       VAR(&ELENB) TYPE(*CHAR) LEN(4) /* List job +
                  entry length in binary 4 form */
            DCL       VAR(&LENTRY) TYPE(*CHAR) LEN(1156) /* +
                  Retrieve area for list job schedule entry */
            DCL       VAR(&INFOSTS) TYPE(*CHAR) LEN(1) /* Retrieve +
                  area for information status */
            DCL       VAR(&JOBNAM) TYPE(*CHAR) LEN(10) /* Retrieve +
                  area for job name */
            DCL       VAR(&ENTRY#) TYPE(*CHAR) LEN(6) /* Retrieve +
                  area for entry number  */
            DCL       VAR(&USERNM) TYPE(*CHAR) LEN(10) /* Retrieve +
                  area for user name */

/*                                                                  */
/* Start of code                                                    */
/*                                                                  */

/*                                                                  */
/* You may want to monitor for additional messages here.            */
/*                                                                  */

/*                                                                  */
/* This creates the user space.  The user space will be 256 bytes   */
/* and will be initialized to blanks.                               */
/*                                                                  */

            CALL      PGM(QUSCRTUS) PARM(&USRSPC 'CHGSCDEUSR' +
                  X'00000100' ' ' '*ALL      ' 'CHGSCDEUSR +
                  TEMPORARY USER SPACE                    ')
            MONMSG    MSGID(CPF3C00) EXEC(GOTO CMDLBL(ERROR))

/*                                                                  */
/* This lists job schedule entries of the name specified.           */
/*                                                                  */

PARTLIST:   CALL      PGM(QWCLSCDE) PARM(&USRSPC 'SCDL0200' +
                  &JOBNAME &CNTHDL 0)

/*                                                                  */
/* Retrieve the generic header from the user space.                 */
/*                                                                  */

            CALL      PGM(QUSRTVUS) PARM(&USRSPC X'00000001' +
                  X'0000008C' &GENHDR)
            MONMSG    MSGID(CPF3C00) EXEC(GOTO CMDLBL(ERROR))
/*                                                                  */
/* Get the information status for the list from the generic header. */
/* If it is incomplete, go to BADLIST label and send out 'Bad list' */
/* message.                                                         */
/*                                                                  */

            CHGVAR    VAR(&LSTSTS) VALUE(%SST(&GENHDR 104 1))
```

```
            IF          COND(&LSTSTS = 'I') THEN(GOTO CMDLBL(BADLIST))

/*                                                                  */
/* Get the number of entries returned. Convert to decimal and       */
/* if zero go to NOENTS label to send out 'No entries' message.     */
/*                                                                  */

            CHGVAR      VAR(&NUMENTB) VALUE(%SST(&GENHDR 133 4))
            CHGVAR      VAR(&NUMENT) VALUE(%BIN(&NUMENTB))
            IF          COND(&NUMENT = 0) THEN(GOTO CMDLBL(NOENTS))

/*                                                                  */
/* Get the list entry length and the list entry offset.             */
/* These values are used to set up the starting position.           */
/*                                                                  */

            CHGVAR      VAR(&ELENB) VALUE(%SST(&GENHDR 137 4))
            CHGVAR      VAR(&OFFSETB) VALUE(%SST(&GENHDR 125 4))
            CHGVAR      VAR(%BIN(&STRPOSB)) VALUE(%BIN(&OFFSETB) + 1)

/*                                                                  */
/* This loops for the number of entries until no more entries are   */
/* found and goes to the ALLDONE label.                             */
/*                                                                  */

STARTLOOP:  IF          COND(&NUMENT = 0) THEN(GOTO CMDLBL(PARTCHK))

/*                                                                  */
/* This retrieves the list entry.                                   */
/*                                                                  */
            CALL        PGM(QUSRTVUS) PARM(&USRSPC &STRPOSB &ELENB +
                          &LENTRY)
            MONMSG      MSGID(CPF3C00) EXEC(GOTO CMDLBL(ERROR))
/*                                                                  */
/* This copies the information status, job name, entry number, and  */
/* user name.                                                       */
/*                                                                  */

            CHGVAR      VAR(&INFOSTS) VALUE(%SST(&LENTRY 1 1))
            CHGVAR      VAR(&JOBNAM)  VALUE(%SST(&LENTRY 2 10))
            CHGVAR      VAR(&ENTRY#)  VALUE(%SST(&LENTRY 12 10))
            CHGVAR      VAR(&USERNM)  VALUE(%SST(&LENTRY 547 10))

/*                                                                  */
/* This checks to make sure the list entry contains the user name.  */
/* If it does, the user name is compared to the old user name       */
/* passed in.  If either of these checks fails, this entry will     */
/* be skipped.                                                      */
/*                                                                  */

            IF          COND(&INFOSTS *NE ' ') THEN(GOTO +
                          CMDLBL(ENDLOOP))

            IF          COND(&USERNM *NE &OLDUSER) THEN(GOTO +
                          CMDLBL(ENDLOOP))

/*                                                                  */
/* This code will issue the CHGJOBSCDE command for the entry.       */
/*                                                                  */

            CHGJOBSCDE JOB(&JOBNAM) ENTRYNBR(&ENTRY#) USER(&NEWUSER)
            MONMSG      MSGID(CPF1620) EXEC(GOTO CMDLBL(NOCHG))
            SNDPGMMSG   MSG('Entry' *BCAT &JOBNAM *BCAT &ENTRY# +
                          *BCAT 'was changed.')
            GOTO        CMDLBL(ENDLOOP)
NOCHG:      SNDPGMMSG   MSG('Entry' *BCAT &JOBNAM *BCAT &ENTRY# +
                          *BCAT 'was NOT changed.')
```

```
/*                                                                   */
/* At end of loop, set new decimal position to the next entry and    */
/* decrement the loop counter by one.                                */
/*                                                                   */

ENDLOOP:    CHGVAR     VAR(%BIN(&STRPOSB)) VALUE(%BIN(&STRPOSB) +
                       + %BIN(&ELENB))
            CHGVAR     VAR(&NUMENT) VALUE(&NUMENT - 1)
            GOTO       CMDLBL(STARTLOOP)

/*                                                                   */
/* This sends a message that no entries were found.                  */
/*                                                                   */

NOENTS:     SNDPGMMSG  MSG('No entries found.')
            GOTO       CMDLBL(ALLDONE)

/*                                                                   */
/* This sends a message that the list was incomplete.                */
/*                                                                   */

BADLIST:    SNDPGMMSG  MSG('Incomplete list in the user space. +
                            See joblog for details.')
            GOTO       CMDLBL(ALLDONE)

/*                                                                   */
/* This sends a message that an unexpected error occurred.           */
/*                                                                   */

ERROR:      SNDPGMMSG  MSG('Unexpected error. +
                            See joblog for details.')
            GOTO       CMDLBL(ALLDONE)

/*                                                                   */
/* This will check for a partial list in the user space and          */
/* finish processing the rest of the list.                           */
/*                                                                   */

PARTCHK:    IF         COND(&LSTSTS = 'C') THEN(GOTO CMDLBL(ALLDONE))
/*                                                                   */
/* Retrieve the header information from the user space.              */
/* Use this information to get the rest of the list.                 */
/*                                                                   */

            CHGVAR     VAR(&HDROFFB) VALUE(%SST(&GENHDR 121 4))
            CHGVAR     VAR(&HDRLENB) VALUE(%SST(&GENHDR 117 4))
            CALL       PGM(QUSRTVUS) PARM(&USRSPC &HDROFFB +
                         &HDRLENB &HDRINFO)
            MONMSG     MSGID(CPF3C00) EXEC(GOTO CMDLBL(ERROR))
            CHGVAR     VAR(&CNTHDL) VALUE(%SST(&HDRINFO 11 16))
            GOTO       CMDLBL(PARTLIST)


/*                                                                   */
/* All done.  Now the temporary user space is deleted.               */
/*                                                                   */

ALLDONE:    DLTUSRSPC  USRSPC(QTEMP/%SST(&USRSPC 1 10))
            MONMSG     MSGID(CPF0000)
            ENDPGM
```

To create the CL program, specify the following:

```
CRTCLPGM PGM(QGPL/CHGSCDEUSR) SRCFILE(QGPL/QCLSRC)
```

You can change the command to:

- Specify different parameters that the Change Job Schedule Entry (CHGJOBSCDE) command can change.
- Provide a menu to select job schedule entries to be changed.

Creating Your Own Telephone Directory

To create a telephone directory, you must use the following $USIDXCRT ILE C program to create a user index, and you must use the $USIDXEX ILE C program to insert the entries into a telephone directory.

To set up the program to create a user index, specify the following:

```
/*****************************************************************/
/*  PROGRAM:  $USIDXCRT                                          */
/*                                                               */
/*  LANGUAGE:  ILE C for OS/400                                  */
/*                                                               */
/*  DESCRIPTION:  THIS PROGRAM CREATES A USER INDEX NAMED "TESTIDX" */
/*   IN THE LIBRARY "QGPL".                                      */
/*                                                               */
/*  APIs USED:  QUSCRTUI                                         */
/*                                                               */
/*****************************************************************/
#include <quscrtui.h>
main()
{
/*****************************************************************/
/*    Set up the parameters to be used in the call to 'QUSCRTUI'    */
/*****************************************************************/
char idx_name[]= "TESTIDX   QGPL      ";
char ext_atr[]= "TESTER    ";
char entry_lgth_att[] = "F";
int entry_lngth = 50;
char key_insert[] = "1";
int key_lngth = 15;
char imm_update[] = "0";
char optim[] = "0";
char auth[] = "*CHANGE   ";
char desc[] = "Description .....    ";
/*****************************************************************/
/*  Call the 'QUSCRTUI' program to create the user index.        */
/*****************************************************************/
  QUSCRTUI(idx_name,ext_atr,entry_lgth_att,entry_lngth,key_insert,
           key_lngth,imm_update,optim,auth,desc);
}
```

To compile the program that creates the user index, specify the following:

```
CRTBNDC PGM(QGPL/$USIDXCRT) SRCFILE(QGPL/QCSRC)
```

To insert entries into the user index, use the following ILE C program:

```
/*****************************************************************/
/*  PROGRAM:  $USIDXEX                                           */
/*                                                               */
/*  LANGUAGE:  ILE C for OS/400                                  */
/*                                                               */
/*  DESCRIPTION:  THIS PROGRAM USES A USER INDEX TO KEEP TRACK OF   */
/*    NAMES AND PHONE NUMBERS.  THERE ARE TWO OPERATIONS THAT ARE   */
/*    DEMONSTRATED IN THIS EXAMPLE.  THE FIRST IS THE INSERTION OF  */
/*    AN ENTRY INTO THE INDEX, AND SECONDLY THE FINDING OF A GIVEN  */
/*    INDEX ENTRY.                                               */
/*    THE INDEX IS KEYED ON THE LAST NAME, THEREFORE ENTER AS MUCH  */
/*    OF THE NAME AS YOU KNOW AND THE PROGRAM WILL LIST ALL ENTRIES */
```

```
/*    MATCHING YOUR STRING (IN ALPHABETICAL ORDER).                */
/*                                                                 */
/*  APIs USED:  NONE                                               */
/*                                                                 */
/*****************************************************************/
#include <stdio.h>
#include <string.h>
#include <miindex.h>
#include <miptrnam.h>
#include <stdlib.h>
#include <ctype.h>

_SYSPTR index;
_IIX_Opt_List_T  ins_option_list;
_IIX_Opt_List_T  *fnd_option_list;
char Name_And_Num[50];
char In_Name[50];
char Out_Num[5000];
char response[1];
char name[35];
char number[15];
int  Ent_Found,count,start,length_of_entry;


/*****************************************************************/
/* Procedure to copy 'cpylngth' elements of 'string2' into the   */
/* new string, 'string1';  starting at position 'strpos'.        */
/*****************************************************************/

void strncpyn(string1,string2,strpos,cpylngth)
 char string1[],string2[];
 int strpos,cpylngth;
{
 int x = 0;
 while (x < cpylngth)
  string1[x++]=string2[strpos++];
} /*strncpyn*/


/*****************************************************************/
/* Procedure to convert any string into uppercase, where applicable */
/*****************************************************************/

void convert_case(string1)
 char string1[];
{
 int x = 0;
 while (x < (strlen(string1))) {
  string1[x] = toupper(string1[x]);
  x++;
 } /*while*/
} /*convert_case*/


main()
{
fnd_option_list = malloc(sizeof(_IIX_Opt_List_T)
                         +99*sizeof(_IIX_Entry_T));

/*****************************************************************/
/*  Resolve to the index created in $USIDXCRT.                   */
/*****************************************************************/

  index = rslvsp(_Usridx,"TESTIDX","QGPL",_AUTH_ALL);

/*****************************************************************/
/*  Set up the insert option list                                */
```

```
/*****************************************************************/

  ins_option_list.Rule = _INSERT_REPLACE;
  ins_option_list.Arg_Length = 50;
  ins_option_list.Occ_Count = 1;
  ins_option_list.Entry[0].Entry_Length = 50;
  ins_option_list.Entry[0].Entry_Offset = 0;

/*****************************************************************/
/*  Set up the find option list                                  */
/*****************************************************************/

  fnd_option_list->Rule = _FIND_EQUALS;
  fnd_option_list->Occ_Count = 100;

/*****************************************************************/
/*  Loop until the choice 'Q' is entered at the menu             */
/*****************************************************************/

 while (1==1) {
  printf("\n\n**********************\n");
  printf("*   TELEPHONE INDEX   *\n");
  printf("**********************\n");
  printf("* 'A'  Add name & num *\n");
  printf("* 'L'  List a number  *\n");
  printf("* 'Q'  Quit index     *\n");
  printf("**********************\n");
  gets(response);
  if ((strncmp(response,"A",1)==0)||(strncmp(response,"a",1)==0))
   { printf("\nEnter name to add. ex(Last, First)\n");
     gets(name);
     convert_case(name);
     printf("\nEnter number to add. ex(999-9999)\n");
     gets(number);
     strcpy(name,strcat(name,"     "));
     strcpy(Name_And_Num,strcat(name,number));
     printf("\nName and number to add is => %s\n",Name_And_Num);
     insinxen(index,Name_And_Num,Integrated Netfinity Server_option_list);
   } /* if 'a'*/
  if ((strncmp(response,"L",1)==0)||(strncmp(response,"l",1)==0))
   {
     printf("\nEnter name to find. ex(Last, First)\n");
     gets(In_Name);
     convert_case(In_Name);
     fnd_option_list->Arg_Length = strlen(In_Name);
     fndinxen(Out_Num,index,fnd_option_list,In_Name);
     length_of_entry = fnd_option_list->Entry[0].Entry_Length;
     Ent_Found = fnd_option_list->Ret_Count;
     if (Ent_Found == 0)
       printf("\nName not found in index => %s\n",In_Name);
     else {
       if (Ent_Found > 1)  {
        printf("\n%d occurences found,\n",Ent_Found);
        count = 0;
        start = 0;
        while (count++ < Ent_Found) {
         printf("Name and number is => %s\n",Out_Num);
         start = start + length_of_entry;
         strncpyn(Out_Num,Out_Num,start,length_of_entry);
        } /* while */
       }else
        printf("\nName and number is => %s\n",Out_Num);
     }  /*else*/
   }  /*if 'l'*/
  if ((strncmp(response,"Q",1)==0)||(strncmp(response,"q",1)==0))
   { break; }
 }  /*while*/
```

```
} /*$USIDXEX*/
```

To create the ILE C program to insert entries into the user index, specify the following:

```
CRTBNDC PGM(QGPL/$USIDXEX) SRCFILE(QGPL/QCSRC)
```

Creating and Manipulating a User Index

The following example shows how to create and manipulate a user index with a call from an MI program. For another example using the QUSCRTUI API, see Creating Your Own Telephone Directory.

```
/*****************************************************************/
/*                                                               */
/* PROGRAM:  GLOBALV                                             */
/*                                                               */
/* LANGUAGE:  MI/IRP                                             */
/*                                                               */
/* DESCRIPTION:  MAINTAINS AN INDEPENDENT INDEX.  EACH INDEX ENTRY  */
/*               CONTAINS 100 BYTES OF USER DATA.  THE ENTRIES ARE  */
/*               KEYED TWO 10 BYTE VALUES:  THE USER PROFILE AND A  */
/*               VALUE IDENTIFIER.                               */
/*                                                               */
/* APIs USED:  QUSCRTUI                                          */
/*                                                               */
/* PARAMETERS:                                                   */
/*                                                               */
/*   PARM  TYPE       DESCRIPTION                                */
/*                                                               */
/*    1    CHAR(1)    FUNCTION:                                  */
/*                                                               */
/*                    'U':  UPDATE GLOBALV INFORMATION           */
/*                    'R':  RETRIEVE GLOBALV INFORMATION         */
/*                                                               */
/*    2    CHAR(10)   USER PROFILE                               */
/*                                                               */
/*                    THE NAME OF THE USER PROFILE FOR WHICH     */
/*                    INFORMATION IS TO BE SAVED OR RETRIEVED.   */
/*                                                               */
/*    3    CHAR(10)   VALUE ID                                   */
/*                                                               */
/*                    THE NAME OF THE GLOBALV VARIABLE ID FOR WHICH */
/*                    INFORMATION IS TO BE SAVED OR RETRIEVED.   */
/*                                                               */
/*    4    CHAR(100)  VALUE                                      */
/*                                                               */
/*                    IF FUNCTION IS 'U', THIS VALUE SHOULD CONTAIN */
/*                    THE NEW VALUE TO BE ASSOCIATED WITH THE    */
/*                    USER ID AND VALUE ID.                      */
/*                                                               */
/*                    IF FUNCTION IS 'R', THIS VARIABLE WILL BE  */
/*                    SET TO THE VALUE ASSOCIATED WITH THE USER ID  */
/*                    AND VALUE ID.  IF NO VALUE EXISTS, *NONE   */
/*                    IS SPECIFIED.                              */
/*                                                               */
/*****************************************************************/

ENTRY * (GLOBALV_PARM) EXT;

/*****************************************************************/
/* PARAMETER VALUE POINTERS FOR GLOBALV.                         */
/*****************************************************************/

DCL SPCPTR GV_REQUEST@ PARM;
DCL SPCPTR GV_USERID@ PARM;
```

```
DCL SPCPTR GV_VALUEID@ PARM;
DCL SPCPTR GV_VALUE@ PARM;

/*****************************************************************/
/* PARAMETER VALUES FOR GLOBALV.                                 */
/*****************************************************************/

DCL DD GV_REQUEST CHAR(1) BAS(GV_REQUEST@);
DCL DD GV_USERID CHAR(10) BAS(GV_USERID@);
DCL DD GV_VALUEID CHAR(10) BAS(GV_VALUEID@);
DCL DD GV_VALUE CHAR(100) BAS(GV_VALUE@);

/*****************************************************************/
/* PARAMETER LIST FOR GLOBALV.                                   */
/*****************************************************************/

DCL OL GLOBALV_PARM (GV_REQUEST@
                    ,GV_USERID@
                    ,GV_VALUEID@
                    ,GV_VALUE@
                    ) PARM EXT;

/*****************************************************************/
/* ARGUMENT VALUES FOR CREATE USER INDEX (QUSCRTUI) API.         */
/*****************************************************************/

DCL DD UI_NAME CHAR(20) INIT("GLOBALV   QGPL      ");
DCL DD UI_ATTR CHAR(10) INIT("          ");
DCL DD UI_EATR CHAR(1) INIT("F");
DCL DD UI_ELEN BIN(4) INIT(120);
DCL DD UI_KATR CHAR(1) INIT("1");
DCL DD UI_KLEN BIN(4) INIT(20);
DCL DD UI_IUPD CHAR(1) INIT("0");
DCL DD UI_OPT  CHAR(1) INIT("0");
DCL DD UI_AUT  CHAR(10) INIT("*CHANGE   ");
DCL DD UI_TEXT CHAR(50)
       INIT("GLOBALV INDEX                                    ");

/*****************************************************************/
/* POINTERS TO ARGUMENT VALUES FOR QUSCRTUI API.                 */
/*****************************************************************/

DCL SPCPTR UI_NAME@ INIT(UI_NAME);
DCL SPCPTR UI_ATTR@ INIT(UI_ATTR);
DCL SPCPTR UI_EATR@ INIT(UI_EATR);
DCL SPCPTR UI_ELEN@ INIT(UI_ELEN);
DCL SPCPTR UI_KATR@ INIT(UI_KATR);
DCL SPCPTR UI_KLEN@ INIT(UI_KLEN);
DCL SPCPTR UI_IUPD@ INIT(UI_IUPD);
DCL SPCPTR UI_OPT@  INIT(UI_OPT);
DCL SPCPTR UI_AUT@  INIT(UI_AUT);
DCL SPCPTR UI_TEXT@ INIT(UI_TEXT);

/*****************************************************************/
/* ARGUMENT LIST FOR QUSCRTUI API.                               */
/*****************************************************************/

DCL OL QUSCRTUI_ARG (UI_NAME@
                    ,UI_ATTR@
                    ,UI_EATR@
                    ,UI_ELEN@
                    ,UI_KATR@
                    ,UI_KLEN@
                    ,UI_IUPD@
                    ,UI_OPT@
                    ,UI_AUT@
                    ,UI_TEXT@
```

```
                    ) ARG;

/*****************************************************************/
/* SYTSEM POINTER TO QUSCRTUI API *PGM OBJECT.                   */
/*****************************************************************/

DCL SYSPTR QUSCRTUI INIT("QUSCRTUI",TYPE(PGM));

/*****************************************************************/
/* SYSTEM POINTER TO GLOBALV *USRIDX OBJECT.                     */
/*****************************************************************/

DCL SYSPTR INX@;

DCL DD INX_OBJECTID CHAR(34);
DCL DD INX_OBJECTID_TYPE CHAR(2) DEF(INX_OBJECTID) POS(1)
       INIT(X'0E0A');
DCL DD INX_OBJECTID_NAME CHAR(30) DEF(INX_OBJECTID) POS(3)
       INIT('GLOBALV                        ');
DCL DD INX_OBJECTID_AUT CHAR(2) DEF(INX_OBJECTID) POS(33)
       INIT(X'0000');

/*****************************************************************/
/* EXCEPTION MONITOR TO DETECT 2201X EXCEPTIONS (OBJECT NOT FOUND)  */
/*****************************************************************/

DCL EXCM EXCM_NOOBJECT EXCID(H"2201") INT(CREATE_INDEX) IMD;

/*****************************************************************/
/* PASA INVOCATION ENTRY FOR RETURN FROM EXCEPTION.              */
/*****************************************************************/

DCL DD RTN_NOOBJECT CHAR(18) BDRY(16);
DCL SPCPTR RTN_NOOBJECT@ INIT(RTN_NOOBJECT);
DCL DD RTN_NOOBJECT_ADDR CHAR(16) DEF(RTN_NOOBJECT);
DCL DD RTN_NOOBJECT_OPT CHAR(1) DEF(RTN_NOOBJECT) POS(18)
       INIT(X'00');

/*****************************************************************/
/* RECEIVER VARIABLE FOR INDEPENDENT INDEX OPERATIONS.           */
/*****************************************************************/

DCL DD INX_RECEIVER CHAR(120);
DCL SPCPTR INX_RECEIVER@ INIT(INX_RECEIVER);

/*****************************************************************/
/* OPTION TEMPLATE FOR INDEPENDENT INDEX OPERATIONS.             */
/*****************************************************************/

DCL DD INX_OPT CHAR(14);
DCL SPCPTR INX_OPT@ INIT(INX_OPT);
DCL SPC INX_OPT_SPC BAS(INX_OPT@);
DCL DD INX_OPT_RULE CHAR(2) DIR;
DCL DD INX_OPT_ARGL BIN(2) DIR;
DCL DD INX_OPT_ARGO BIN(2) DIR;
DCL DD INX_OPT_OCCC BIN(2) DIR;
DCL DD INX_OPT_RTNC BIN(2) DIR;
DCL DD INX_OPT_ELEN BIN(2) DIR;
DCL DD INX_OPT_EOFF BIN(2) DIR;

/*****************************************************************/
/* ARGUMENT VARIABLE FOR INDEPENDENT INDEX OPERATIONS.           */
/*****************************************************************/

DCL DD INX_ARG CHAR(120);
DCL SPCPTR INX_ARG@ INIT(INX_ARG);
```

```
/*****************************************************************/
/* START OF CODE                                                 */
/*****************************************************************/

  MATINVE RTN_NOOBJECT_ADDR,*,X'03'; /* MATERIALIZE THIS PROGRAM'S  */
                                     /* INVOCATION ENTRY IN THE     */
                                     /* PASA.  THIS ENTRY IS USED   */
                                     /* WHEN RETURNING FROM THE     */
                                     /* EXCEPTION HANDLER BELOW.    */

  RSLVSP INX@,INX_OBJECTID,*,*; /* RESOLVE TO "GLOBALV" USER INDEX  */
                                /* OBJECT.  IF THE OBJECT DOES NOT  */
                                /* EXIST, THEN THE X'2201' EXCEPTION*/
                                /* IS RETURNED, CAUSING THE "OBJECT */
                                /* NOT FOUND" EXCEPTION HANDLER AT  */
                                /* THE END OF THE PROGRAM TO RUN.   */



  CMPBLA(B) GV_REQUEST,'U'/NEQ(NOT_UPDATE); /* IF GV_REQUEST ¬= U   */
                                            /* BRANCH TO NOT_UPDATE */

    /* SET UP OPTIONS FOR INSERT INDEPENDENT INDEX ENTRY (INSINXEN) */
    /* OPERATION.                                                  */

    CPYBLA INX_OPT_RULE,X'0002';     /* RULE= INSERT.              */
    CPYNV  INX_OPT_OCCC,1;           /* OCCURRENCE COUNT = 1.      */
    CPYBLA INX_ARG(1:10),GV_USERID;  /* SPECIFY INDEX ENTRY.       */
    CPYBLA INX_ARG(11:10),GV_VALUEID;
    CPYBLA INX_ARG(21:100),GV_VALUE;
    INSINXEN INX@,INX_ARG@,INX_OPT@; /* INSERT THE INDEX ENTRY.    */
    RTX *;                           /* RETURN                     */

NOT_UPDATE:

  CMPBLA(B) GV_REQUEST,'R'/NEQ(NOT_RETRIEVE); /* IF GV_REQUEST ¬= R */
                                              /* GOTO NOT_RETRIEVE. */

    /* SET UP OPTIONS FOR FIND INDEPENDENT INDEX ENTRY (FNDINXEN)   */
    /* OPERATION.                                                   */

    CPYBLA INX_OPT_RULE,X'0001';    /* RULE= FIND WITH EQUAL KEY.   */
    CPYNV  INX_OPT_ARGL,20;         /* ARGUMENT LENGTH= 20.         */
    CPYNV  INX_OPT_OCCC,1;          /* OCCURRENCE COUNT=1.          */
    CPYBLA INX_ARG(1:10),GV_USERID; /* SPECIFY SEARCH ARGUMENT.     */
    CPYBLA INX_ARG(11:10),GV_VALUEID;
    FNDINXEN INX_RECEIVER@,INX@,INX_OPT@,INX_ARG@; /* FIND ENTRY.   */
    CMPNV(B) INX_OPT_RTNC,1/EQ(FOUND_ENTRY); /* IF RETURN_COUNT = 1 */
                                             /* GOTO FOUND_ENTRY.   */
      CPYBLAP GV_VALUE,'*NONE',' '; /* ENTRY WAS NOT FOUND, SPECIFY */
                                    /* VALUE OF *NONE.              */
      RTX *;                        /* RETURN                       */

FOUND_ENTRY:

      CPYBLA GV_VALUE,INX_RECEIVER(21:100); /* ENTRY WAS FOUND,     */
                                            /* COPY VALUE TO USER   */
                                            /* PARAMETER.           */
      RTX *;                                /* RETURN               */

NOT_RETRIEVE:

  RTX *; /* UNKNOWN FUNCTION CODE.  RETURN.                        */

/*****************************************************************/
/* "OBJECT NOT FOUND" EXCEPTION HANDLER.                         */
/*****************************************************************/
```

```
ENTRY CREATE_INDEX INT;

  MODEXCPD EXCM_NOOBJECT,X'0000',X'01'; /* TURN OFF EXCEPTION       */
                                        /* MONITOR.                */

  CALLX QUSCRTUI,QUSCRTUI_ARG,*; /* USE QUSCRTUI API TO CREATE THE  */
                                 /* USER INDEX OBJECT.              */

  RTNEXCP RTN_NOOBJECT@; /* RETURN FROM THE EXCEPTION HANDLER AND   */
                         /* RETRY THE OPERATION.                    */
PEND;
```

Creating a Batch Machine

These ILE C programs emulate a batch machine. One program, $USQEXSRV, acts as a server and takes the entries off a user queue and then runs the request through the Execute Command (QCMDEXC) API. The other program, $USQEXREQ, acts as a requester and puts the entries into a user space. The APIs used in this example are:

- Create User Queue (QUSCRTUQ)
- Execute Command (QCMDEXC)

Requester Program ($USQEXREQ)

The following is a requester program using ILE C:

```
/*****************************************************************/
/*  PROGRAM:  $USQEXREQ                                          */
/*                                                               */
/*  LANGUAGE:  ILE C for OS/400                                  */
/*                                                               */
/*  DESCRIPTION:  THIS PROGRAM ENTERS COMMANDS TO BE PROCESSED ONTO */
/*    A QUEUE CALLED 'TESTQ' IN LIBRARY 'QGPL'.  THE USER WILL BE   */
/*    PROMPTED TO ENTER AS MANY COMMANDS (UNDER 51 CHARACTERS) AS   */
/*    IS DESIRED.  WHEN THE USER WISHES TO END THE PROGRAMS,        */
/*    ALL THAT NEED BE DONE IS ENTER 'quit' AT THE PROMPT.          */
/*                                                               */
/*  APIs USED:                                                   */
/*                                                               */
/*****************************************************************/
#include <stdio.h>
#include <string.h>
#include <miqueue.h>
#include <miptrnam.h>


main()
{
_ENQ_Msg_Prefix_T e_msg_prefix;
_SYSPTR queue;
char INMsg[100];

/*****************************************************************/
/*  Resolve to the queue created by $USQEXSRV.                   */
/*****************************************************************/

queue = rslvsp(_Usrq,"TESTQ","QGPL",_AUTH_ALL);
e_msg_prefix.Msg_Len = 100;

/*****************************************************************/
/*  Loop until the user enters 'quit' as the command.            */
```

```
/*****************************************************************/

  while (1) {
    printf("\nEnter command to put on queue, or 'quit' \n ");
    scanf("%100s", INMsg);
    gets(INMsg);
    printf("\nCommand entered was ==> %.100s\n",INMsg);

/*****************************************************************/
/*  Check to see if the user entered 'quit' as the command.      */
/*  If true then break out of the 'while' loop.                  */
/*****************************************************************/

    if ((strncmp(INMsg,"quit",4) == 0)||(strncmp(INMsg,"QUIT",4) == 0))
      { break; }

/*****************************************************************/
/*  Add the user-entered command to the queue.                   */
/*****************************************************************/

    enq(queue,&e_msg_prefix,INMsg);
    strcpy(INMsg," ");
  } /*while*/

/*****************************************************************/
/*  Add the command end to the queue which causes the            */
/*  server program ($USQEXSRV) to end                            */
/*****************************************************************/

  strcpy(INMsg,"END");
  enq(queue,&e_msg_prefix,INMsg);
} /* $USQEXREQ */
```

To create the requester program using ILE C, specify the following:

```
CRTBNDC PGM(QGPL/$USQEXREQ) SRCFILE(QGPL/QCSRC)
```

Server Program ($USQEXSRV)

The following is the server program using ILE C:

```
/*****************************************************************/
/*  PROGRAM:  $USQEXSRV                                          */
/*                                                               */
/*  LANGUAGE:  ILE C for OS/400                                  */
/*                                                               */
/*  DESCRIPTION:  THIS PROGRAM EXTRACTS COMMANDS TO BE RUN FROM  */
/*    A QUEUE CALLED 'TESTQ' IN LIBRARY 'QGPL'.  THE COMMANDS WILL*/
/*    BE EXTRACTED AND RUN IN FIFO ORDER.  THE QUEUE WILL BE     */
/*    CREATED PRIOR TO USE AND SHOULD BE DELETED AFTER EACH USE  */
/*    OF THIS EXAMPLE PROGRAM.  THIS PROGRAM END WHEN IT         */
/*    EXTRACTS THE COMMAND 'END' FROM THE QUEUE.                 */
/*       THE FLOW IS AS FOLLOWS:                                 */
/*         (1)  CREATE THE USER QUEUE                            */
/*         (2)  ENTER LOOP                                       */
/*         (3)     WAIT FOREVER FOR A COMMAND ON THE QUEUE       */
/*         (4)      IF COMMAND IS 'END' THEN EXIT LOOP           */
/*         (5)      ELSE RUN COMMAND, RESTART LOOP               */
/*         (6)  END LOOP                                         */
/*    FOR BEST RESULTS, THIS PROGRAM CAN BE CALLED BY THE USER, THEN*/
/*    THE $USQEXREQ SHOULD BE CALLED FROM ANOTHER SESSION.       */
/*                                                               */
/* APIs USED:  QCMDEXC, QUSCRTUQ                                 */
/*                                                               */
```

```
/*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <micomput.h>
#include <miqueue.h>
#include <miptrnam.h>
#include <quscrtuq.h>
#include <qcmdexc.h>


main()
{
_DEQ_Msg_Prefix_T d_msg_prefix;
_SYSPTR           queue;
char              OUTMsg[100];
int               cmd_name_lngth;
decimal(15,5)     pack_name_lngth;
char              igc_param[] = "IGC";

/*****************************************************************/
/*    Set up the parameters to be used in the call to 'QUSCRTUQ'    */
/*****************************************************************/

char q_name[]= "TESTQ     QGPL      ";
char ext_atr[]= "TESTER    ";
char q_type[]= "F";
int key_lngth = 0;
int max_msg_s = 100;
int int_msgs  = 10;
int add_msgs = 50;
char auth[]  = "*ALL      ";
char desc[] = "Description .....    ";

/*****************************************************************/
/*  Call the 'QUSCRTUQ' program to create the user queue.          */
/*****************************************************************/

  QUSCRTUQ(q_name,ext_atr,q_type,key_lngth,max_msg_s,int_msgs,
           add_msgs,auth,desc);

/*****************************************************************/
/*  Resolve to the queue created above.                            */
/*****************************************************************/

  queue = rslvsp(_Usrq,"TESTQ","QGPL",_AUTH_ALL);

/*****************************************************************/
/*  Set the deq operation to wait for command indefinitely.        */
/*****************************************************************/

  d_msg_prefix.Wait_Forever = 1;

/*****************************************************************/
/*  Loop until the command 'END' is extracted from the queue       */
/*****************************************************************/

  while (1) {
    deq(&d_msg_prefix,OUTMsg,queue);

/*****************************************************************/
/*  Check to see if the command extracted is 'END'                 */
/*  If true then break out of the 'while' loop.                    */
/*****************************************************************/

    if (strncmp(OUTMsg,"END",3) == 0)
      { break; }
```

```
    cmd_name_lngth = strlen(OUTMsg);

/*****************************************************************/
/*  Convert the integer in cmd_name_lngth to a packed decimal    */
/*****************************************************************/

    cpynv(NUM_DESCR(_T_PACKED,15,5), &pack_name_lngth,
          NUM_DESCR(_T_SIGNED,4,0), &cmd_name_lngth);

/*****************************************************************/
/*  Execute the command extracted from the queue                 */
/*****************************************************************/

    QCMDEXC(OUTMsg,pack_name_lngth,igc_param);
  } /* while */
}  /* $USQEXSRV */
```

To create the server program using ILE C, specify the following:

```
CRTBNDC PGM(QGPL/$USQEXSRV) SRCFILE(QGPL/QCSRC)
```

Defining Queries

This topic includes several examples that use the Query (QQQQRY) API. The examples define the following query functions:

- A simple query to perform ordering
- A join query
- A join query with selection grouping and ordering

The following QQAPI header (or include) file and the QQFUNCS query code are used by all the examples. The example programs follow the QQAPI header and QQFUNCS code.

QQAPI Header

```
/*****************************************************************/

#ifndef _QQAPIH
#define _QQAPIH


/*****************************************************************/
/*****************************************************************/
/*                                                               */
/* FUNCTION:  Defines constants and structures for use           */
/*            with the QQQQRY API examples.                      */
/*                                                               */
/* LANGUAGE:  ILE C for OS/400                                   */
/*                                                               */
/* APIs USED: None                                               */
/*                                                               */
/*****************************************************************/
/*****************************************************************/


/* The following define will enable some debug procedures and code */
/* #define QQDEBUG */

/* Query Open options */
#define QO_INPUT   1
```

```
#define QO_OUTPUT  2
#define QO_UPDATE  4
#define QO_DELETE  8

/* simple defines */
#define ON    1
#define OFF   0

/* user defined limits - change as needed */
#define MAX_ORDERBY    20
  /* max number of order by fields (8000 max)*/
#define MAX_JOINTESTS  20
  /* max number of join tests (999 max)*/
#define MAX_GROUPBY    20
  /* max number of order by fields (120 max)*/
/* storage sizes - increase if needed */
#define QDT_SIZE      6000
#define FORMAT_SIZE   5000
#define SELECT_SIZE   5000
#define AP_SIZE      65535     /* Initialize access plan size to 64K */


/* Required definitions - do NOT change, hard limits */
#define MAX_FILES      32      /* Maximum number of files in a query */
#define REQ_REL       "01"     /* Required value for release field */
#define REQ_VER       "00"     /* Required value for version field */
#define QFLD_SIZE      30      /* QQ API field size - see qqqqry.h */

/* define error code structure */
typedef struct
{
    int bytes_provided;
    int bytes_available;
    char msgid[(7]);
    char reserved;
    char data[(512]);
} error_code;

/* define attribute record for QUSCUSAT API */
typedef _Packed struct
{
    int  numAttrs;
    int  key;
    int  length;
    _Packed union {
      long spaceSize;    /* key = 1 */
      char initialValue; /* key = 2 */
      char autoExtend;   /* key = 3 */
    } data;
} QUSCUSAT_T;

/* define access plan structure */
typedef _Packed struct
{
    _SPCPTR storagePtr;
    long size;
    char reserved[(28]);
} ACCPLN_T;


/* Function prototypes: */
void dumpPtr(char *, char *, int );
char *strcnv400(char *, int );
int strcpy400(char *, char *, int );
void initUFCB(QDBUFCB_T *, int , Qdb_Qddfmt_t *);
void initQDT(QDBQH_T *, char , int , int ,
  char , int );
```

```
void initFile(QDBQFHDR_T *, char , char );
void initFormat(Qdb_Qddfmt_t *, char *);
void initSelection(QDBQS_T *);
void initOrderBy(QDBQKH_T *);
void initGroupBy(QDBQGH_T *);
void initJoin(QDBQJHDR_T *);
int addFile(QDBQFHDR_T *, QDBQN_T *,
  char *, char *, char *, char *);
int getRecordFmt(Qdb_Qddfmt_t *, long,
  char *, char *, char *);
long copyField(Qdb_Qddfmt_t *, char *, int ,
  Qdb_Qddfmt_t *);
void setFieldUsage(Qdb_Qddfmt_t *, char *, char );
int addSelectField(QDBQS_T *, char *, int );
int addSelectLiteral(QDBQS_T *, void *, int );
int addSelectOperator(QDBQS_T *, char *);
int addOrderBy(QDBQKH_T *, QDBQKF_T *,
  char *, int );
int addGroupBy(QDBQGH_T *, QDBQGF_T *,
  char *, int );
int addJoinTest(QDBQJHDR_T *, QDBQJFLD_T *, char *,
  int , char *, int , char *);
void addQDTsection(QDBQH_T *, char *, int , int *);
long createAccessPlanSpace(ACCPLN_T *, char *, long );
int saveQDT(QDBQH_T *, ACCPLN_T *);
int saveAccessPlan(ACCPLN_T *);
int loadQDT(QDBQH_T *, ACCPLN_T *);
long loadAccessPlan(ACCPLN_T *, char *);

#endif
/*****************************************************************/
```

QQFUNCS Query Code

```
/*****************************************************************/
#include <stdio.h>
#include <string.h>
#include <qdbrtvfd.h>
#include <qqqqry.h>
#include <quscrtus.h>
#include <qusptrus.h>
#include <quscusat.h>
#include <qusrusat.h>
#include "qqapi.h"

/*****************************************************************/
/*****************************************************************/
/*                                                               */
/* FUNCTION:  This module contains all of the functions          */
/*            used by the examples to build the API information. */
/*                                                               */
/* LANGUAGE:  ILE C for OS/400                                   */
/*                                                               */
/* APIs USED: QDBRTVFD, QUSCRTUS, QUSCUSAT, QUSPTRUS, QUSRUSAT   */
/*                                                               */
/*****************************************************************/
/*****************************************************************/

#ifdef QQDEBUG
/* dumpPtr(comment string, pointer, length)
 - prints a comment then dumps data in hexadecimal starting at the
   given pointer location for the specified length */
void dumpPtr(char *text, char *ptr, int len)
{
```

```
    int i;

    printf("%s\n", text);
    for (i=0; i < len; i++, ptr++)
    {
        printf("%02X ", (int) *ptr);
        if ((i+1) % 16 == 0)
           printf("\n");
    }
    printf("\n");
}
#endif


/* strcnv400(source string, string length)
 - convert an OS/400 string to a zero terminated string */
char *strcnv400(char *str, int len)
{
static char buffer[256];

    strncpy(buffer, str, len);
    buffer[len] = (char) 0;
    return(buffer);
}


/* strcpy400(destination string, source string, source length)
 - copy a zero terminated string to an OS/400 string, pad with blanks
   if necessary */
int strcpy400(char *dest, char *src, int len)
{
    int i;

    if ((i = strlen(src)) > len)
        len = i;
    if (len)
        memcpy(dest, src, strlen(src));
    if (i < len)
        memset((dest+i), ' ', len-i);
    return(len);
}


/* initUFCB(ufcb, open options, record format)
 - initialize the UFCB structure */
void initUFCB(QDBUFCB_T *ufcbPtr, int openFlags,
  Qdb_Qddfmt_t *formatPtr)
{
    _Packed struct qufcb *ufcb;

    /* verify parameters */
    if (ufcbPtr == NULL || openFlags == 0)
    {
        printf("Invalid UFCB settings\n");
        return;
    }
    /* Clear the entire UFCB */
    memset((void *) ufcbPtr, (char) 0, sizeof(QDBUFCB_T));
    /* Now start initializing values */
    ufcb = &ufcbPtr-&gt;qufcb;
    strcpy400((char *) ufcb->relver.release, REQ_REL,
      sizeof(ufcb->relver.release));
    strcpy400((char *) ufcb->relver.version, REQ_VER,
      sizeof(ufcb->relver.version));
    /* Blocked Records (BLKRCD) should be on if CPYFRMQRYF is used */
    ufcb->markcnt.flg2brcd = ON;
    ufcb->parameter.maximum = MAXFORMATS;
```

```
    /* Set the open option */
    if (openFlags&QO_INPUT)
        ufcb->open.flagui = ON;
    if (openFlags&QO_OUTPUT)
        ufcb->open.flaguo = ON;
    if (openFlags&QO_UPDATE)
        ufcb->open.flaguu = ON;
    if (openFlags&QO_DELETE)
        ufcb->open.flagud = ON;
    /* set up options to match _Ropen options */
    ufcb->parameter.keyfdbk = KEYFDBK;
    ufcb->parameter.keyonoff = ON; /* Key feedback ON */
    ufcb->parameter.filedep = FILEDEP;
    ufcb->parameter.fldonoff = ON; /* File dependent I/O ON */
    /* turn the rest of the parameters off */
    ufcb->parameter.seqonly = NOTSEQUPROC;
    ufcb->parameter.primrln1 = NOTRECORDLTH;
    ufcb->parameter.commitc = NOTCOMITCTL;
    /* if the format is supplied,
      define it in the UFCB and do level checking */
    if (formatPtr != NULL)
    {
        ufcb->parameter.lvlchk = LEVELCK;
        ufcb->parameter.lvlonoff = ON; /* Level check ON */
        ufcb->parameter.curnum = 1; /* only one format */
        /* set the format name and format level identifier */
        ufcb->parameter.recfmts = FORMATSEQ;
        memcpy(ufcb->parameter.formats[0].name, formatPtr->Qddfname,
          sizeof(ufcb->parameter.formats[0].name));
        memcpy(ufcb->parameter.formats[0].number, formatPtr->Qddfseq,
          sizeof(ufcb->parameter.formats[0].number));
    }
    else /* no format and level checking */
    {
        ufcb->parameter.lvlchk = NOTLEVELCK;
        ufcb->parameter.recfmts = NOTFORMATSEQ;
    }
    ufcb->ufcbend  = ENDLIST;
}


/* initQDT(qdt, options...)
 - initialize the QDT header */
void initQDT(QDBQH_T *qdtHdr, char alwCpyDta,
  int optAllAp, int statusMsgs,
  char optimize, int forRows)
{
    if (qdtHdr == NULL)
    {
        printf("Invalid QDT settings\n");
        return;  /* invalid pointer */
    }
    /* Clear the entire QDT */
    memset((void *) qdtHdr, (char) 0, sizeof(QDBQH_T));
    /* set the initial QDT space used size */
    qdtHdr->qdbspcsize = sizeof(QDBQH_T);
    /* QDT options... */
    /* ordering not specified */
    qdtHdr->qdbqkeyo = -1;
    /* set optimize parameter (ALLIO, FIRSTIO, MINWAIT) */
    if (optimize == QDBQFINA || optimize == QDBQFINF ||
      optimize == QDBQFINM || optimize == QDBQFINC)
        qdtHdr->qdbqfin = optimize; /* OPTIMIZE() parameter */
    else
        qdtHdr->qdbqfin = QDBQFINA; /* default to OPTIMIZE(*ALLIO) */
    /* set allow copy data parameter (YES, NO, OPTIMIZE) */
    if (alwCpyDta == QDBQTEMN || alwCpyDta ==  QDBQTEMO ||
```

```
      alwCpyDta == QDBQTEMA)
        qdtHdr->qdbqtem = alwCpyDta; /* ALWCPYDTA() parameter */
    else
        qdtHdr->qdbqtem = QDBQTEMA; /* default to ALWCPYDTA(*YES) */
    /* status messages (YES, NO) */
    qdtHdr->qdbqattr.qdbqnst = statusMsgs ? ON : OFF;
    /* optimize all access path parameter (YES, NO) */
    qdtHdr->qdbqdt_7.qdbqopta = optAllAp ? ON : OFF;
    /* optimizer for n rows parameter */
    qdtHdr->qdbq_optmrows = forRows > 0 ? forRows : 0;
}


/* initFile(file section, join type, join order option)
 - initialize the file header section */
void initFile(QDBQFHDR_T *fileHdr, char joinType, char joinOrder)
{
    if (fileHdr == NULL)
    {
        printf("Invalid File Header settings\n");
        return;  /* invalid pointer */
    }
    /* Clear the header */
    memset((void *) fileHdr, (char) 0, sizeof(QDBQFHDR_T));
    /* File Spec options... */
    /* inner, partial outer or exception join type */
    if (joinType == QDBQINNJ || joinType == QDBQOUTJ ||
      joinType == QDBQEXCJ)
        fileHdr->qdbqmfop = joinType;
    else
        fileHdr->qdbqmfop = QDBQINNJ;
    /* join order - any order or join as specified */
    fileHdr->qdbqmfor = joinOrder == QDBQMFON ? QDBQMFON : QDBQMFOA;
}


/* initFormat(format section, format name)
 - initialize the format header section */
void initFormat(Qdb_Qddfmt_t *formatHdr, char *name)
{
    if (formatHdr == NULL)
    {
        printf("Invalid Format Header settings\n");
        return;  /* invalid pointer */
    }
    /* Clear the header */
    memset((void *) formatHdr, (char) 0, sizeof(Qdb_Qddfmt_t));
    /* Format Spec options... */
    strcpy400(formatHdr->Qddfname, name, sizeof(formatHdr->Qddfname));
    formatHdr->Qddfrcid = 65535;
    formatHdr->Qddfsrcd = 65535;
    formatHdr->Qddflgs.Qddfrsid = 1;
    memset(formatHdr->Qddfseq, ' ', sizeof(formatHdr->Qddfseq));
    memset(formatHdr->Qddftext, ' ', sizeof(formatHdr->Qddftext));
    /* Format size (so far) */
    formatHdr->Qddbyava = sizeof(Qdb_Qddfmt_t);
    formatHdr->Qddbyrtn = formatHdr->Qddbyava;
}


/* initSelection(selection section)
 - initialize the selection header section */
void initSelection(QDBQS_T *selectHdr)
{
    if (selectHdr == NULL)
    {
        printf("Invalid selection settings\n");
```

```
        return;  /* invalid pointer */
    }
    /* Clear the header */
    memset((void *) selectHdr, (char) 0, sizeof(QDBQS_T));
    /* set initial selection spec size (minus dummy selection spec) */
    selectHdr->qdbqsl = sizeof(QDBQS_T) - sizeof(selectHdr->qdbqspec);
}


/* initOrderBy(orderby section)
 - initialize order by header section */
void initOrderBy(QDBQKH_T *orderByHdr)
{
    if (orderByHdr == NULL)
    {
        printf("Invalid Order By settings\n");
        return;  /* invalid pointer */
    }
    /* Clear the header */
    memset((void *) orderByHdr, (char) 0, sizeof(QDBQKH_T));
}


/* initGroupBy(groupby section)
 - initialize group by header section */
void initGroupBy(QDBQGH_T *groupByHdr)
{
    if (groupByHdr == NULL)
    {
        printf("Invalid Group By settings\n");
        return;  /* invalid pointer */
    }
    /* Clear the header */
    memset((void *) groupByHdr, (char) 0, sizeof(QDBQGH_T));
}


/* initJoin(join section)
 - initialize join header section */
void initJoin(QDBQJHDR_T *joinHdr)
{
    if (joinHdr == NULL)
    {
        printf("Invalid Join settings\n");
        return;  /* invalid pointer */
    }
    /* Clear the header */
    memset((void *) joinHdr, (char) 0, sizeof(QDBQKH_T));
    /* set initial join spec size */
    joinHdr->qdbqjln = sizeof(QDBQJHDR_T);
}


/* addFile (file section, file spec section, file name, file library,
   file member, file format)
 - add file information to the file section */
int addFile(QDBQFHDR_T *fileHdr, QDBQN_T *fileSpec,
  char *filename, char *library, char *member, char *format)
{
    int i;
    QDBQFLMF_T *fileSpecPtr;

    if (fileHdr == NULL || fileSpec == NULL || filename == NULL)
        return(0);  /* invalid data */
    if (fileHdr->qdbqfilnum == MAX_FILES)
        return(0);  /* no more files allowed */
    /* increment the count of file specs */
```

```
    i = fileHdr->qdbqfilnum++;
    /* initialize the file spec area */
    memset((void *) &fileSpec[i], (char) 0, sizeof(QDBQN_T));
    fileSpecPtr = (QDBQFLMF_T *) &fileSpec[i].qdbqflmf;
    /* fill in the data... */
    strcpy400(fileSpecPtr->qdbqfile, filename,
      sizeof(fileSpecPtr->qdbqfile));
    if (library == NULL)
        strcpy400(fileSpecPtr->qdbqlib, QDBQLIBL,
          sizeof(fileSpecPtr->qdbqlib));
    else
        strcpy400(fileSpecPtr->qdbqlib, library,
          sizeof(fileSpecPtr->qdbqlib));
    if (member == NULL)
        strcpy400(fileSpecPtr->qdbqmbr, QDBQFRST,
          sizeof(fileSpecPtr->qdbqmbr));
    else
        strcpy400(fileSpecPtr->qdbqmbr, member,
          sizeof(fileSpecPtr->qdbqmbr));
    if (format == NULL)
        strcpy400(fileSpecPtr->qdbqfmt, QDBQONLY,
          sizeof(fileSpecPtr->qdbqfmt));
    else
        strcpy400(fileSpecPtr->qdbqfmt, format,
          sizeof(fileSpecPtr->qdbqfmt));
    /* return the amount of storage used in the file specs */
    return(fileHdr->qdbqfilnum*sizeof(QDBQN_T));
}


/* getRecordFmt(format, format storage size(max),
   file name, file library, file format)
 - get a record format (using QDBRTVFD) */
int getRecordFmt(Qdb_Qddfmt_t *formatPtr, long spaceSize,
  char *filename, char *libname, char *formatname)
{
    error_code errcod;
    char override = '1';  /* process overrides */
    char fileLibname[20];
    char outFilLib[20];
    char format[10];

    if (formatPtr == NULL || filename == NULL)
        return(0);  /* missing data */
    errcod.bytes_provided = 512;
    errcod.msgid[0] = (char) 0;
    /* set up temporary variables... */
    strcpy400(fileLibname, filename, 10);
    if (libname == NULL)
        strcpy400(&fileLibname[10], QDBQLIBL, 10);
    else
        strcpy400(&fileLibname[10], libname, 10);
    if (formatname == NULL)
        strcpy400(format, filename, 10);
    else
        strcpy400(format, formatname, 10);
    /* call the RTVFD API to get the record format */
    QDBRTVFD((char *) formatPtr, spaceSize, outFilLib,
      "FILD0200",
      fileLibname, format, &override,
        "*LCL      ", "*EXT      ", &errcod);
    if (errcod.msgid[0])
    {
        printf("API QDBRTVFD failed\n");
        printf("msgid = %7s\n", strcnv400(errcod.msgid,
          sizeof(errcod.msgid)));
    }
```

```
    if (formatPtr->Qddbyrtn != formatPtr->Qddbyava)
        return(0);  /* missing data */
    /* return total storage used in format */
    return(formatPtr->Qddbyrtn);
}


/* copyField(format, field name, file number, existing format)
 - copy a field from an existing format */
long copyField(Qdb_Qddfmt_t *formatPtr, char *fieldName, int fieldFile,
  Qdb_Qddfmt_t *oldFormatPtr)
{
    int i;
    long fieldSize;
    char padField[30];
    Qdb_Qddffld_t *fieldPtr, *oldFieldPtr;

    if (formatPtr == NULL || fieldName == NULL || oldFormatPtr == NULL)
        return(0);  /* missing data */
    strcpy400(padField, fieldName, 30);
    /* set up field pointers */
    fieldPtr = (Qdb_Qddffld_t *) ((char *) formatPtr +
      formatPtr->Qddbyava);
    oldFieldPtr = (Qdb_Qddffld_t *) (oldFormatPtr + 1);
    /* loop through all the fields, looking for a match */
    for (i=0; i < oldFormatPtr->Qddffldnum; i++,
      oldFieldPtr = (Qdb_Qddffld_t *) ((char *) oldFieldPtr +
      oldFieldPtr->Qddfdefl))
    /* if a match was found... */
    if (memcmp(oldFieldPtr->Qddfflde, padField, 30) == 0)
    {
        /* copy the field over */
        fieldSize = oldFieldPtr->Qddfdefl;
        memcpy(fieldPtr, oldFieldPtr, fieldSize);
        /* set the file number it was defined in */
        fieldPtr->Qddfjref = fieldFile;
        /* increment the format header information */
        formatPtr->Qddffldnum++;
        formatPtr->Qddfrlen += fieldPtr->Qddffldb;
        formatPtr->Qddbyava += fieldSize;
        formatPtr->Qddbyrtn = formatPtr->Qddbyava;
        break;
    }
    /* return total storage used in format */
    return(formatPtr->Qddbyrtn);
}


/* setFieldUsage(format, field name, usage)
 - set the field usage in a format */
void setFieldUsage(Qdb_Qddfmt_t *formatPtr, char *fieldName, char usage)
{
    int i;
    char padField[30];
    Qdb_Qddffld_t *fieldPtr;

    if (formatPtr == NULL)
        return;  /* missing data */
    if (fieldName != NULL)
        strcpy400(padField, fieldName, 30);
    /* set up field pointers */
    fieldPtr = (Qdb_Qddffld_t *) (formatPtr + 1);
    /* loop through all the fields, looking for a match */
    for (i=0; i < formatPtr->Qddffldnum; i++,
      fieldPtr = (Qdb_Qddffld_t *) ((char *) fieldPtr +
      fieldPtr->Qddfdefl))
    /* if all fields to be set or a match was found... */
```

```
    if (fieldName == NULL ||
      memcmp(fieldPtr->Qddfflde, padField, 30) == 0)
        fieldPtr->Qddffiob = usage;
}


/* addSelectField(section section, field name, file number for field)
 - add a selection for a file field to the selection section */
int addSelectField(QDBQS_T *selectHdr, char *fieldName, int fieldFile)
{
    QDBQSIT_T *selectItemPtr;
    QDBQSOPF_T *selectFldPtr;
    int itemSize;

    if (selectHdr == NULL || fieldName == NULL)
        return(0);  /* invalid data */
    /* set up all the section for adding a field */
    selectItemPtr = (QDBQSIT_T *) ((char *) selectHdr +
      selectHdr->qdbqsl);
    itemSize = sizeof(QDBQSIT_T) - sizeof(selectItemPtr->qdbqsitm);
    memset((void *) selectItemPtr, (char) 0, itemSize);
    selectFldPtr = (QDBQSOPF_T *) ((char *) selectItemPtr + itemSize);
    memset((void *) selectFldPtr, (char) 0, sizeof(QDBQSOPF_T));
    /* set up the selection item information for a field */
    selectItemPtr->qdbqslen = itemSize + sizeof(QDBQSOPF_T);
    /* length */
    selectItemPtr->qdbqsitt = QDBQOPF; /* type is field */
    /* now set up the field */
    strcpy400(selectFldPtr->qdbqsofn, fieldName,
      sizeof(selectFldPtr->qdbqsofn));
    selectFldPtr->qdbqsofj = fieldFile;
    /* update the header statistics */
    selectHdr->qdbqsnum++;  /* increment number of select specs */
    selectHdr->qdbqsl += selectItemPtr->qdbqslen;  /* total length */
    /* return the total storage now in the selection section */
    return(selectHdr->qdbqsl);
}


/* addSelectLiteral(selection section, literal, size of literal data)
 - add a selection for a literal to the selection section */
int addSelectLiteral(QDBQS_T *selectHdr, void *literal, int sizeLit)
{
    QDBQSIT_T *selectItemPtr;
    QDBQSOCH_T *selectLitPtr;
    void *selectDataPtr;
    int itemSize;

    if (selectHdr == NULL || literal == NULL || sizeLit < 1)
        return(0);  /* invalid data */
    /* set up all the sections for adding a literal */
    selectItemPtr = (QDBQSIT_T *)
      ((char *) selectHdr + selectHdr->qdbqsl);
    itemSize = sizeof(QDBQSIT_T) - sizeof(selectItemPtr->qdbqsitm);
    memset((void *) selectItemPtr, (char) 0, itemSize);
    selectLitPtr = (QDBQSOCH_T *) ((char *) selectItemPtr + itemSize);
    memset((void *) selectLitPtr, (char) 0, sizeof(QDBQSOCH_T));
    selectDataPtr = (void *) (selectLitPtr + 1);
    /* set up the selection item information for a literal */
    selectItemPtr->qdbqslen = itemSize + sizeof(QDBQSOCH_T) + sizeLit;
    selectItemPtr->qdbqsitt = QDBQOPC;  /* literal type */
    /* now set up the literal */
    selectLitPtr->qdbqsocl = sizeLit;  /* literal size */
    selectLitPtr->qdbqsoft = '\xFF';
      /* use job format for date/time fields */
    memcpy(selectDataPtr, literal, sizeLit);
      /* save the literal value */
```

```
    /* update the header statistics */
    selectHdr->qdbqsnum++;  /* increment number of select specs */
    selectHdr->qdbqsl += selectItemPtr->qdbqslen;  /* total length */
    /* return the total storage now in the selection section */
    return(selectHdr->qdbqsl);
}


/* addSelectOperator(selection section, operator type)
 - add a selection for an operator to the selection section */
int addSelectOperator(QDBQS_T *selectHdr, char *operator)
{
    QDBQSIT_T *selectItemPtr;
    QDBQSOPR_T *selectOprPtr;
    QDBQSOP2_T *selectWldPtr;
    int itemSize;
    int oprSize;

    if (selectHdr == NULL || operator == NULL)
        return(0);  /* invalid data */
    /* set up all the sections for adding an operator */
    selectItemPtr = (QDBQSIT_T *)
      ((char *) selectHdr + selectHdr->qdbqsl);
    itemSize = sizeof(QDBQSIT_T) - sizeof(selectItemPtr->qdbqsitm);
    memset((void *) selectItemPtr, (char) 0, itemSize);
    selectOprPtr = (QDBQSOPR_T *) ((char *) selectItemPtr + itemSize);
    oprSize = sizeof(QDBQSOPR_T) + sizeof(QDBQSOP2_T);
    memset((void *) selectOprPtr, (char) 0, oprSize);
    /* set up the selection item information for an operator */
    selectItemPtr->qdbqslen = itemSize + oprSize;  /* length */
    selectItemPtr->qdbqsitt = QDBQOPTR;  /* operator type */
    /* now set up the operator */
    memcpy(selectOprPtr->qdbqsop, operator,
      sizeof(selectOprPtr->qdbqsop));
    /* wildcard operator set up */
    if (memcmp(operator, QDBQWILD, 2) == 0)
    {
       selectOprPtr->qdbqswc1 = '_';
       selectOprPtr->qdbqswc2 = '*';
       selectWldPtr = (QDBQSOP2_T *) (selectOprPtr + 1);
       memcpy(selectWldPtr->qdbqsdb1,"\42_", 2);
       memcpy(selectWldPtr->qdbqsdb2,"\42*", 2);
    }
    /* update the header statistics */
    selectHdr->qdbqsnum++;  /* increment number of select specs */
    selectHdr->qdbqsl += selectItemPtr->qdbqslen;  /* total length */
    /* return the total storage now in the selection section */
    return(selectHdr->qdbqsl);
}


/* addOrderBy(orderby section, orderby specs section, key field name,
   descend sort option
 - add an order by to the order by section */
int addOrderBy(QDBQKH_T *orderByHdr, QDBQKF_T *orderByFld,
  char *keyfield, int descend)
{
    int i;
    QDBQKF_T *orderByFldPtr;

    if (orderByHdr == NULL || orderByFld == NULL || keyfield == NULL)
        return(0);
    if (orderByHdr->qdbqknum == MAX_ORDERBY)
        return(0);
    /* increment the order by spec counter */
    i = orderByHdr->qdbqknum++;
    /* add the new orderby data */
```

```
    orderByFldPtr = &orderByFld[i];
    memset((void *) orderByFldPtr, (char) 0, sizeof(QDBQKF_T));
    strcpy400(orderByFldPtr->qdbqkfld, keyfield,
        sizeof(orderByFldPtr->qdbqkfld));
    orderByFldPtr->qdbqksq.qdbqksad = (descend) ? ON : OFF;
    /* return the space used by the order by specs */
    return(orderByHdr->qdbqknum*sizeof(QDBQKF_T));
}


/* addGroupBy(groupby section, groupby field spec section,
   groupby field name, file number of groupby field)
 - add a group by to the group by section */
int addGroupBy(QDBQGH_T *groupByHdr, QDBQGF_T *groupByFld,
  char *groupfield, int fromFile)
{
    int i;
    QDBQGF_T *groupByFldPtr;

    if (groupByHdr == NULL || groupByFld == NULL || groupfield == NULL)
        return(0);
    if (groupByHdr->qdbqgfnum == MAX_GROUPBY)
        return(0);
    /* increment the group by spec counter */
    i = groupByHdr->qdbqgfnum++;
    /* add the new groupby data */
    groupByFldPtr = (QDBQGF_T *) &groupByFld[i];
    memset((void *) groupByFldPtr, (char) 0, sizeof(QDBQGF_T));
    strcpy400(groupByFldPtr->qdbqgfld, groupfield,
        sizeof(groupByFldPtr->qdbqgfld));
    groupByFldPtr->qdbqgflj = fromFile;
    /* return the space used by the group by specs */
    return(groupByHdr->qdbqgfnum*sizeof(QDBQGF_T));
}


/* addJoinTest(join section, join test section, join from field name,
   join from file number, join to field name, join to file number,
   join operator)
 - add a join test to the join section */
int addJoinTest(QDBQJHDR_T *joinHdr,
  QDBQJFLD_T *joinSpec, char *fromFld,
  int fromFile, char *toFld, int toFile, char *joinOp)
{
    int i;
    QDBQJFLD_T *joinSpecPtr;

    if (joinHdr == NULL || joinSpec == NULL)
        return(0);
    if (joinHdr->qdbqjknum == MAX_JOINTESTS)
        return(0);
    /* increment the join test counter */
    i = joinHdr->qdbqjknum++;
    memset((void *) &joinSpec[i], (char) 0, sizeof(QDBQJFLD_T));
    /* add the new join data */
    joinSpecPtr = &joinSpec[i];
    strcpy400(joinSpecPtr->qdbqjfnm, fromFld,
      sizeof(joinSpecPtr->qdbqjfnm));
    joinSpecPtr->qdbqjfnum = fromFile;  /* 1, 2, 3, etc */
    strcpy400(joinSpecPtr->qdbqjtnm, toFld,
      sizeof(joinSpecPtr->qdbqjtnm));
    joinSpecPtr->qdbqjtnum = toFile;  /* 1, 2, 3, etc */
    /* Join operator - see #defines in QQ API include */
    strcpy400(joinSpecPtr->qdbqjop, joinOp,
      sizeof(joinSpecPtr->qdbqjop));
    /* set size of entire join spec */
    joinHdr->qdbqjln += sizeof(QDBQJFLD_T);
```

```
    /* return the space used by the join tests */
    return(joinHdr->qdbqjknum*sizeof(QDBQJFLD_T));
}


/* addQDTsection(qdt, new section, size of new section, qdt offset)
 - place a new section into the QDT */
void addQDTsection(QDBQH_T *qdtHdr, char *newSection,
  int newSize, int *offset)
{
    char *sectionPtr;

    /* position to the current end of the QDT */
    sectionPtr = (char *) qdtHdr + qdtHdr->qdbspcsize;
    /* append in the new section data */
    memcpy(sectionPtr, newSection, newSize);
    /* if an offset is to be stored, remember it now */
    if (offset != NULL)
        *offset = qdtHdr->qdbspcsize;
    /* update the QDT size */
    qdtHdr->qdbspcsize += newSize;
}


/* createAccessPlanSpace(access plan, user space name, size)
 - creates a *USRSPC object for storing the access plan */
long createAccessPlanSpace(ACCPLN_T *accessPlan, char *name,
  long spaceSize)
{
    QUSCUSAT_T chgAttr;
    _SPCPTR usrSpcPtr;
    char library[10];
    char value = (char) 0;
    char text[50];
    error_code errcode;

    errcode.bytes_provided = 512;

    strcpy400(text,"Access Plan for QQ API example",50);
    /* Create the User Space */
    QUSCRTUS(name,
             "ACCESSPLAN",
             spaceSize,
             &value,
             "*ALL      ",
             text,
             "*YES      ",
             &errcode,
             "*USER     ");
    if (errcode.msgid[0])
    {
        printf("Create User Space API failed!\n");
        printf("msgid = %7s\n", strcnv400(errcode.msgid,
          sizeof(errcode.msgid)));
        return(-1);
    }

    /* Change the User Space to allow Auto-Extend */
    strcpy400(library,&name[10],10);
    chgAttr.numAttrs = 1;
    chgAttr.key = 3; /* Auto extend */
    chgAttr.length = sizeof(char);
    chgAttr.data.autoExtend = '1';
    QUSCUSAT(library,
             name,
             &chgAttr,
             &errcode);
```

```
    if (errcode.msgid[0])
    {
        printf("Change User Space Attributes FAILED!\n");
        printf("msgid = %7s\n", strcnv400(errcode.msgid,
          sizeof(errcode.msgid)));
        return(-1);
    }

    /* Retrieve Space Pointer to the User Space */
    QUSPTRUS(name,
             &usrSpcPtr,
             &errcode);
    if (errcode.msgid[0])
    {
        printf("Retrieve Space Pointer to User Space FAILED!\n");
        printf("msgid = %7s\n", strcnv400(errcode.msgid,
          sizeof(errcode.msgid)));
        return(-1);
    }
    /* Now move to the access plan itself (on 16 byte boundary) */
    accessPlan->storagePtr = (_SPCPTR) ((char*) usrSpcPtr + 16);

    return(0);
}


/* saveAccessPlan(access plan)
 - update the size in the access plan (QQQQRY actually wrote the data) */
int saveAccessPlan(ACCPLN_T *accessPlan)
{
    _SPCPTR usrSpcPtr;

    /* Position to the start of the user space */
    usrSpcPtr = (_SPCPTR) ((char*) accessPlan->storagePtr - 16);
    /* Write the access plan size out at the start */
    memcpy(usrSpcPtr, (void *) &accessPlan->size,
      sizeof(accessPlan->size));
#ifdef QQDEBUG
    printf("AP size = %ld\n", accessPlan->size);
#endif
    return(0);
}


/* saveQDT(qdt, access plan)
 - append the QDT to the end of the access plan */
int saveQDT(QDBQH_T *qdtPtr, ACCPLN_T *accessPlan)
{
    _SPCPTR usrSpcPtr;

    /* Position to the just after the access plan */
    usrSpcPtr = (_SPCPTR) ((char*) accessPlan->storagePtr +
      accessPlan->size);
    /* Write the QDT size out */
    memcpy(usrSpcPtr, &qdtPtr->qdbspcsize, sizeof(qdtPtr->qdbspcsize));
#ifdef QQDEBUG
    printf("qdt size = %ld\n", qdtPtr->qdbspcsize);
#endif
    /* Move up the user space pointer */
    usrSpcPtr = (_SPCPTR) ((char *) usrSpcPtr + 16);
    /* Write the QDT itself out */
    memcpy(usrSpcPtr, qdtPtr, qdtPtr->qdbspcsize);
    return(0);
}


/* loadQDT(qdt, access plan)
```

```
 - load the QDT from the end of the access plan */
int loadQDT(QDBQH_T *qdtPtr, ACCPLN_T *accessPlan)
{
    _SPCPTR usrSpcPtr;

    /* Position to the just after the access plan */
    usrSpcPtr = (_SPCPTR) ((char*) accessPlan->storagePtr +
      accessPlan->size);
    /* Write the QDT size out */
    memcpy((void *) &qdtPtr->qdbspcsize, usrSpcPtr,
      sizeof(qdtPtr->qdbspcsize));
#ifdef QQDEBUG
    printf("qdt size = %ld\n", qdtPtr->qdbspcsize);
#endif
    /* Move up the user space pointer */
    usrSpcPtr = (_SPCPTR) ((char *) usrSpcPtr + 16);
    /* Write the QDT itself out */
    memcpy((void *) qdtPtr, usrSpcPtr, qdtPtr->qdbspcsize);
    return(qdtPtr->qdbspcsize);
}


/* loadAccessPlan(access plan, userspace name)
 - loads an access plan from a *USRSPC object */
long loadAccessPlan(ACCPLN_T *accessPlan, char *name)
{
    Qus_SPCA_0100_t usrSpcAttr;
    _SPCPTR usrSpcPtr;
    error_code errcode;

    errcode.bytes_provided = 512;
    errcode.msgid[0] = (char) 0;

    /* Retrieve Space Pointer to the User Space */
    QUSPTRUS(name, &usrSpcPtr, &errcode);
    if (errcode.msgid[0])
    {
        printf("Retrieve Space Pointer to User Space FAILED!\n");
        printf("msgid = %7s\n", strcnv400(errcode.msgid,
          sizeof(errcode.msgid)));
        return(0);
    }

    /* Retrieve Size of Access Plan */
    QUSRUSAT(&usrSpcAttr,
            sizeof(Qus_SPCA_0100_t),
            "SPCA0100",
            name,
            &errcode);
    if (errcode.msgid[0])
    {
        printf("Retrieve User Space Attributes FAILED!\n");
        printf("msgid = %7s\n", strcnv400(errcode.msgid,
          sizeof(errcode.msgid)));
        return(0);
    }
#ifdef QQDEBUG
    else
    {
        printf("Original User Space Attributes\n");
        printf("Bytes Returned  ==> %d\n",usrSpcAttr.Bytes_Returned);
        printf("Bytes Available ==> %d\n",usrSpcAttr.Bytes_Available);
        printf("Space Size      ==> %d\n",usrSpcAttr.Space_Size);
        printf("Auto Extend     ==> %c\n",
          usrSpcAttr.Automatic_Extendability);
    }
#endif
```

```
    /* Pull the access plan size out first */
    memcpy((void *) &accessPlan->size, usrSpcPtr,
      sizeof(accessPlan->size));
#ifdef QQDEBUG
    printf("AP size = %ld\n", accessPlan->size);
#endif
    /* Now move to the access plan itself (on 16 byte boundary) */
    accessPlan->storagePtr = (_SPCPTR) ((char*) usrSpcPtr + 16);

    return(accessPlan->size);
}


/*****************************************************************/
```

Defining a Simple Query

This QQQQRY API example defines a simple query to perform ordering. The following is the equivalent SQL:

```
SELECT * FROM OPENFILE1
ORDER BY LNAME

/*****************************************************************/
/*  PROGRAM:  QQAPI1                                             */
/*                                                               */
/*  LANGUAGE:  ILE C FOR OS/400                                  */
/*                                                               */
/*  DESCRIPTION:  THIS PROGRAM DEFINES A SIMPLE QUERY TO PERFORM    */
/*    ORDERING.                                                  */
/*                                                               */
/*  APIs USED:  QQQQRY                                           */
/*                                                               */
/*****************************************************************/


#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <recio.h>
#include <qdbrtvfd.h>
#include <qqqqry.h>
#include "qqapi.h"

/* get the record format from the file */
#pragma mapinc("recfmt","APIQQ/OPENFILE1(OPENFILE1)","input","p z",,)
#include "recfmt"



/* main - start of the program
 *
 * Flow:
 * - initialize variables
 * - override to set up sharing
 * - build various QDT sections
 * - build QDT with those sections
 * - QQQQRY to run the query
 * - open the data path
 * - read the data and display it
 * - close the data paths
 *
 */
main()
{
```

```
/* record I/O variables */
_RIOFB_T *feedback;
_RFILE *file1;
APIQQ_OPENFILE1_OPENFILE1_i_t recBuf;
int recCount = 0;

/* Query variables */
QDBUFCB_T ufcbBuf;
char qdtBuf[QDT_SIZE];
char formatBuf[FORMAT_SIZE];
QDBQH_T *qdtPtr;
Qdb_Qddfmt_t *formatPtr;
QDBQFHDR_T fileHdr;
QDBQN_T fileSpec[MAX_FILES];
QDBQKH_T orderByHdr;
QDBQKF_T orderByFld[MAX_ORDERBY];
int formatSize;
int fileSpecSize;
int orderBySize;
error_code errcod;


errcod.bytes_provided = 512;
/* initialize the pointers */
qdtPtr = (QDBQH_T *) qdtBuf;
formatPtr = (Qdb_Qddfmt_t *) formatBuf;

/* initialize the headers */
initQDT(qdtPtr, QDBQTEMO, ON, ON, QDBQFINA, 0);
initFile(&fileHdr, QDBQINNJ, QDBQMFOA);
initOrderBy(&orderByHdr);

/* set up override to allow sharing */
system("OVRDBF FILE(OPENFILE1) SHARE(*YES)");
/* Note:  If level checking is not done
   (ie. no format on initUFCB) then
   the override above must specify LVLCHK(*NO) */

/* build the individual QDT sections */
fileSpecSize = addFile(&fileHdr, fileSpec, "OPENFILE1",
  NULL, NULL, NULL);
formatSize = getRecordFmt(formatPtr, FORMAT_SIZE, "OPENFILE1",
  NULL, NULL);
orderBySize = addOrderBy(&orderByHdr, orderByFld, "LNAME", OFF);

/* initialize the UFCB */
initUFCB(&ufcbBuf, QO_INPUT, formatPtr);

/* Now build the real QDT... */
addQDTsection(qdtPtr, (char *) &fileHdr,
  sizeof(fileHdr), &qdtPtr->qdbqfilo);
addQDTsection(qdtPtr, (char *) fileSpec, fileSpecSize, NULL);
addQDTsection(qdtPtr, (char *) formatPtr,
  formatSize, &qdtPtr->qdbqfldo);
addQDTsection(qdtPtr, (char *) &orderByHdr, sizeof(orderByHdr),
  &qdtPtr->qdbqkeyo);
addQDTsection(qdtPtr, (char *) orderByFld, orderBySize, NULL);

/* Finally, run the query! */
QQQQRY("RUNQRY    ", (char *) &ufcbBuf, qdtBuf, NULL, NULL,
  &errcod);
if (errcod.msgid[0])
{
    printf("API QQQQRY failed\n");
    printf("msgid = %7s\n", strcnv400(errcod.msgid,
      sizeof(errcod.msgid)));
}
```

```
    /* Now access the data */
    if ((file1 = _Ropen("OPENFILE1", "rr riofb=N")) == NULL)
    {
        printf("Error opening file\n");
        exit(1);
    }

    /* Perform any record I/O here... */
    _Rformat(file1, "OPENFILE1 ");
    printf("First name   Last name         State\n");
    feedback = _Rreadn(file1, (void *) &recBuf, sizeof(recBuf), __DFT);
    while (feedback->num_bytes == sizeof(recBuf))
    {
       recCount++;
       printf("%s   ", strcnv400(recBuf.FNAME, sizeof(recBuf.FNAME)));
       printf("%s   ", strcnv400(recBuf.LNAME, sizeof(recBuf.LNAME)));
       printf("%s\n",  strcnv400(recBuf.STATE, sizeof(recBuf.STATE)));
       feedback = _Rreadn(file1, (void *) &recBuf,
         sizeof(recBuf), __DFT);
    }
    printf("%d records selected\n", recCount);

    /* Close the file */
    _Rclose(file1);

    /* close out the QDT file handle */
    system("RCLRSC");
}
```

Defining a Join Query

This QQQQRY API example defines a join query. The following is the equivalent SQL:

```
SELECT * FROM OPENFILE1 A, OPENFILE2 B
WHERE STATE = 'AK' AND
      A.ACCTNUM = B.CUSTNUM

/*********************************************************************/
/*  PROGRAM:  QQAPI7                                                 */
/*                                                                   */
/*  LANGUAGE:  ILE C FOR OS/400                                      */
/*                                                                   */
/*  DESCRIPTION:  THIS PROGRAM DEFINES A JOIN QUERY.                 */
/*                                                                   */
/*  APIs USED:  QQQQRY                                               */
/*                                                                   */
/*********************************************************************/


#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <recio.h>
#include <qdbrtvfd.h>
#include <qqqqry.h>
#include "qqapi.h"

/* get the record format from the file */
#pragma mapinc("recfmt","APIQQ/FORMAT1(FORMAT1)","input","p z",,)
#include "recfmt"



/* main - start of the program
```

```
 *
 * Flow:
 * - initialize variables
 * - override to set up sharing
 * - build various QDT sections
 * - build QDT with those sections
 * - QQQQRY to run the query
 * - open the data path
 * - read the data and display it
 * - close the data paths
 *
 */
main()
{
    /* record I/O variables */
    _RIOFB_T *feedback;
    _RFILE *file1;
    APIQQ_FORMAT1_FORMAT1_i_t recBuf;
    int recCount = 0;

    /* Query variables */
    QDBUFCB_T ufcbBuf;
    char qdtBuf[QDT_SIZE];
    char formatBuf[FORMAT_SIZE];
    char selectBuf[SELECT_SIZE];
    QDBQH_T *qdtPtr;
    Qdb_Qddfmt_t *formatPtr;
    QDBQS_T *selectPtr;
    QDBQFHDR_T fileHdr;
    QDBQN_T fileSpec[MAX_FILES];
    QDBQJHDR_T joinHdr;
    QDBQJFLD_T joinSpec[MAX_JOINTESTS];
    int formatSize;
    int fileSpecSize;
    int selectSize;
    int joinSize;
    error_code errcod;


    errcod.bytes_provided = 512;
    /* initialize the pointers */
    qdtPtr = (QDBQH_T *) qdtBuf;
    formatPtr = (Qdb_Qddfmt_t *) formatBuf;
    selectPtr = (QDBQS_T *) selectBuf;

    /* initialize the headers */
    initQDT(qdtPtr, QDBQTEMO, ON, ON, QDBQFINA, 0);
    initFile(&fileHdr, QDBQINNJ, QDBQMFOA);
    initSelection(selectPtr);
    initJoin(&joinHdr);

    /* set up override to allow sharing */
    system("OVRDBF FILE(OPENFILE1) SHARE(*YES) LVLCHK(*NO)");
    /* Note:  If level checking is not done
       (ie. no format on initUFCB) then
       the override above must specify LVLCHK(*NO) */

    /* build the individual QDT sections */
    addFile(&fileHdr, fileSpec, "OPENFILE1", NULL, NULL, NULL);
    fileSpecSize = addFile(&fileHdr, fileSpec, "OPENFILE2",
      NULL, NULL, NULL);
    formatSize = getRecordFmt(formatPtr, FORMAT_SIZE, "FORMAT1",
      NULL, NULL);
    joinSize = addJoinTest(&joinHdr, joinSpec, "ACCTNUM", 1,
      "CUSTNUM" , 2, "EQ");
    /* build selection test:  STATE = 'AK' */
    addSelectField(selectPtr, "STATE", 1);
```

```
    addSelectLiteral(selectPtr, "'AK'", 4);
    selectSize = addSelectOperator(selectPtr, QDBQEQ);

    /* initialize the UFCB */
    initUFCB(&ufcbBuf, QO_INPUT, NULL);

    /* Now build the real QDT... */
    addQDTsection(qdtPtr, (char *) &fileHdr,
      sizeof(fileHdr), &qdtPtr->qdbqfilo);
    addQDTsection(qdtPtr, (char *) fileSpec, fileSpecSize, NULL);
    addQDTsection(qdtPtr, (char *) formatPtr,
      formatSize, &qdtPtr->qdbqfldo);
    addQDTsection(qdtPtr, (char *) &joinHdr,
      sizeof(joinHdr),&qdtPtr->qdbqjoio);
    addQDTsection(qdtPtr, (char *) joinSpec, joinSize, NULL);
    addQDTsection(qdtPtr, (char *) selectPtr,
      selectSize, &qdtPtr->qdbqselo);

    /* Finally, run the query! */
    QQQQRY("RUNQRY    ", (char *) &ufcbBuf, qdtBuf, NULL, NULL,
      &errcod);
    if (errcod.msgid[0])
    {
        printf("API QQQQRY failed\n");
        printf("msgid = %7s\n", strcnv400(errcod.msgid,
          sizeof(errcod.msgid)));
    }
    /* Now access the data */
    if ((file1 = _Ropen("OPENFILE1", "rr riofb=N")) == NULL)
    {
        printf("Error opening file\n");
        exit(1);
    }

    /* Perform any record I/O here... */
    _Rformat(file1, "FORMAT1");
    printf("Last name        Item name\n");
    feedback = _Rreadn(file1, (void *) &recBuf, sizeof(recBuf), __DFT);
    while (feedback->num_bytes == sizeof(recBuf))
    {
       recCount++;
       printf("%s  ", strcnv400(recBuf.LNAME, sizeof(recBuf.LNAME)));
       printf("%s\n", strcnv400(recBuf.ITEMNAME,
         sizeof(recBuf.ITEMNAME)));
       feedback = _Rreadn(file1, (void *) &recBuf,
         sizeof(recBuf), __DFT);
    }
    printf("%d records selected\n", recCount);

    /* Close the file */
    _Rclose(file1);

    /* close out the QDT file handle */
    system("RCLRSC");
}
```

Defining a Join Query with Selection, Grouping, and Ordering

This QQQQRY API example defines a join query with selection, grouping, and ordering. The following is the equivalent SQL:

```
SELECT LNAME, FNAME, ITEMCODE, ITEMNAME, STATUS
FROM OPENFILE1, OPENFILE2
WHERE STATE = 'AK' AND CUSTNUM = ACCTNUM
GROUP BY LNAME, FNAME, ITEMCODE, ITEMNAME, STATUS
ORDER BY ITEMNAME
```

```
/*****************************************************************/
/*  PROGRAM:  QQAPI11                                            */
/*                                                               */
/*  LANGUAGE:  ILE C FOR OS/400                                  */
/*                                                               */
/*  DESCRIPTION:  THIS PROGRAM DEFINES A JOIN QUERY WITH SELECTION  */
/*    GROUPING AND ORDERING.                                     */
/*                                                               */
/*  APIs USED:  QQQQRY                                           */
/*                                                               */
/*****************************************************************/


#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <qdbrtvfd.h>
#include <qqqqry.h>
#include "qqapi.h">



/* main - start of the program
 *
 * Flow:
 * - initialize variables
 * - override to set up sharing
 * - build various QDT sections
 * - build QDT with those sections
 * - QQQQRY to run the query
 * - open the data path
 * - read the data and display it
 * - close the data paths
 *
 */
main()
{
    /* file I/O variables */
#define REC_SIZE  52
    FILE *file1;
    char recBuf[REC_SIZE];
    int recCount = 0, found;

    /* Query variables */
    QDBUFCB_T ufcbBuf;
    char qdtBuf[QDT_SIZE];
    char formatBuf[FORMAT_SIZE];
    char tempFormatBuf[FORMAT_SIZE];
    char selectBuf[SELECT_SIZE];
    QDBQH_T *qdtPtr;
    Qdb_Qddfmt_t *formatPtr;
    Qdb_Qddfmt_t *tempFormatPtr;
    QDBQS_T *selectPtr;
    QDBQFHDR_T fileHdr;
    QDBQN_T fileSpec[MAX_FILES];
    QDBQJHDR_T joinHdr;
    QDBQJFLD_T joinSpec[MAX_JOINTESTS];
    QDBQKH_T orderByHdr;
    QDBQGH_T groupByHdr;
    QDBQKF_T orderByFld[MAX_ORDERBY];
    QDBQGF_T groupByFld[MAX_GROUPBY];
    int formatSize;
    int fileSpecSize;
    int orderBySize;
    int groupBySize;
    int selectSize;
```

```
int joinSize;
error_code errcod;

memset( (void *) &errcod, (char) 0, sizeof(error_code) );
errcod.bytes_provided = 512;
/* initialize the pointers */
qdtPtr = (QDBQH_T *) qdtBuf;
formatPtr = (Qdb_Qddfmt_t *) formatBuf;
tempFormatPtr = (Qdb_Qddfmt_t *) tempFormatBuf;
selectPtr = (QDBQS_T *) selectBuf;

/* initialize the headers */
initQDT(qdtPtr, QDBQTEMO, ON, ON, QDBQFINA, 0);
initFile(&fileHdr, QDBQINNJ, QDBQMFOA);
initFormat(formatPtr, "JOINFMT01");
initOrderBy(&orderByHdr);
initGroupBy(&groupByHdr);
initSelection(selectPtr);
initJoin(&joinHdr);

/* set up override to allow sharing */
system("OVRDBF FILE(OPENFILE1) SHARE(*YES) LVLCHK(*NO)");
/* Note:  If level checking is not done
   (ie. no format on initUFCB) then
   the override above must specify LVLCHK(*NO) */

/* build the individual QDT sections */
addFile(&fileHdr, fileSpec, "OPENFILE1", NULL, NULL, NULL);
fileSpecSize = addFile(&fileHdr, fileSpec, "OPENFILE2",
  NULL, NULL, NULL);
/* get the first format and copy some fields */
getRecordFmt(tempFormatPtr, FORMAT_SIZE, "OPENFILE1",
  NULL, NULL);
copyField(formatPtr, "LNAME", 1, tempFormatPtr);
copyField(formatPtr, "FNAME", 1, tempFormatPtr);
/* clear the old format data */
memset(tempFormatPtr, 0, FORMAT_SIZE);
/* get the second format and copy some more fields */
getRecordFmt(tempFormatPtr, FORMAT_SIZE, "OPENFILE2", NULL, NULL);
copyField(formatPtr, "ITEMCODE", 2, tempFormatPtr);
copyField(formatPtr, "ITEMNAME", 2, tempFormatPtr);
formatSize = copyField(formatPtr, "STATUS", 2, tempFormatPtr);
/* set all the fields to input only */
setFieldUsage(formatPtr, NULL, 1);
/* build selection test:  STATE = 'AK' */
addSelectField(selectPtr, "STATE", 1);
addSelectLiteral(selectPtr, "'AK'", 4);
selectSize = addSelectOperator(selectPtr, QDBQEQ);
joinSize = addJoinTest(&joinHdr, joinSpec, "ACCTNUM", 1,
  "CUSTNUM" , 2, "EQ");
orderBySize = addOrderBy(&orderByHdr, orderByFld,
  "ITEMNAME", OFF);
addGroupBy(&groupByHdr, groupByFld, "LNAME", 0);
addGroupBy(&groupByHdr, groupByFld, "FNAME", 0);
addGroupBy(&groupByHdr, groupByFld, "ITEMCODE", 0);
addGroupBy(&groupByHdr, groupByFld, "ITEMNAME", 0);
groupBySize = addGroupBy(&groupByHdr, groupByFld, "STATUS", 0);


/* initialize the UFCB */
initUFCB(&ufcbBuf, QO_INPUT, NULL);
/* set up for sequential only processing since it is a group by */
ufcbBuf.qufcb.parameter.seqonly = SEQUPROC;
ufcbBuf.qufcb.parameter.seqonoff = ON;
ufcbBuf.qufcb.parameter.numonoff = ON;
ufcbBuf.qufcb.parameter.numrecs = 1;
```

```
    /* Now build the real QDT... */
    addQDTsection(qdtPtr, (char *) &fileHdr,
      sizeof(fileHdr), &qdtPtr->qdbqfilo);
    addQDTsection(qdtPtr, (char *) fileSpec, fileSpecSize, NULL);
    addQDTsection(qdtPtr, (char *) formatPtr,
      formatSize, &qdtPtr->qdbqfldo);
    addQDTsection(qdtPtr, (char *) &joinHdr,
      sizeof(joinHdr),&qdtPtr->qdbqjoio);
    addQDTsection(qdtPtr, (char *) joinSpec, joinSize, NULL);
    addQDTsection(qdtPtr, (char *) selectPtr,
      selectSize, &qdtPtr->qdbqselo);
    addQDTsection(qdtPtr, (char *) &orderByHdr, sizeof(orderByHdr),
      &qdtPtr->qdbqkeyo);
    addQDTsection(qdtPtr, (char *) orderByFld, orderBySize, NULL);
    addQDTsection(qdtPtr, (char *) &groupByHdr, sizeof(groupByHdr),
      &qdtPtr->qdbqgrpo);
    addQDTsection(qdtPtr, (char *) groupByFld, groupBySize, NULL);

    /* Finally, run the query! */
    QQQQRY("RUNQRY    ", (char *) &ufcbBuf, qdtBuf,
      NULL, NULL, &errcod);
    if (errcod.msgid[0])
    {
        printf("API QQQQRY failed\n");
        printf("msgid = %7s\n", strcnv400(errcod.msgid,
          sizeof(errcod.msgid)));
    }
    /* Now access the data */
    if ((file1 = fopen("OPENFILE1", "rb")) == NULL)
    {
        printf("Error opening file\n");
        exit(1);
    }

    /* Perform any record I/O here... */
    printf("Last name        First name  Code   \
Item                St\n");
    found = fread((void *) &recBuf, REC_SIZE, 1, file1);
    while (found)
    {
       recCount++;
       printf("%s  ", strcnv400(recBuf, 15));
       printf("%s  ", strcnv400(&recBuf[15], 10));
       printf("%s  ", strcnv400(&recBuf[25], 5));
       printf("%s  ", strcnv400(&recBuf[30], 20));
       printf("%s\n", strcnv400(&recBuf[50], 2));
       found = fread((void *) &recBuf, REC_SIZE, 1, file1);
    }
    printf("%d records selected\n", recCount);

    /* Close the file */
    fclose(file1);

    /* close out the QDT file handle */
    system("RCLRSC");
}
```

Generating and Sending an Alert

The following ILE RPG program uses both alert APIs.

```
     H
     D*************************************************************
     D*************************************************************
```

```
D*
D*  Program Name: ALERTS
D*
D*  Programming Language: ILE RPG for OS/400
D*
D*  Description:  This program uses alert APIs.  First, it
D*                calls the Generate Alert (QALGENA) API to
D*                generate an alert without sending a message
D*                to QSYSOPR or QHST message queue.  Then it
D*                uses the Send Alert (QALSNDA) API to send
D*                the alert to the OS/400 alert manager.
D*
D*  Header Files Included: QUSEC - Error Code Parameter
D*
D*****************************************************************
D*****************************************************************
D*
D* Error Code parameter include
D*
D/COPY QSYSINC/QRPGLESRC,QUSEC
D*
D*
D* Miscellaneous data structure
D*
DRCVVAR           S            512
DRCVLEN           S              9B 0 INZ(%SIZE(RCVVAR))
DALERT_SIZE       S              9B 0
DMSG_FILE         S             20    INZ('QCPFMSG   QSYS')
DMSG_ID           S              7    INZ('CPA2601')
DMSG_DATA         S            100
DMSG_SIZE         S              9B 0 INZ(0)
DALERT_TYPE       S              1    INZ('L')
DORIGIN           S             10    INZ('ALERTS')
C*
C* Beginning of mainline
C*
C* Set error handling
C*
C                   EVAL      QUSBPRV = %SIZE(QUSEC)
C*
C* Start by generating an alert for a specific message
C*
C                   CALL      'QALGENA'
C                   PARM                    RCVVAR
C                   PARM                    RCVLEN
C                   PARM                    ALERT_SIZE
C                   PARM                    MSG_FILE
C                   PARM                    MSG_ID
C                   PARM                    MSG_DATA
C                   PARM                    MSG_SIZE
C                   PARM                    QUSEC
C*
C* If no error reported, send the generated alert
C*
C     QUSBAVL       IFEQ      0
C                   CALL      'QALSNDA'
C                   PARM                    RCVVAR
C                   PARM                    ALERT_SIZE
C                   PARM                    ALERT_TYPE
C                   PARM                    ORIGIN
C                   PARM                    QUSEC
C*
C* If error on send, then display the error message
C*
C     QUSBAVL       IFNE      0
C     QUSEI         DSPLY
C                   END
```

```
     C*
     C* If error on generation, then display the error message
     C*
     C                   ELSE
     C     QUSEI         DSPLY
     C                   END
     C*
     C                   EVAL      *INLR = '1'
     C                   RETURN
     C*
     C* End of MAINLINE
     C*
```

Diagnostic Reporting

The following example program illustrates the use of the Send Nonprogram Message API, QMHSNDM, the Receive Program Message API, QMHRCVPM, and the Change Exception Message API, QMHCHGEM. The program produces a diagnostic report of errors that occur when the QMHSNDM API is used to send a message to more than one message queue.

The program calls the QMHSNDM API to send a message to message queues that do not exist. The QMHSNDM API returns a generic exception message, CPF2469. This message indicates that the API also returned one or more diagnostic messages describing the errors. After the program receives the exception message and verifies that it is message CPF2469, it uses the QMHCHGEM API to handle the exception message. The QMHRCVPM API is used to receive the diagnostic messages. The program prints the exception message, the diagnostic messages, and the message help.

Diagnostic Report (DIAGRPT) Program

```
/*************************************************************************/
/*                                                                       */
/* MODULE NAME:  DIAGRPT - Diagnostic Report                             */
/* LANGUAGE:     ILE C for OS/400                                        */
/*                                                                       */
/* FUNCTION:     This module will produce a diagnostic report that       */
/*               could be used in diagnosing the errors that             */
/*               occurred using the QMHSNDM API to send a message        */
/*               to multiple message queues.                             */
/*                                                                       */
/*               This program purposely causes the QMHSNDM API to        */
/*               try to send a message to message queues that do         */
/*               not exist.  As a result, the generic CPF2469            */
/*               exception is returned indicating that one or more       */
/*               diagnostic messages were returned identifying the       */
/*               error(s) on the send operation.                         */
/*                                                                       */
/*               The program looks for and handles the CPF2469           */
/*               exception.  It then receives and prints out the         */
/*               exception and the previous diagnostics.                 */
/*                                                                       */
/* Dependency:   A print file must be created before calling             */
/*               program DIAGRPT.  The print file should be created      */
/*               using the following command:                            */
/*                                                                       */
/*               CRTPRTF FILE(PRTDIAG) CTLCHAR(*FCFC)                    */
/*                       CHLVAL((1 (13)))                                */
/*************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <except.h>
#include <qmhchgem.h>                    /* From QSYSINC/H        */
```

```
#include <qmhrcvpm.h>                    /* From QSYSINC/H         */
#include <qmhsndm.h>                     /* From QSYSINC/H         */
#include <qusec.h>                       /* From QSYSINC/H         */

#define  DIAG_TYPE "02"
#define  BUF_SIZE  80

/*****************************************************************/
/* Type definition for error code structure                      */
/*****************************************************************/
typedef struct error_code_struct
  {
     Qus_EC_t   ec_fields;
     char       Exception_Data[100];
  } error_code_struct;

/*****************************************************************/
/* Type definition for qualified name structure                  */
/*****************************************************************/
typedef struct qual_name_struct
  {
   char  name[10];
   char  libr[10];
  } qual_name_struct;

/*****************************************************************/
/* Type definition for message information structure used on the  */
/* receive.  F is the fixed portion of the record  and V is the  */
/* variable length portion of the record.                        */
/*****************************************************************/
typedef struct msg_info_struct
  {
   Qmh_Rcvpm_RCVM0200_t   F;
   char                   V[1200];
  } msg_info_struct;

FILE  *prtf;
char  buf[80];
char  received[7];
int   exception_count;


/*****************************************************************/
/* Function to handle errors received on the API calls.          */
/*****************************************************************/
static void excp_handler(_INTRPT_Hndlr_Parms_T *excp_info)
 {
   error_code_struct Error_Code;

   /* If the exception is CPF2469, increment the exception counter,  */
   /* and mark the exception as handled by the QMHCHGEM API         */

   if (strncmp(excp_info->Msg_Id,"CPF2469",7) == 0) {
      memcpy(received,(excp_info->Msg_Id),7);
      exception_count++;
      QMHCHGEM(&(excp_info->Target), 0,
               (char *)(&(excp_info->Msg_Ref_Key)),
               "*HANDLE   ", "", 0, &Error_Code);
   }
 }


/*****************************************************************/
/* BuildQList:  Routine to build the message queue list.         */
/*****************************************************************/
void BuildQList(  qual_name_struct *QueueList, int NumQueue)
{
```

```
  int i;

  strncpy(QueueList[0].name,"QPGMR     ",10);
  strncpy(QueueList[1].name,"SNOOPY    ",10);
  strncpy(QueueList[2].name,"QSECOFR   ",10);
  strncpy(QueueList[3].name,"PEANUTS   ",10);
  strncpy(QueueList[4].name,"QUSER     ",10);

  for (i = 0; i < NumQueue ; i++ )
    {
     strncpy(QueueList[i].libr,"*LIBL     ",10);
    }
}


/*********************************************************************/
/* PrintError: Routine to print error information and exit.          */
/*********************************************************************/
void PrintError(char *errstring, char exception[7])
{

  memset(buf,' ',BUF_SIZE);
  buf[0] = '0';
  strncpy(buf+1,errstring,strlen(errstring));
  fwrite(buf,1,BUF_SIZE,prtf);

  memset(buf,' ',BUF_SIZE);
  buf[0] = '0';
  strncpy(buf+1,"Exception received->",20);
  strncpy(buf+21,exception,strlen(exception));
  fwrite(buf,1,BUF_SIZE,prtf);
  fclose(prtf);
  exit(1);
 }


/*********************************************************************/
/* PrintData: Routine to print varying length character string data.*/
/*********************************************************************/
void PrintData(char *strname, void *strptr, int strlgth)
{
  char *strdata = strptr;
  int i,lgth,remain;

  /* Write the description and the data that will fit on one line   */
  memset(buf,' ',BUF_SIZE);
  buf[0] = '0';
  lgth = strlen(strname);
  strncpy(buf+1,strname,lgth);
  lgth++;

  /* remain = MIN(strlgth,80 - lgth)  */
  remain = (strlgth < 80 - lgth) ? strlgth : 80 - lgth;
  strncpy(buf+lgth,strdata,remain);
  fwrite(buf,1,BUF_SIZE,prtf);

  /* Now write the remainder of the data */
  if (strlgth > (80 - lgth ))
    {
     /* Adjust pointer to data not printed yet */
     strdata = strdata + (80 - lgth);

     for (i = 0; i < strlgth; i = i + 70, strdata = strdata + 70 )
       {
        /* lgth = MIN(strlgth-i,70)  */
        lgth = (strlgth-i < 70) ? strlgth-i : 70;
```

```
        memset(buf,' ',BUF_SIZE);
        strncpy(buf,"0         ",10);
        memcpy(buf+10,strdata,lgth);
        fwrite(buf,1,BUF_SIZE,prtf);
       }

    }
}


/*********************************************************************/
/* PrintMessage: Routine to print the message data and text.         */
/*********************************************************************/
void PrintMessage(msg_info_struct *Msg)

{
  char *DataPtr;     /* Pointer to the varying length character data*/
  int  DataLen;      /* Length of the varying length character data */
  char CharType[10]; /* Message type as a string  */

  PrintData("Message ID->",Msg->F.Message_Id,7);
  /* Convert Message Type to a character string to be printed out   */
  if (memcmp(Msg->F.Message_Type,"02",2)==0)
    strncpy(CharType,"DIAGNOSTIC", 10);
  else if (memcmp(Msg->F.Message_Type,"15",2)==0)
    strncpy(CharType,"ESCAPE    ",10);
  PrintData("Message Type->",CharType,10);

  /* First point to the beginning of the message data               */
  /* in the structure and get the length of data returned.          */
  DataPtr = Msg->V;
  DataLen = Msg->F.Length_Data_Returned;
  /* If there is non-blank data, print it out                       */
  if ((DataLen > 0) && (strspn(DataPtr," ") < DataLen))
    PrintData("Message data received->",DataPtr,DataLen);

  /* Point to the beginning of the message text field and get the   */
  /* length of message text returned.                               */
  DataPtr += DataLen;
  DataLen = Msg->F.Length_Message_Returned;
  /* If there is non-blank text, print it out                       */
  if ((DataLen > 0) && (strspn( DataPtr," ") < DataLen))
    PrintData("Message text received->",DataPtr,DataLen);

  /* Now update to point to the beginning of the message            */
  /* help text field and get the length of message help text        */
  /* returned.                                                      */
  DataPtr += DataLen;
  DataLen = Msg->F.Length_Help_Returned;
  /* If there is non-blank message help text, print it out          */
  if ((DataLen > 0) && (strspn( DataPtr," ") < DataLen))
    PrintData("Message help text received->",DataPtr,DataLen);
  strncpy(buf,"-                                              ",43);
  fwrite(buf,1,BUF_SIZE,prtf);
}

 /*********************************************************************/
 /*                                                                   */
 /*  Start of main program.                                           */
 /*                                                                   */
 /*********************************************************************/

main()
{


 error_code_struct ErrorCode;
```

```
qual_name_struct  MsgQList[5];
qual_name_struct  MsgFile;
qual_name_struct  RpyMsgQ;

msg_info_struct   MsgInfo;

char MsgData[128];
char MsgText[512];
char MsgHelp[512];
char PgmMsgQ[10];
char MsgType[10];
char MsgAction[10];
char Format[8];
char MsgId[7];
char MsgKey[4];

int  MsgTextLen;
int  MsgInfoLen;
int  NumMsgQ;
int  PgmCount;
int  WaitTime;
int  morediag;


/* Initialize variables */
exception_count = 0;
memcpy(ErrorCode.ec_fields.Exception_Id,"       ",7);
ErrorCode.ec_fields.Bytes_Provided = 0;

memcpy(MsgId,"       ",7);
memcpy(MsgFile.name,"          ",10);
memcpy(MsgFile.libr,"          ",10);
strcpy(MsgText,"This is an immediate, informational message");
MsgTextLen = strlen(MsgText);
memcpy(MsgType,"*INFO     ",10);
memcpy(RpyMsgQ.name,"          ",10);
memcpy(RpyMsgQ.libr,"          ",10);

/* Build the list of message queues to send the message to         */
NumMsgQ = 5;
BuildQList(MsgQList,NumMsgQ);


/* Enable the exception handler around the call to QMHSNDM         */
#pragma exception_handler(excp_handler, 0, 0, _C2_MH_ESCAPE)

/* Send the message to the list of message queues.                 */
QMHSNDM( MsgId,
        &MsgFile,
         MsgText,
         MsgTextLen,
         MsgType,
        &MsgQList,
         NumMsgQ,
        &RpyMsgQ,
        &MsgKey,
        &ErrorCode);

/* Disable the exception handler                                   */
#pragma disable_handler

/* If an error occurred on the send, produce an exception report   */
/* identifying what errors occurred.                               */
if (exception_count != 0)
  {
```

```
/* Open printer file using first character forms control and    */
/* write the header information.                                */
prtf = fopen ("PRTDIAG", "wb type=record recfm=FA lrecl=80");
memset(buf,' ',BUF_SIZE);
strncpy(buf,"1                          DIAGNOSTIC REPORT",43);
fwrite(buf,1,BUF_SIZE,prtf);
strncpy(buf,"                           -----------------",43);
fwrite(buf,1,BUF_SIZE,prtf);
strncpy(buf,"-                                           ",43);
fwrite(buf,1,BUF_SIZE,prtf);

/* Do the setup to first receive the exception signalled.       */
memcpy(Format,"RCVM0200",8);
memcpy(PgmMsgQ,"*         ",10);
memcpy(MsgType,"*EXCP     ",10);
memcpy(MsgKey,"    ",4);
memcpy(MsgAction,"*OLD      ",10);
PgmCount = 0;
WaitTime = 0;
MsgInfoLen = 1276;

/* Now change bytes_provided to 116 so that if any errors occur */
/* on the receive, the error information will be returned in the*/
/* error code structure instead of generating more exceptions   */
/* which will clutter up the program message queue.             */
ErrorCode.ec_fields.Bytes_Provided = 116;

/* Receive the last exception type message on the program       */
/* message queue                                                */
QMHRCVPM(&MsgInfo,
          MsgInfoLen,
          Format,
          PgmMsgQ,
          PgmCount,
          MsgType,
          MsgKey,
          WaitTime,
          MsgAction,
         &ErrorCode);

/* Test for any errors on the receive */
if (ErrorCode.ec_fields.Bytes_Available > 0)
  {
   PrintError("QMHRCVPM - Did not complete successfully",
              ErrorCode.ec_fields.Exception_Id);
  }

/* An exception message was received successfully.  Now see if  */
/* the message received is the same exception that was signalled*/
/* If not, there is an error.                                   */
if (strncmp(MsgInfo.F.Message_Id,received,7) != 0)
  {
   PrintError("QMHRCVPM - Wrong exception received",
              MsgInfo.F.Message_Id);
  }


/* The exception message was received successfully.             */
/* Print the message data and text for the exception message.   */
PrintMessage(&MsgInfo);

/* If the message was the generic CPF2469, there are one or     */
/* more diagnostic messages to go with the CPF2469 on the queue.*/
/* Receive the diagnostic messages previous to the CPF2469 until*/
/* a non-diagnostic message is received or there are no more    */
/* messages.                                                    */
if (strncmp(MsgInfo.F.Message_Id,"CPF2469",7) == 0)
```

```
      {
       memcpy(MsgType,"*PRV      ",10);
       memcpy(MsgKey,MsgInfo.F.Message_Key,4);
       morediag = 1;

       while(morediag)
         {
          /* Receive the previous diagnostic */
          QMHRCVPM(&MsgInfo,
                   MsgInfoLen,
                   Format,
                   PgmMsgQ,
                   PgmCount,
                   MsgType,
                   MsgKey,
                   WaitTime,
                   MsgAction,
                   &ErrorCode);

          /* Test for error on the receive */
          if (ErrorCode.ec_fields.Bytes_Available > 0)
            {
             PrintError("QMHRCVPM - Did not complete successfully",
                        ErrorCode.ec_fields.Exception_Id);
            }

          /* If bytes available = 0 OR the next message is not a    */
          /* diagnostic message, we are done.                       */
          if ((MsgInfo.F.Bytes_Available == 0) ||
              (strncmp(MsgInfo.F.Message_Type,DIAG_TYPE,2) != 0) )
            {
             morediag = 0;
            }
          else                /* A diagnostic was received         */
            {
             /* Print the message data and text for the diagnostic  */
             /* message                                             */
             PrintMessage(&MsgInfo);

             /* Now copy the message key of the diagnostic message  */
             /* received to the MsgKey parameter to use on the next */
             /* call to QMHRCVPM.                                   */
             memcpy(MsgKey,MsgInfo.F.Message_Key,7);
            }

         }                        /* End of while morediag = 1      */

       }                          /* End of if CPF2469 received     */

     /* Write trailer */
     memset(buf,' ',BUF_SIZE);
     strncpy(buf,"-                        END OF DIAGNOSTIC REPORT",48);
     fwrite(buf,1,BUF_SIZE,prtf);

     /* Close the print file */
     fclose(prtf);

    }                             /* End of if error on send        */

}                                 /* End mainline                   */
```

Printed Diagnostic Report

The DIAGRPT program produces a report like this:

```
Message ID->CPF2469
Message Type->ESCAPE
Message text received->Error occurred when sending message.
Message help text received->Recovery  . . . :   See messages
          previously listed for a description of the error.
          Correct the error, and then try the
          command again.

Message ID->CPF2403
Message Type->DIAGNOSTIC
Message data received->PEANUTS   *LIBL
Message text received->Message queue PEANUTS in *LIBL not found.
Message help text received->Cause . . . . . :   The message queue you
        specified was not found in the library you specified.  One
        of the following occurred: -- The queue name was not
        entered correctly. -- The queue does not exist in the
        specified library. -- You specified the wrong library name.
        Recovery  . . . :   Do one of the following and try the
        request again: -- Correct or change the message queue
        name or library name used in the message queue (MSGQ)
        parameter or the to-message queue (TOMSGQ) parameter.
        -- Create the message queue using the Create Message
        Queue (CRTMSGQ) command.


Message ID->CPF2403
Message Type->DIAGNOSTIC
Message data received->SNOOPY    *LIBL
Message text received->Message queue SNOOPY in *LIBL not found.
Message help text received->Cause . . . . . :   The message queue
          you specified was not found in the library you specified.
          One of the following occurred: -- The queue name was not
          entered correctly. -- The queue does not exist in the
          specified library. -- You specified the wrong library
          name. Recovery  . . . :   Do one of the following and
          try the request again: -- Correct or change the message
          queue name or library name used in the message queue
          (MSGQ) parameter or the to-message queue (TOMSGQ)
          parameter. -- Create the message queue using the Create
          Message Queue (CRTMSGQ) command.
                     End of Diagnostic Report
```

Listing Directories

You should call this program with only one parameter, the parameter that represents the directory you want to list.

```
/*****************************************************************/
/*****************************************************************/
/* FUNCTION:  This program lists a directory to a spooled file.   */
/*                                                                */
/* LANGUAGE:  ILE C                                    */
/*                                                                */
/*                                                                */
/* APIs USED:  QHFOPNDR, QHFRDDR, QHFCLODR, QHFLSTFS, QUSCRTUS,   */
/*             QUSRTVUS                                           */
/*                                                                */
/*****************************************************************/
/*****************************************************************/
```

```
/*****************************************************************/
/* INCLUDE FILES                                                 */
/*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <qhfopndr.h>
#include <qhfrddr.h>
#include <qhfclodr.h>
#include <qhflstfs.h>
#include <quscrtus.h>
#include <qusrtvus.h>
#include <qusec.h>

/*****************************************************************/
/* STRUCTURE AND VARIABLE DECLARATIONS                           */
/*****************************************************************/

/*****************************************************************/
/* Parameters for QHFOPNDR                                       */
/*****************************************************************/
char   dir_handle[16];           /* Directory handle             */
int    namelen;                  /* Length of path name          */
char   openinfo[6];              /* Open information             */

typedef struct {
        Qhf_Attr_Selec_Tbl_t fixed;
        int                  offset2;
        int                  offset3;
        int                  att_len1;
        char                 att_name1[8];
        int                  att_len2;
        char                 att_name2[8];
        int                  att_len3;
        char                 att_name3[8];
} selection_struct;

selection_struct select;
int              selectionlen;

/*****************************************************************/
/* Error Code Structure                                          */
/*                                                               */
/* This shows how the user can define the variable length portion */
/* of error code for the exception data.                         */
/*                                                               */
/*****************************************************************/
typedef struct {
     Qus_EC_t   ec_fields;
     char       Exception_Data[100];
        } error_code_t;

error_code_t  error_code;

/*****************************************************************/
/* Parameters for QHFRDDR                                        */
/*****************************************************************/
/* The directory handle is the same as for QHFOPNDR              */

typedef struct {
        Qhf_Data_Buffer_t   fixed;
        int                 num_att;
        int                 offsets[4];
        char                attinfo[276];
} read_buffer;

read_buffer buffer;
```

```
int    result_count;
int    bytes_returned;

/*****************************************************************/
/* Parameters for QHFCLODR                                       */
/*****************************************************************/
/* No additional ones need to be declared                        */

/*****************************************************************/
/* Parameters for QUSCRTUS                                       */
/*****************************************************************/
int    size;
char   text[50];

/*****************************************************************/
/* Parameters for QHFLSTFS                                       */
/*****************************************************************/
/* No additional ones need to be declared                        */

/*****************************************************************/
/* Parameters for QUSRTVUS                                       */
/*****************************************************************/
int    startpos;
int    len;
char   charbin4[4];
char   FSname[10];

/*****************************************************************/
/* Other declarations                                            */
/*****************************************************************/
int    entrypos;
int    numentries;
int    entrylen;
char   *att;
char   name[100];
char   attname[30];
char   attval[30];
int    attnamelen;
int    attvallen;
char   newname[30];
int    filesize;
char   fileatt[10];

typedef struct {
        char     century;
        char     year[2];
        char     month[2];
        char     day[2];
        char     hour[2];
        char     minute[2];
        char     second[2];
} charval;

charval chartime;

int    bytes_used;
int    i;

main(int argc, char *argv[])
{
  char    write_string[100];
  FILE    *stream;

  error_code.ec_fields.Bytes_Provided = 0;

  /*****************************************************************/
  /* Make sure we received the correct number of parameters. The  */
```

```
/* argc parameter will contain the number of parameters that    */
/* was passed to this program. This number also includes the    */
/* program itself, so we need to evaluate argc-1.               */
/****************************************************************/

if (((argc - 1) < 1) || ((argc - 1 > 1)))
/****************************************************************/
/* We did not receive all of the required parameters, or        */
/* received too many. Exit from the program.                    */
/****************************************************************/
{
  exit(1);
}

/****************************************************************/
/* Open QPRINT file so that data can be written to it.  If the  */
/* file cannot be opened, print a message and exit.             */
/****************************************************************/
if((stream = fopen("QPRINT", "wb")) == NULL)
{
  printf("File could not be opened\n");
  exit(1);
}

memset(name, ' ', 100);
memcpy(name, argv[1], 100);
if(!memcmp(name, " ", 1))
{
memcpy(name,"ROOT",4);
fprintf(stream,"Directory listing for path %.100s\n", name);
size = 1;
memcpy(text, "temporary user space used by program DIR          ",
       50);
/****************************************************************/
/* Create the user space for QHFLSTFS to use.                   */
/****************************************************************/
QUSCRTUS("FSLST     QTEMP     ", "TEMPSPACE ", size, " ",
         "*USE      ", text, "*YES      ", &error_code);

/****************************************************************/
/* List the file systems into that space.                       */
/****************************************************************/
QHFLSTFS("FSLST     QTEMP     ", "HFSL0100", &error_code);

/****************************************************************/
/* Get the starting point for the file system entries.          */
/****************************************************************/
startpos = 125;
len = 4;
QUSRTVUS("FSLST     QTEMP     ", startpos, len, charbin4,
         &error_code);

entrypos = *(int *)charbin4;

/****************************************************************/
/* Get the number of entries in the user space.                 */
/****************************************************************/
startpos = 133;
len = 4;

QUSRTVUS("FSLST     QTEMP     ", startpos, len, charbin4,
         &error_code);

numentries = *(int *)charbin4;

/****************************************************************/
/* Find the length of the entries.                              */
```

```
/*****************************************************************/
startpos = 137;
len = 4;

QUSRTVUS("FSLST     QTEMP     ", startpos, len, charbin4,
         &error_code);

entrylen = *(int *)charbin4;

/*****************************************************************/
/* Loop through the entries and get the names of the file        */
/* systems.                                                      */
/*****************************************************************/
for(i=0;i<numentries;++i)
{
   startpos = entrypos + 1;
   len = 10;
   QUSRTVUS("FSLST     QTEMP     ", startpos, len, FSname,
            &error_code);
   /*************************************************************/
   /* List the names into the spooled file.                     */
   /*************************************************************/
   sprintf(write_string," %.10s <DIR>", FSname);
   fprintf(stream, write_string);
   entrypos = entrypos + entrylen;
}

}
else
{
fprintf(stream,"Directory listing for path %.100s\n", name);
/*****************************************************************/
/* Build the attribute selection table for QHFOPNDR.             */
/*****************************************************************/
select.fixed.Number_Attributes = 3;
select.fixed.Offset_First_Attr = 16;
select.offset2 = 28;
select.offset3 = 40;
select.att_len1 = 8;
memcpy(select.att_name1, "QFILSIZE", 8);
select.att_len2 = 8;
memcpy(select.att_name2, "QCRTDTTM", 8);
select.att_len3 = 8;
memcpy(select.att_name3, "QFILATTR", 8);
selectionlen = 52;
memcpy(openinfo, "10    ", 6);

/*****************************************************************/
/* Find the length of the directory name.                        */
/*****************************************************************/
for(i=0;i<100;i++)
{
  if((name[i] == ' ') || (name[i] == '\x00'))
    break;
}
namelen = i;

/*****************************************************************/
/* Open the directory.                                           */
/*****************************************************************/
QHFOPNDR(dir_handle, name, namelen, openinfo, &select, selectionlen,
         &error_code);

/*****************************************************************/
/* Read one entry from the directory.                            */
/*****************************************************************/
QHFRDDR(dir_handle, &buffer, 300, 1, &result_count, &bytes_returned,
```

```
        &error_code);

while(result_count > 0)
{
  memcpy(attname,"                                ",30);
  memcpy(attval,"                                ",30);
  att = buffer.attinfo;
  bytes_used = 20;

  /*************************************************************/
  /* Loop for the number of attributes in the entry.           */
  /*************************************************************/
  for(i=0;i<buffer.num_att;i++)
  {
    memcpy(charbin4, att, 4);
    attnamelen = *(int *)charbin4;
    att += 4;
    bytes_used += 4;
    memcpy(charbin4, att, 4);
    attvallen = *(int *)charbin4;
    att += 8;
    bytes_used += 8;
    memcpy(attname, att, attnamelen);
    att += attnamelen;
    bytes_used += attnamelen;
    memcpy(attval, att, attvallen);
    att += attvallen;
    bytes_used += attvallen;

    /*************************************************************/
    /* Update att so that its first character is the first       */
    /* character of the next attribute entry.                    */
    /*************************************************************/
    if ((bytes_used == buffer.offsets[i+1]) &&
        ((i+1) == buffer.num_att))
      att += (buffer.offsets[i] - bytes_used);

    /*************************************************************/
    /* If the attribute is QNAME, then set newname.              */
    /*************************************************************/
    if(!memcmp(attname, "QNAME", 5))
    {
      memset(newname, ' ', 12);
      memcpy(newname, attval, attvallen);
    }

    /*************************************************************/
    /* If the attribute is QFILSIZE, then set filesize.          */
    /*************************************************************/
    else if(!memcmp(attname, "QFILSIZE", 8))
    {
      memcpy(charbin4, attval, 4);
      filesize = *(int *)charbin4;
    }
    /*************************************************************/
    /* If it was QCRTDTTM, then set the time.                    */
    /*************************************************************/
    else if(!memcmp(attname, "QCRTDTTM", 8))
      memcpy(&chartime, attval, 13);
    /*************************************************************/
    /* Else the attribute was QFILATTR, so set fileatt.          */
    /*************************************************************/
    else
      memcpy(fileatt, attval, 10);
  }

  /*************************************************************/
```

```
    /* If the entry was a directory, list its name and <DIR>.     */
    /*************************************************************/
    if(fileatt[3] == '1')
    {
      sprintf(write_string," %s <DIR>", newname);
      fprintf(stream, write_string);
    }
    /*************************************************************/
    /* If the entry is not a hidden file, list its name and size. */
    /*************************************************************/
    else if(fileatt[1] == '0')
    {
      sprintf(write_string," %s %d", newname, filesize);
      fprintf(stream, write_string);
    }
    /*************************************************************/
    /* If the entry is not a hidden file or directory, list its   */
    /* date of creation.                                         */
    /*************************************************************/
    if(fileatt[1] == '0')
    {
      sprintf(write_string," %.2s-%.2s-%.2s", chartime.month,
              chartime.day, chartime.year);
      fprintf(stream, write_string);
      sprintf(write_string," %.2s:%.2s:%.2s\n", chartime.hour,
              chartime.minute, chartime.second);
      fprintf(stream, write_string);
    }

    QHFRDDR(dir_handle, &buffer, 200, 1, &result_count, &bytes_returned,
            &error_code);

  } /* while */

  } /* else */

  /*************************************************************/
  /* Close the directory.                                      */
  /*************************************************************/
  QHFCLODR(dir_handle, &error_code);

  fclose(stream);

} /* main */
```

Listing Subdirectories

You should call this program with only one parameter, the parameter that represents the directory you want to list.

```
/*************************************************************/
/*************************************************************/
/* FUNCTION:  List the subdirectories of the path passed to the   */
/*            program to a spooled file.                         */
/*                                                               */
/* LANGUAGE:  ILE C                                    */
/*                                                               */
/*                                                               */
/* APIs USED:  QHFOPNDR, QHFRDDR, QHFCLODR                       */
/*                                                               */
/*************************************************************/
/*************************************************************/
```

```
/*****************************************************************/
/* INCLUDE FILES                                                 */
/*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <qhfopndr.h>
#include <qhfrddr.h>
#include <qhfclodr.h>
#include <qusec.h>

char    write_string[100];
FILE    *stream;

void print_subdir(char name[100], int numtabs)
{

/*****************************************************************/
/* Parameters for QHFOPNDR                                       */
/*****************************************************************/
char   dir_handle[16];           /* Directory handle             */
int    namelen;                  /* Length of path name          */
char   openinfo[6];              /* Open information             */

typedef struct {
        Qhf_Attr_Selec_Tbl_t fixed;
        int                  att_len;
        char                 att_name[8];
} selection_struct;

selection_struct select;
int              selectionlen;

/*****************************************************************/
/* Error Code Structure                                          */
/*                                                               */
/* This shows how the user can define the variable length        */
/* portion of error code for the exception data.                 */
/*                                                               */
/*****************************************************************/
typedef struct {
     Qus_EC_t   ec_fields;
     char       Exception_Data[100];
        } error_code_t;

error_code_t  error_code;

/*****************************************************************/
/* Parameters for QHFRDDR                                        */
/*****************************************************************/
/* The directory handle is the same as for QHFOPNDR              */

typedef struct {
        Qhf_Data_Buffer_t   fixed;
        int                 num_att;
        int                 offsets[2];
        char                attinfo[180];
} read_buffer;

read_buffer buffer;
int    result_count;
int    bytes_returned;

/*****************************************************************/
/* Parameters for QHFCLODR                                       */
/*****************************************************************/
/* No additional ones need to be declared                        */
```

```
/*****************************************************************/
/* Other declarations                                            */
/*****************************************************************/
char   *att;
char   attname[30];
char   attval[30];
int    attnamelen;
int    attvallen;
char   newname[30];
int    newnamelen;
int    filesize;
char   fileatt[10];
char   tab[5];
int    bytes_used;
int    i, j;
char   charbin4[4];
char   tempname[100];

  error_code.ec_fields.Bytes_Provided = 0;

  /*****************************************************************/
  /* Build the attribute selection table for QHFOPNDR.             */
  /*****************************************************************/
  select.fixed.Number_Attributes = 1;
  select.fixed.Offset_First_Attr = 8;
  select.att_len = 8;
  memcpy(select.att_name, "QFILATTR", 8);
  selectionlen = 20;
  memcpy(openinfo, "10    ", 6);
  memcpy(tab,"     ", 5);

  /*****************************************************************/
  /* Find the length of the directory name.                        */
  /*****************************************************************/
  for(i=0;i<100;i++)
  {
    if((name[i] == ' ') || (name[i] == '\x00'))
      break;
  }
  namelen = i;

  /*****************************************************************/
  /* Open the directory.                                           */
  /*****************************************************************/
  QHFOPNDR(dir_handle, name, namelen, openinfo, &select, selectionlen,
           &error_code);

  /*****************************************************************/
  /* Read one entry from the directory.                            */
  /*****************************************************************/
  QHFRDDR(dir_handle, &buffer, 200, 1, &result_count, &bytes_returned,
          &error_code);

  fprintf(stream,"\n");
  for(i=0;i<numtabs;i++)
    fprintf(stream, tab);
  fprintf(stream, name);

  while(result_count > 0)
  {
    memcpy(attname,"                                ",30);
    memcpy(attval,"                               ",30);
    att = buffer.attinfo;
    bytes_used = 12;

    /*****************************************************************/
```

```
  /* Loop for the number of attributes in the entry.            */
  /*************************************************************/
  for(i=0;i<buffer.num_att;i++)
  {
    memcpy(charbin4, att, 4);
    attnamelen = *(int *)charbin4;
    att += 4;
    bytes_used += 4;
    memcpy(charbin4, att, 4);
    attvallen = *(int *)charbin4;
    att += 8;
    bytes_used += 8;
    memcpy(attname, att, attnamelen);
    att += attnamelen;
    bytes_used += attnamelen;
    memcpy(attval, att, attvallen);
    att += attvallen;
    bytes_used += attvallen;

    /*************************************************************/
    /* Update att so that its first character is the first       */
    /* character of the next attribute entry.                    */
    /*************************************************************/
    if ((bytes_used == buffer.offsets[i+1]) &&
        ((i+1) == buffer.num_att))
      att += (buffer.offsets[i] - bytes_used);

    /*************************************************************/
    /* If the attribute is QNAME, then set newname and           */
    /* newnamelen just in case the entry is a directory.         */
    /*************************************************************/
    if(!memcmp(attname, "QNAME", 5))
    {
      memcpy(newname, attval, attvallen);
      newnamelen = attvallen;
    }

    /*************************************************************/
    /* Else the attribute was QFILATTR, so set fileatt.          */
    /*************************************************************/
    else
      memcpy(fileatt, attval, 10);
  }

  /*************************************************************/
  /* If the entry was a directory, construct new path name and  */
  /* print_subdir to print the subdirectory.                   */
  /*************************************************************/
  if(fileatt[3] == '1')
  {
    memcpy(tempname, name, 100);
    strcat(name, "/");
    strcat(name, newname);
    memcpy(newname, name, namelen + newnamelen + 1);
    print_subdir(newname, numtabs + 1);
    memcpy(name, tempname, 100);
  }

  QHFRDDR(dir_handle, &buffer, 200, 1, &result_count, &bytes_returned,
          &error_code);

} /* while */

/*************************************************************/
/* Close the directory.                                      */
/*************************************************************/
QHFCLODR(dir_handle, &error_code);
```

```
}/* print_subdir */


main(int argc, char *argv[])
{
  char    dir_name[100];

  /************************************************************/
  /* Make sure we received the correct number of parameters. The  */
  /* argc parameter will contain the number of parameters that    */
  /* was passed to this program. This number also includes the    */
  /* program itself, so we need to evaluate argc-1.               */
  /************************************************************/

  if (((argc - 1) < 1) || ((argc - 1 > 1)))
  /************************************************************/
  /* We did not receive all of the required parameters, or        */
  /* received too many. Exit from the program.                    */
  /************************************************************/
  {
    exit(1);
  }

  /************************************************************/
  /* Open QPRINT file so that data can be written to it.  If the  */
  /* file cannot be opened, print a message and exit.             */
  /************************************************************/
  if((stream = fopen("QPRINT", "wb")) == NULL)
  {
    printf("File could not be opened\n");
    exit(1);
  }

  memset(dir_name, ' ', 100);
  memcpy(dir_name, argv[1], 100);
  if(!memcmp(dir_name, " ", 1))
  {
  fprintf(stream,"No directory specified");
  }
  else
  {
  fprintf(stream,"Directory substructure starting at %.100s", dir_name);
  print_subdir(dir_name, 0);
  }

  fclose(stream);

} /* main */
```

Saving to Multiple Devices

The following example program shows how to save a large library using more than one device at the same time.

```
/************************************************************/
/*  PROGRAM:  SaveBigLib                                    */
/*                                                          */
/*  LANGUAGE: ILE C for OS/400                              */
/*                                                          */
/*  DESCRIPTION:  This is an example program for the use of */
/*                a media definition in a save operation.   */
/*                It saves library BIGLIB in parallel format to  */
```

```
/*                two media files, using tape media library        */
/*                TAPMLB01.                                        */
/*                                                                 */
/*                The flow of this program is as follows:          */
/*                (1) Build media definition input.                */
/*                (2) Create a media definition using              */
/*                    QsrCreateMediaDefinition.                    */
/*                (3) Save library BIGLIB using the media          */
/*                    definition.                                  */
/*                                                                 */
/*  APIs USED:  QsrCreateMediaDefinition, QCMDEXC                  */
/*                                                                 */
/*******************************************************************/

#include <qcmdexc.h>
#include <qsrlib01.h>
#include <qusec.h>
#include <string.h>

/*******************************************************************/
/* Variables for QsrCreateMediaDefinition                          */
/*******************************************************************/
char                      Data_Buffer[1000];
Qsr_TAPE0100_t            *Input_Data;
Qsr_TAPE0100_Device_t     *Device;
Qsr_TAPE0100_File_t       *Media_File;
char                      *Next_Free;
char                      *Volid;
Qus_EC_t                  Err_Code;
int                       Data_Length;
char                      Text[50];

/*******************************************************************/
/* Variables for QCMDEXC                                           */
/*******************************************************************/
char          Cmd_String[100];
decimal(15,5) Cmd_Length;

/*******************************************************************/
/* Start of main()                                                 */
/*******************************************************************/

int main (int argc, char *argv[]) {

/*******************************************************************/
/*  Specify input data for QsrCreateMediaDefinition.               */
/*******************************************************************/
/*-----------------------------------------------------------------*/
/*  Build general media definition input data.                     */
/*  Use one device with two parallel device resources.             */
/*-----------------------------------------------------------------*/
memset(Data_Buffer,0,sizeof(Data_Buffer));
Input_Data = (Qsr_TAPE0100_t*)Data_Buffer;
Next_Free = (char*)(Input_Data + 1);
Input_Data->Maximum_Resources    = 2;
Input_Data->Minimum_Resources    = 2;
Input_Data->Offset_First_Device  = Next_Free - Data_Buffer;
Input_Data->Device_Count         = 1;

/*-----------------------------------------------------------------*/
/*  Build input data for the first device.                         */
/*  Use device TAPMLB01 with two media files.                      */
/*-----------------------------------------------------------------*/
Device = (Qsr_TAPE0100_Device_t*)Next_Free;
Next_Free = (char*)(Device + 1);
memcpy(Device->Device_Name,"TAPMLB01  ",10);
Device->Offset_First_File = Next_Free - Data_Buffer;
```

```
Device->File_Count        = 2;

/*-------------------------------------------------------------------*/
/*  Build input data for the first media file for device TAPMLB01. */
/*  Use the default sequence number, and volumes VOL11 and VOL12.  */
/*-------------------------------------------------------------------*/
Media_File = (Qsr_TAPE0100_File_t*)Next_Free;
Next_Free = (char*)(Media_File + 1);
Media_File->Sequence_Number        = 0;
Media_File->Offset_First_Volume_Id = Next_Free - Data_Buffer;
Media_File->Volume_Id_Count        = 2;
Media_File->Volume_Id_Length       = 6;
Media_File->Starting_Volume        = 1;
Data_Length = Media_File->Volume_Id_Count
            * Media_File->Volume_Id_Length;
Volid = Next_Free;
memcpy(Volid,"VOL11 VOL12 ",Data_Length);
if (Data_Length % 4)                       /* Ensure that Next_Free */
  Data_Length += (4 - (Data_Length % 4));  /* is incremented by a   */
Next_Free += Data_Length;                  /* multiple of 4.        */
Media_File->Offset_Next_File = Next_Free - Data_Buffer;

/*-------------------------------------------------------------------*/
/*  Build input data for the second media file for device TAPMLB01. */
/*  Use the default sequence number, and volumes VOL21 and VOL22.   */
/*-------------------------------------------------------------------*/
Media_File = (Qsr_TAPE0100_File_t*)Next_Free;
Next_Free = (char*)(Media_File + 1);
Media_File->Sequence_Number        = 0;
Media_File->Offset_First_Volume_Id = Next_Free - Data_Buffer;
Media_File->Volume_Id_Count        = 2;
Media_File->Volume_Id_Length       = 6;
Media_File->Starting_Volume        = 1;
Data_Length = Media_File->Volume_Id_Count
            * Media_File->Volume_Id_Length;
Volid = Next_Free;
memcpy(Volid,"VOL21 VOL22 ",Data_Length);
if (Data_Length % 4)                       /* Ensure that Next_Free */
  Data_Length += (4 - (Data_Length % 4));  /* is incremented by a   */
Next_Free += Data_Length;                  /* multiple of 4.        */

/*********************************************************************/
/*  Create the media definition.                                     */
/*********************************************************************/
Data_Length = Next_Free - Data_Buffer;
memset(Text,' ',sizeof(Text));
memcpy(Text,"Save BIGLIB",11);
QsrCreateMediaDefinition(
                  "SAVEBIGLIBQTEMP     ",  /* Media definition      */
                                           /*   name, library       */
                  Data_Buffer,             /* Input data            */
                  Data_Length,             /* Length of data        */
                  "TAPE0100",              /* Format name           */
                  "*USE      ",            /* Public authority      */
                  Text,                    /* Text description      */
                  1,                       /* Replace if it exists  */
                  &Err_Code);              /* Error code            */

/*********************************************************************/
/*  Save library BIGLIB using the media definition.                  */
/*********************************************************************/
strcpy(Cmd_String,
       "SAVLIB LIB(BIGLIB) DEV(*MEDDFN) MEDDFN(QTEMP/SAVEBIGLIB)");
Cmd_Length = strlen(Cmd_String);
QCMDEXC(Cmd_String,Cmd_Length);

return 0;
```

```
}
```

Scanning String Patterns

A typical use of the QCLSCAN API is to allow the work station user to retrieve all records that contain a specified pattern.

Example 1

Assume a 20-character database field containing only uppercase characters and the pattern 'ABC' is scanned for. The user program calls the QCLSCAN API for each database record read. The parameters would be as follows:

Field Name	Result
STRING	The 20-byte field to be scanned
STRLEN	20
STRPOS	1
PATTERN	'ABC'
PATLEN	3
TRANSLATE	'0'
TRIM	'0'
WILD	' '
RESULT	A value returned to your program

The following describes some fields and the results of the scan:

```
Scan  String               Result  Comments
 1    ABCDEFGHIJKLMNOPQRST   001
 2    XXXXABCXXXXXXXXXXXXX   005
 3    abcXXXXXXXXXXXXXXXX   000    Translation not requested
 4    XXXABCXXXXXABCXXXXXX   004    First occurrence found; see note
 5    ABABABABBCACCBACBABA   000    Not found
 6    ABABABCABCABCABCABCA   005
```

Note: In scan 4, the string has two places where the pattern could have been found. Since the STRPOS value is 1, the first value (position 004) was found. If the value of STRPOS had been 4, the result would still have been 004. If the STRPOS value had been in a range of 5 through 12, the result would have been 012.

Example 2

Assume a 25-character database field containing only uppercase characters and a user program that will prompt for the pattern to be scanned, which will not exceed 10 characters. The work station user is allowed to enter 1 through 10 characters to search with and trailing blanks will be trimmed from the pattern. The program would call the QCLSCAN program for each database record read. The program parameters would be as follows:

Field Name	Result
STRING	The 25-byte field to be scanned
STRLEN	25
STRPOS	1
PATTERN	Varies
PATLEN	10
TRANSLATE	'0'
TRIM	'1'

WILD	''
RESULT	A value returned to your program

The following describes some fields and the results of the scan:

```
Scan  String                      Pattern        Result  Comments
 1    ABCDEFGHIJKLMNOPQRSTUVWXY  'CDE        '   003
 2    ABCDEFGHIJKLMNOPQRSTUVWXY  'CDEFGH     '   003
 3    ABCDEFGHIJKLMNOPQRSTUVWXY  'CDEFGHIJKL '   003
 4    XXXXABCXXXXXXXXXXXXXXXXXX  'ABCD       '   000    Not found
 5    abcXXXXXXXXXXXXXXXXXXXXXX  'ABC        '   000    Not translated
 6    ABCXXXXXABC EXXXXXXXXXXXX  'ABC E      '   009
 7    XXXABCXXXXXABCXXXXXXXXXXX  'ABC        '   004    See note
```

Note: In scan 7, the string has two places where the pattern could be found. Since the STRPOS value is 1, only the first value (position 004) is found. If the value of STRPOS were 4, the result would still be 004. If the STRPOS value were in the range of 5 through 12, the result would be 012.

Example 3

Assume a 25-character database field containing either uppercase or lowercase characters. The user program prompts for the pattern to be scanned, which does not exceed 5 characters. The work station user can enter 1 through 5 characters to be found. The system trims trailing blanks from the pattern. If the user enters an asterisk (*) in the pattern, the asterisk is handled as a wild character. The program calls the QCLSCAN program for each database record read. The parameters are as follows:

Field Name	**Result**
STRING	The 25-byte field to be scanned
STRLEN	25
STRPOS	1
PATTERN	Varies
PATLEN	5
TRANSLATE	'1' (See note 1)
TRIM	'1'
WILD	'*'
RESULT	A value returned to your program

The following describes some fields and the results of the scan:

```
Scan  String                      Pattern      Result  Comments
 1    ABCDEFGHIJKLMNOPQRSTUVWXY  'CDE      '   003
 2    ABCDEFGHIJKLMNOPQRSTUVWXY  'C*E      '   003
 3    abcdefghijklmnopqrstuvwxy  'C***G     '   003    See note 1
 4    abcdefghijklmnopqrstuvwxy  'ABCD     '   001
 5    abcXXXXXXXXXXXXXXXXXXXXX  'C*E      '   000    Not found
 6    XXXAbcXXXXXabcXXXXXXXXXXX  'ABC      '   004    See note 2
 7    ABCDEFGHIJKLMNOPQRSTUVWXY  '*BC      '  -003    See note 3
 8    ABCDEFGHIJKLMNOPQRSTUVWXY  '         '  -004    See note 4
```

Notes:

1. When field translation is specified (the TRANSLATE parameter is specified as '1'), the string is translated to uppercase characters before scanning occurs; the data in the string is not changed.

2. In scan 6, the string has two places where the pattern could have been found. Since the STRPOS value is 1, the first value (position 004) was found.

3. In scan 7, the wild character (*) is the first character in the trimmed pattern. Wild characters cannot be the first character in a pattern.

4. In scan 8, the trimmed pattern is blank.

Using COBOL Program to Call APIs

This example COBOL program uses the example error handler in Error Handler for Example COBOL Program.

Note: In order for this example to run successfully, the error program, ACERRF24 (shown in Error Handler for Example COBOL Program), must exist in a library called UTCBL.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  ACF24.
      *************************************************************
      *************************************************************
      *
      * FUNCTION:  SHOWS HOW TO CALL THE VARIOUS APIs, WHILE
      *            TESTING THAT THEY WORK PROPERLY.
      *
      * LANGUAGE:  COBOL FOR OS/400
      *
      * APIs USED: QLRRTVCE, QLRCHGCM, QLRSETCE
      *
      *************************************************************
      *************************************************************
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.  IBM-AS400.
       OBJECT-COMPUTER.  IBM-AS400.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 old.
          05  oldname       PIC X(10).
          05  oldlibr       PIC X(10).
       77 scope             PIC X VALUE "P".
       01 errparm.
          05 input-l        PIC S9(6) BINARY VALUE ZERO.
          05 output-l       PIC S9(6) BINARY VALUE ZERO.
          05 exception-id   PIC X(7).
          05 reserved       PIC X(1).
          05 exception-data PIC X(50).
       01  new.
          05  newname       PIC X(10) VALUE "ACERRF24".
          05  newlibr       PIC X(10) VALUE "UTCBL".
       77  newlib           PIC X(10).
       PROCEDURE DIVISION.
       main-proc.
           DISPLAY "in ACF24".
           PERFORM variation-01 THRU end-variation.
           STOP RUN.
       variation-01.
      *************************************************************
      *                                                           *
      * This variation addresses the situation where there is no  *
      * pending COBOL main, so no pending error handler can exist. *
      *                                                           *
      *************************************************************
           DISPLAY "no pending so expect nothing but error LBE7052".
           MOVE SPACES TO old exception-id.
      *************************************************************
      * By setting error parm > 8, expect escape message          *
      * LBE7052 to be returned in error parameter.                *
      *************************************************************
           MOVE LENGTH OF errparm TO input-l.
           CALL "QLRRTVCE" USING old scope errparm.
```

```
     IF exception-id IS NOT =  "LBE7052" THEN
       DISPLAY "** error - expected LBE7052"
     ELSE
       DISPLAY "LBE7052 was found"
     END-IF.
*****************************************************************
* Reset input-l to ZERO,  thus any further errors will cause *
* COBOL program to stop.                                        *
*****************************************************************
     MOVE 0 TO input-l.
     MOVE SPACES TO old exception-id.
 variation-02.
*****************************************************************
*                                                               *
* This variation creates a pending run unit.  It then makes  *
* sure that no pending error handler has been set.           *
*                                                               *
*****************************************************************
     DISPLAY "create pending run unit".
     CALL "QLRCHGCM" USING errparm.
*****************************************************************
*                                                               *
* No pending error handler exists so *NONE should be         *
* returned.                                                     *
*                                                               *
*****************************************************************


     CALL "QLRRTVCE" USING old scope errparm.
     DISPLAY "Retrieved Error Handler is=" old.
     IF oldname IS NOT = "*NONE" THEN
       DISPLAY "** error - expected *NONE for error handler"
     END-IF.
     MOVE 0 TO input-l.
     MOVE SPACES TO old exception-id.
 variation-03.
*****************************************************************
*                                                               *
* This variation sets an error handler for the pending       *
* run unit and then does another check to make sure it       *
* was really set.                                               *
*                                                               *
*****************************************************************
     CALL "QLRSETCE" USING new scope newlib old errparm.
     IF oldname IS NOT = "*NONE"
       DISPLAY "** error in oldname "
     END-IF.
     IF newlib IS NOT = "UTCBL"
       DISPLAY "** error in new library "
     END-IF.
*****************************************************************
* Call the retrieve API to check to make sure that the       *
* set API worked.                                               *
*****************************************************************
     MOVE SPACES TO old exception-id.
     CALL "QLRRTVCE" USING old scope errparm.
     DISPLAY "Retrieved Error Handler is=" old.
     IF oldname IS NOT = "ACERRF24" OR oldlibr IS NOT = "UTCBL"
       DISPLAY "** error - expected ACERRF24 error handler"
     END-IF.
 end-variation.
```

Error Handler for Example COBOL Program

This example error handler works with Using COBOL Program to Call APIs.

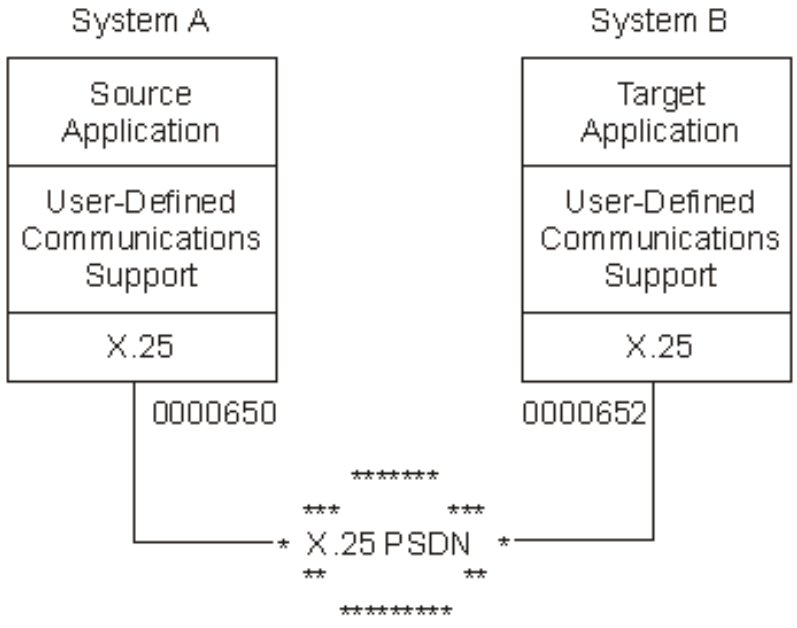
```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  ACERRF24.
*************************************************************
*************************************************************
*
* FUNCTION:  Error handler for preceding example COBOL program
*
* LANGUAGE:  COBOL FOR OS/400
*
* APIs USED: None
*
*************************************************************
*************************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  IBM-AS400.
 OBJECT-COMPUTER.  IBM-AS400.
 SPECIAL-NAMES. SYSTEM-CONSOLE IS SYSCON.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77 scope PIC X VALUE "P".
 01 errparm.
    05 FILLER PIC X(30).
 LINKAGE SECTION.
 77  cobol-id PIC X(7).
 77  valid-responses PIC X(6).
 01  progr.
     05  progname PIC X(10).
     05  proglibr PIC X(10).
 77  system-id PIC X(7).
 77  len-text PIC S9(9) COMP-4.
 01  subtext.
     03  subchars PIC X OCCURS 1 TO 230 TIMES
         DEPENDING ON len-text.
 77  retcode PIC X(1).
 PROCEDURE DIVISION USING cobol-id, valid-responses,
          progr, system-id, subtext, len-text, retcode.
 main-proc.
*********************************************************
* check for typical messages and take appropriate action *
*********************************************************
     EVALUATE cobol-id
     WHEN "LBE7604"
*********************************************************
* stop literal, let the user see the message             *
*********************************************************
       MOVE SPACE TO retcode
     WHEN "LBE7208"
*********************************************************
* accept/display, recoverable problem answer G to continue
*********************************************************
       MOVE "G" TO retcode
     WHEN OTHER
*********************************************************
* for all other messages signal system operator and      *
*  end the current run unit                              *
*********************************************************
       DISPLAY "COBOL Error Handler ACERRF24 "
               "Found message " cobol-id
               " Issued from program " progr
               UPON syscon
       DISPLAY " Ended current run unit"
```

```
            UPON syscon
  MOVE "C" TO retcode
END-EVALUATE.
GOBACK.
```

Using the User-Defined Communications Programs for File Transfer

This section provides a simple example showing how X.25-oriented applications use the user-defined communications support to connect to remote systems. Two user-defined application programs written in the ILE C programming language are used to illustrate a simple file transfer between two systems over an X.25 packet-switching data network (PSDN). Although an X.25 example is shown, many of the same concepts can be applied to applications running over token-ring and Ethernet local area networks (LANs). For the purposes of the examples, the APIs are referred to by their call names. The includes *header*, *hexconv*, and *typedefs* are not in QSYSINC. These includes are only documented in the examples.

For this example, the following network configuration will be used.



X.25 Overview

In this example X.25 network, the source application on System A is responsible for establishing a switched virtual circuit, or connection to the target application running on System B. This is done by using the remote network address (System B's address) of X'0000652'. When the target application on System B is initialized, it waits for notification of an incoming call packet before proceeding. Once the virtual circuit is established, the source application reads records from a file into its output buffer and sends them to the target application using normal X.25 data transfer procedures. While receiving the file data, the target application writes the data to a local file on System B. When the file transfer completes, the source application closes the connection by issuing an X.25 clear request packet and ends. When receiving the clear indication packet, the target application also ends.

User-Defined Communications Support Overview

Both the source and target applications call the Query Line Description (QOLQLIND) API to obtain information about the local X.25 line being used. This information is stored in a local control block for use in establishing the peer connection during X.25 connection processing. Both applications also call the Enable Link (QOLELINK) API to enable the link for future communications. The iSeries server line name, communications handle, and remote DTE address are passed to both programs as arguments to the C function main(). For simplicity, the user space names and data queue name on the call to the QOLELINK API are coded directly in the

applications.

Note: Keyed data queue support is used by both applications. The key length is 3 and the keys used are source (SRC) and target (TGT) for the source and target applications, respectively.

Activating Filters

Once the links have been enabled and both applications have read their respective enable-complete entries from their data queues, the target application program calls the Set Filter (QOLSETF) API to activate a filter. The filter activated then identifies the protocol of the local X.25 service user. This filter is used by the user-defined communications support on System B to route incoming calls. The actual filter type activated is X'00' (for X.25 PID) and its associated value is X'21'. For more information concerning filters, see . After activating the X'21' filter, the target application waits for the source application to request a connection.

Establishing a Connection

The source application calls the Send Data (QOLSEND) API with a X'B000' operation in its output data buffer to establish a switched virtual circuit (SVC) to the target application. Included in the first byte of the call user data is the protocol ID of the target application, or X'21'. When the user-defined communications support on System B sees the incoming call packet with the first byte of user data equal to a previously activated filter, the call is routed to the process responsible for activating that filter. In this case, the target application will receive notification of an incoming call since it previously activated filter X'21'.

While waiting for the incoming call, the target application calls the Receive Data (QOLRECV) API to receive a X'B201' operation with incoming call data. After doing so, the target application accepts the X.25 connection by calling the QOLSEND API with a X'B400' operation in its output data buffer. See for more information.

Sending Data

Once the peer connection is established between the source and target applications running on System A and System B respectively, the file transfer takes place. The source application reads records from a local file and calls the QOLSEND API with X'0000' operations in its output data buffer to transfer the file data to System B. This process continues until the entire contents of the source file has been sent to System B.

Receiving Data

After accepting the X.25 connection, the target application waits until its data queue receives incoming-data entries. When the first entry is read from the queue, the QOLRECV API is called to determine which operation was received. Barring failure, the target application should receive a X'0001' operation as a result of the QOLRECV API call. The data contained in the input data buffer is the file data received from System A. While receiving the file data, the target application writes the data to a local file. This process continues until the entire contents of the file is received from System A. The target application then assumes the file transfer is complete when an operation other than a X'0001' operation is received after a successful call to the QOLRECV API. Most likely, the first non-X'0001' operation received will be X'B301' operation, signalling that the user-defined communications support running on System B received an SVC clear indication.

Clearing the Connection and Disabling Links

Once the entire contents of the file has been read and sent to System B, the source application calls the QOLSEND API with a X'B100' operation in its output data buffer to clear the X.25 connection. Afterwards, the source application closes its local file, disables its local link by calling the QOLDLINK API, and ends.

When the source application program sends a X'B100' operation, it causes the target application to receive a X'B301' operation. After receiving this operation, the target application program calls the QOLSEND API with a X'B100' operation to locally close the connection between itself and the user-defined communications support. Afterwards, the target application closes its local file, disables its local link by calling the QOLDLINK API, and ends.

Using Timers and the Data Queue Support

Both the source and target application programs use the user-defined communications support timer service to manage the reception of certain operations. This is done by setting a timer before checking the data queue for an entry. For example, the target application sets a timer to manage the reception of file data from the source application. If the timer expires, the user-defined communications support places a timer-expired entry on the application's data queue. The target application then assumes when receiving this entry that the source application ended abnormally. The target application can then take the appropriate action to end itself.

ILE C Compiler Listings

Below are the listings for the source and target applications described in the previous paragraphs. Note the reference numbers (for example, **(1)**) in the listings. Detailed explanations of each reference number block are found in Source application program listing references and Target application program listing references.

The target application compiler listing can be found in Target Application on System B Listing.

Source Application on System A Listing

In this example, the source application is the initiator of all meaningful work. In summary, the source program listed on the following pages does the following:

- Calls the QOLQLIND API to get local X.25 line information
- Opens the local file
- Calls the QOLELINK API to establish a link for communications
- Calls the QOLSEND API with X'B000' operation to establish a peer (SVC) connection
- Sends the local file to the target system using X'0000' operations
- Calls the QOLSEND API with X'B100' operation to clear the peer (SVC) connection
- Calls the QOLDLINK API to disable the link
- Calls the QOLTIMER API to manage the reception of data queue entries

To create the program using ILE C, use the Create Bound C (CRTBNDC) command.

```
Program name  . . . . . :(SOURCE)
  Library name  . . . . . :     UDCS_APPLS
Source file . . . . . :   QCSRC
  Library name  . . . . . :     UDCS_APPLS
Source member name  . . . . . :   SOURCE
Text Description  . . . . . :   Source Application Example
Output  . . . . . :   *NONE
Compiler options  . . . . . :   *SOURCE  *NOXREF  *SHOWUSR
                              :   *SHOWSYS  *NOSHOWSKP  *NOEXPMAC
                              :   *NOAGR *NOPPONLY  *NODEBUG
                              :   *GEN *NOSECLVL  *PRINT  *LOGMSG
                              :   *USRINCPATH
Checkout Options  . . . . . :   *NOAGR
Optimization  . . . . . :   *NONE
Inline Options:
  Inliner . . . . . :   *OFF
  Mode  . . . . . :   *NOAUTO
  Threshold . . . . . :   250
  Limit . . . . . :   2000
Debugging View  . . . . . :   *NONE
Define Names  . . . . . :   *NONE
Language Level  . . . . . :   *SOURCE
Source Margins:
  Left margin . . . . . :   1
  Right margin  . . . . . :   32754
Sequence columns:
  Left column . . . . . :   *NONE
  Right column  . . . . . :
Message flagging level  . . . . :   0
Compiler messages:
  Message limit . . . . . :   *NOMAX
  Message limit severity  . . . :   30
Replace Program Object  . . . . :   *YES
```

```
 User Profile  . . . . . :   *USER
 Authority . . . . . :   *LIBCRTAUT
 Target Release  . . . . . :   *CURRENT
 System includes . . . . . :   *YES

/*****************************************************************/
/**  Program Name: Source Application Program Example           **/
/**                                                             **/
/**                                                             **/
/**  Function:                                                  **/
/**  This is the source application program example that uses   **/
/**  X.25 services provided by the user-defined communications  **/
/**  support to transfer a simple file to the target application **/
/**  program running on system B.  This program performs the    **/
/**  following:                                                 **/
/**     01.  Open the source file name INFILE.                  **/
/**     02.  Call QOLQLIND API to obtain local line information.**/
/**     03.  Enable a link.                                     **/
/**     04.  Send a 'B000'X operation (call request).           **/
/**     05.  Receive a 'B001'X operation (call confirmation).   **/
/**     06.  Read record(s) from the file opened in step 1). and**/
/**          send '0001'X operation(s) to transfer the file to  **/
/**          the target application program.                    **/
/**     07.  Send a 'B100'X operation (clear call request).     **/
/**     08.  Receive a 'B101'X operation.                       **/
/**     09.  Disable the link enabled in step 3).               **/
/**                                                             **/
/**  A data queue will be actively used to manage the operation **/
/**  of this program.  Data queue support will be used to monitor**/
/**  for the completion of the enable and disable routines, as  **/
/**  well as timer expirations and incoming data.  Timers are   **/
/**  used to ensure that there will never be an infinite wait on**/
/**  the data queue.  If a timer expires, the link enabled will **/
/**  be disabled and the program will stop.                     **/
/**                                                             **/
/** Inputs:                                                     **/
/** The program expects the following input parameters          **/
/**     Line Name:   This is the name of the line description   **/
/**                  that will be used to call the QOLELINK API.**/
/**                  The line must be an X.25 line with at least**/
/**                  one SVC of type *SVCBOTH or *SVCOUT.       **/
/**                                                             **/
/**     CommHandle:  This is the logical name that will be used **/
/**                  to identify the link enabled.              **/
/**                                                             **/
/**     Remote DTE Address:  The is the Local Network Address   **/
/**                          of System B.                       **/
/**                                                             **/
/**                                                             **/
/** Outputs:                                                    **/
/** Current status of the file transfer will be provided when   **/
/** running this program.  If an error should occur, then a     **/
/** message will be displayed indicating where the error occurred**/
/** and the program will end.        If the program completes   **/
/** successfully, a "successful completion" message will be     **/
/** posted.                                                     **/
/**                                                             **/
/** Language:  ILE C for OS/400                                 **/
/**                                                             **/
/** APIs used: QOLELINK, QUSPTRUS, QOLRECV, QOLSEND, QOLDLINK,  **/
/**            QOLTIMER, QRCVDTAQ                               **/
/**                                                             **/
/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
#include "header"
#include "typedef"
```

```
#include "hexconv"
(1)

/************   Typedef  Declarations      ******************/
(2)

void senddata(sendparms *a, char *b, desc *c, char *d, char *e, int f);
void sndformat1(sendparms *a,char *b, char *c, char *d, qlindparms *f);
void sndformat2 (sendparms *a, char *b, char *c);
void setfilters (hdrparms *a);
void byte (char *a, int b, char *c, int d);
void printespec (espec *a);
void settimer(unsigned short *a,char *b,qentry *c,usrspace *d,char *e);
void dequeue (int a, char *b, qentry *c, usrspace *d);
void x25lind (qlindparms *a, char *b);
int getline (char *a, int b, FILE *c);
void disablelink (disableparms *a, char *b, usrspace *c);
void handler (disableparms a, usrspace *b);

void _GetExcData(_INTRPT_Hndlr_Parms_T *parms);
/*********************************************************/
/***************   Start Main Program   ******************/
/*********************************************************/
main (int argc, char *argv[])
{
/************  Variable  Declarations      ******************/
  usrspace inbuff,       /*  Input Data Buffer */
           indesc,       /*  Input Buffer Descriptor  */
           outbuff,      /*  Output Data Buffer  */
           outdesc,      /*  Output Buffer Descriptor */
           qname;        /*  Data Queue  */
  int length,              /* Data Queue key length  */
      linesiz,             /* Length of line that is read in */
      i= 0;                /* counter  */
  unsigned short expctid;  /*  Message ID that is expected */
  char commhandle[10],     /*  Command Line Parameter  */
       *buffer,            /* Pointer to buffer     */
       rmtdte[18],         /* Remote DTE read in       */
       line[132],          /* Line to read in       */
       key[256];           /* Data Queue key identifier  */
  desc *descriptor;        /* Pointer to buffer descriptor  */
  /** definitions for the API functions  **/
  enableparms enable;
  disableparms disable;
  sendparms send;
  recvparms recv;
  setfparms setf;
  timerparms timer;
  qlindparms qlind;
  qentry dataq;
  hdrparms *header;

(3)

  /***---  Open the file to send to remote side        ----**/
  if ((fptr = fopen("UDCS_APPLS/INFILE(INFILE)", "r")) == NULL)
     {
     printf("Unable to open source input file in UDCS_APPLS LIB.\n");
     printf("The Program was terminated.\n\n");
     return;
     }
  /***---  Open the display file as our input screen.  ----**/
  if ((screen = fopen("ERRORSPEC", "ab+ type=record")) == NULL)
     {
     printf("Unable to open display file.\n");
     printf("The Program was terminated.\n\n");
     return;
```

```
   }
/** set the exception handler  **/

signal(SIGALL,SIG_DFL);
/**  Clear the command line Parameters  **/
strncpy(enable.linename, "          ", 10);  /* Clear linename */
strncpy(commhandle, "          ", 10);       /* Clear Commhandle*/
strncpy(rmtdte, "                ", 17);    /* Clear Remote DTE*/
/**  Receive command line Parameters  **/
strncpy(enable.linename, argv[1], strlen(argv[1]));
strncpy(commhandle, argv[2], strlen(argv[2]));
strncpy(rmtdte, argv[3], strlen(argv[3]));
rmtdte[strlen(argv[3])] = '\0';
/** Initialize the user spaces  **/
strncpy(inbuff.library, "UDCS_APPLS", 10);    /*  Input Buffer */
strncpy(inbuff.name, "SOURCEIBUF", 10);
strncpy(indesc.library, "UDCS_APPLS", 10);    /* Input B Desc  */
strncpy(indesc.name, "SOURCEBDSC", 10);
strncpy(outbuff.library, "UDCS_APPLS", 10);   /*  Output Buffer*/
strncpy(outbuff.name, "SOURCEOBUF", 10);
strncpy(outdesc.library, "UDCS_APPLS", 10);   /* Output B Desc */
strncpy(outdesc.name, "SOURCEODSC", 10);
strncpy(qname.library, "UDCS_APPLS", 10);   /*  Data queue  */
strncpy(qname.name, "X25DTAQ   ", 10);
/*****   retrieve the line description information  ******/
x25lind (&qlind, enable.linename);
if ((qlind.retcode != 0) || (qlind.reason != 0))
   {
   printf("Query line description failed.\n");
   printf("Return code = %d\n", qlind.retcode);
   printf("Reason code = %d\n\n", qlind.reason);
   return;
   }
/*****   Hard Code the QOLELINK Input Parameters  ******/
enable.maxdtax25 = 512;
enable.keylength = 3;
strncpy (enable.keyvalue, "SND", 3);
```

(4)

```
/************************************************/
/************  Enable the line  *****************/
/************************************************/
QOLELINK (&(enable.retcode), &(enable.reason), &(enable.tdusize),\
    &(enable.numunits), &(enable.maxdtalan), &(enable.maxdtax25),\
    (char *)&inbuff, (char *)&indesc, (char *)&outbuff,\
    (char *)&outdesc, &(enable.keylength), enable.keyvalue,\
    (char *)&qname, enable.linename, commhandle);
if ((enable.retcode != 0) || (enable.reason != 0))
   {
   printf("Line %.10s with Commhandle %.10s was NOT ENABLED.\n",\
           enable.linename, commhandle);
   printf("Return code = %d\n", enable.retcode);
   printf("Reason code = %d\n\n", enable.reason);
   return;
   }
```

(5)

```
/*-------- Set a timer for Enable Link  ---------**/
expctid = 0xF0F0;
settimer(&expctid, "Enable", &dataq, &qname, commhandle);
if (expctid != 0xF0F0)
   {
   disablelink (&disable, commhandle, &qname);
   return;
   }
```

(6)

```
/*****************************************************************/
/**************   Set up a Call Request Packet  *******************/
/*****************************************************************/
/****   Get pointers to the user spaces.   ******/
QUSPTRUS(&outbuff, &buffer);
QUSPTRUS(&outdesc, &descriptor);
send.ucep = 26;              /*  set the UCEP number  */
send.operation = 0xB000;     /*  send a call request  */
send.numdtaelmnts = 1;       /*  send one data unit   */
/**-----------    Send the packet   ---------**/
sndformat1 (&send, buffer, rmtdte, commhandle, &qlind);
if ((send.retcode != 0) || (send.reason != 0))
   {
   printf("Call request packet not sent\n");
   printf("Return code = %d\n", send.retcode);
   printf("Reason code = %d\n", send.reason);
   printf("new pcep %d\n", send.newpcep);
   printespec(&(send.errorspecific));
   disablelink (&disable, commhandle, &qname);
   return;
   }
```

(7)

```
/*****************************************************************/
/************   Receive the Call CONFIRMATION packet       ********/
/*****************************************************************/
/*-------- Set a timer to receive a message  ---------**/
expctid = 0xF0F3;
settimer(&expctid, "Rcv Call", &dataq, &qname, commhandle);
if (expctid != 0xF0F3)
   {
   disablelink (&disable, commhandle, &qname);
   return;
   }
QOLRECV (&(recv.retcode), &(recv.reason), &(recv.ucep),\
         &(recv.pcep), &(recv.operation), &(recv.numdtaunits),\
         &(recv.dataavail), &(recv.errorspecific), commhandle);
if ((recv.retcode != 0) || (recv.reason != 0))
   {
   printf("Recv Call reqst resp failed\n");
   printf("return code %d\n", recv.retcode);
   printf("reason code %d\n", recv.reason);
   printespec(&(send.errorspecific));
   disablelink (&disable, commhandle, &qname);
   return;
   }
/* Interpret the Received Operation  */
if (recv.operation != 0xB001)
   {
   printf("Recvd opr %x instead of opr B001\n", recv.operation);
   disablelink (&disable, commhandle, &qname);
   return;
   }
printf("We have an X.25 SVC connection\n\n");
```

(8)

```
/*****************************************************************/
/***************   Send the file to the target application *******/
/*****************************************************************/
send.pcep = send.newpcep;       /*  set the PCEP number  */
/***************   Send the Mbr LGRF in file DOC  ****************/
linesiz = getline(line, 92, fptr);   /* Get first record  **/
```

```
  while (linesiz != 0)
     {
     /***************   Send a Packet of Data    ***************/
     /****   Get pointers to the user spaces.   ******/
     QUSPTRUS(&outbuff, &buffer);
     QUSPTRUS(&outdesc, &descriptor);
     send.operation = 0x0000;
     send.numdtaelmnts = 1;
     /**-----   Send the packet     -------------**/
     senddata (&send, buffer, descriptor, commhandle, line, linesiz);
     if ((send.retcode != 0) || (send.reason != 0))
        {
        printf("Data NOT sent for commhandle %.9s\n", commhandle);
        printf("Return code = %d\n", send.retcode);
        printf("Reason code = %d\n", send.reason);
        printf("new pcep %d\n", send.newpcep);
        printespec(&(send.errorspecific));
        disablelink (&disable, commhandle, &qname);
        return;
        }
     i = i + 1;
     printf("Data %d Sent for commhandle %.9s.\n\n", i, commhandle);
     linesiz = getline(line, 92, fptr);  /** Get next record  **/
     }                          /***  End While loop  ***/
  /*************************************************************/
  /***************   Set up a Clear Request Packet  **************/
  /*************************************************************/
```

(9)

```
  /****   Get pointers to the user spaces.   ******/
  QUSPTRUS(&outbuff, &buffer);
  QUSPTRUS(&outdesc, &descriptor);
  send.operation = 0xB100;         /**  send clear request  **/
  send.numdtaelmnts = 1;           /**  send one data unit  **/
  /**-----------    Send the packet   ---------**/
  sndformat2 (&send, buffer, commhandle);
  if ((send.retcode != 0) || (send.reason != 0))
     {
     printf("Clear request packet not sent\n");
     printf("Return code = %d\n", send.retcode);
     printf("Reason code = %d\n", send.reason);
     printf("new pcep %d\n", send.newpcep);
     printespec(&(send.errorspecific));
     disablelink (&disable, commhandle, &qname);
     return;
     }
```

(10)

```
  /*************************************************************/
  /***********   Receive the Clear Request Response packet   *****/
  /*************************************************************/
  /*-------- Set a timer to receive a message  ---------**/
  expctid = 0xF0F3;
  settimer(&expctid, "Rv Clr Rqt", &dataq, &qname, commhandle);
  if (expctid != 0xF0F3)
     {
     disablelink (&disable, commhandle, &qname);
     return;
     }
  /***********   Call QOLRECV to Receive the Clear Response  *****/
  /****   Get pointers to the user spaces.   ******/
  QUSPTRUS (&inbuff, &buffer);
  QUSPTRUS (&indesc, &descriptor);
  QOLRECV (&(recv.retcode), &(recv.reason), &(recv.ucep),\
           &(recv.pcep), &(recv.operation), &(recv.numdtaunits),\
           &(recv.dataavail), &(recv.errorspecific), commhandle);
  if ((recv.retcode != 0) || (recv.reason != 0))
```

```
     {
     printf("Recv clear response failed\n");
     printf("return code %d\n", recv.retcode);
     printf("reason code %d\n", recv.reason);
     printespec(&(send.errorspecific));
     disablelink (&disable, commhandle, &qname);
     return;
     }
  /* Interpret the Received Operation  */
  if (recv.operation != 0xB101)
     {
     printf("Recvd opr %x instead of opr B101\n", recv.operation);
     disablelink (&disable, commhandle, &qname);
     return;
     }
  /*********************************************/
  /***  Disable the link and end the program  ****/
  /*********************************************/
  disablelink (&disable, commhandle, &qname);
  printf("******   SOURCE completed successfully   ******\n\n");
}    /* End Main */
/*****************************************************************/
/************** Start Subroutine Section    **********************/
/*****************************************************************/
/*****************************************************************/
/***************     Send a Packet of Data  ******************/
```

(11)

```
void senddata (sendparms *send,
               char *buffer,
               desc *descriptor,
               char *commhandle,
               char *line,
               int linesiz)
{
  descriptor->length = linesiz;
  descriptor->more = 0;
  descriptor->qualified = 0;
  descriptor->interrupt = 0;
  descriptor->dbit = 0;
  strncpy (buffer, line, linesiz);
  QOLSEND (&(send->retcode), &(send->reason), &(send->errorspecific),\
           &(send->newpcep), &(send->ucep), &(send->pcep), \
           commhandle, &(send->operation), &(send->numdtaelmnts));
}  /* End senddata Subroutine  */
/*****************************************************/
/************  Routine to fill X.25 Format I   ***********/
void sndformat1 (sendparms *send,
                 char *buffer,
                 char *rmtdte,
                 char *commhandle,
                 qlindparms *qlind)
{
  format1 *output = (format1 *) buffer;
  register int counter;
  register querydata *qd;
  qd = (querydata *)&(qlind->userbuffer);
  output->type = 2;                 /*  SVC used  */
  output->logchanid = 0x0;
  output->sendpacksize = qd->x25data.defsend;
  output->sendwindsize = qd->x25data.windowsend;
  output->recvpacksize = qd->x25data.defrecv;
  output->recvwindsize = qd->x25data.windowrecv;
  output->dtelength = strlen(rmtdte);
  byte(output->dte, 16, rmtdte, strlen(rmtdte));
  output->dbit = 0;
  output->cug = 0;
```

```
  output->cugid = 0;
  output->reverse = 0;
  output->fast = 0;
  output->faclength = 0;
  byte(output->facilities, 109, "", 0);
  output->calllength = 1;
  byte(output->callud, 128, "21", 2);   /* Contains Remote PID */
  output->misc[0] = 0;     /* change to 0x80 for reset support */
  output->misc[1] = 0;
  output->misc[2] = 0;
  output->misc[3] = 0;
  output->maxasmsize = 16383;
  output->autoflow = 32;
  QOLSEND (&(send->retcode), &(send->reason), &(send->errorspecific),\
           &(send->newpcep), &(send->ucep), &(send->pcep),\
           commhandle, &(send->operation), &(send->numdtaelmnts));
}  /* End sndformat1 Subroutine  */
/*********************************************************/
/************  Routine to fill X.25 Format II   ***********/
void sndformat2 (sendparms *send,
                 char *buffer,
                 char *commhandle)
{
  format2 *output = (format2 *) buffer;
  output->type = 1;
  output->cause = 'FF';
  output->diagnostic = 'FF';
  output->faclength = 0;
  byte(output->facilities, 109, "", 0);
  output->length = 0;
  byte(output->userdata, 128, "", 0);
  QOLSEND (&(send->retcode), &(send->reason), &(send->errorspecific),\
           &(send->newpcep), &(send->ucep), &(send->pcep),\
           commhandle, &(send->operation), &(send->numdtaelmnts));
}  /* End sndformat2 Subroutine  */
```

(12)

```
/*****************************************/
/********    Routine to disable   ***********/
void disablelink (disableparms *disable,
                  char *commhandle,
                  usrspace *qname)
{
unsigned short expctid;
qentry dataq;
  disable->vary = 1;     /*  Hard coded to be varied off  */
  QOLDLINK (&(disable->retcode), &(disable->reason),\
                        commhandle, &(disable->vary));
  if ((disable->retcode != 0) && (disable->reason != 00))
     {
     printf ("Link %.10s did not disabled.\n", commhandle);
     printf ("return code = %d\n", disable->retcode);
     printf ("reason code = %d\n\n", disable->reason);
     }
  /**-------  Set a timer to receive disable complete msg --------**/
  expctid = 0xF0F1;
  settimer(&expctid, "Disable", &dataq,  qname, commhandle);
  if (expctid != 0xF0F1)
     {
     printf("Disable link did not complete successfully");
     return;
     }
  printf ("%.10s link disabled \n", commhandle);
  /**  close the files  **/
  fclose(fptr);
  fclose(screen);
```

```
}  /* End disablelink Subroutine  */
/*************************************************************/
/**    Routine to convert string to Hexadecimal format   ******/
void byte (char *dest,
           int dlength,
           char *source,
           int slength)
{
  register int counter;
  char holder[2];
  for (counter=0;counter<dlength;counter++)
   dest[counter]=0;
  for (counter=slength-1;counter>=0;counter--)
   if (isxdigit(source[counter]))
     {
      holder[0]=source[counter];
      holder[1]='\0';
      if  (counter % 2 == 0)
        dest[counter/2] += (char) hextoint(holder)*16;
      else dest[counter/2] += (char) hextoint(holder);
     }
}  /* End byte Subroutine  */
/*************************************************************/
/**    Routine to display the ErrorSpecific output       ******/
void printespec(espec *errorspecific)
{
  especout outparms;

  sprintf(outparms.hwecode, "%.8X", errorspecific->hwecode);
  sprintf(outparms.timestamp, "%.8X%.8X", errorspecific->timestamphi,\
                                    errorspecific->timestamplo);
  sprintf(outparms.elogid, "%.8X", errorspecific->elogid);
  if (errorspecific->flags & 0x40)
    outparms.fail = 'Y';
  else outparms.fail = 'N';
  if (errorspecific->flags & 0x20)
    outparms.zerocodes = 'Y';
  else outparms.zerocodes = 'N';
  if (errorspecific->flags & 0x10)
    outparms.qsysopr = 'Y';
  else outparms.qsysopr = 'N';
  sprintf(outparms.cause,"%.2X", errorspecific->cause);
  sprintf(outparms.diagnostic, "%.2X", errorspecific->diagnostic);
  sprintf(outparms.erroffset, "%.6d", errorspecific->erroroffset);
  fwrite(&outparms, 1, sizeof(especout), screen);
  fread("", 0, 0, screen);
}  /* End printespec Subroutine  */
```

(13)

```
/*************   Set a timer and dequeue next entry  ******/
void settimer (unsigned short *expctid,
               char *process,
               qentry *dataq,
               usrspace *qname,
               char *commhandle)
{
timerparms timer;
disableparms disable;
int length;
char key[6];
  timer.interval = 20000;      /*  Set timer for 20 seconds */
  timer.establishcount = 1;
  timer.keylength = 3;          /*  Set key value  */
  strncpy(timer.keyvalue, "SRC", 3);
  timer.operation = 1;           /*  Set a timer  */
  QOLTIMER (&(timer.retcode), &(timer.reason), timer.handleout,\
           timer.handlein, (char *)qname, &(timer.operation),\
```

```
           &(timer.interval), &(timer.establishcount),\
           &(timer.keylength), timer.keyvalue, timer.userdata);
  if ((timer.retcode != 0) || (timer.reason != 0))
     {
     printf("%s timer failed while being set.\n", process);
     printf("Return code = %d\n", timer.retcode);
     printf("Reason code = %d\n\n", timer.reason);
     }
/**-------  Dequeue an entry  --------**/
  strncpy(key, "SRC",3);
  length = 3;
  dequeue (length, key, dataq, qname);
  /*** Cancel timer  ***/
  if (dataq->msgid != 0xF0F4)
     {
     strncpy(timer.handlein, timer.handleout, 8);
     timer.operation = 2;           /*  Set one timer  */
     QOLTIMER (&(timer.retcode), &(timer.reason), timer.handleout,\
           timer.handlein, (char *)qname, &(timer.operation),\
           &(timer.interval), &(timer.establishcount),\
           &(timer.keylength), timer.keyvalue, timer.userdata);
     if ((timer.retcode != 0) || (timer.reason != 0))
        {
        printf("%s timer failed while being canceled\n", process);
        printf("Return code = %d\n", timer.retcode);
        printf("Reason code = %d\n\n", timer.reason);
        }
     }
  if (dataq->msgid != *expctid)
      {
      printf ("A %.4X message ID was received instead of %.4X\n",\
              dataq->msgid, *expctid);
      printf ("%s completion message was not received\n", process);
      *expctid = dataq->msgid;
      }
}  /* End settimer Subroutine  */
/************************************************************/
/*******   Dequeues the Incoming Message and processes it ******/
void dequeue (int length,
              char *key,
              qentry *dataq,
              usrspace *qname)
{
  char fldlen[3],
       waittime[3],
       keylen[2],
       senderid[2],
       *pointer,
       order[2];
  register int counter;
  waittime[0] = 0;
  waittime[1] = 0;
  waittime[2] = 0x1D;   /*  Hard code a delay of infinite   */
  keylen[0] = 0;
  keylen[1] = 0x3F;   /* Hard code a keylength of 3  */
  senderid[0] = 0;
  senderid[1] = 0x0F;
  strncpy(order, "EQ", 2);
  fflush(stdin);
  pointer = (char *)dataq;
  for (counter = 0; counter < 336; counter++)
    pointer[counter] = 0;
  strncpy (dataq->type, "       ", 7);
  while ((strncmp(dataq->type, "*USRDFN", 7) != 0) || (fldlen == 0))
     QRCVDTAQ(qname->name, qname->library, fldlen, dataq, waittime,\
           order, keylen, key, senderid,"");
}  /* End dequeue Subroutine  */
```

(14)

```
/*********************************************************/
/**  x25lind:  Retrieve X.25 line description information  **/
void x25lind (qlindparms *qlind, char *linename)
{
register int counter;
  for(counter=0;counter<256;counter++)
     qlind->userbuffer[counter]=0;
  qlind->format = 0x01;
  QOLQLIND (&(qlind->retcode), &(qlind->reason), &(qlind->nbytes),\
            qlind->userbuffer, linename, &(qlind->format));
}  /*  End x25lind Subroutine  */
/*****************************************************/
/**  Getline:  Read a record into line and return length   **/
int getline (char *line, int max, FILE *fptr)
{
   if (fgets(line, max, fptr) == NULL)
      return 0;
   else
      return strlen(line);
}  /* End getline Subroutine  */
/*************************************************/
```

Source application program listing references

The following reference numbers and explanations correspond to the reference numbers in the source application's program listing.

(1) Some general C structure declarations used by both the source and target application programs.

(2) Function prototypes of the internal functions used in this program.

(3) Call the C library routines fopen() and signal() to open the source file and set up a signal handler to process OS/400 exceptions, respectively. An example of an exception would be accessing a data area with a NULL pointer. If an exception situation is encountered, SIG_DFL, the default handler, will be called in order for the program to end.

(4) Call the QOLQLIND API to retrieve local configuration information from the iSeries server line description about that will be used for communications. Next, call the QOLELINK API to enable the line description using the line name and communications handle passed as input parameters to this program.

(5) Call the QOLTIMER API to time the completion of the enable link operation. If the timer expires before the enable-complete entry is posted on the this program's data queue, then this program will end.

(6) Call the QOLSEND API with a X'B000' operation to establish a connection to the target application program.

(7) Monitor the source program's data queue for the call confirmation. The source program will be notified of the call confirmation by call the QOLRECV API and receiving a X'B001' operation in the program's input buffer.

(8) This is the main send loop for the source program. The data from the source file is placed one line at a time in the output buffer and then the QOLSEND API is called to send one data unit of the file to System B. This process repeats until the contents of the entire file have been transmitted to the target application.

(9) Call the QOLSEND API with a X'B100' operation to clear the peer connection.

(10) The source program will check its data queue for a response to the clear packet sent to the target system. Once the response is received, the program will clean up, call the QOLDLINK API to disable the link previously enabled, and end.

(11) The following C functions illustrate the various user-defined communications support APIs.

(12) This procedure illustrates a call to the QOLDLINK API. Note the vary option is set to vary off the associated iSeries server *USRDFN network device.

(13) The settimer() calls the QOLTIMER API requesting timers for 20000 milliseconds, or twenty seconds. After setting a timer, the settimer() will call the dequeue() to remove an entry from the program's data queue.

(14) The x25lind() illustrates calling the QOLQLIND API.

Target Application on System B Listing

The target application waits for the source application to initiate the file transfer. The following list summarizes the actions of the target application:

- Calls the QOLQLIND API to get local X.25 line information
- Opens the local file
- Calls the QOLELINK API to establish a link for communications
- Calls the QOLSETF API to activate an X.25 protocol ID filter
- Calls the QOLRECV API to receive the X'B201' operation (incoming call)
- Calls the QOLSEND API with a X'B400' operation to accept the SVC connection
- Receives the file from the target system using X'0001' operations
- Calls the QOLRECV API to receive the X'B301' (connection failure notification)
- Call the QOLSEND API with 'B100' operation to locally close the SVC connection
- Calls the QOLDLINK API to disable the link
- Calls the QOLTIMER API to manage the reception of data queue entries

To create the program using ILE C, use the Create Bound C (CRTBNDC) command.

Explanations of the reference numbers in the listing can be found in Target application program listing references.

```
Program name  . . . . . :(TARGET)
  Library name  . . . . . :      UDCS_APPLS
Source file . . . . . :   QCSRC
  Library name  . . . . . :      UDCS_APPLS
Source member name  . . . . . :   TARGET
Text Description  . . . . . :  Target Application Example
Output  . . . . . :   *NONE
Compiler options  . . . . . :   *SOURCE  *NOXREF *NOSHOWUSR
                            :   *NOSHOWSYS *NOSHOWSKP *NOEXPMAC
                            :   *NOAGR *NOPPONLY *NODEBUG
                            :   *GEN  *NOSECLVL  *PRINT  *LOGMSG
                            :   *USRINCPATH
Checkout Options  . . . . . :   *NOAGR
Optimization  . . . . . :   *NONE
Inline Options:
  Inliner . . . . . :   *OFF
  Mode  . . . . . :   *NOAUTO
  Threshold . . . . . :   250
  Limit . . . . . :   2000
Debugging View  . . . . . :   *NONE
Define Names  . . . . . :   *NONE
Language Level  . . . . . :   *SOURCE
Source Margins:
  Left margin . . . . . :   1
  Right margin  . . . . . :   32754
Sequence columns:
  Left column . . . . . :   *NONE
  Right column  . . . . . :
Message flagging level  . . . . :   0
Compiler messages:
  Message limit . . . . . :   *NOMAX
  Message limit severity  . . . :   30
Replace Program Object  . . . . :   *YES
User Profile  . . . . . :   *USER
Authority . . . . . :   *LIBCRTAUT
Target Release  . . . . . :   *CURRENT
System includes . . . . . :   *YES
```

```
/*****************************************************************/
/**                                                             **/
/**  Program Name: Target Application Program Example           **/
/**                                                             **/
/**                                                             **/
/**  Function:                                                  **/
/**  This is the target application program example that uses   **/
/**  X.25 services provided by the user-defined communications  **/
/**  support to receive a simple file from the source application **/
/**  program running on System A.  This program performs the    **/
/**  following:                                                 **/
/**     01.  Open the target file named OUTFILE.                **/
/**     02.  Call QOLQLIND to obtain local line information.    **/
/**     03.  Enable a link.                                     **/
/**     04.  Set a Filter on the enabled link.                  **/
/**     05.  Receive a 'B101'X operation (incoming call).       **/
/**     06.  Send a 'B400'X operation (accept call).            **/
/**     07.  Receive '0001'X operation(s) (incoming data) from  **/
/**          the source application program and write it to the **/
/**          file opened in step 1).                            **/
/**     08.  Receive a 'B301'X operation (clear call indication). **/
/**     09.  Send a 'B100'X operation to respond locally to the **/
/**          clearing of the connection.                        **/
/**     10.  Disable the link enabled in step 3).               **/
/**                                                             **/
/**  A data queue will be actively used to manage the operation **/
/**  of this program.  Data queue support will be used to monitor **/
/**  for the completion of the enable and disable routines, as  **/
/**  well as timer expirations and incoming data.  Timers are   **/
/**  used to ensure that there will never be an infinite wait on **/
/**  the data queue.  If a timer expires, the link enabled will **/
/**  be disabled and the program will stop.                     **/
/**                                                             **/
/**                                                             **/
/** Inputs:                                                     **/
/** The program expects the following input parameters:         **/
/**      Line Name:   This is the name of the line description  **/
/**                   that will be used to call the QOLELINK API. **/
/**                   The line must be an X.25 line with at least **/
/**                   one SVC of type *SVCBOTH or *SVCIN.       **/
/**                                                             **/
/**      CommHandle:  This is the logical name that will be used **/
/**                   to identify the link enabled.             **/
/**                                                             **/
/**      Remote DTE Address:  The is the Local Network Address  **/
/**                           of system A.                      **/
/**                                                             **/
/**                                                             **/
/** Outputs:                                                    **/
/** Current status of the file transfer will be provided when   **/
/** running this program. If an error should occur, then a      **/
/** message will be displayed indicating where the error occurred **/
/** and the program will end.        If the program completes   **/
/** successfully, a "successful completion" message will be     **/
/** posted.                                                     **/
/**                                                             **/
/** Language:  ILE C for OS/400                                 **/
/**                                                             **/
/** APIs used: QOLELINK, QUSPTRUS, QOLRECV, QOLSEND, QOLDLINK,  **/
/**            QRCVDTAQ, QOLTIMER                               **/
/**                                                             **/
/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
#include "header"
```

```
#include "typedef"
#include "hexconv"
void senddata(sendparms *a, char *b, desc *c, char *d, char *e, int f);
void sndformat1(sendparms *a,char *b, char *c, char *d, qlindparms *e);
void sndformat2 (sendparms *a, char *b, char *c);
void setfilters (hdrparms *a);
void byte (char *a, int b, char *c, int d);
void printespec (espec *a);
void settimer(unsigned short *a,char *b,qentry *c,usrspace *d,char *e);
void dequeue (int a, char *b, qentry *c, usrspace *d);
void putdata (char *a, int b, FILE *c);
void x25lind (qlindparms *a, char *b);
void disablelink (disableparms *a, char *b, usrspace *c);
void handler (disableparms a, usrspace *b);

void _GetExcData(_INTRPT_Hndlr_Parms_T *parms);
/*******************************************************/
/***************   Start Main Program   ****************/
/*******************************************************/
main (int argc, char *argv[])
{
/************  Variable  Declarations      ******************/
  usrspace inbuff,       /*  Input Data Buffer */
           indesc,       /*  Input Buffer Descriptor  */
           outbuff,      /*  Output Data Buffer  */
           outdesc,      /*  Output Buffer Descriptor */
           qname;        /*  Data Queue  */
  int length,               /* Data Queue key length  */
      inc, i, j;                  /*  counters  */
  unsigned short expctid;      /*  Message ID that is expected */
  char commhandle[10],    /*  Command Line Parameter  */
       rmtdte[17],        /*  Remote DTE Address  */
       *buffer,              /* Pointer to buffer     */
       key[256];          /* Data Queue key identifier  */
  desc *descriptor;           /* Pointer to buffer descriptor  */
/**  definitions for API functions  **/
  enableparms enable;
  disableparms disable;
  sendparms send;
  recvparms recv;
  setfparms setf;
  timerparms timer;
  qlindparms qlind;
  qentry dataq;
  hdrparms *header;
  /******  Annnddddddd....  they're off!!   **********/
(1)

  /***---  Open the file to put the received data.     ----**/
  if ((fptr = fopen("UDCS_APPLS/OUTFILE))", "w")) == NULL)
    {
     printf("Unable to open target output file in UDCS_APPLS LIB.\n");
     printf("The Program was terminated.\n\n");
     return;
    }
  /***---  Open the display file for error handling.   ----**/
  if ((screen = fopen("ERRORSPEC", "ab+ type = record")) == NULL)
    {
     printf("Unable to open display file.\n");
     printf("The Program was terminated.\n\n");
     return;
    }
  /***---  Set the Exception Handler        ----**/

  signal(SIGALL,SIG_DFL);
  /**  Clear the command line parameters  **/
  strncpy(enable.linename, "          ", 10);    /* Clear linename */
```

```
  strncpy(commhandle, "          ", 10);     /* Clear Commhandle */
  strncpy(rmtdte, "                ", 17);     /* Clear Remote DTE */
  /**  Receive command line Parameters  **/
  strncpy(enable.linename, argv[1], strlen(argv[1]));
  strncpy(commhandle, argv[2], strlen(argv[2]));
  strncpy(rmtdte, argv[3], strlen(argv[3]));
  rmtdte[strlen(argv[3])] = '\0';
  /**   Initialize the user spaces  **/
  strncpy(inbuff.library, "UDCS_APPLS", 10);    /*  Input Buffer */
  strncpy(inbuff.name, "TARGETIBUF", 10);
  strncpy(indesc.library, "UDCS_APPLS", 10);    /* Input B Desc  */
  strncpy(indesc.name, "TARGETIDSC", 10);
  strncpy(outbuff.library, "UDCS_APPLS", 10);   /*  Output Buffer*/
  strncpy(outbuff.name, "TARGETOBUF", 10);
  strncpy(outdesc.library, "UDCS_APPLS", 10);   /* Output B Desc */
  strncpy(outdesc.name, "TARGETODSC", 10);
  strncpy(qname.library, "UDCS_APPLS", 10);   /*  Data queue  */
  strncpy(qname.name, "X25DTAQ   ", 10);
  /*****   retrieve the line description information  ******/
  x25lind (&qlind, enable.linename);
  if ((qlind.retcode != 0) || (qlind.reason != 0))
     {
     printf("Query line description failed.\n");
     printf("Return code = %d\n", qlind.retcode);
     printf("Reason code = %d\n\n", qlind.reason);
     return;
     }
  /*****   Hard Code the QOLELINK Input Parameters  ******/
  enable.maxdtax25 = 512;
  enable.keylength = 3;
  strncpy(enable.keyvalue, "RCV", 3);
```

(2)

```
  /**-------  Enable the link  -----------**/
  QOLELINK (&(enable.retcode), &(enable.reason), &(enable.tdusize),\
      &(enable.numunits), &(enable.maxdtalan), &(enable.maxdtax25),\
      (char *)&inbuff, (char *)&indesc, (char *)&outbuff,\
      (char *)&outdesc, &(enable.keylength), enable.keyvalue,\
      (char *)&qname, enable.linename, commhandle);
  if ((enable.retcode != 0) || (enable.reason != 0))
     {
     printf("Line %.10s with Commhandle %.10s was NOT ENABLED.\n",\
             enable.linename, commhandle);
     printf("Return code = %d\n", enable.retcode);
     printf("Reason code = %d\n\n", enable.reason);
     return;
     }
```

(3)

```
  /**-------  Set a timer for Enable link  --------**/
  expctid = 0xF0F0;
  settimer(&expctid, "Enable", &dataq, &qname, commhandle);
  if (expctid != 0xF0F0)
     {
     disablelink (&disable, commhandle, &qname);
     return;
     }
  /*************************************************************/
  /*******----  Set a Filter for the Link  --------*********/
  /*************************************************************/
```

(4)

```
  QUSPTRUS(&outbuff, &header);   /*  get the output buffer pointer */
  header->function = 1;          /*  add a filter   */
  header->type = 0;              /*  X.25 PID only  */
  header->number = 1;            /*  set 1 filter   */
  header->length = 16;           /*  X.25 filter length  */
```

```
  setfilters(header);            /*  Fill in the filter format */
  /*******----  Set the filter for the Link  --------*********/
  QOLSETF (&(setf.retcode), &(setf.reason), &(setf.erroffset),\
              commhandle);
  if ((setf.retcode != 0) || (setf.reason != 0))
     {
     printf("Set Filters Return Code = %.2d\n", setf.retcode);
     printf("Set Filters Reason Codes = %.4d\n", setf.reason);
     printf("Set Filters Error Offset = %.4d\n", setf.erroffset);
     return;
     }
  /*************************************************************/
  /****  Receive the incoming call packet and accept the call **/
  /*************************************************************/
  /**-------  Set a timer to receive data  --------**/
  expctid = 0xF0F3;
  settimer(&expctid, "Inc Call ", &dataq, &qname, commhandle);
  if (expctid != 0xF0F3)
     {
     disablelink (&disable, commhandle, &qname);
     return;
     }
```

(5)

```
  /**********  Receive the Incoming Data     **********/
  QUSPTRUS (&inbuff, &buffer);
  QUSPTRUS (&indesc, &descriptor);
  QOLRECV (&(recv.retcode), &(recv.reason), &(recv.ucep),\
           &(recv.pcep), &(recv.operation), &(recv.numdtaunits),\
           &(recv.dataavail), &(recv.errorspecific), commhandle);
  if ((recv.retcode != 0) || (recv.reason != 0))
     {
     printf("Recv incoming call packet failed\n");
     printf("return code %d\n", recv.retcode);
     printf("reason code %d\n", recv.reason);
     printespec(&(send.errorspecific));
     disablelink (&disable, commhandle, &qname);
     return;
     }
  /*** Interpret the Received Operation  ***/
  if (recv.operation != 0xB201)
     {
     printf("Recvd operation %x instead of B201", recv.operation);
     disablelink (&disable, commhandle, &qname);
     return;         /****  End the program  ***/
     }
```

(6)

```
  /*****************************************************************/
  /** Send a response to accept the call and establish a connection */
  /*****************************************************************/
  /****   Get pointers to the user spaces.   ******/
  QUSPTRUS(&outbuff, &buffer);
  QUSPTRUS(&outdesc, &descriptor);
  /*******   Set up Send Packet    *********/
  send.ucep = 62;                   /*  set UCEP to be 62  */
  send.pcep = recv.pcep;            /*  get the PCEP number  */
  send.operation = 0xB400;          /*  send a call request response*/
  send.numdtaelmnts = 1;            /*  send one data unit   */
  /**-----   Send the packet   ----------------**/
  sndformat1 (&send, buffer, rmtdte, commhandle, &qlind);
  if ((send.retcode != 0) || (send.reason != 0))
     {
     printf("Data NOT sent for commhandle %.9s\n", commhandle);
     printf("Return code = %d\n", send.retcode);
     printf("Reason code = %d\n", send.reason);
     printf("new pcep %d\n\n", send.newpcep);
```

```
   printespec(&(send.errorspecific));
   disablelink (&disable, commhandle, &qname);
   return;
   }
printf("An X.25 SVC connection was completed\n\n");
```

(7)

```
/*********************************************************/
/****    Receive Incoming Data    ************************/
/*********************************************************/
/**-------  Set a timer to receive data  --------**/
expctid = 0xF0F3;
settimer(&expctid, "Inc Data ", &dataq, &qname, commhandle);
if (expctid != 0xF0F3)
   {
   disablelink (&disable, commhandle, &qname);
   return;
   }
/*******---  Receive the Incoming Data     ----******/
/** Get pointer to user space  **/
QUSPTRUS (&inbuff, &buffer);
QUSPTRUS (&indesc, &descriptor);
/**  Receive the data   **/
QOLRECV (&(recv.retcode), &(recv.reason), &(recv.ucep),\
       &(recv.pcep), &(recv.operation), &(recv.numdtaunits),\
       &(recv.dataavail), &(recv.errorspecific), commhandle);
if ((recv.retcode != 0) || (recv.reason != 0))
   {
   printf("Recv op for first data unit failed\n");
   printf("return code %d\n", recv.retcode);
   printf("reason code %d\n", recv.reason);
   printespec(&(send.errorspecific));
   disablelink (&disable, commhandle, &qname);
   return;
   }
```

(8)

```
/*************************************************************/
/*******   Start a loop to read in all the incoming data    ***/
/*************************************************************/
i = 1;
while (recv.operation == 0x0001)
   {
   printf("%d Data Recvd {%.4x}.\n\n", i++, recv.operation);
   /**  Store all the data units in the file  **/
   for (j = 1; j <= recv.numdtaunits; j++) {
      putdata (buffer + (j - 1)*enable.tdusize,\
               descriptor->length, fptr);
      descriptor = (desc *)((char *)descriptor + sizeof(desc));
      }   /* for */
   /**-------  Set a timer to wait for more data  -------**/
   if (recv.dataavail == 0)
      {
      /** Set timer  **/
      expctid = 0xF0F3;
      settimer(&expctid, "Wt Inc Dta", &dataq, &qname, commhandle);
      if (expctid != 0xF0F3)
         {
         disablelink (&disable, commhandle, &qname);
         return;
         }
      }
   /** Get pointer to user space  **/
   QUSPTRUS (&inbuff, &buffer);
   QUSPTRUS (&indesc, &descriptor);
```

```
    /**  Receive the data   **/
    QOLRECV (&(recv.retcode), &(recv.reason), &(recv.ucep),\
            &(recv.pcep), &(recv.operation), &(recv.numdtaunits),\
            &(recv.dataavail), &(recv.errorspecific), commhandle);
    }    /**  End  Receive data  while loop   ******/
```

(9)

```
  /*****************************************************/
  /*********** Receive the Clear indication ***********/
  /*****************************************************/
  if ((recv.retcode != 83) || (recv.reason != 4002))
     {
     printf("Recv opr for clear request failed\n");
     printf("return code %d\n", recv.retcode);
     printf("reason code %d\n", recv.reason);
     printespec(&(send.errorspecific));
     disablelink (&disable, commhandle, &qname);
     return;
     }
  /* Interpret the Received Operation  */
  if (recv.operation != 0xB301)
     {
     printf("Recvd operation %x instead of B301", recv.operation);
     disablelink (&disable, commhandle, &qname);
     return;         /****  end the program  ***/
     }
```

(10)

```
  /*****************************************************************/
  /***********  Send local response to clear indication ***********/
  /*****************************************************************/
  /****   Get pointers to the user spaces.   ******/
  QUSPTRUS(&outbuff, &buffer);
  QUSPTRUS(&outdesc, &descriptor);
  /*******   Set up the packet          ***************/
  send.operation = 0xB100;       /*  send a clear request packet  */
  send.numdtaelmnts = 1;         /*  send one data unit  */
  /**-----   Send the packet   ----------------**/
  sndformat2 (&send, buffer, commhandle);
  if ((send.retcode != 0) && (send.reason != 0))
     {
     printf("Response not sent for clear connection\n");
     printf("Return code = %d\n", send.retcode);
     printf("Reason code = %d\n", send.reason);
     printf("new pcep %d\n\n", send.newpcep);
     printespec(&(send.errorspecific));
     disablelink (&disable, commhandle, &qname);
     return;
     }
  /*****************************************************/
  /*********** Receive the Clear Confirmation  **********/
  /*****************************************************/
  /**-------  Set a timer to receive data  --------**/
  expctid = 0xF0F3;
  settimer(&expctid, "Clr Cnfrm", &dataq, &qname, commhandle);
  if (expctid != 0xF0F3)
     {
     disablelink (&disable, commhandle, &qname);
     return;
     }
  if ((recv.retcode != 00) || (recv.reason != 0000))
     {
     printf("Recv failed for clear confirmation\n");
     printf("return code %d\n", recv.retcode);
     printf("reason code %d\n", recv.reason);
```

```
     printespec(&(send.errorspecific));
     disablelink (&disable, commhandle, &qname);
     return;
     }
  /* Interpret the Received Operation  */
  if (recv.operation != 0xB101)
     {
     printf("Recvd opr %x instead of opr B301\n", recv.operation);
     disablelink (&disable, commhandle, &qname);
     return;
     }
```

(11)

```
  /*************************************/
  /**  disable the link and end program  **/
  /*************************************/
  disablelink (&disable, commhandle, &qname);
  printf("TARGET application completed OK!\n\n");
}    /* End Main */
/*****************************************************************/
/************** Start Subroutine Section    **********************/
/*****************************************************************/
/*****************************************************************/
/************  Routine to fill X.25 Format I   ***********/
void sndformat1 (sendparms *send,
                 char *buffer,
                 char *rmtdte,
                 char *commhandle,
                 qlindparms *qlind)
{
  format1 *output = (format1 *) buffer;
  register int counter;
  register querydata *qd;
  qd = (querydata *)&(qlind->userbuffer);
  output->type = 0;                 /*  not used  */
  output->logchanid = 0x0;
  output->sendpacksize = qd->x25data.defsend;
  output->sendwindsize = qd->x25data.windowsend;
  output->recvpacksize = qd->x25data.defrecv;
  output->recvwindsize = qd->x25data.windowrecv;
  output->dtelength = strlen(rmtdte);               /*  not used  */
  byte(output->dte, 16, rmtdte, strlen(rmtdte));    /*  not used  */
  output->dbit = 0;
  output->cug = 0;                                  /*  not used  */
  output->cugid = 0;                                /*  not used  */
  output->reverse = 0;                              /*  not used  */
  output->fast = 0;                                 /*  not used  */
  output->faclength = 0;
  byte(output->facilities, 109, "", 0);
  output->calllength = 0;
  byte(output->callud, 128, "00", 2);
  output->misc[0] = 0;
  output->misc[1] = 0;
  output->misc[2] = 0;
  output->misc[3] = 0;
  output->maxasmsize = 16383;
  output->autoflow = 32;
  QOLSEND (&(send->retcode), &(send->reason), &(send->errorspecific),\
           &(send->newpcep), &(send->ucep), &(send->pcep),\
           commhandle, &(send->operation), &(send->numdtaelmnts));
}  /*  End sndformat1 Subroutine  */
/*****************************************************************/
/************  Routine to fill X.25 Format II   ***********/
void sndformat2 (sendparms *send,
                 char *buffer,
                 char *commhandle)
```

```
{
  format2 *output = (format2 *) buffer;
  output->type = 1;
  output->cause = 'FF';
  output->diagnostic = 'FF';
  output->faclength = 0;
  byte(output->facilities, 109, "", 0);
  output->length = 0;
  byte(output->userdata, 128, "", 0);
  QOLSEND (&(send->retcode), &(send->reason), &(send->errorspecific),\
           &(send->newpcep), &(send->ucep), &(send->pcep),\
           commhandle, &(send->operation), &(send->numdtaelmnts));
}  /*  End sndformat2 Subroutine  */
/*****************************************************************/
/**************   Fill in the Buffer for the Filter  ***************/
void setfilters (hdrparms *header)
{
  x25filter *filters;
  filters = (x25filter *)header->filters;
  filters[0].pidlength = 1;
  filters[0].pid = 0x21;            /*  set the protocol ID  */
  filters[0].dtelength = 0;         /* no DTE used in filter */
  byte(filters[0].dte, 12, "", 0);
  filters[0].flags = 0x0;
  filters[0].flags += 0x80;      /* Set Reverse Charging to no  */
  filters[0].flags += 0x40;      /*  Set Fast Select to no  */
}  /*  End setfilters Subroutine  */
/*****************************************/
/********    Routine to disable   ***********/
void disablelink (disableparms *disable,
                  char *commhandle,
                  usrspace *qname)
{
  qentry dataq;
  unsigned short expctid;
  disable->vary = 1;     /*  Hard code device to vary off  */
  /**  Call disable link  **/
  QOLDLINK (&(disable->retcode), &(disable->reason),\
                        commhandle, &(disable->vary));
  if ((disable->retcode != 0) && (disable->reason != 00))
     {
     printf ("Link %.10s did not disabled.\n", commhandle);
     printf ("return code = %d\n", disable->retcode);
     printf ("reason code = %d\n\n", disable->reason);
     }
  else
     printf ("%.10s link disabled \n", commhandle);
  /**-------  Set a timer to receive message  --------**/
  expctid = 0xF0F1;
  settimer(&expctid, "Disable  ", &dataq, qname, commhandle);
  if (expctid != 0xF0F1)
     {
     printf("Disable link did not complete successfully");
     return;
     }
  /**  close the files  **/
  fclose(fptr);
  fclose(screen);
}  /*  End disablelink Subroutine  */
/*********************************************************/
/**    Routine to convert string to Hexadecimal format   ******/
void byte (char *dest,
           int dlength,
           char *source,
           int slength)
{
  register int counter;
```

```
  char holder[2];
  for (counter=0;counter<dlength;counter++)
   dest[counter]=0;
  for (counter=slength-1;counter>=0;counter--)
   if isxdigit(source[counter])
     {
      holder[0]=source[counter];
      holder[1]='\0';
      if  (counter % 2 == 0)
        dest[counter/2] += (char) hextoint(holder)*16;
      else dest[counter/2] += (char) hextoint(holder);
     }
}  /*  End byte Subroutine  */
/*************************************************************/
/**    Routine to display the ErrorSpecific output       ******/
/*************************************************************/
void printespec(espec *errorspecific)
{
  especout outparms;

  sprintf(outparms.hwecode, "%.8X", errorspecific->hwecode);
  sprintf(outparms.timestamp, "%.8X%.8X", errorspecific->timestamphi,\
                                          errorspecific->timestamplo);
  sprintf(outparms.elogid, "%.8X", errorspecific->elogid);
  if (errorspecific->flags & 0x40)
    outparms.fail = 'Y';
  else outparms.fail = 'N';
  if (errorspecific->flags & 0x20)
    outparms.zerocodes = 'Y';
  else outparms.zerocodes = 'N';
  if (errorspecific->flags & 0x10)
    outparms.qsysopr = 'Y';
  else outparms.qsysopr = 'N';
  sprintf(outparms.cause,"%.2X", errorspecific->cause);
  sprintf(outparms.diagnostic, "%.2X", errorspecific->diagnostic);
  sprintf(outparms.erroffset, "%.6d", errorspecific->erroroffset);
  fwrite(&outparms, 1, sizeof(especout), screen);
  fread("", 0, 0, screen);
}  /*  End printespec Subroutine  */
/*************************************************************/
/*******   Dequeues the Incoming Message and processes it ******/
void dequeue (int length,
              char *key,
              qentry *dataq,
              usrspace *qname)
{
  char fldlen[3],
       waittime[3],
       keylen[2],
       senderid[2],
       *pointer,
       order[2];
  register int counter;
  waittime[0] = 0;
  waittime[1] = 0;
  waittime[2] = 0x1D;   /*  Hard code a delay of infinite   */
  keylen[0] = 0;
  keylen[1] = 0x3F;  /* Hard code a keylength of 3  */
  senderid[0] = 0;
  senderid[1] = 0x0F;
  strncpy(order, "EQ", 2);
  /*  Clear the data structures  **/
  fflush(stdin);
  pointer = (char *)dataq;
  for (counter = 0; counter < 336; counter++)
    pointer[counter] = 0;
  strncpy (dataq->type, "       ", 7);
```

```
  while ((strncmp(dataq->type, "*USRDFN", 7) != 0) || (fldlen == 0))
     QRCVDTAQ(qname->name, qname->library, fldlen, dataq, waittime,\
           order, keylen, key, senderid,"");
}  /*  End dequeue Subroutine  */
/*********************************************************/
/*************   Set a timer and dequeue next entry  ******/
void settimer (unsigned short *expctid,
               char *process,
               qentry *dataq,
               usrspace *qname,
               char *commhandle)
{
timerparms timer;
disableparms disable;
int length;
char key[6];
  timer.interval = 20000;              /*  set timer for 20 seconds */
  timer.establishcount = 1;            /*  set establish count to 1 */
  timer.keylength = 3;                 /*  key value  */
  strncpy(timer.keyvalue, "TGT", 3);   /*  set key value /
  timer.operation = 1;                 /*  set a timer  */
  /* Call QOLTIMER   */
  QOLTIMER (&(timer.retcode), &(timer.reason), timer.handleout,\
           timer.handlein, (char *)qname, &(timer.operation),\
           &(timer.interval), &(timer.establishcount),\
           &(timer.keylength), timer.keyvalue, timer.userdata);
  if ((timer.retcode != 0) || (timer.reason != 0))
     {
     printf("%s timer failed while being set.\n", process);
     printf("Return code = %d\n", timer.retcode);
     printf("Reason code = %d\n\n", timer.reason);
     }
  /**-------  Dequeue an entry  --------**/
  strncpy(key, "TGT", 3);
  length = 3;
  dequeue (length, key, dataq, qname);
  /***----     Cancel timer   -----***/
  if (dataq->msgid != 0xF0F4)
     {
     strncpy(timer.handlein, timer.handleout, 8);
     timer.operation = 2;           /*  Cancel one timer  */
     QOLTIMER (&(timer.retcode), &(timer.reason), timer.handleout,\
           timer.handlein, (char *)qname, &(timer.operation),\
           &(timer.interval), &(timer.establishcount),\
           &(timer.keylength), timer.keyvalue, timer.userdata);
     if ((timer.retcode != 0) || (timer.reason != 0))
        {
        printf("%s timer failed while being canceled\n", process);
        printf("Return code = %d\n", timer.retcode);
        printf("Reason code = %d\n\n", timer.reason);
        }
     }
  if (dataq->msgid != *expctid)
     {
     printf ("A %.4X message ID was received instead of %.4X\n",\
              dataq->msgid, *expctid);
     printf ("%s completion message was not received\n", process);
     *expctid = dataq->msgid;
     }
}  /*  End settimer Subroutine  */
/*********************************************************/
/**  x25lind:  Read a record into buf and return length   **/
void x25lind (qlindparms *qlind, char *linename)
{
register int counter;
  for(counter=0;counter<256;counter++)
     qlind->userbuffer[counter]=0;
```

```
  qlind->format = 0x01;
  QOLQLIND (&(qlind->retcode), &(qlind->reason), &(qlind->nbytes),\
            qlind->userbuffer, linename, &(qlind->format));
}  /*  End x25lind Subroutine  */
/*************************************************************/
/**  putdata:  Read a record into buf and return length   **/
void putdata (char *buf,
              int dtalen,
              FILE *fptr)
{
int i;
   for (i = 0; i < dtalen; i++)
      fwrite(buf + i, 1, 1, fptr);
}  /*  End putdata Subroutine  */
```

Target application program listing references

The following reference numbers and explanations correspond to the reference numbers in the target application's program listing.

(1) Call the C library routines fopen() and signal() to open the target file and set up a signal handler to process OS/400 exceptions, respectively. If an exception situation is encountered, the handler() will be called to perform clean-up in order for the program to end.

(2) Call the QOLELINK API to enable the line description using the line name and communications handle passed as input parameters to this program.

(3) Call the QOLTIMER API to time the completion of the enable link operation. If the timer expires before the enable-complete message is posted on the this program's data queue, then this program will end.

(4) Call the QUSPTRUS API to obtain a pointer to the beginning of the output buffer user space. The output buffer will be used to construct a filter list for the call to the QOLSETF API.

(5) Call the QOLRECV API to receive inbound data after reading an incoming data message that was posted on the program's data queue by the user-defined communications support. Since these programs are operating using the communications services of X.25, the first data unit the target program should see is a X'B201' operation signalling an incoming call was received.

(6) Call the QOLSEND API with a X'B400' operation to accept the incoming X.25 call. A connection is now established between the source and target application programs.

(7) The target program will now set a timer by calling the QOLTIMER API and wait for incoming data. If the timer expires before any incoming data is received, then this program will call the QOLDLINK API, and end.

(8) This is the main receive loop for the target program. When data is received from the source program, it will be written to the target file opened during the initialization of this program. The loop will process until a message other than incoming-data entry is read from the program's data queue.

(9) Call the QOLSEND API with a X'B001' operation to locally close the connection.

(10) Receives a X'B101' operation from the user-defined communications support. This is a local confirmation of X'B100' operation.

(11) Call the QOLDLINK API to disable the link previously enabled and end.

Includes for Source and Target Programs

The following three includes are used by both the preceding source and target programs. They are not in an OS/400 library.

```
/*****************************************************************/
/*****************************************************************/
/* Include Name: Header                                          */
/*                                                               */
/*                                                               */
/* Function:                                                     */
/*   Type define and declare the structures used to interface    */
/*   to the user-defined communications APIs.  These structures  */
/*   are used by both the source and target application.         */
/*                                                               */
/*                                                               */
/* LANGUAGE:  ILE C for OS/400                                   */
/*                                                               */
```

```
/* APIs USED:  QOLDLINK, QOLELINK, QOLSEND, QOLRECV, QOLSETF,      */
/*             QOLTIMER, QUSPTRUS, QRCVDTAQ, QCLRDTAQ, QOLQLIND    */
/*                                                                 */
/*****************************************************************/
/*****************************************************************/

FILE *screen;
FILE *rptr;
FILE *fptr;

#include <qoldlink.h>
#include <qolelink.h>
#include <qolsend.h>
#include <qolrecv.h>
#include <qolsetf.h>
#include <qoltimer.h>
#include <qusptrus.h>
#include <qrcvdtaq.h>
#include <qclrdtaq.h>
#include <qolqlind.h>

/************   Typedef  Declarations      ******************/

typedef struct usrspace
    {
      char name[10];
      char library[10];
    } usrspace;

typedef struct enableparms     /*  Enable parameters  */
  {
   int retcode,              /*  Output  */
       reason,               /*  Output  */
       tdusize,              /*  Output  */
       numunits,             /*  Output  */
       maxdtalan,            /*  Output  */
       maxdtax25,            /*  Input   */
       keylength;            /*  Input   */
char keyvalue[256],     /*  Input  */
     linename[10];      /*  Input  */
  } enableparms;

typedef struct disableparms     /*  Disable parameters  */
  {
  int retcode,             /*  Output  */
      reason;              /*  Output  */
  char vary;               /*  Input  */
  } disableparms;

typedef struct setfparms     /*  Set Filters parameters  */
  {
  int retcode,             /*  Output  */
      reason,              /*  Output  */
      erroffset;           /*  Output  */
  } setfparms;

typedef _Packed struct hdrparms /* Filter header */
  {
   char function;
   char type;
   unsigned short number;
   unsigned short length;
   char filters[1];
  } hdrparms;

typedef _Packed struct x25filter /* X.25 filter */
  {
```

```
   char pidlength;
   char pid;
   char dtelength;
   char dte[12];
   char flags;
  } x25filter;

typedef struct sendparms     /*  Send parameters  */
  {
 espec errorspecific;       /*  Output  */
   int retcode,             /*  Output  */
       reason,              /*  Output  */
       newpcep,             /*  Output  */
       ucep,                /*  Input  */
       pcep,                /*  Input   */
       numdtaelmnts;        /*  Input   */
unsigned short operation;   /*  Input  */
  } sendparms;

typedef struct recvparms     /*  Receive parameters  */
  {
 espec errorspecific;       /*  Output  */
   int retcode,             /*  Output  */
       reason,              /*  Output  */
       newpcep,             /*  Output  */
       ucep,                /*  Output  */
       pcep,                /*  Output   */
       numdtaunits;         /*  Output   */
  char dataavail;           /*  Output  */
unsigned short operation;   /*  Output  */
  } recvparms;

typedef struct timerparms     /*  Timer parameters  */
  {
   int retcode,             /*  Output  */
       reason,              /*  Output  */
       interval,            /*  Input  */
       establishcount,      /*  Input  */
       keylength;           /*  Input   */
  char handleout[8],    /*  Output  */
       handlein[8],     /*  Input  */
       operation,           /*  Input  */
       keyvalue[256],   /*  Input  */
       userdata[60];    /*  Input  */
  } timerparms;


typedef struct especout
  {
   char hwecode[8];
   char timestamp[16];
   char elogid[8];
   char fail;
   char zerocodes;
   char qsysopr;
   char cause[2];
   char diagnostic[2];
   char erroffset[6];
  } especout;

typedef struct qlindparms     /*  Query line parameters  */
  {
   int retcode,             /*  Output  */
       reason,              /*  Output  */
       nbytes;              /*  Output  */
   char userbuffer[256];
   char format;
```

```
  } qlindparms;

typedef _Packed union content       /*  Queue support parameters  */
  {
    _Packed struct other
      {
       char commhandle[10];
       char reserved[58];
      } other;
    _Packed struct enable
      {
       char commhandle[10];
       char status;
       char reserved[57];
      } enable;
    _Packed struct timer
      {
       char timerhandle[8];
       char userdata[60];
      } timer;
  } content;

typedef _Packed struct qentry      /*  Queue  parameters  */
  {
    char type[10];
    unsigned short msgid;
    content message;
    char key[256];
  } qentry;
```

The following typedef include has new type declarations used by both source and target programs.

```
/*********************************************************************/
/*********************************************************************/
/* Include Name: Typedef                                             */
/*                                                                   */
/* Function:                                                         */
/*   Define the buffer spaces used to pass the data to the           */
/*   APIs.                                                           */
/*                                                                   */
/*                                                                   */
/* LANGUAGE:  ILE C for OS/400                                       */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
/*These definitions and C library #include files are either global, or
  are used by multiple modules in the Open FM API driver.*/

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <xxcvt.h>
#include <string.h>
#include <ctype.h>
#include <recio.h>

typedef struct queuein
    {
      char library[10];
      char name[10];
      char option;
    } queuein;

typedef struct namelib
    {
      char library[10];
```

```
      char name[10];
    } namelib;

typedef _Packed struct format1
  {
   char type;
   char reserved1;
   unsigned short logchanid;
   unsigned short sendpacksize;
   unsigned short sendwindsize;
   unsigned short recvpacksize;
   unsigned short recvwindsize;
   char reserved2[7];
   char dtelength;
   char dte[16];
   char reserved3[8];
   char dbit;
   char reserved4[7];
   char cug;
   char cugid;
   char reverse;
   char fast;
   char faclength;
   char facilities[109];
   char reserved5[48];
   unsigned short calllength;
   char callud[128];
   char reserved6[128];
   unsigned char misc[4];         /* control flags */
   unsigned int maxasmsize;
   unsigned short autoflow;
} format1;

typedef _Packed struct format2
  {
   unsigned short type;
   char cause;
   char diagnostic;
   char reserved[4];
   char faclength;
   char facilities[109];
   char reserved2[48];
   unsigned short length;
   char userdata[128];
  } format2;

typedef _Packed struct desc
  {
   unsigned short length;
   char more;              /*These 4 char's are only used for X.25.*/
   char qualified;
   char interrupt;
   char dbit;
   char reserved[26];
  } desc;

typedef _Packed struct llcheader
  {
   unsigned short headerlength;
   char macaddr[6];
   char dsap;
   char ssap;
   char priority;
   char priorctl;
   unsigned short routlen;
   unsigned short userdtalen;
   char data[1];
```

```
  } llcheader;

typedef _Packed struct espec
  {
   char reserved[2];
   unsigned int hwecode;
   unsigned int timestamphi;
   unsigned int timestamplo;
   unsigned int elogid;
   char reserved2[10];
   char flags;
   char cause;
   char diagnostic;
   char reserved3;
   unsigned int erroroffset;
   char reserved4[4];
  } espec;

typedef struct tableentry
  {
    char handle[10];
    char type;
    char inbuff[20];
    char indesc[20];
    char outbuff[20];
    char outdesc[20];
    unsigned int totaldusize;
    struct tableentry *next;
  } tableentry;

/*******  Data structure for X.25 line      ********/
/*******  descriptions as returned by QOLQLIND.  ******/

typedef struct x25info
  {
   char addrlen;
   char addr[9];
   char addrtype;
   char insert;
   char modulus;
   char dtedce;
   unsigned short maxsend;
   unsigned short maxrecv;
   unsigned short defsend;
   unsigned short defrecv;
   char windowsend;
   char windowrecv;
   unsigned short numlc;
   char lcinfo[4];
  } x25info;

typedef struct querydata
  {
   char header[12];    /* line header info */
   x25info x25data;        /* preliminary data */
  } querydata;

/*****************************************************************/
/*****************************************************************/
/* Include Name: Hexconv                                         */
/*                                                               */
/* Function:                                                     */
/*   This include brings in procedures to convert hexadecimal    */
/*   to integer values and vice versa.                           */
/*                                                               */
/*                                                               */
/* LANGUAGE:  ILE C for OS/400                                   */
```

```
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
#include <stdio.h>

unsigned int hextoint(char *);

char *inttohex(decimal,hex)  /*Converts a 4-byte integer into a
                               string of 2 uppercase hex characters.*/
unsigned int decimal;
char *hex;

{
  sprintf(hex,"%.2X",decimal);
  return(hex);
}

unsigned int hextoint(hex)   /*Converts a string containing hex
                               digits into a 4-byte integer.   */
char *hex;
{
  int decimal;

  sscanf(hex,"%x",&decimal);
  return(decimal);
}
```

Using the Control Device (QTACTLDV) API

This example shows how the QTACTLDV (Control Device) API could be used to send a diagnostic command to a tape device.

```
/* Usage example for QTACTLDV API                                    */
/*                                                                   */
/*                                                                   */

 #include <string.h>
 #include <stdio.h>
 #include <qtactldv.h>
 #include <qusec.h>

 /*********************************************************************/
 /*  Typedef structure for QTACTLDV                                   */
 /*********************************************************************/
 typedef struct {                          /* QTACTLDV command data  */
   Qta_CTLD0100_t  data;                   /*   command data         */
   char        cmd_str[6];                 /*   command string       */
 } cmd_struct;

 /*********************************************************************/
 /*  Typedef structure for Error code                                 */
 /*********************************************************************/
 typedef struct {                          /* Error code structure   */
   Qus_EC_t    Edata;                      /*   Error code data      */
                                           /* CPF67C8 excp data      */
   char        dev_nam[10];                /*   Device name          */
   char        reason_cd[3];               /*   Reason code          */
   char        resv1[3];                   /*   Reserved             */
 } EC_struct;

 /*********************************************************************/
 /*  Constants                                                        */
 /*********************************************************************/

 #define  SNDRSNS "\x03\x00\x00\x00\x12\x00"  /* Request sense       */
```

```
                                                /* command string      */
#define  SNDDIAG "\x1D\x04\x00\x00\x00\x00"  /* Send diagnostic     */
                                                /* command string      */


main(int argc,char *argv[])
{
/*********************************************************************/
/*                                                                   */
/*  START OF MAINLINE                                                */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/*  Variables for QTACTLDV                                           */
/*********************************************************************/
 char                  device[10];        /* device name            */
 char                  send_buff[256];    /* send buffer            */
 int                   send_buff_len;     /* length of send buffer  */
 char                  recv_buff[256];    /* receive buffer         */
 int                   recv_buff_len;     /* length of recv buffer  */
 int                   cmd_data_len;      /* length of command data */
 int                   i;                 /* counter variable       */

 EC_struct             EC;                /* error code structure   */
 cmd_struct            Cmd;               /* struct for QTACTLDV    */

 memcpy(device,argv[1],10);               /* copy device name       */

/*********************************************************************/
/*  OPEN connection                                                  */
/*********************************************************************/
 send_buff_len = 0;                       /* no send buffer         */
 recv_buff_len = 0;                       /* no receive buffer      */
 cmd_data_len = 0;                        /* no command data        */
 EC.Edata.Bytes_Provided = 32;            /* No exceptions          */

 QTACTLDV(device,                        /* device name             */
          FUNOPEN,                       /* requested function      */
          send_buff,                     /* send buffer             */
          send_buff_len,                 /* length of send buffer   */
          recv_buff,                     /* receive buffer          */
          recv_buff_len,                 /* length of receive buffer */
          CTLD0100,                      /* command format          */
          &Cmd,                          /* command data            */
          cmd_data_len,                  /* length of command data  */
          &EC);                          /* Error Code              */

if (EC.Edata.Bytes_Available>0)   /* If there was an error          */
{
  /* Handle the error                                               */
}

/*********************************************************************/
/*  Send Diagnostic command                                          */
/*********************************************************************/
 send_buff_len = 0;                       /* no send buffer         */
 recv_buff_len = 0;                       /* no recv buffer         */
 cmd_data_len = sizeof(Cmd);              /* size of command struct */
 EC.Edata.Bytes_Provided = 32;            /* No exceptions          */
 Cmd.data.Data_transfer_direction = XFRNONE;  /* No data transfer   */
 Cmd.data.Requested_transfer_length = 0;      /* 0 transfer length  */
 Cmd.data.Ignore_length_errors = RPTLERR;     /* report length errs */
 Cmd.data.Command_timeout = 600;              /* 10 minute timeout  */
 Cmd.data.Type_of_command = CMDSCSI;          /* SCSI command       */
 Cmd.data.Offset_to_command_string = 32;      /* offset 32          */
 Cmd.data.Length_of_command_string = 6;       /* 6 byte command     */
```

```
 Cmd.data.Reserved1=0;                          /* reserved           */
 memcpy(&Cmd.cmd_str, SNDDIAG, 6);              /* command string     */

 QTACTLDV(device,                 /* device name                     */
          FUNCMD,                 /* requested function              */
          send_buff,              /* send buffer                     */
          send_buff_len,          /* length of send buffer           */
          recv_buff,              /* receive buffer                  */
          recv_buff_len,          /* length of receive buffer        */
          CTLD0100,               /* command format                  */
          &Cmd,                   /* command data                    */
          cmd_data_len,           /* length of command data          */
          &EC);                   /* Error code                      */

if (EC.Edata.Bytes_Available>0)   /* If there was an error           */
{
  /* See what message was returned                                   */
  if (strncmp(EC.Edata.Exception_Id,"CPF67C8",7)==0) /* Command
                                                        failed msg  */
  {
    /*****************************************************************/
    /* Check the data returned with CPF67C8                          */
    /*****************************************************************/

    if (strncmp(EC.reason_cd,"\x02\xC0", 2) == 0) /* Device detected
                                                      error   */
    {
      /* Check the SCSI completion status                            */
      if (EC.reason_cd[2]=='\x02')      /* Check condition status    */
      {
        /***************************************************/
        /*  Send Request Sense command                     */
        /***************************************************/
        send_buff_len = 0;               /* no send buffer         */
        recv_buff_len = 18;              /* length of recv buffer  */
        cmd_data_len = sizeof(Cmd);      /* size of command struct */
        Cmd.data.Data_transfer_direction = XFRRECV; /* receive     */
        Cmd.data.Requested_transfer_length = 18;    /* 18 bytes    */
        Cmd.data.Ignore_length_errors = IGNLERR; /* ignore length
                                                    errors  */
        Cmd.data.Command_timeout = 60;          /* 60 sec timeout  */
        Cmd.data.Type_of_command = CMDSCSI;     /* SCSI command    */
        Cmd.data.Offset_to_command_string = 32; /* offset 32       */
        Cmd.data.Length_of_command_string = 6;  /* 6 byte cmd      */
        Cmd.data.Reserved1=0;                   /* reserved        */
        memcpy(&Cmd.cmd_str, SNDRSNS, 6);       /* command string  */

        EC.Edata.Bytes_Provided = 32;    /* No exceptions          */

        QTACTLDV(device,         /* device name                     */
                 FUNCMD,         /* requested function              */
                 send_buff,      /* send buffer                     */
                 send_buff_len,  /* length of send buffer           */
                 recv_buff,      /* receive buffer                  */
                 recv_buff_len,  /* length of receive buffer        */
                 CTLD0100,       /* command format                  */
                 &Cmd,           /* command data                    */
                 cmd_data_len,   /* length of command data          */
                 &EC);           /* Error code                      */

        if (EC.Edata.Bytes_Available>0)   /* If there was an error  */
        {
          /* Handle error on request sense command                 */
        }
        else
        {
          /* Parse the request sense data to determine what action  */
```

```
          /* to take.                                                    */
        }
      }
      else if (EC.reason_cd[2]=='\x08')  /* Busy status                 */
      {
        /* Try the command again later                                  */
      }
      else                        /* Unexpected completion status       */
      {
        /* Send error message for unexpected completion status        */
      }
    }

    else if (strncmp(EC.reason_cd,"\x02\xC1\x00", 3) == 0)
                                          /* Selection timeout          */
    {
      /* Send message that device might be powered off                */
    }

    /* Add else if for the other reason codes here                     */

    else
    {
      /* Send error message for unexpected reason code                */
    }
  }

  else
  {
    /* Handle other messages                                          */
  }
}
else
{
  /* No error                                                         */
}

/*******************************************************************/
/*  CLOSE connection                                               */
/*******************************************************************/
 send_buff_len = 0;                       /* no send buffer         */
 recv_buff_len = 0;                       /* no receive buffer      */
 cmd_data_len = 0;                        /* no command data        */
 EC.Edata.Bytes_Provided = 32;            /* No exceptions          */

 QTACTLDV(device,                        /* device name             */
          FUNCLOS,                       /* requested function      */
          send_buff,                     /* send buffer             */
          send_buff_len,                 /* length of send buffer   */
          recv_buff,                     /* receive buffer          */
          recv_buff_len,                 /* length of receive buffer */
          CTLD0100,                      /* command format          */
          &Cmd,                          /* command data            */
          cmd_data_len,                  /* length of command data  */
          &EC.Edata);                    /* Error code              */

if (EC.Edata.Bytes_Available>0)   /* If there was an error          */
{
  /* Handle the error                                               */
}

}
```

Using a Data Queue

The following examples explain three methods to process data queue files.

Example 1: Waiting up to 2 Hours to Receive Data from Data Queue

In the following example, program B specifies to wait up to 2 hours (7200 seconds) to receive an entry from the data queue. Program A sends an entry to data queue DTAQ1 in library QGPL. If program A sends an entry within 2 hours, program B receives the entries from this data queue. Processing begins immediately. If 2 hours elapse without program A sending an entry, program B processes the time-out condition because the field length returned is 0. Program B continues receiving entries until this time-out condition occurs. The programs are written in CL; however, either program could be written in any high-level language.

The data queue is created with the following command:

```
CRTDTAQ DTAQ(QGPL/DTAQ1) MAXLEN(80)
```

In this example, all data queue entries are 80 bytes long.

In program A, the following statements relate to the data queue:

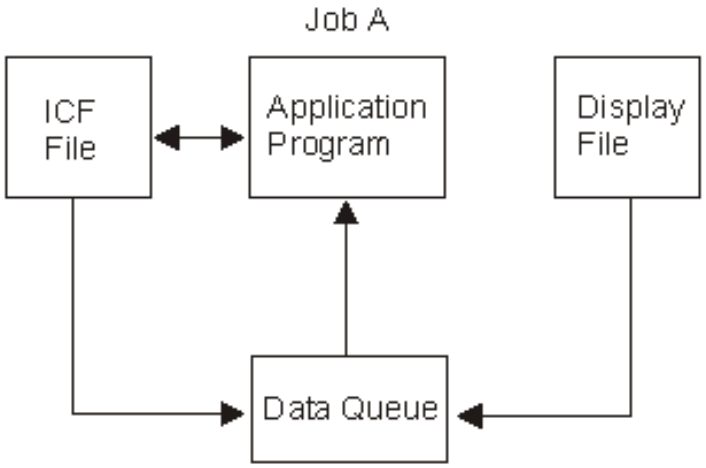
```
PGM
DCL  &FLDLEN  *DEC  LEN(5 0)  VALUE(80)
DCL  &FIELD  *CHAR  LEN(80)
.
.(determine data to be sent to the queue)
.
CALL  QSNDDTAQ  PARM(DTAQ1 QGPL &FLDLEN &FIELD)
.
.
.
```

In program B, the following statements relate to the data queue:

```
PGM
 DCL  &FLDLEN  *DEC  LEN(5 0)  VALUE(80)
 DCL  &FIELD  *CHAR  LEN(80)
 DCL  &WAIT  *DEC  LEN(5 0)  VALUE(7200)   /* 2 hours */
  .
  .
  .
LOOP:   CALL QRCVDTAQ  PARM(DTAQ1
          QGPL &FLDLEN &FIELD &WAIT)
IF (&FLDLEN *NE 0) DO   /* Entry received */
  .
  . (process data from data queue)
  .
    GOTO LOOP /* Get next entry from data queue */
    ENDDO
  .
  . (no entries received for
           2 hours; process time-out condition)
  .
```

Example 2: Waiting for Input from a Display File and an ICF File

The following example is different from the usual use of data queues because there is only one job. The data queue serves as a communications object within the job rather than between two jobs.



In this example, a program is waiting for input from a display file and an ICF file. Instead of alternately waiting for one and then the other, a data queue is used to allow the program to wait on one object (the data queue). The program calls QRCVDTAQ and waits for an entry to be placed on the data queue that was specified on the display file and the ICF file. Both files specify the same data queue. Two types of entries are put on the queue by display data management and ICF data management support when the data is available from either file. ICF file entries start with *ICFF and display file entries start with *DSPF.

The display file or ICF file entry that is put on the data queue is 80 characters in length and contains the field attributes described in the following table. Therefore, the data queue that is specified using the CRTDSPF, CHGDSPF, OVRDSPF, CRTICFF, CHGICFF, and OVRICFF commands must have a length of at least 80 characters.

Position (and Data Type)	**Description**
1 through 10 (character)	The type of file that placed the entry on the data queue. This field will have one of two values: *ICFF for ICF file *DSPF for display file If the job receiving the data from the data queue has only one display file or one ICF file open, then this is the only field needed to determine what type of entry has been received from the data queue.
11 through 12 (binary)	The unique identifier for the file. The value of the identifier is the same as the value in the open feedback area for the file. This field should be used by the program receiving the entry from the data queue only if there is more than one file with the same name placing entries on the data queue.
13 through 22 (character)	The name of the display file or ICF file. This is the name of the file actually opened, after all overrides have been processed, and is the same as the file name found in the open feedback area for the file. This field should be used by the program receiving the entry from the data queue only if there is more than one display file or ICF file that is placing entries on the data queue.
23 through 32 (character)	The library where the file is located. This is the name of the library, after all overrides have been processed, and is the same as the library name found in the open feedback area for the file. This field should be used by the program receiving the entry from the data queue only if there is more than one display file or ICF file that is placing entries on the data queue.
33 through 42 (character)	The program device name, after all overrides have been processed. This name is the same as that found in the program device definition list of the open feedback area. For file type *DSPF, this is the name of the display device where the command or Enter key was pressed. For file type *ICFF, this is the name of the program device where data is available. This field should be used by the program receiving the entry from the data queue only if the file that placed the entry on the data queue has more than one device or session invited prior to receiving the data queue entry.
43 through 80 (character)	Reserved.

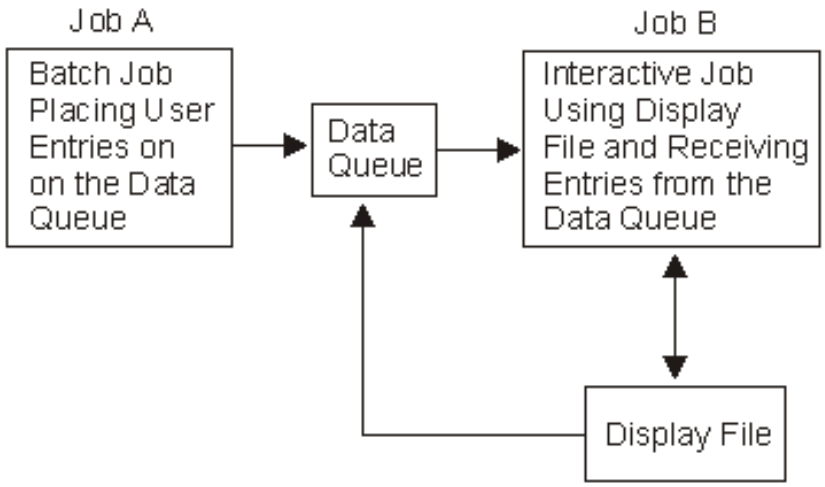
The following example shows coding logic that the application program previously described might use:

.

```
  .
  .
  .
 OPEN DSPFILE ...
      /* Open the Display file. DTAQ parameter specified on*/
      /* CRTDSPF, CHGDSPF, or OVRDSPF for the file.        */

 OPEN ICFFILE ...
      /* Open the ICF file. DTAQ parameter specified on    */
      /* CRTICFF, CHGICFF, or OVRICFF for the file.        */

.
.
DO
  WRITE DSPFILE
       /* Write with Invite for the Display file           */
  WRITE ICFFILE
       /* Write with Invite for the ICF file               */

  CALL QRCVDTAQ
       /* Receive an entry from the data queue specified   */
       /* on the DTAQ parameters for the files. Entries    */
       /* are placed on the data queue when the data is    */
       /* available from any invited device or session     */
       /* on either file.                                  */
       /* After the entry is received, determine which file */
       /* has data available, read the data, process it,   */
       /* invite the file again and return to process the  */
       /* next entry on the data queue.                    */
  IF 'ENTRY TYPE' FIELD = '*DSPF     ' THEN
                                /*  Entry is from display */
    DO                          /*  file. Since this entry*/
                                /*  does not contain the  */
                                /* data received, the data*/
                                /* must be read from the  */
                                /* file before it can be  */
      READ DATA FROM DISPLAY FILE /* processed.            */
      PROCESS INPUT DATA FROM DISPLAY FILE
      WRITE TO DISPLAY FILE     /* Write with Invite      */
    END
  ELSE                          /* Entry is from ICF      */

                                /*  file. Since this entry*/
                                /* does not contain the   */
                                /* data received, the data*/
                                /* must be read from the  */
                                /* file before it can be  */
                                /* processed.             */
      READ DATA FROM ICF  FILE
      PROCESS INPUT DATA FROM ICF FILE
      WRITE TO ICF FILE         /* Write with Invite      */
  LOOP BACK TO RECEIVE ENTRY FROM DATA QUEUE
.
.
.
 END
```

Example 3: Waiting for Input from a Display File and a Data Queue

In the following example, the program in Job B is waiting for input from a display file that it is using and for input to arrive on the data queue from Job A. Instead of alternately waiting for the display file and then the data queue, the program waits for one object, the data queue.



The program calls QRCVDTAQ and waits for an entry to be placed on the data queue that was specified on the display file. Job A is also placing entries on the same data queue. There are two types of entries put on this queue, the display file entry and the user-defined entry. The display file entry is placed on the data queue by display data management when data is available from the display file. The user-defined entry is placing on the data queue by Job A.

The structure of the display file entry is described in the previous example.

The structure of the entry placed on the queue by Job A is defined by the application programmer.

The following example shows coding logic that the application program in Job B might use:

```
  .
  .
  .
  .
  OPEN DSPFILE ...
             /* Open the Display file. DTAQ parameter specified on*/
             /* CRTDSPF, CHGDSPF, or OVRDSPF for the file.        */


.
.
DO
  WRITE DSPFILE   /* Write with Invite for the Display file      */

  CALL QRCVDTAQ
             /* Receive an entry from the data queue specified    */
             /* on the DTAQ parameter for the file. Entries       */
             /* are placed on the data queue either by Job A or   */
             /* by display data management when data is           */
             /* available from any invited device on the display  */
             /* file.                                             */
             /* After the entry is received, determine what type  */
             /* of entry it is, process it, and return to receive */
             /* the next entry on the data queue.                 */
  IF 'ENTRY TYPE' FIELD = '*DSPF     ' THEN
                                       /* Entry is from display */
    DO                                 /*  file. Since this entry*/
                                       /* does not contain the   */
                                       /* data received, the data*/
                                       /* must be read from the  */
                                       /* file before it can be  */
      READ DATA FROM DISPLAY FILE      /* processed.             */
      PROCESS INPUT DATA FROM DISPLAY FILE
      WRITE TO DISPLAY FILE
                                        /* Write with Invite     */
```

```
    END
   ELSE                                  /* Entry is from Job A.  */
                                         /* This entry contains   */
                                         /* the data from Job A,  */
                                         /* so no read is required*/
                                         /* before processing the */
                                         /* data.                 */
      PROCESS DATA QUEUE ENTRY FROM JOB A
   LOOP BACK TO RECEIVE ENTRY FROM DATA QUEUE
 .
 .
 .
 END
```

Using Environment Variables

This program displays the value of an environment variable and then sets the environment variable to a new value.

Use the Create C Module (CRTCMOD) and the Create Program (CRTPGM) commands to create this program.

Call this program with one parameter to display the environment variable specified by that parameter. Call this program with two parameters to set the environment variable specified by the first parameter to the value specified by the second parameter.

```
/*****************************************************************/
/*****************************************************************/
/*                                                               */
/* FUNCTION:  Display the value of an environment variable and   */
/*            then set the environment variable to a new value.  */
/*                                                               */
/* LANGUAGE:  ILE C for OS/400                                   */
/*                                                               */
/* APIs USED: getenv(), putenv()                                 */
/*                                                               */
/*****************************************************************/
/*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>

#define BUFLEN 1024

int main(int argc, char *argv[])
{
    int   num=0;              /* counter                          */
    int   rc;                 /* API return code                  */
    int   l1, l2;             /* lengths of the two parameters    */
    char  *envvar=NULL;       /* pointer to an environment variable*/
    char  **envvaridx=NULL;   /* pointer to an envvar pointer     */
    char  envstring[BUFLEN];
                              /* buffer to construct putenv request*/

                              /* Show a small usage message.      */
    if ((argc != 2) && (argc != 3)) {
        printf("Usage: %s <ENV_VAR> <new_value>\n OR \n"
               "Usage: %s <ENV_VAR>\n", argv[0], argv[0]);
        printf("Sets an environment variable to a user requested\n"
            "value\n"
            "OR\nDisplays the value of a single environment variable\n");
        exit(1);
      }

    if (argc == 2) {
```

```
        /* Called just to display the environment variables.       */
        envvar = getenv(argv[1]);
        if (envvar == NULL) {
            printf("No environment variable %s set\n",
                   argv[1]);
          }
        else {
            printf("Environment variable: %s\n", envvar);
          }
        return 0;
      }

    /* ELSE, called to set an environment variable.                */

    /* Check the size of the parameters and construct a string of  */
    /* the form "VAR=string" which is valid input to putenv.       */
    l1 = strlen(argv[1]);
    l2 = strlen(argv[2]);
    if (l1+l2+2 >= BUFLEN) {
        printf("Only 1024 characters total allowed for this test\n");
        exit(1);
      }
    memcpy(envstring, argv[1], l1);
    envstring[l1] = '=';
    memcpy(&envstring[l1+1], argv[2], l2);
    envstring[l1+l2+1]='\0';

    /* Now that the string is built, let's see if the environment  */
    /* variable was already set.                                   */
    envvar = getenv(argv[1]);
    if (envvar == NULL) {
        printf("Setting NEW environment variable %s\n",
               envstring);
      }
    else {
        printf("Resetting OLD environment variable from: %s\n to %s\n",
               envvar, envstring);
      }

    /* Now actually set the environment variable.                  */
    rc = putenv(envstring);
    if (rc < 0) {
        printf("putenv failed, errno = %d\n", errno);
        return -1;
      }
    printf("Environment variable set\n");
    return 0;
}
```

Saving and Restoring System-Level Environment Variables

The following two-part example illustrates how to save the current set of system-level environment variables and restore them later.

Saving System-Level Environment Variables

This program stores the system-level environment variables and the associated CCSIDs in a file for restoring later.

Use the Create C Module (CRTCMOD) and the Create Program (CRTPGM) commands to create this program.

Call this program with one parameter (the file to store the variable list and the CCSIDs).

```
/*****************************************************************/
/*****************************************************************/
/*                                                               */
/* FUNCTION:  Save the system-level environment variable list    */
/*            and the CCSIDs in a file                           */
/*                                                               */
/* LANGUAGE:  ILE C for OS/400                                   */
/*                                                               */
/* APIs USED: Qp0zGetAllSysEnv()                                 */
/*                                                               */
/*****************************************************************/
/*****************************************************************/

#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <qp0z1170.h>


int main(int argc, char *argv[])
{

    int fd, bw, rc;
    int listBufSize, ccsidBufSize, *ccsidBuf;
    char *listBuf;
    int numvar, sl, sc;

    if(argc != 2)
    {
      printf("Usage: call %s <filename>\n",argv[0]);
      printf("Example: call %s '/tmp/sev'\n",argv[0]);
      return -1;
    }

    sl = listBufSize = 1000;
    sc = ccsidBufSize = 1000;
    listBuf = (char *)malloc(listBufSize);
    ccsidBuf = (int *)malloc(ccsidBufSize);

    /* Create a file of specified name */
    /* If it exists, it is cleared out */
    /* Opened for writing              */
    fd = open(argv[1], O_CREAT | O_WRONLY | O_TRUNC, S_IRWXU);
    if(fd == -1)
    {
      printf("open() failed. errno = %d\n", errno);
      return -1;
    }


    rc = Qp0zGetAllSysEnv(listBuf, &listBufSize, ccsidBuf,
                          &ccsidBufSize, NULL);
    if(rc != 0)
    {
      /* If there are no variables to save, write a  */
      /* zero into the file and return success       */

      if(rc == ENOENT)
      {
        numvar = 0;
        bw = write(fd, &numvar, sizeof(int));
        close(fd);
        printf("No system-level environment variables to save");
```

```
      return 0;
    }

    if(rc != ENOSPC)
    {
      printf("Error using  Qp0zGetAllSysEnv(), errno = %d\n", rc);
      return -1;
    }

    /* rc = ENOSPC. size of buffer is not enough */
    /* change buffer size and try again */

    /* If listBuf is not large enough, */
    /* allocate more space             */
    if(listBufSize > sl)
    {
      listBuf = (char *)realloc(listBuf, listBufSize);
    }

    /* If ccsidBuf is too small, allocate */
    /* more space                         */
    if(ccsidBufSize > sc)
    {
      ccsidBuf = (int *)realloc(ccsidBuf, ccsidBufSize);
    }

    rc = Qp0zGetAllSysEnv(listBuf, &listBufSize,
                          ccsidBuf, &ccsidBufSize, NULL);
    if(rc != 0)
    {
      printf("Error using Qp0zGetAllSysEnv(), errno = %d\n", rc);
      return -1;
    }
  }


  /* Write the contents of the buffer into the file */
  /* First write the total number of ccsid values   */
  /* This is the total number of variables          */

  numvar = ccsidBufSize/sizeof(int);

  bw = write(fd, &numvar, sizeof(int));
  if(bw == -1)
  {
    printf("write() of total number of ccsids failed. errno = %d\n",
errno);
    return -1;
  }


  /* Next write the ccsid values */

  bw = write(fd, ccsidBuf, ccsidBufSize);
  if(bw == -1)
  {
    printf("write() of ccsid values failed. errno = %d\n", errno);
    return -1;
  }

  /* Now write the size (in bytes) of the listBuf */

  bw = write(fd, &listBufSize, sizeof(int));
  if(bw == -1)
  {
    printf("write() of listBufSize failed. errno = %d\n", errno);
    return -1;
```

```
    }

    /* Finally write the listBuf containing the variable strings*/

    bw = write(fd, listBuf, listBufSize);
    if(bw == -1)
    {
      printf("write() of listBuf failed. errno = %d\n", errno);
      return -1;
    }

    /* Close the file */
    rc = close(fd);
    if(rc != 0)
    {
      printf("close() failed. errno = %d\n", errno);
      return -1;
    }

    printf("System-level environment variables saved\n");
    return 0;
}
```

Restoring System-Level Environment Variables

This program reads the system-level environment variable list from a file and then sets the system-level environment variables.

Use the Create C Module (CRTCMOD) and the Create Program (CRTPGM) commands to create this program.

Call this program with one parameter (the name of the file in which the system-level environment variables were stored).

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:  Restore the system-level environment variable list  */
/*            and the associated CCSIDs stored in a file          */
/*                                                                   */
/* LANGUAGE:  ILE C for OS/400                                       */
/*                                                                   */
/* APIs USED: Qp0zPutSysEnv()                                        */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <qp0z1170.h>


int main(int argc, char *argv[])
{
    int fd, rc, br, i, numvar;
    int ccsidBufSize = 0, listBufSize = 0, *ccsidBuf;
    char *listBuf;


    if (argc != 2)
    {
      printf("Usage: call %s <filename>\n",argv[0]);
      printf("Example: call %s '/tmp/sev'\n",argv[0]);
      return -1;
    }
```

```
 /* Open the file specified */

 fd = open(argv[1], O_RDONLY);
 if(fd == -1)
 {
   printf("open() failed. errno = %d\n", errno);
   return -1;
 }

 /* Get the number of variables */
 br = read(fd, &numvar, sizeof(int));
 if(br == -1)
 {
   printf("read() failed. errno = %d\n", errno);
   return -1;
 }

 /* Could delete the existing system-level environment */
 /* variables and have only the restored values.       */
 /* If so desired, could call Qp0zDltSysEnv() to do so */

 /* If there aren't any elements in the file, skip the rest of */
 /* the reads and go to the end                                */

 if(numvar > 0)
 {
   ccsidBufSize = numvar*sizeof(int);
   ccsidBuf = (int *)malloc(ccsidBufSize);

   /* Read the ccsid values and put it in ccsidBuf */
   br = read(fd, ccsidBuf, ccsidBufSize);
   if(br == -1)
   {
     printf("read() failed. errno = %d\n", errno);
     return -1;
   }


   /* Read the size of the list buffer and put it in listBufSize */
   br = read(fd, &listBufSize, sizeof(int));
   if(br == -1)
   {
     printf("read() failed. errno = %d\n", errno);
     return -1;
   }

   listBuf = (char *)malloc(listBufSize);

   /* Finally read the strings themselves */
   br = read(fd, listBuf, listBufSize);
   if(br == -1)
   {
     printf("read() failed. errno = %d\n", errno);
     return -1;
   }
}

/* Walk through the buffer and get the   */
/* name=value strings one by one         */
/* Use Qp0zPutSysEnv() to set the values */

for(i = 0; i < numvar; i++)
{
  rc = Qp0zPutSysEnv(listBuf, ccsidBuf[i], NULL);
  if(rc != 0)
  {
```

```
      printf("Qp0zPutSysEnv() failed. rc=%d\n",rc);
      return -1;
    }

    listBuf += strlen(listBuf) + 1;
  }

  close(fd);
  printf("System-level environment variables restored\n");
  return 0;

}
```

Using ILE Common Execution Environment Data APIs

The following examples show how to call the ILE Common Execution Environment (CEE) Date APIs for ILE COBOL and ILE RPG.

```
       PROCESS NOMONOPRC.
      *************************************************************
      *
      * This sample ILE COBOL program demonstrates how to call the
      * Common Execution Environment (CEE) Date APIs.  The program
      * accepts two parameters.  The first is the date in character
      * form and the second the format of the date.  For instance
      * CALL CEEDATES ('10131955' 'MMDDYYYY') causes the program
      * to treat the date as October 13 1955).
      *
      * The program then displays on the console the numeric day of
      * the week for that date (Sunday = 1) and the named day of
      * week for that date.
      *
      *************************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CEEDATES.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
         SPECIAL-NAMES.
         LINKAGE TYPE PROCEDURE FOR "CEEDAYS" USING ALL DESCRIBED,
         LINKAGE TYPE PROCEDURE FOR "CEEDYWK" USING ALL DESCRIBED,
         LINKAGE TYPE PROCEDURE FOR "CEEDATE" USING ALL DESCRIBED.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  Lilian-Date         PIC S9(9)  BINARY.
       01  Day-of-Week-Numeric PIC S9(9)  BINARY.
       01  Day-of-Week-Alpha   PIC X(10).
       01  Day-of-Week-Format  PIC X(10)         VALUE "Wwwwwwwwwz".
       LINKAGE SECTION.
       01  Sample-Date         PIC X(8).
       01  Date-Format         PIC X(8).
       PROCEDURE DIVISION USING Sample-Date, Date-Format.
       SAMPLE.
      *
      * Convert formatted date to Lilian date
      *
           CALL "CEEDAYS" USING Sample-Date
                                Date-Format
                                Lilian-Date
                                OMITTED.
      *
      * Get numeric day of week from Lilian date
      *
```

```
      CALL "CEEDYWK" USING Lilian-Date
                           Day-of-Week-Numeric
                           OMITTED.
 *
 * Get day of week from Lilian date
 *
      CALL "CEEDATE" USING Lilian-Date
                           Day-of-Week-Format
                           Day-of-Week-Alpha
                           OMITTED.
      DISPLAY "Day of week = " Day-of-Week-Numeric UPON CONSOLE.
      DISPLAY "Day of week = " Day-of-Week-Alpha UPON CONSOLE.
      STOP RUN.
```

```
D*************************************************************
D*
D* This sample ILE RPG program demonstrates how to call the
D* Common Execution Environment (CEE) Date APIs.  The program
D* accepts two parameters.  The first is the date in character
D* form and the second the format of the date.  For instance
D* CALL CEEDATES ('10131955' 'MMDDYYYY') causes the program
D* to treat the date as October 13 1955).
D*
D* The program must be compiled with DFTACTGRP(*NO)
D*
D* The program then displays on the console the numeric day of
D* the week for that date (Sunday = 1) and the named day of
D* week for that date.
D*
D*************************************************************
DLilianDate       s             10i 0
DDayOfWkN         s             10i 0
DDayOfWkA         s             10
DDayOfWkFmt       s             10    inz('Wwwwwwwwwz')
C     *entry        plist
C                   parm                    SampleDate        8
C                   parm                    DateFormat        8
C*
C* Convert formatted date to Lilian date
C*
C                   callb(d)  'CEEDAYS'
C                   parm                    SampleDate
C                   parm                    DateFormat
C                   parm                    LilianDate
C                   parm                    *OMIT
C*
C* Get numeric day of week from Lilian date
C*
C                   callb(d)  'CEEDYWK'
C                   parm                    LilianDate
C                   parm                    DayOfWkN
C                   parm                    *OMIT
C*
C* Get day of week from Lilian date
C*
C                   callb(d)  'CEEDATE'
C                   parm                    LilianDate
C                   parm                    DayOfWkFmt
C                   parm                    DayOfWkA
C                   parm                    *OMIT
C*
C     DayOfWkN      dsply
C     DayOfWkA      dsply
C                   eval      *inlr = '1'
C                   return
```

Using the Generic Terminal APIs

The following two examples illustrate programs that implement a generic terminal and a simple interpreter.

Terminal Program

This program starts and runs a generic terminal.

This program demonstrates the use of the generic terminal functions Qp0zStartTerminal(), Qp0zRunTerminal(), Qp0zEndTerminal(), and Qp0zGetTerminalPid().

Use the Create Bound C Program (CRTBNDC) command to create this program (see Creating the Terminal and Interpreter Programs).

Call this program with no parameters (see Calling the Terminal Program).

```
/* Includes */
#include <qp0ztrml.h>
#include <qp0z1170.h>
#include <stdlib.h>
#include <stdio.h>

/* Constants */
#define NUM_PREDEFINED_ENVS 2

/* Global Variables */
extern char **environ;

int main (int argc, char *argv[])
{
  char *args[2];             /* Argument array */
  int envCount;              /* Count of currently defined env vars */
  int index;                 /* Loop index */
  char **envp;               /* For walking environ array */
  char **envs;               /* Environment variable array */
  Qp0z_Terminal_T handle;    /* Terminal handle */
  Qp0z_Terminal_Attr_T ta;   /* Terminal attributes */
  pid_t pid;                 /* Process ID of interpreter */
  int rc;                    /* Return code */

  /*****************************************************************/
  /* Build the argument array.  */
  /*****************************************************************/

  args[0] = "/QSYS.LIB/QGPL.LIB/ECHOINT.PGM";
  args[1] = NULL;

  /*****************************************************************/
  /* Build the environment variable array. */
  /*****************************************************************/

  /* Make sure environ is set in this activation group. */
  Qp0zInitEnv();

  /* Count the number of environment variables currently defined in this
     process.  Qp0zStartTerminal() will make sure the interpreter
     process does not have too many environment variables. */
  for (envCount = 0, envp = environ; *envp != NULL; ++envp, ++envCount);

  /* Allocate storage for the environment variable array. */
  envs = (char **)malloc(sizeof(char *) *
                         (envCount + NUM_PREDEFINED_ENVS));
  if (envs == NULL) {
```

```
    perror("malloc() failed");
    exit(1);
  }

  /* Copy the current environment variables to the array. */
  for (index = 0; environ[index] != NULL; ++index) {
    envs[index] = environ[index];
  }

  /* Set QIBM_USE_DESCRIPTOR_STDIO variable for using descriptors.  This
     will override the current value of the variable. */
  envs[index++] = "QIBM_USE_DESCRIPTOR_STDIO=Y";

  /* Null terminate array of environment variables. */
  envs[index] = NULL;

  /*********************************************************************/
  /* Set the terminal attributes.  */
  /*********************************************************************/

  memset(&ta, '\0', sizeof(Qp0z_Terminal_Attr_T));
  ta.Buffer_Size = 8196;
  ta.Inherit.pgroup = SPAWN_NEWPGROUP;
  ta.Title = "Echo Interpreter";
  ta.Cmd_Key_Line1 = "F3=Exit F9=Retrieve";
  ta.Cmd_Key_Line2 = "F17=Top F18=Bottom";

  /*********************************************************************/
  /* Start and run the terminal.  */
  /*********************************************************************/

  /* Start the terminal. */
  if (Qp0zStartTerminal(handle, args, envs, ta) != 0) {
    perror("Qp0zStartTerminal() failed");
    exit(1);
  }

  /* Get the PID of the interpreter process. */
  if (Qp0zGetTerminalPid(handle, &pid) != 0) {
    perror("Qp0zGetTerminalPid() failed");
    exit(1);
  }

  /* Run the terminal. */
  rc = Qp0zRunTerminal(handle);
  switch (rc) {
    case QP0Z_TERMINAL_F12:
    case QP0Z_TERMINAL_F3:
    case QP0Z_TERMINAL_ENDED:
      /* Do nothing */
      break;

    default:
      perror("Qp0zRunTerminal() failed");
      exit(1);
      break;
  }

  /* End the terminal. */
  Qp0zEndTerminal(handle);

  return 0;
}
```

Interpreter Program

This program is a simple echo interpreter that is used by the terminal program (see Terminal Program).

Use the Create Bound C Program (CRTBNDC) command to create this program (see Creating the Terminal and Interpreter Programs).

```
/* Echo interpreter */
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

static void SignalHandler(int);

int main (int argc, char *argv[])
{
  char buffer[8192];         /* Buffer for reading input */
  struct sigaction sigact;   /* Signal action */

  /* Set up a signal handler for SIGHUP.  The terminal
     sends this signal when the user presses F3 to exit. */
  sigemptyset(&sigact.sa_mask);
  sigact.sa_flags = 0;
  sigact.sa_handler = SignalHandler;
  if (sigaction(SIGHUP, &sigact, NULL) != 0) {
    perror("sigaction(SIGHUP) failed.");
    exit(2);
  }

  /* Set up a signal handler for SIGINT.  The terminal sends
     this signal when the user presses SysReq 2. */
  sigemptyset(&sigact.sa_mask);
  sigact.sa_flags = 0;
  sigact.sa_handler = SignalHandler;
  if (sigaction(SIGINT, &sigact, NULL) != 0) {
    perror("sigaction(SIGINT) failed.");
    exit(2);
  }

  /* Switch stdout to use line-mode buffering. */
  setvbuf(stdout, NULL, _IOLBF, 128);
  printf("Echo interpreter starting ...\n");
  printf("Enter text:\n");

  /* Do forever. */
  while (1) {
    /* Read a line from stdin. */
    gets(buffer);

    /* End and clean up any allocated
       resources when stdin is closed. */
    if (feof(stdin)) {
      printf("Echo interpreter ending ...\n");
      exit(0);
    }

    /* Echo the line to stdout. */
    printf("%s\n", buffer);
  } /* End of while */

  return 0;
}

void
SignalHandler(int signo)
{
```

```
  printf("Ending for signal %d\n", signo);
  exit(1);
}
```

Creating the Terminal and Interpreter Programs

The following examples show how to create the example programs (Terminal Program and Interpreter Program). These examples assume that the source for the terminal program is member TERMINAL in the file QGPL/QCSRC and that the source for the interpreter program is member INTERPRET in the file QGPL/QCSRC.

Create the terminal program:

```
CRTBNDC PGM(QGPL/TERMINAL)
        SRCFILE(QGPL/QCSRC)
        SRCMBR(TERMINAL)
        SYSIFCOPT(*IFSIO)
        TEXT('Example Terminal program')
```

Create the interpreter program:

```
CRTBNDC PGM(QGPL/INTERPRET)
        SRCFILE(QGPL/QCSRC)
        SRCMBR(INTERPRET)
        SYSIFCOPT(*IFSIO)
        TEXT('Example Interpreter program')
```

Calling the Terminal Program

The following example shows how to start the example program:

```
CALL PGM(QGPL/TERMINAL)
```

Using Profile Handles

The following example illustrates how to generate, change, and release profile handles in a CL program.

```
/*****************************************************************/
/*****************************************************************/
/*                                                               */
/* FUNCTION:  Illustrates how to generate, change, and release   */
/*            profile handles in a CL program.                   */
/*                                                               */
/* LANGUAGE:  CL                                                 */
/*                                                               */
/* APIs USED: QSYGETPH    - Get Profile Handle                   */
/*            QWTSETP     - Set Profile                          */
/*            QSYRLSPH    - Release Profile Handle               */
/*                                                               */
/*****************************************************************/
/*****************************************************************/

PGM (&USERID &PWD)

/* Declare the variables needed by this program:            */
DCL        VAR(&USERID) TYPE(*CHAR) LEN(10)
DCL        VAR(&PWD) TYPE(*CHAR) LEN(10)
DCL        VAR(&SECOFR) TYPE(*CHAR) LEN(10) VALUE(QSECOFR)
DCL        VAR(&SECPWD) TYPE(*CHAR) LEN(10) VALUE(*NOPWD)
```

```
DCL        VAR(&PRFHNDL1) TYPE(*CHAR) LEN(12)
DCL        VAR(&PRFHNDL2) TYPE(*CHAR) LEN(12)

/* Generate profile handles for the QSECOFR user ID and    */
/* for the user ID passed to this program:                 */

CALL       PGM(QSYGETPH) PARM(&SECOFR &SECPWD &PRFHNDL1)
CALL       PGM(QSYGETPH) PARM(&USERID &PWD &PRFHNDL2)

/* Change the user for this job to the user ID passed to   */
/* this program:                                           */

CALL       PGM(QWTSETP) PARM(&PRFHNDL2)

/* This program is now running under the user ID passed to */
/* this program.                                           */

/* Now change the user ID for this job back to the QSECOFR */
/* user ID:                                                */

CALL       PGM(QWTSETP) PARM(&PRFHNDL1)

/* The profile handles generated in this program can now   */
/* be released:                                            */

CALL       PGM(QSYRLSPH) PARM(&PRFHNDL1)
CALL       PGM(QSYRLSPH) PARM(&PRFHNDL2)

ENDPGM
```

Using Registration Facility APIs

The following is an example of how to use the registration facility in one of your programs. The example does not include any of the programs that are being called, nor does it show anything but an excerpt of the calling program.

The first thing to do, after deciding in what program object the exit point is to be placed, is to register that exit point. It is also important to remember that the exit point format defines what the exit program data looks like. Here is an example ILE C program that registers an exit point named QIBM_QXIC_TSTXPOINTA.

```
/*****************************************************************/
/*  PROGRAM:      RegisterPoint                                  */
/*                                                               */
/*  LANGUAGE:     ILE C for OS/400                               */
/*                                                               */
/*  DESCRIPTION:  This program registers an exit point in an     */
/*                application.                                   */
/*                                                               */
/* APIs USED:     QusRegisterExitPoint                           */
/*                                                               */
/*****************************************************************/
#include <string.h>
#include <qusec.h>
#include <qusrgfa1.h>

/*****************************************************************/
/*  Structure for the control block                              */
/*****************************************************************/
typedef _Packed struct Control_x{
        int                 Num_Vlen_Recs;
        Qus_Vlen_Rec_4_t    Vlen_Rec_1;
        char                Description[50];
} Control_Rec;
```

```
int main () {

   Qus_EC_t     Error_Code     = {0};

   char         EPnt_Name[20]  = "QIBM_QXIC_TSTXPOINTA";
   char         EPnt_F_Name[8] = "USUSOOOO";
   int          EProg_Number   = -1;
   Control_Rec  EPnt_Controls  = {0};

/*************************************
*** INITIALIZING ALL STRUCTURES::      ***
*************************************/

   Error_Code.Bytes_Provided = sizeof(Error_Code);

   EPnt_Controls.Num_Vlen_Recs = 1;
   EPnt_Controls.Vlen_Rec_1.Length_Vlen_Record = 62;
   EPnt_Controls.Vlen_Rec_1.Control_Key = 8;
   EPnt_Controls.Vlen_Rec_1.Length_Data = 50;
   memcpy( EPnt_Controls.Description , "Example Exit Point" , 17 );

   QusRegisterExitPoint (EPnt_Name,
                         EPnt_F_Name,
                         &EPnt_Controls,
                         &Error_Code);
   if ( Error_Code.Bytes_Available ) {
      printf("\nEXCEPTION : %s",Error_Code.Exception_Id);
      exit (1);
   }

   return(0);
}
```

After an exit point has been registered, exit programs must be added to that point, indicating the possible calls based on run-time conditions. The following is an example in ILE C, of how to add an exit program to a registered exit point. The exit program named TSTXITPROGQGPL is added to the exit point registered in the previous example named QIBM_QXIC_TSTXPOINTA.

```
/*****************************************************************/
/*  PROGRAM:      AddProgram                                      */
/*                                                                */
/*  LANGUAGE:     ILE C for OS/400                                */
/*                                                                */
/*  DESCRIPTION:  This program adds an exit program to a registered */
/*                exit point.                                     */
/*                                                                */
/* APIs USED:     QusAddExitProgram                               */
/*                                                                */
/*****************************************************************/

#include <qusec.h>
#include <qusrgfa1.h>

/*****************************************************************/
/*  Structure for the Exit Program Attributes                     */
/*****************************************************************/

typedef _Packed struct Xit_Att{
        int                  Num_Vlen_Recs;
        Qus_Vlen_Rec_4_t     ADPG_Vlen;
        int                  CCSID;
        char                 Reserved;
} Xit_Attributes;

int main () {

   Qus_EC_t       Error_Code       = {0};
```

```
   char           EPnt_Name[20]    = "QIBM_QXIC_TSTXPOINTA";
   char           EPnt_F_Name[8]   = "USUSOOOO";
   int            EProg_Number     = -1;
   char           Q_EProg_Name[20] = "TSTXITPROGQGPL      ";
   char           EProg_Data[10]   = "EXAMPLE   ";
   int            Len_EProg_Data   = sizeof(EProg_Data);
   Xit_Attributes EProg_Attributes;


   Error_Code.Bytes_Provided=sizeof(Error_Code);

   EProg_Attributes.Num_Vlen_Recs=1;
   EProg_Attributes.ADPG_Vlen.Length_Vlen_Record=16;
   EProg_Attributes.ADPG_Vlen.Control_Key=3;
   EProg_Attributes.ADPG_Vlen.Length_Data=4;
   EProg_Attributes.CCSID = 37;

   QusAddExitProgram (EPnt_Name,
                      EPnt_F_Name,
                      EProg_Number,
                      Q_EProg_Name,
                      EProg_Data,
                      Len_EProg_Data,
                      &EProg_Attributes,
                      &Error_Code);
   if ( Error_Code.Bytes_Available ) {
      printf("\nEXCEPTION : %s",Error_Code.Exception_Id);
      exit (1);
   }
   return(0);
}
```

When you have registered an exit point and have added the exit programs to that exit point, you can do exit program calls from within your application. The information needed to do the calls is obtained from the Retrieve Exit Information API. In the following sample a conditional call is made based on the exit point information.

```
/*****************************************************************/
/*  PROGRAM:      RetrieveAndProcess                             */
/*                                                               */
/*  LANGUAGE:     ILE C for OS/400                               */
/*                                                               */
/*  DESCRIPTION:  This is an excerpt of a program that retrieves */
/*                information on an exit point, and does processing */
/*                based on that information.                     */
/*                                                               */
/* APIs USED:     QusRetrieveExitInformation                     */
/*                                                               */
/*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <qusec.h>
#include <qusrgfa2.h>

/*****************************************************************/
/*  Structure for Selection Criteria on the Retrieve             */
/*****************************************************************/
typedef _Packed struct RTVEI_Select_C_x {
        Qus_Selcrtr_t        Crit;
        Qus_Select_Entry_t   Select_Entry;
        char                 RTV_String[10];
} RTVEI_Select_C;

/*****************************************************************/
/*  Conv_Lib converts the library name to a null terminated string  */
/*****************************************************************/
```

```
char * Conv_Lib(char in_lib[], char *tmp) {
      int x = 0;

      while ( (in_lib[x] != ' ') && x!=10 ) {
         *tmp=in_lib[x++];
         tmp++;
      }
      return(tmp);
}

int main() {

   Qus_EXTI0200_t       *EXTI0200;
   Qus_EXTI0200_Entry_t *EXTI0200_Entry;
   char                 *Pgm_Data;

   Qus_EC_t        Error_Code= {0};

   char            EPnt_Name[20]  = "QIBM_QXIC_TSTXPOINTA";
   char            EPnt_F_Name[8] = "USUSOOOO";
   int             EProg_Number   = -1;

   int             Counter;
   char            *tmp_str;
   char            *lib;

   char            Handle[16]     = "               ";
   int             Length_Of_R_Var;
   char            RTVEI_Format_Name[8];
   RTVEI_Select_C  EProg_Select_C = {0};


/*************************************
*  Initializing Structures           *
*************************************/

   Error_Code.Bytes_Provided = sizeof(Error_Code);

   tmp_str=(char *)malloc(sizeof(char));
   lib=(char *)malloc(sizeof(char));
   EXTI0200=(Qus_EXTI0200_t *) malloc ( ( sizeof( Qus_EXTI0200_t ) +
        sizeof( Qus_EXTI0200_Entry_t ) + MAX_PGM_DATA_SIZE ) * 2 );
   EProg_Select_C.Crit.Number_Sel_Criteria = 1;
   EProg_Select_C.Select_Entry.Size_Entry = 26;
   EProg_Select_C.Select_Entry.Comp_Operator = 1;
   EProg_Select_C.Select_Entry.Start_Pgm_Data = 0;
   EProg_Select_C.Select_Entry.Length_Comp_Data = 10;
   memcpy( EProg_Select_C.RTV_String , "EXAMPLE   " , 10 );

   Length_Of_R_Var = (sizeof( Qus_EXTI0200_t ) +
                sizeof( Qus_EXTI0200_Entry_t ) +
                MAX_PGM_DATA_SIZE) *2;
   memcpy( RTVEI_Format_Name , "EXTI0200" , 8 );

   QusRetrieveExitInformation (Handle,
                               EXTI0200,
                               Length_Of_R_Var,
                               RTVEI_Format_Name,
                               EPnt_Name,
                               EPnt_F_Name,
                               EProg_Number,
                               &EProg_Select_C,
                               &Error_Code);
   if ( Error_Code.Bytes_Available ) {
      printf("\nEXCEPTION : %s",Error_Code.Exception_Id);
      exit (1);
   }
```

```
/*****************************************************
*  Call all of the preprocessing exit programs returned *
*****************************************************/
   Counter=EXTI0200->Number_Programs_Returned;

   while ( Counter-- ) {

     EXTI0200_Entry = (Qus_EXTI0200_Entry_t *) EXTI0200;
     EXTI0200_Entry = (Qus_EXTI0200_Entry_t *)((char *)EXTI0200 +
                       EXTI0200->Offset_Program_Entry);
     Pgm_Data = (char *) EXTI0200_Entry;
     Pgm_Data += EXTI0200_Entry->Offset_Exit_Data;

     Conv_Lib(EXTI0200_Entry->Program_Library,lib);

     sprintf( tmp_str , "CALL  %s/%.10s %.*s" ,
          lib,
          EXTI0200_Entry->Program_Name,
          EXTI0200_Entry->Length_Exit_Data,
          Pgm_Data );
     system( tmp_str );
/*****************************************************
*  This is where Error Handling on the exit program     *
*  would be done.                                       *
*****************************************************/
     if ( Counter ) {
         memcpy(EXTI0200->Continue_Handle,Handle,16);
         QusRetrieveExitInformation(Handle,
                                    EXTI0200,
                                    Length_Of_R_Var,
                                    RTVEI_Format_Name,
                                    EPnt_Name,
                                    EPnt_F_Name,
                                    EProg_Number,
                                    &EProg_Select_C,
                                    &Error_Code);
         if ( Error_Code.Bytes_Available ) {
            printf("\nEXCEPTION : %s",Error_Code.Exception_Id);
            exit (1);
         }
      }
   }

   return(0);
}
```

Using Semaphores and Shared Memory

The following two examples illustrate programs that support the client/server model.

Server Program

This program acts as a server to the client program (see Client Program). The buffer is a shared memory segment. The process synchronization is done using semaphores.

Use the Create C Module (CRTCMOD) and the Create Program (CRTPGM) commands to create this program.

Call this program with no parameters before calling the client program.

```
/*****************************************************************/
/*****************************************************************/
```

```
/*                                                                   */
/* FUNCTION:  This program acts as a server to the client program.   */
/*                                                                   */
/* LANGUAGE:  ILE C for OS/400                                       */
/*                                                                   */
/* APIs USED: semctl(), semget(), semop(),                           */
/*            shmat(), shmctl(), shmdt(), shmget()                   */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/

#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>

#define SEMKEY 8888             /* Key passed into semget operation  */
#define SHMKEY 9999             /* Key passed into shmget operation  */

#define NUMSEMS 2               /* Num of sems in created sem set    */
#define SIZEOFSHMSEG 50         /* Size of the shared mem segment    */

#define NUMMSG 2                /* Server only doing two "receives"
                                   on shm segment                    */

int main(int argc, char *argv[])
{
    int rc, semid, shmid, i;
    void *shm_address;
    struct sembuf operations[2];
    struct shmid_ds shmid_struct;
    short  sarray[NUMSEMS];

    /* Create a semaphore set with the constant key.  The number of  */
    /* semaphores in the set is two.  If a semaphore set already     */
    /* exists for the key, return an error. The specified permissions*/
    /* give everyone read/write access to the semaphore set.         */

    semid = semget( SEMKEY, NUMSEMS, 0666 | IPC_CREAT | IPC_EXCL );
    if ( semid == -1 )
      {
        printf("main: semget() failed\n");
        return -1;
      }

    /* Initialize the first semaphore in the set to 0 and the        */
    /* second semaphore in the set to 0.                             */
    /*                                                               */
    /* The first semaphore in the sem set means:                     */
    /*        '1' --  The shared memory segment is being used.       */
    /*        '0' --  The shared memory segment is freed.            */
    /* The second semaphore in the sem set means:                    */
    /*        '1' --  The shared memory segment has been changed by  */
    /*                the client.                                    */
    /*        '0' --  The shared memory segment has not been         */
    /*                changed by the client.                         */

    sarray[0] = 0;
    sarray[1] = 0;


    /* The '1' on this command is a no-op, because the SETALL command*/
    /* is used.                                                      */
    rc = semctl( semid, 1, SETALL, sarray);
    if(rc == -1)
      {
```

```
    printf("main: semctl() initialization failed\n");
    return -1;
  }

/* Create a shared memory segment with the constant key.  The    */
/* size of the segment is a constant.  The specified permissions */
/* give everyone read/write access to the shared memory segment. */
/* If a shared memory segment already exists for this key,       */
/* return an error.                                              */
shmid = shmget(SHMKEY, SIZEOFSHMSEG, 0666 | IPC_CREAT | IPC_EXCL);
if (shmid == -1)
  {
    printf("main: shmget() failed\n");
    return -1;
  }

/* Attach the shared memory segment to the server process.       */
shm_address = shmat(shmid, NULL, 0);
if ( shm_address==NULL )
  {
    printf("main: shmat() failed\n");
    return -1;
  }
printf("Ready for client jobs\n");

/* Loop only a specified number of times for this example.       */
for (i=0; i < NUMMSG; i++)
  {
    /* Set the structure passed into the semop() to first wait   */
    /* for the second semval to equal 1, then decrement it to    */
    /* allow the next signal that the client writes to it.       */
    /* Next, set the first semaphore to equal 1, which means     */
    /* that the shared memory segment is busy.                   */
    operations[0].sem_num = 1;
                                /* Operate on the second sem     */
    operations[0].sem_op = -1;
                                /* Decrement the semval by one   */
    operations[0].sem_flg = 0;
                                /* Allow a wait to occur         */

    operations[1].sem_num = 0;
                                /* Operate on the first sem      */
    operations[1].sem_op =  1;
                                /* Increment the semval by 1     */
    operations[1].sem_flg = IPC_NOWAIT;
                                      /* Do not allow to wait    */

    rc = semop( semid, operations, 2 );
    if (rc == -1)
      {
        printf("main: semop() failed\n");
        return -1;
      }

    /* Print the shared memory contents.                         */
    printf("Server Received : \"%s\"\n", (char *) shm_address);

    /* Signal the first semaphore to free the shared memory.     */
    operations[0].sem_num = 0;
    operations[0].sem_op  = -1;
    operations[0].sem_flg = IPC_NOWAIT;

    rc = semop( semid, operations, 1 );
    if (rc == -1)
      {
        printf("main: semop() failed\n");
        return -1;
```

```
        }

      }  /* End of FOR LOOP */

    /* Clean up the environment by removing the semid structure,     */
    /* detaching the shared memory segment, and then performing      */
    /* the delete on the shared memory segment ID.                   */

    rc = semctl( semid, 1, IPC_RMID );
    if (rc==-1)
      {
        printf("main: semctl() remove id failed\n");
        return -1;
      }
    rc = shmdt(shm_address);
    if (rc==-1)
      {
        printf("main: shmdt() failed\n");
        return -1;
      }
    rc = shmctl(shmid, IPC_RMID, &shmid_struct);
    if (rc==-1)
      {
        printf("main: shmctl() failed\n");
        return -1;
      }
return 0;
}
```

Client Program

This program acts as a client to the server program (see Server Program). The program is run after the message Ready for client jobs appears from the server program.

Use the CRTCMOD and CRTPGM commands to create this program.

Call this program with no parameters after calling the server program.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:  This program acts as a client to the server program.   */
/*                                                                   */
/* LANGUAGE:  ILE C for OS/400                                       */
/*                                                                   */
/* APIs USED: semget(), semop(),                                     */
/*            shmget(), shmat(), shmdt()                             */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/

#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>

#define SEMKEY 8888
#define SHMKEY 9999

#define NUMSEMS 2
#define SIZEOFSHMSEG 50

int main(int argc, char *argv[])
{
```

```
struct sembuf operations[2];
void         *shm_address;
int semid, shmid, rc;

/* Get the already created semaphore ID associated with key.     */
/* If the semaphore set does not exist, then it will not be      */
/* created, and an error will occur.                             */
semid = semget( SEMKEY, NUMSEMS, 0666);
if ( semid == -1 )
  {
    printf("main: semget() failed\n");
    return -1;
  }

/* Get the already created shared memory ID associated with key. */
/* If the shared memory ID does not exist, then it will not be   */
/* created, and an error will occur.                             */

shmid = shmget(SHMKEY, SIZEOFSHMSEG, 0666);
if (shmid == -1)
  {
    printf("main: shmget() failed\n");
    return -1;
  }

/* Attach the shared memory segment to the client process.       */
shm_address = shmat(shmid, NULL, 0);
if ( shm_address==NULL )
  {
    printf("main: shmat() failed\n");
    return -1;
  }

/* First, check to see if the first semaphore is a zero.  If it  */
/* is not, it is busy right now.  The semop() command will wait  */
/* for the semaphore to reach zero before running the semop().   */
/* When it is zero, increment the first semaphore to show that   */
/* the shared memory segment is busy.                            */
operations[0].sem_num = 0;
                                /* Operate on the first sem      */
operations[0].sem_op =  0;
                                /* Wait for the value to be=0    */
operations[0].sem_flg = 0;
                                /* Allow a wait to occur         */

operations[1].sem_num = 0;
                                /* Operate on the first sem      */
operations[1].sem_op =  1;
                                /* Increment the semval by one   */
operations[1].sem_flg = 0;
                                /* Allow a wait to occur         */

rc = semop( semid, operations, 2 );
if (rc == -1)
  {
    printf("main: semop() failed\n");
    return -1;
  }

strcpy((char *) shm_address, "Hello from Client");

/* Release the shared memory segment by decrementing the in-use  */
/* semaphore (the first one).  Increment the second semaphore to */
/* show that the client is finished with it.                     */
operations[0].sem_num = 0;
                                /* Operate on the first sem      */
operations[0].sem_op =  -1;
```

```
                                    /* Decrement the semval by one   */
    operations[0].sem_flg = 0;
                                    /* Allow a wait to occur         */

    operations[1].sem_num = 1;
                                    /* Operate on the second sem     */
    operations[1].sem_op =  1;
                                    /* Increment the semval by one   */
    operations[1].sem_flg = 0;
                                    /* Allow a wait to occur         */

    rc = semop( semid, operations, 2 );
    if (rc == -1)
      {
        printf("main: semop() failed\n");
        return -1;
      }

    /* Detach the shared memory segment from the current process.    */
    rc = shmdt(shm_address);
    if (rc==-1)
      {
        printf("main: shmdt() failed\n");
        return -1;
      }

return 0;
}
```

Using SNA/Management Services Transport APIs

This example shows a source and target application using network management transport APIs to send and receive management services data. The example compiles in ILE C.

Source Application Program

This source application program sends a request to a target application.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:                                                         */
/*  This is a source application that uses the management services   */
/*  transport APIs.  It does the following:                          */
/*   1.  Prompts for the network ID and CP name of the remote system */
/*       where target application MSTTARG has been started.          */
/*   2.  Prompts for data to be sent to MSTTARG.                     */
/*   3.  Prompts for whether or not a reply is required.             */
/*   4.  Sends a management services transport request to MSTTARG.   */
/*   5.  Repeats steps 2-4 until QUIT is entered.                    */
/*                                                                   */
/* Note: MSTTARG may be ended by this application by sending it the  */
/*        string "ENDRMTAPP".                                        */
/*                                                                   */
/* LANGUAGE:  ILE C for OS/400                                       */
/*                                                                   */
/* APIs USED: QNMSTRAP, QNMENDAP, QNMRCVDT,                          */
/*            QNMSNDRQ, QNMCHGMN, QNMENDAP                           */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
/* Includes                                                          */
```

```
/*********************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define NOERROR "NOERROR"
#define RQSONLY "*RQS      "
#define RQSRPY "*RQSRPY   "

/*-------------------------------------------------------------------*/
/* Type definitions                                                  */
/*-------------------------------------------------------------------*/
typedef int HANDLE;                /* typedef for handle             */
typedef char APPLNAME[8];          /* typedef for application name   */
typedef char NETID[8];             /* typedef for network ID         */
typedef char CPNAME[8];            /* typedef for control point name*/
typedef char MODENAME[8];          /* typedef for mode name          */
typedef char SENSECODE[8];         /* typedef for SNA sense code (in
                                      character format)              */
typedef char LIBNAME[10];          /* typedef for library name       */
typedef char QNAME[10];            /* typedef for data queue name    */
typedef char MSGID[7];             /* typedef for message ID         */
typedef char EXCPDATA[48];         /* typedef for exception data     */
typedef char CATEGORY[8];          /* typedef for category           */
typedef char APPLTYPE[10];         /* typedef for application type   */
typedef char REPLREG[10];          /* typedef for replace
                                      registration                   */
typedef char DATARCVD[10];         /* typedef for data received      */
typedef char REQTYPE[10];          /* typedef for request type       */
typedef char POSTRPL[10];          /* typedef for post reply         */
typedef char REQUESTID[53];        /* typedef for request ID         */
typedef char SRBUFFER[500];        /* typedef for send/receive
                                      buffer.  This program limits
                                      the amount of data to be sent
                                      or received to 500 bytes.  The
                                      maximum size of a management
                                      services transport buffer is
                                      31739.                         */

typedef struct {                   /* Library-qualified data queue
                                      name                           */
   QNAME data_queue_name;          /* data queue name                */
   LIBNAME library_name;           /* library name                   */
  } QUALQNAME;

typedef struct {                   /* Error code structure           */
   int bytes_provided;             /* number of bytes provided       */
   int bytes_available;            /* number of bytes available      */
   MSGID exception_ID;             /* exception ID                   */
   char reserved_area;             /* reserved                       */
   EXCPDATA exception_data;        /* exception data                 */
  } ERRORCODE;

typedef struct {                   /* Notification record structure */
   char record_type[10];           /* Record type                    */
   char function[2];               /* Function                       */
   HANDLE handle;                  /* Handle                         */
   REQUESTID req_id;               /* Request ID                     */
   char reserved[11];              /* Reserved area                  */
  } NOTIFRCD;

typedef struct {                   /* Receiver variable structure   */
   int bytes_provided;             /* number of bytes provided       */
   int bytes_available;            /* number of bytes available      */
   SRBUFFER received_data;         /* received data                  */
  } RECEIVERVAR;

typedef struct {                   /* Qualified application name    */
```

```
   NETID network_id;               /* Network ID                    */
   CPNAME cp_name;                 /* Control point name            */
   APPLNAME app_name;              /* Application name              */
  } QUALAPPL;

/*-------------------------------------------------------------------*/
/* External program declarations                                     */
/*-------------------------------------------------------------------*/
#pragma linkage(QNMSTRAP, OS)       /* Start application API         */
extern void QNMSTRAP (HANDLE *handle, /* pointer to handle           */
                      APPLNAME *applname, /* pointer to appl name    */
                      QUALQNAME *qualqname, /* pointer to data queue
                                    name                            */
                      ERRORCODE *errorcode); /* pointer to error code
                                    parameter                       */

#pragma linkage(QNMENDAP, OS)       /* End application API           */
extern void QNMENDAP (HANDLE *handle, /* pointer to handle           */
                      ERRORCODE *errorcode); /* pointer to error code
                                    parameter                       */

#pragma linkage(QNMRCVDT, OS)       /* Receive data API              */
extern void QNMRCVDT (HANDLE *handle, /* pointer to handle           */
                      RECEIVERVAR *rcvvar, /* pointer to receiver
                                    variable                        */
                      int *rcvvarln, /* pointer to receiver variable
                                    length                          */
                      REQUESTID *reqid, /* pointer to request ID     */
                      QUALAPPL *qualappl, /* pointer to remote
                                    application name                */
                      DATARCVD *datarcvd, /* pointer to type of data
                                    received                        */
                      int *waittim, /* pointer to wait time          */
                      ERRORCODE *errorcode); /* pointer to error code
                                    parameter                       */

#pragma linkage(QNMSNDRQ, OS)       /* Send request API              */
extern void QNMSNDRQ (HANDLE *handle, /* pointer to handle           */
                      QUALAPPL *qualappl, /* pointer to remote
                                    application name                */
                      REQUESTID *reqid, /* pointer to request ID     */
                      SRBUFFER *sndbuf, /* pointer to send buffer    */
                      int *sndbufln, /* pointer to send buffer length*/
                      REQTYPE *reqtype, /* pointer to request type   */
                      POSTRPL *postrpl, /* pointer to post reply     */
                      int *waittim, /* pointer to wait time          */
                      ERRORCODE *errorcode); /* pointer to error code
                                    parameter                       */

#pragma linkage(QNMCHGMN, OS)       /* Change mode name API          */
extern void QNMCHGMN (HANDLE *handle, /* pointer to handle           */
                      MODENAME *modename, /* pointer to mode name    */
                      ERRORCODE *errorcode); /* pointer to error code
                                    parameter                       */

void check_error_code (char func_name[8]); /* Used to check error code
                                                                    */
void get_network_id (void);         /* Get network ID of destination
                                       node                         */
void get_cp_name (void);            /* Get CP name of destination
                                       node                         */
void process_replies(void);         /* Process replies received from
                                       destination application      */

/*-------------------------------------------------------------------*/
/* Global declarations                                               */
/*-------------------------------------------------------------------*/
```

```
 HANDLE appl_handle;                /* Handle of application         */
 ERRORCODE error_code_struc =       /* Error code parameter          */
           {sizeof(error_code_struc),  /* Initialize bytes provided  */
            0,                      /* initialize bytes available    */
            NOERROR};               /* initialize error code         */
 char input_line[80];               /* Input data                    */
 QUALAPPL qual_appl =               /* Qualified application name    */
        {"        ","        ","         "};
 REQUESTID req_id;                  /* Returned request ID           */
 int wait_time = -1;                /* Wait time = wait forever      */

/*-------------------------------------------------------------------*/
/* Start of main.                                                    */
/*-------------------------------------------------------------------*/
int main ()
{
 APPLNAME appl_name = "MSTSOURC";   /* Application name to be used   */
 QUALQNAME data_queue_parm =        /* Data queue name to be used    */
           {"*NONE     ", "          "}; /* Initialize structure     */
 NOTIFRCD notif_record;             /* Area to contain notification
                                       record                      */
 CATEGORY category = "*NONE   ";    /* SNA/Management Services function
                                       set group                   */
 APPLTYPE appl_type = "*FPAPP    "; /* Application type              */
 REPLREG replace_reg = "*YES      "; /* Replace registration = *YES  */
 int sys_result;                    /* Result of system function     */
 char end_msg[] = "ENDRMTAPPL";     /* If this data is received then
                                       the application will end      */
 char incoming_data[] = "01";       /* Incoming data constant        */
 SRBUFFER send_buffer;              /* Send buffer                   */
 int data_length;                   /* Length of send data           */
 char input_char;                   /* Input character               */
 REQTYPE req_type;                  /* Request type                  */
 POSTRPL post_reply = "*NO       "; /* Don't post any received replies
                                                                    */
 MODENAME mode_name = "#INTER  ";   /* Mode name = #INTER            */

/*-------------------------------------------------------------------*/
/* Start of code                                                     */
/*-------------------------------------------------------------------*/
 QNMSTRAP (&appl_handle,
           &appl_name,
           &data_queue_parm,
           &error_code_struc);      /* Start application             */
 check_error_code("QNMSTRAP");      /* Check error code              */
 QNMCHGMN (&appl_handle,
           &mode_name,
           &error_code_struc);      /* Change mode name              */
 check_error_code("QNMCHGMN");      /* Check error code              */
 get_network_id();                  /* Get network ID                */
 get_cp_name();                     /* Get CP name                   */
 memcpy(qual_appl.app_name,
        "MSTTARG ",
        sizeof(qual_appl.app_name)); /* Copy application name        */
 printf ("Enter message to send to remote application or "
         "QUIT to end\n");
 gets(input_line);
 while (memcmp(input_line,
               "QUIT",
               sizeof("QUIT")) != 0) /* While an ending string
                                        has not been entered        */
   {
     data_length = strlen(input_line); /* Get length of message     */
     memcpy(send_buffer,
            input_line,
            data_length);           /* Put message in send buffer    */
     printf("Reply necessary? (Y or N)\n"); /* Prompt for reply
```

```
                                    indicator                    */
    gets(input_line);              /* Get reply character          */
    input_char = toupper(input_line[0]); /* Convert character to
                                    uppercase                    */
    while (strlen(input_line) != 1 ||
           (input_char != 'Y' &&
           input_char != 'N'))
      {
        printf("Please type Y or N\n");
        gets(input_line);          /* Get reply character          */
        input_char = toupper(input_line[0]); /* Convert character to
                                    uppercase                    */
      }
    if (input_char == 'Y')
      {
        memcpy(req_type,
               RQSRPY,
               sizeof(req_type));  /* Indicate request should have
                                    a reply                      */
      }
    else
      {
        memcpy(req_type,
               RQSONLY,
               sizeof(req_type));  /* Indicate request should not have
                                    a reply                      */
      }
    QNMSNDRQ (&appl_handle,
              &qual_appl,
              &req_id,
              &send_buffer,
              &data_length,
              &req_type,
              &post_reply,
              &wait_time,
              &error_code_struc);  /* Send request to remote
                                    application                  */
    check_error_code("QNMSNDRQ");  /* Check error code             */
    if (input_char == 'Y')
      {
        process_replies();         /* Process one or more received
                                    replies                      */
      }
    printf ("Enter message to send to remote application or "
            "QUIT to end\n");
    gets(input_line);
   }
 QNMENDAP (&appl_handle,
           &error_code_struc);     /* End the application          */

 return 0;
}

/*--------------------------------------------------------------------*/
/* process_replies function                                           */
/*--------------------------------------------------------------------*/
void process_replies ()
{
 RECEIVERVAR receiver_var =        /* Receiver variable            */
           {sizeof(receiver_var)}; /* Initialize bytes provided    */
 int rcv_var_len = sizeof(receiver_var); /* Length of receiver
                                    variable                     */
 DATARCVD data_rcvd = "*NODATA   "; /* Type of data received        */
 QUALAPPL qual_appl;               /* Sender of reply              */

 printf ("Received reply(s):\n");
 while (memcmp(data_rcvd,
```

```
               "*RPYCPL   ",
               sizeof(data_rcvd)) != 0) /* While final reply has not
                                        been received                */
     {
       strncpy(receiver_var.received_data,
               "\0",
               sizeof(receiver_var.received_data)); /* Null out
                                        data buffer                  */
       QNMRCVDT (&appl_handle,
                 &receiver_var,
                 &rcv_var_len,
                 &req_id,
                 &qual_appl,
                 &data_rcvd,
                 &wait_time,
                 &error_code_struc);  /* Receive reply               */
       check_error_code("QNMRCVDT");  /* Check error code            */
       printf("%1.500s\n",receiver_var.received_data); /* Print out
                                                          reply       */
     }
}

/*--------------------------------------------------------------------*/
/* get_network_id function.                                           */
/*--------------------------------------------------------------------*/
void get_network_id ()
{
 int count;
 printf("Enter network ID of remote system where MSTTARG "
        "application has been started\n"); /* Prompt for network
                                        ID                           */
 gets(input_line);                  /* Get network ID                */
 while (strlen(input_line) <= 0 ||
        strlen(input_line) > 8)     /* While network ID is not valid */
   {
     printf("Network ID is too long or too short - try again\n");
     gets(input_line);              /* Get network ID                */
   }
 memcpy(qual_appl.network_id,
        input_line,
        strlen(input_line));        /* Copy network ID               */
 for (count=0; count < strlen(input_line); count++)
   qual_appl.network_id[count] =
                 toupper(qual_appl.network_id[count]); /* Convert
                                        input to uppercase           */
}

/*--------------------------------------------------------------------*/
/* get_cp_name function.                                              */
/*--------------------------------------------------------------------*/
void get_cp_name ()
{
 int count;
 printf("Enter CP name of remote system where MSTTARG application "
        "has been started\n");      /* Prompt for CP name            */
 gets(input_line);                  /* Get CP name                   */
 while (strlen(input_line) <= 0 ||
        strlen(input_line) > 8)     /* While CP name is not valid    */
   {
     printf("CP name is too long or too short - try again\n");
     gets(input_line);              /* Get CP name                   */
   }
 memcpy(qual_appl.cp_name,
        input_line,
        strlen(input_line));        /* Copy CP name                  */
 for (count=0; count < strlen(input_line); count++)
   qual_appl.cp_name[count] =
```

```
                    toupper(qual_appl.cp_name[count]); /* Convert
                                        input to uppercase           */
}

/*--------------------------------------------------------------------*/
/* check_error_code -                                                 */
/*--------------------------------------------------------------------*/
void check_error_code (char func_name[8])
{
 char *sense_ptr = error_code_struc.exception_data + 36; /*
                                        Pointer to sense code in
                                        exception data               */
 SENSECODE sense_code;              /* SNA sense code                */
 if (error_code_struc.bytes_available != 0) /* Error occurred?       */
   {
     printf("\n\nError occurred calling %1.8s.\n",func_name);
     memcpy(sense_code,
            sense_ptr,
            sizeof(sense_code));    /* Copy sense code from exception
                                        data                         */
     printf("Error code is %1.7s, SNA sense code is %1.8s.\n",
                  error_code_struc.exception_ID,
                  sense_code);
     if (memcmp(func_name,
            "QNMSTRAP",
            sizeof(func_name)) != 0)  /* Error did not occur on
                                        start application?           */
       {
         QNMENDAP (&appl_handle,
                   &error_code_struc); /* End the application        */

       }
     exit(EXIT_FAILURE);            /* Exit this program             */
   }
}
```

Target Application Program

This target application receives requests from and returns replies to source applications.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:                                                         */
/* This is a target application that uses the management services    */
/* transport APIs.  It receives management services transport        */
/* requests from source application MSTSOURC and displays the data   */
/* contained in the request.  If the request specifies that a        */
/* reply needs to be sent, this program accepts input from the       */
/* keyboard and sends one or more replies to the source application. */
/*                                                                   */
/* LANGUAGE:  ILE C for OS/400                                       */
/*                                                                   */
/* APIs USED: QNMSTRAP, QNMENDAP, QNMREGAP, QNMDRGAP,                */
/*            QNMRCVDT, QNMSNDRP, QNMRCVOC, QRCVDTAQ,                */
/*            QNMENDAP                                               */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
/* Includes                                                          */
/*********************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```
#define NOERROR "NOERROR"
#define REQUEST "*RQS      "
#define REQREPLY "*RQSRPY   "
#define REPLYINC "*RPYINCPL "
#define REPLYCMP "*RPYCPL   "

/*--------------------------------------------------------------------*/
/* Type definitions                                                   */
/*--------------------------------------------------------------------*/
typedef int HANDLE;                /* typedef for handle              */
typedef char APPLNAME[8];          /* typedef for application name    */
typedef char NETID[8];             /* typedef for network ID          */
typedef char CPNAME[8];            /* typedef for control point name  */
typedef char SENSECODE[8];         /* typedef for SNA sense code
                                      (in character format)           */
typedef char LIBNAME[10];          /* typedef for library name        */
typedef char QNAME[10];            /* typedef for data queue name     */
typedef char MSGID[7];             /* typedef for message ID          */
typedef char EXCPDATA[48];         /* typedef for exception data      */
typedef char CATEGORY[8];          /* typedef for category            */
typedef char APPLTYPE[10];         /* typedef for application type    */
typedef char REPLREG[10];          /* typedef for replace
                                      registration                    */
typedef char DATARCVD[10];         /* typedef for data received       */
typedef char REPLYTYPE[10];        /* typedef for reply type          */
typedef char REQUESTID[53];        /* typedef for request ID          */
typedef char PACKED5[3];           /* typedef for PACKED(5,0) field   */
typedef char SRBUFFER[500];        /* typedef for send/receive
                                      buffer.  This program limits
                                      the amount of data to be sent
                                      or received to 500 bytes.  The
                                      maximum size of a management
                                      services transport buffer is
                                      31739.                          */

typedef struct {                   /* Library-qualified data queue
                                      name                            */
   QNAME data_queue_name;          /* data queue name                 */
   LIBNAME library_name;           /* library name                    */
  } QUALQNAME;

typedef struct {                   /* Error code structure            */
   int bytes_provided;             /* number of bytes provided        */
   int bytes_available;            /* number of bytes available       */
   MSGID exception_ID;             /* exception ID                    */
   char reserved_area;             /* reserved                        */
   EXCPDATA exception_data;        /* exception data                  */
  } ERRORCODE;

typedef struct {                   /* Notification record structure   */
   char record_type[10];           /* Record type                     */
   char function[2];               /* Function                        */
   HANDLE handle;                  /* Handle                          */
   REQUESTID req_id;               /* Request ID                      */
   char reserved[11];              /* Reserved area                   */
  } NOTIFRCD;

typedef struct {                   /* Receiver variable structure     */
   int bytes_provided;             /* number of bytes provided        */
   int bytes_available;            /* number of bytes available       */
   SRBUFFER received_data;         /* received data                   */
  } RECEIVERVAR;

typedef struct {                   /* Qualified application name      */
   NETID network_id;               /* Network ID                      */
   CPNAME cp_name;                 /* Control point name              */
   APPLNAME app_name;              /* Application name                */
```

```
  } QUALAPPL;

/*-------------------------------------------------------------------*/
/* External program declarations                                     */
/*-------------------------------------------------------------------*/
#pragma linkage(QNMSTRAP, OS)       /* Start application API         */
extern void QNMSTRAP (HANDLE *handle, /* pointer to handle           */
                      APPLNAME *applname,  /* pointer to application
                                   name                             */
                      QUALQNAME *qualqname,/* pointer to data queue
                                   name                             */
                      ERRORCODE *errorcode); /* pointer to error code
                                   parameter                        */

#pragma linkage(QNMENDAP, OS)       /* End application API           */
extern void QNMENDAP (HANDLE *handle, /* pointer to handle           */
                      ERRORCODE *errorcode); /* pointer to error code
                                   parameter                        */

#pragma linkage(QNMREGAP, OS)       /* Register application API      */
extern void QNMREGAP (HANDLE *handle, /* pointer to handle           */
                      CATEGORY *category, /* pointer to category     */
                      APPLTYPE *appltype, /* pointer to application
                                   type                             */
                      REPLREG *replreg, /* pointer to replace
                                   registration parameter           */
                      ERRORCODE *errorcode); /* pointer to error code
                                   parameter                        */

#pragma linkage(QNMDRGAP, OS)       /* Deregister application API    */
extern void QNMDRGAP (HANDLE *handle, /* pointer to handle           */
                      ERRORCODE *errorcode); /* pointer to error code
                                    set group                       */

#pragma linkage(QNMRCVDT, OS)       /* Receive data API              */
extern void QNMRCVDT (HANDLE *handle, /* pointer to handle           */
                      RECEIVERVAR *rcvvar, /* pointer to receiver
                                   variable                         */
                      int *rcvvarln, /* pointer to receiver variable
                                   length                           */
                      REQUESTID *reqid, /* pointer to request ID     */
                      QUALAPPL *qualappl, /* pointer to remote
                                   application name                 */
                      DATARCVD *datarcvd, /* pointer to type of data
                                   received                         */
                      int *waittim, /* pointer to wait time          */
                      ERRORCODE *errorcode); /* pointer to error code
                                   parameter                        */

#pragma linkage(QNMSNDRP, OS)       /* Send reply API                */
extern void QNMSNDRP (HANDLE *handle, /* pointer to handle           */
                      REQUESTID *reqid, /* pointer to request ID     */
                      SRBUFFER *sndbuf, /* pointer to send buffer    */
                      int *sndbufln, /* pointer to send buffer length*/
                      REPLYTYPE *rpltype, /* pointer to reply type   */
                      int *waittim, /* pointer to wait time          */
                      ERRORCODE *errorcode); /* pointer to error code
                                   parameter                        */

#pragma linkage(QNMRCVOC, OS)       /* Receive operation completion API
                                                                     */
extern void QNMRCVOC (HANDLE *handle, /* pointer to handle           */
                      REQUESTID *reqid, /* pointer to request ID     */
                      QUALAPPL *qualappl, /* pointer to remote
                                   application name                 */
                      ERRORCODE *errorcode); /* pointer to error code
                                   parameter                        */
```

```
#pragma linkage(QRCVDTAQ, OS)       /* Receive data queue            */
extern void QRCVDTAQ (QNAME *queue_name, /* pointer to queue name    */
                      LIBNAME *lib_name, /* pointer to library name  */
                      PACKED5 *rcd_len, /* pointer to record length  */
                      NOTIFRCD *notifrcd, /* pointer to notification
                                          record                    */
                      PACKED5 *waittime); /* pointer to wait time    */

void check_error_code (char func_name[8]); /* Used to check error
                                          code                      */

/*-------------------------------------------------------------------*/
/* Global declarations                                               */
/*-------------------------------------------------------------------*/
 HANDLE appl_handle;                /* Handle of application         */
 ERRORCODE error_code_struc =       /* Error code parameter          */
           {sizeof(error_code_struc),  /* Initialize bytes provided  */
            0,                      /* initialize bytes available    */
            NOERROR};               /* initialize error code         */
/*-------------------------------------------------------------------*/
/* Start of main function                                            */
/*-------------------------------------------------------------------*/
int main ()
{
/*-------------------------------------------------------------------*/
/* Local declarations                                                */
/*-------------------------------------------------------------------*/
 APPLNAME appl_name = "MSTTARG ";   /* Application name to be used   */
 QUALQNAME data_queue_parm =        /* Data queue name to be used    */
           {"MSTDTAQ   ", "QTEMP     "}; /* Initialize structure     */
 NOTIFRCD notif_record;             /* Area to contain notification
                                       record                       */
 RECEIVERVAR receiver_var =         /* Receiver variable             */
           {sizeof(receiver_var)};  /* Initialize bytes provided     */
 QUALAPPL qual_appl;                /* Qualified application name    */
 DATARCVD data_rcvd;                /* Type of data received         */
 CATEGORY category = "*NONE   ";    /* SNA/Management Services function
                                        set group                   */
 APPLTYPE appl_type = "*FPAPP    "; /* Application type              */
 REPLREG replace_reg = "*YES      "; /* Replace registration = *NO   */
 REPLYTYPE reply_cmp = REPLYCMP;    /* Complete reply                */
 REPLYTYPE reply_inc = REPLYINC;    /* Incomplete reply              */
 int sys_result;                    /* Result of system function     */
 int rcv_var_len = sizeof(receiver_var); /* Length of receiver
                                       variable                     */
 PACKED5 wait_time_p = "\x00\x00\x1D"; /* Packed value for wait time
                                       = -1, that is, wait forever  */
 PACKED5 record_len;                /* Length of received data queue
                                       record                       */
 int wait_forever = -1;             /* Integer value for wait time =
                                       -1, that is, wait forever    */
 int no_wait = 0;                   /* Do not wait for I/O to
                                       complete                     */
 char end_msg[] = "ENDRMTAPPL";     /* If this data is received then
                                       the application will end     */
 char incoming_data[] = "01";       /* Incoming data constant        */
 char inbuf[85];                    /* Input buffer                  */
 SRBUFFER send_buffer;              /* Send buffer for sending
                                       replies                      */
 int reply_len;                     /* Length of reply data          */

/*-------------------------------------------------------------------*/
/* Start of executable code                                          */
/*-------------------------------------------------------------------*/
 sys_result = system("DLTDTAQ DTAQ(QTEMP/MSTDTAQ)"); /* Delete
                                       previous data queue (if any) */
```

```
sys_result = system("CRTDTAQ DTAQ(QTEMP/MSTDTAQ) MAXLEN(80)"); /*
                                    Create data queue            */
QNMSTRAP (&appl_handle,
          &appl_name,
          &data_queue_parm,
          &error_code_struc);      /* Start application           */
check_error_code("QNMSTRAP");      /* Check error code            */
QNMREGAP (&appl_handle,
          &category,
          &appl_type,
          &replace_reg,
          &error_code_struc);      /* Register the application    */
check_error_code("QNMREGAP");      /* Check error code            */
while (memcmp(receiver_var.received_data,
              end_msg,
              sizeof(end_msg)) != 0)
  {                                /* Loop until an ending string
                                      has been sent by the requesting
                                      application                 */
    QRCVDTAQ (&data_queue_parm.data_queue_name,
              &data_queue_parm.library_name,
              &record_len,
              &notif_record,
              &wait_time_p);       /* Receive indication from data
                                      queue                       */
    if (memcmp(notif_record.function,
               incoming_data,
               sizeof(incoming_data)) == 0) /* Incoming data was
                                      received?                   */
      {
        strncpy(receiver_var.received_data,
                "\0",
                sizeof(receiver_var.received_data)); /* Null out the
                                      receive buffer              */
        QNMRCVDT (&appl_handle,
                  &receiver_var,
                  &rcv_var_len,
                  &notif_record.req_id,
                  &qual_appl,
                  &data_rcvd,
                  &wait_forever,
                  &error_code_struc); /* Receive data using the
                                      request ID in the notification*/
        check_error_code("QNMRCVDT"); /* Check error code          */
        printf("%1.500s\n",receiver_var.received_data); /* Display
                                      the received data           */
        if (memcmp(data_rcvd,
                   REQREPLY,
                   sizeof(data_rcvd)) == 0) /* Request requires
                                      a reply?                    */
          {
            printf("Please enter your replies (a null line "
                   "indicates that you are finished)\n"); /* Display
                                      a prompt message            */
            gets(inbuf);           /* Get the reply data          */
            reply_len = strlen(inbuf); /* Get length of reply     */
            while (reply_len != 0)  /* While no null string was input
                                                                  */
              {
                memcpy(send_buffer,inbuf,strlen(inbuf)); /* Copy
                                      data to send buffer         */
                QNMSNDRP (&appl_handle,
                          &notif_record.req_id,
                          &send_buffer,
                          &reply_len,
                          &reply_inc,
                          &no_wait,
```

```
                        &error_code_struc); /* Send a reply to the
                                    source application (specify
                                    "not last" reply).  The results
                                    of this operation will be
                                    obtained later using the
                                    receive operation completion
                                    API.                          */
                  gets(inbuf);       /* Get the next reply          */
                  reply_len = strlen(inbuf); /* Get length of reply   */
                }
              QNMSNDRP (&appl_handle,
                        &notif_record.req_id,
                        &send_buffer,
                        &reply_len,
                        &reply_cmp,
                        &no_wait,
                        &error_code_struc); /* Send final reply (this
                                    contains no data).  The results
                                    of this operation will be
                                    obtained later using the
                                    receive operation completion
                                    API.                          */
            }
          else
            {                        /* A reply is not required      */
              if (memcmp(data_rcvd,
                  REQUEST,
                  sizeof(data_rcvd)) != 0) /* Something other than a
                                    request was received?         */
                {
                  printf("Incorrect data was received, "
                         "data_rcvd = %1.10s\n", data_rcvd); /* Print
                                    value of data_rcvd            */
                }
            }

        }
      else
        {                           /* A send completion was received
                                       for a previous send reply
                                       operation                  */
          QNMRCVOC (&appl_handle,
                    &notif_record.req_id,
                    &qual_appl,
                    &error_code_struc);/* Receive operation completion*/
          check_error_code("QNMRCVOC"); /* Check error code          */
          printf("Reply was sent successfully.\n"); /* Error code was
                                    OK                            */
        }
    }
 QNMDRGAP (&appl_handle,
           &error_code_struc);      /* Deregister the application    */
 QNMENDAP (&appl_handle,
           &error_code_struc);      /* End the application           */

 return 0;
}

/*-------------------------------------------------------------------*/
/* check_error_code -                                                */
/*                                                                   */
/*   This function validates the error code parameter returned on    */
/*   the call to a management services transport API program.  If    */
/*   an error occurred, it displays the error that occurred and      */
/*   ends this program.                                              */
/*-------------------------------------------------------------------*/
void check_error_code (char func_name[8])
```

```
{
 char *sense_ptr = error_code_struc.exception_data + 36; /*
                                     Pointer to sense code in
                                     exception data               */
 SENSECODE sense_code;              /* SNA sense code              */
 if (error_code_struc.bytes_available != 0) /* Error occurred?      */
   {
     printf("\nError occurred calling %1.8s.\n",func_name);
     memcpy(sense_code,
            sense_ptr,
            sizeof(sense_code));    /* Copy sense code from exception
                                     data                         */
     printf("Error code is %1.7s, SNA sense code is %1.8s.\n",
                  error_code_struc.exception_ID,
                  sense_code);
     if (memcmp(func_name,
            "QNMSTRAP",
            sizeof("QNMSTRAP")) != 0)  /* Error did not occur on
                                     start application?           */
       {
         QNMENDAP (&appl_handle,
                   &error_code_struc); /* End the application      */

       }
     exit(EXIT_FAILURE);            /* Exit this program           */
   }
}
```

Using Source Debugger APIs

The ILE source debugger APIs allow an application developer to write a debugger for ILE programs. One might ask why this would ever be done when an ILE debugger is provided with OS/400. There are several reasons why an application developer might want to use these APIs to write a different ILE debugger:

- A debugger running on a workstation could be built that would debug ILE programs running on the iSeries server. This would allow a debugger to be written that would take advantage of Windows and other ease-of-use interfaces available on the workstation. The workstation debugger would communicate with code running on the iSeries server. The code running on the iSeries server would use the debugger APIs.
- The writer of an ILE compiler on the iSeries server might want to write a debugger to take advantages of the features of the language. The OS/400 debugger is a more general-purpose debugger made for all ILE languages.
- A debugger could be written with functions not available on the OS/400 ILE debugger.

Source Debugger APIs--Overview

The ILE source debugger APIs can be divided into several groups. These include APIs that:

- Start and end the debug session
- Add programs and modules to debug
- Manipulate text views in a program
- Add and remove breakpoints, steps, and so on

Besides APIs, there are two user exits that get called:

- The Source Debug program gets called when the Start Debug (STRDBG), Display Module Source (DSPMODSRC), and End Debug (ENDDBG) CL commands are entered.

- The Program Stop Handler gets called when an ILE program being debugged hits a breakpoint, step, and so on.

To demonstrate how these APIs are used, this topic presents an example debugger with complete code examples and an explanation of what the APIs do.

The ILE debugger that comes with OS/400 uses the debugger APIs just as a user-written debugger would. There is nothing special about the OS/400 debugger. Its functions could be done by an application developer using the debugger APIs and other OS/400 APIs.

A Simple Debugger--Scenario

Consider a simple scenario in which the user wishes to debug an ILE program.

1. From the command entry screen, the user enters the Start Debug (STRDBG) command, passing it the name of an ILE program to debug.

   ```
   STRDBG P1
   ```

2. The ILE debugger screen is displayed, showing the source of a module in the ILE program being debugged. From this screen, the user adds a breakpoint and then exits.

3. Back at the command entry screen, the user runs the ILE program that is being debugged.

   ```
   CALL P1
   ```

4. The ILE program hits the breakpoint previously set. The ILE debugger screen is displayed, highlighting in the source where the program has stopped at the breakpoint.

5. The user displays a variable in the program being debugged.

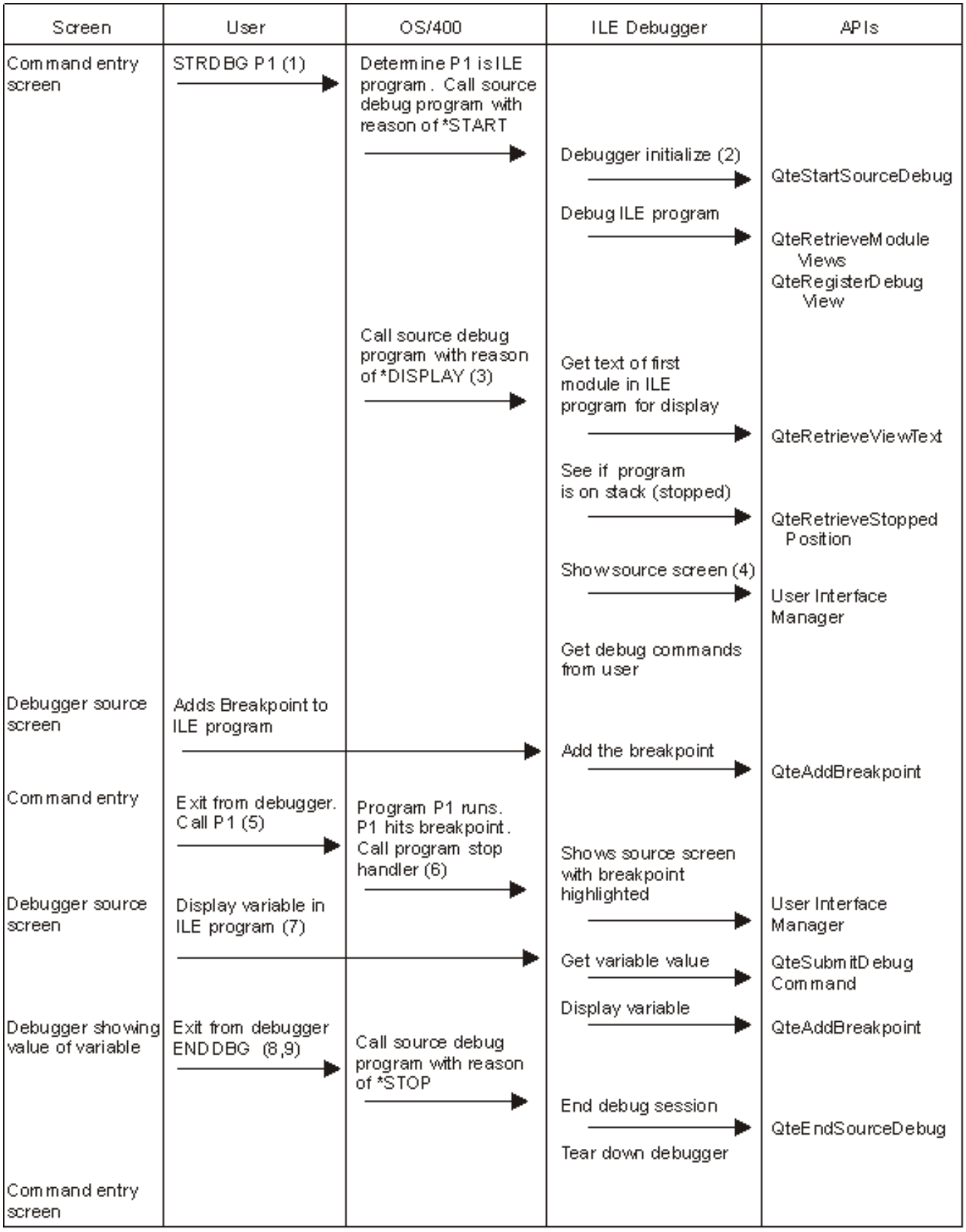
6. The user exits the ILE debugger, allowing the ILE program to run to completion. The program ends.

7. Back at the command entry screen, the user ends the debug session.

   ```
   ENDDBG
   ```

This is the simplest of debug scenarios, but it illustrates how OS/400, the debugger user exits, and the debugger APIs interact.

The following figure shows the various interactions.



A detailed explanation of the scenario follows:

1. The Start Debug (STRDBG) CL command is used to start the debug session. By default, if an ILE program is specified on the command, the OS/400 ILE debugger user exit is called. A different user exit (called the Source Debug program) can be

specified on the Start Debug command by specifying a program name on the SRCDBGPGM parameter.

When the Source Debug program is called, it is passed a reason field, which indicates why it was called. The *START reason is passed to it by the Start Debug command, indicating that the ILE debugger is to start itself and do any necessary initialization. When the *START reason is indicated, the names of any ILE programs on the Start Debug command are also passed to the Source Debug program.

2. In this scenario, the system Source Debug program initializes itself. It calls the QteStartSourceDebug API, which tells the system that ILE debugging is to be done. The name of a program stop handler program is passed to this API. The stop handler is a program that the system calls when an ILE program hits a breakpoint, step, or other condition where the system stops the program for the debugger.

   The Source Debug program must indicate to the system that the ILE programs specified on the Start Debug command are to be debugged. To do this, the QteRetrieveModuleViews API is called, once for each ILE program specified on the Start Debug command. In this scenario, the API is called, passing it the name of program P1. The purpose of the API is to return information about the ILE program, including the modules and views of the program. A view is the source text that is displayed by the debugger for a particular module.

   Once information about the ILE program is obtained, one or more views of the program must be registered. Once a view is registered, the system can perform various functions on that view in behalf of the debugger application. For performance reasons, only the views the user is interested in displaying should be registered.

   The Source Debug program is now done performing the function for the *START reason. It exits, returning control to the Start Debug command.

3. By default, if an ILE program is specified on the Start Debug command, the ILE debug screen is displayed. To indicate to the ILE debugger that a screen is to be put up, the Source Debug program is called by the command again, this time with a reason of *DISPLAY.

   Because this is the first time any views for P1 are to be displayed, the ILE debugger must retrieve the text to display. The first view of the first module of the program is selected as the default view to display.

   The Source Debug program calls the QteRetrieveViewText API to retrieve the text associated with the default view. Next, in case this program is already on the stack and stopped, the QteRetrieveStoppedPosition API is called to check. If the program were on the stack, the source would be positioned to the statement where the program was stopped, and that line would be highlighted. In this scenario, the program is not yet on the stack, so the first line of the source will appear at the top of the screen, and no line will be highlighted.

   The Source Debug program next calls User Interface Manager (UIM) APIs to display the source on the screen.

4. At this point, the source screen is displayed showing the text of the first view in the first module of the first ILE program specified on the Start Debug command. From this screen, the user can enter debug commands or do other options provided by the debugger application.

   In this scenario, the user adds a breakpoint to a line in the ILE program P1 being debugged. When a command is entered, the UIM APIs call a program which is part of the ILE debugger to process the command.

   To process the breakpoint, the QteAddBreakpoint is called. It is passed a view number which indicates the view being displayed, and a line number in that view. A breakpoint is added to the program by the API.

5. Back to the UIM screen, the user exits the ILE debugger. Once at the command entry screen, the user then runs the program P1 which has the breakpoint.

6. When P1 hits the breakpoint, the system calls the program stop handler defined by the QteStartSourceDebug API. The Program Stop Handler calls UIM to put up the source for the module where the program has stopped because of the breakpoint. The line is highlighted to show the user exactly where the program has stopped.

7. From the source debugger screen, the user displays a variable in program P1 which is stopped at the breakpoint. UIM calls the debugger to process the command. The debugger calls the QteSubmitDebugCommand API, which retrieves the value of the variable to be displayed. The debugger then displays this value on the screen.

8. The user now exits from the source debugger screen. This allows P1, which was stopped at a breakpoint, to continue running. When P1 ends, the user is back at the command entry screen.

9. The user ends the debug session by entering the End Debug (ENDDBG) CL command. The system calls the Source Debug program, passing it a reason of *STOP. The Source Debug program calls the QteEndSourceDebug API to indicate to the system that ILE debugging has ended. It then tears down its own environment (closes files, frees space, and so on) and then ends. The End Debug command completes, and the user is back to the command entry, the debug session having ended.

Source Debugger--Example

This section discusses an example ILE debugger that demonstrates the use of some of the ILE debugger APIs. Each function in the C program is discussed along with the APIs that they call. Although the entire program listing is printed later (see Debugger Code--Sample), each function or piece of code is printed with the section where it is discussed to make reading the code easier.

The example debugger does not use all ILE debugger APIs. Its function is limited. After the discussion of the code, the APIs and some functions not covered are discussed.

Compiling the Debugger

The Create C Module (CRTCMOD) command compiles the source code of the debugger. It is compiled into module DEBUG.

The Create Program (CRTPGM) command creates program DEBUG from module DEBUG. It is necessary to bind to service program QTEDBGS so that the calls to the debugger APIs are resolved. It is also important to use activation group QTEDBGAG. This is an activation group that cannot be destroyed while the job is in debug mode. Thus, all static variables in program DEBUG remain intact throughout the debugging of the ILE program. Only when ENDDBG is entered can the activation group be destroyed, even if the Reclaim Resources (RCLRSC) CL command is entered.

Starting the Debugger

The example debugger consists of a single program called DEBUG. The program is used as the Source Debug program as well as the Program Stop Handler. The program determines how many parameters it is being called with, and with this information it does the function of one or the other of the user exits.

The debugger can debug only one ILE program. This program is specified on the Start Debug CL command. The program cannot be removed from debug until ENDDBG is done. No new programs can be added.

To debug an ILE program P1 with this sample debugger, the following CL command could be entered:

```
STRDBG P1 SRCDBGPGM(DEBUG)
```

Note that DEBUG must be in the library list when STRDBG is done.

If the command is done, P1 is called twice, once as a Source Debug program given a reason of *START, and again as a Source Debug program given a reason of *DISPLAY.

Other variations of the Start Debug command can be given with different results. For example, the following CL command causes DEBUG to be called only once with a reason of *START:

```
STRDBG P1 SRCDBGPGM(DEBUG) DSPMODSRC(*NO)
```

This is because STRDBG has been told not to display the debug screen, so the *DISPLAY reason is not given until the user does the Display Module Source (DSPMODSRC) CL command.

The following example does not even call DEBUGGER:

```
STRDBG SRCDBGPGM(DEBUG)
```

This is because no ILE program is specified. If an ILE program receives an unmonitored message and the ILE debugger needs to be called, DEBUG is first called with *START as a Source Debug program. Also, if Display Module Source is entered, the *START and then the *DISPLAY reason is passed to DEBUG.

Using the Debugger

When the debugger is started, it allows simple debugging commands to be entered. The C session manager is put up, which scrolls the users commands and the debugger output. To see a list of the allowable commands, enter HELP.

The "list views" command shows all of the views available in the program being debugged. The text description of the view is listed, with a sequential number. This number is used by the "switch" command to switch to that view.

The "list text" command prints out the text of the current view. Text has a line number next to it. The line number is used when setting breakpoints or other debug commands.

The switch command switches the current view. The current view is the view used when setting breakpoints, displaying variables,

viewing text, and so on.

The "quit" command exits the debugger.

Other commands are interpreted by the QteSubmitDebugCommand API. This API will be discussed later. An example command that can be entered is "break n", where n is the line number in the current view. These commands are similar to the ones allowed in the ILE debugger shipped with OS/400.

Header Files Used in Debugger

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <qtedbgs.h>
```

Besides the normal C library header files, an API header file, qtedbgs.h is included. This file defines the functions exported from service program QTEDBGS. This service program contains the ILE debugger APIs.

Global Variables

```
static _TE_VEWL0100_T *pgm_dbg_dta = NULL;
static long current_view = 0;       /* current view - defaults to 1st*/
static _TE_OBJLIB_T program_lib;    /* name and lib of pgm debugged  */
```

These are global variables that hold information about the program being debugged. These variables do not go away when program DEBUG exits, because they are stored in the activation group which is not destroyed until the debug session has completed.

The name and library of the program are stored, as is the current view being debugged. Also, a pointer to a structure returned by the QteRetrieveModuleViews is saved, as this information is needed when debugging the various views of the program.

PgmList_t

```
typedef struct {
    _TE_OBJLIB_T PgmLib;            /* Name and Library of program   */
    _TE_NAME_T PgmType;             /* program type, *PGM or *SRVPGM */
} PgmList_t;
```

This is the structure of the name, library, and type of the program being debugged.

main()

```
main (int argc, char *argv[]) {
  if (argc == 4)                    /* called as source debug program*/
    HandleSession(argv[1], (PgmList_t *)argv[2], *(int
*)argv[3]);
  else if (argc == 8)               /* called as program stop handler*/
    HandleStop((_TE_OBJLIB_T *)argv[1], argv[2],
argv[3], argv[4],
              (long *)argv[5], *(int *)argv[6],
argv[7]);
}
```

Program DEBUG can be called in two ways. When it is called by the STRDBG, DSPMODSRC, and ENDDBG CL commands, it is called as the Source Debug program user exit. It is passed three parameters.

DEBUG can also be called when a program being debug hits a breakpoint or step. In this case, it is passed seven parameters.

DEBUG therefore can determine why it was called by counting the number of parameters it was passed. Remember that argc includes the program name as the first argument passed.

If argc is 4 (three parameters passed to DEBUG), function HandleSession is called, and the three parameters passed to DEBUG are passed to it, typecasted as needed.

If argc is 8 (seven parameters passed to DEBUG), function HandleStop is called, and the seven parameters passed to DEBUG are passed to it, typecasted as needed.

If any other number of parameters are passed to DEBUG, it cannot have been called from the OS/400 debug support, so DEBUG will just exit.

HandleSession()

```
void HandleSession(char reason[10],
                   PgmList_t ProgramList[],
                   int ProgramListCount) {

  if (memcmp(reason,"*START    ",10) == 0) /* reason is *START       */
    StartUpDebugger(ProgramList, ProgramListCount);
  else if ( memcmp(reason,"*STOP     ",10) == 0) /* reason is *STOP  */
    TearDownDebugger();
  else if ( memcmp(reason,"*DISPLAY  ",10) == 0) /* reason *DISPLAY  */
    ProcessCommands();
}
```

When DEBUG is called as a session handler, it is passed three parameters. The first parameter is a 10-character array containing a reason field. This contains the reason why the session handler is called.

When DEBUG is first called, it is passed a reason of *START, indicating that the debugger is to initialize for an ILE debug session. When this reason is given, the second parameter contains a list of ILE programs specified on the STRDBG command, and the third parameter contains the number of programs specified on parameter two. From 0 to 10 ILE programs can be specified.

When the user wishes to see the ILE debugger screen, either from STRDBG or DSPMODSRC, a reason of *DISPLAY is passed. When the user enters ENDDBG, the *STOP reason is passed, indicating that the ILE debug session is ending. The second and third parameters are not used when the reason is *DISPLAY or *STOP.

The code tests for a reason and calls the appropriate function. There is one function for each reason that can be passed.

TearDownDebugger()

```
void TearDownDebugger(void) {
  _TE_ERROR_CODE_T errorCode = {8}; /* errors will be ignored        */

  /* Call EndSourceDebug to get out of ILE debug mode               */
  QteEndSourceDebug(&errorCode);

  exit(0);                          /* destroy activation group     */
}
```

This function is called when the user enters ENDDBG. The debugger calls the QteEndSourceDebug API which ends ILE debugging. Since an 8 is passed as the number of bytes provided, the message ID and error data from an error are not returned to the caller. Thus, any errors from this API (there should not be any) are ignored.

The exit() function is called, which destroys the activation group. Thus, all global data defined in the program's variables are lost. This is ok, since the debug session is ending at this point.

StartUpDebugger()

```
void StartUpDebugger(PgmList_t ProgramList[],
                     int ProgramListCount) {

  _TE_ERROR_CODE_T errorCode = {0}; /* exceptions are generated     */
  _TE_OBJLIB_T StopHandler = {"DEBUG     ", "*LIBL     "};
  int i;

  if (ProgramListCount!=1) {        /* is only 1 pgm passed on STRDBG*/
    printf("Exactly ONE program must be specified on STRDBG.\n");
    TearDownDebugger();             /* end debugger                 */
  }

  /* Copy program name to global variables                          */
  memcpy(&program_lib, &ProgramList->PgmLib, 20);

  /* Call StartSourceDebug: giving the name and library of the      */
  /* stop handler.  This will start ILE debug mode                  */
```

```
  QteStartSourceDebug(&StopHandler, &errorCode);

  AddProgram();                     /* add program to debug          */
}
```

This function is passed the second and third parameters which were passed from the system when it called DEBUG with a reason of *START. These parameters are the list of programs to be added to debug and the number of programs in the list. This simple example debugger can only debug one program, so if any other number of programs were specified on STRDBG, the debugger just exits.

StartUpDebugger first stores the program/library element passed to it in a global variable available to all functions. This is the name and library of the program being debugged. It then calls the QteStartSourceDebug API to tell the system that an ILE debug session is to begin. The name and library of program DEBUG are passed to this API as the Program Stop Handler. Thus, whenever the program being debugged is stopped by the debugger, program DEBUG will be called.

Finally, the function calls AddProgram to add the single program to debug.

AddProgram()

```
void AddProgram(void) {

  _TE_ERROR_CODE_T errorCode = {0};    /* Signal exceptions on error */
  _TE_NAME_T       Library;            /* Lib returned               */
  _TE_TIMESTAMP_T  TimeStamp;          /* TimeStamp returned         */
  int viewIndex;
  long int         iViewID;
  long int         iViewLines;
  long rtvModViewDataLength = 8;       /* size of receiver buffer    */
  char tempBuffer[8];        /* enough room for header only*/
  int i, tempModuleCount;

  /* Call QteRetrieveModuleViews to determine the number of bytes    */
  /* the receiver variable needs to be to hold all of the views for  */
  /* the program.                                                    */
  pgm_dbg_dta = (_TE_VEWL0100_T *)tempBuffer;
  QteRetrieveModuleViews((char *)pgm_dbg_dta, &rtvModViewDataLength,
                         "VEWL0100", &program_lib,
                         "*PGM      ", "*ALL      ", Library,
                         &errorCode);

  /* Get a buffer large enough to hold all view information          */
  rtvModViewDataLength = pgm_dbg_dta->BytesAvailable;
  pgm_dbg_dta = (_TE_VEWL0100_T *)malloc(rtvModViewDataLength);

  /* Call QteRetrieveModuleViews again, passing a big enough buffer. */
  QteRetrieveModuleViews((char *)pgm_dbg_dta, &rtvModViewDataLength,
                         "VEWL0100", &program_lib,
                         "*PGM      ", "*ALL      ", Library,
                         &errorCode);

  /* If number of elements is zero, program is not debuggable.       */
  if (pgm_dbg_dta->NumberElements == 0) {
    printf("Program %.10s in Library %.10s cannot be debugged.",
           program_lib.obj, program_lib.lib);
    TearDownDebugger();
  }

  /* Put the library returned by Retrieve Module Views in PgmLib     */
  memcpy(program_lib.lib, Library, sizeof(_TE_NAME_T));

  /* Register all views in the program                               */
  for (i=0; i < pgm_dbg_dta->NumberElements; i++) {
    QteRegisterDebugView(&iViewID, &iViewLines, Library, TimeStamp,
                         &program_lib, "*PGM      ",
                         pgm_dbg_dta->Element[i].ModuleName,
                         &pgm_dbg_dta->Element[i].ViewNumber,
                         &errorCode);
```

```
    /* overwrite unneeded ViewNumber with obtained view id           */
    pgm_dbg_dta->Element[i].ViewNumber = iViewID;
  }
}
```

The heart of this function is the two calls to the QteRetrieveModuleViews API and the call to QteRegisterDebugView API.

The QteRetrieveModuleViews API returns information about an ILE program. It returns this information in a structure of type _TE_VEWL0100_T. This is a fairly complex structure that has the following fields:

```
typedef _Packed struct {            /* format VEWL0100               */
  long int BytesReturned;           /* number of bytes returned      */
  long int BytesAvailable;          /* number of bytes available     */
  long int NumberElements;          /* number of elements returned   */
  _Packed struct {                  /* one element                   */
    _TE_NAME_T ModuleName;          /* name of module in program     */
    _TE_NAME_T ViewType;            /* type of view:                 */
    _TE_COMPILER_ID_T CompilerID;   /* compiler ID                   */
    _TE_NAME_T MainIndicator;       /* main indicator                */
    _TE_TIMESTAMP_T TimeStamp;      /* time view was created         */
    _TE_TEXTDESC_T ViewDescription; /* view description              */
    char Reserved[3];
    long int ViewNumber;            /* view number within module     */
    long int NumViews;              /* number of views in this module*/
  } Element[1];                     /* one element        */
} _TE_VEWL0100_T;
```

This structure has a header portion which holds the number of bytes returned by the API (BytesReturned), the number of bytes that can be returned by the API, used when there is not enough room for the API to return all of its data (BytesAvailable), and the number of elements (views) returned by the API (NumberElements).

Since there is no way to know in advance how many views a program has, the QteRetrieveModuleViews API should be called once with only enough storage to return the number of bytes that the API needs to return all of its information. Thus, the first call to the API provides only 8 bytes of storage for the API to return its data. This allows the API to fill in the BytesAvailable field.

QteRetrieveModuleViews is passed a buffer to hold the receiver variable and the length of that buffer (in this case, 8 bytes). It is also passed a format name which identifies the structure of the receiver variable. The only allowable format name at this time is VEWL0100. A structure containing the program name and library name of the ILE program is passed. Also, the program type is passed. In this example debugger, only *PGM objects can be debugged, but it is possible to debug *SRVPGM objects using the ILE debugger APIs.

The name of the module is provided, in which case information about that module is returned. *ALL indicates that information about all modules in the program is to be returned. A return library variable is passed. This is so that when *LIBL is passed as a library name, the real library name can be obtained, making subsequent API calls faster because the library list won't have to be searched again.

Finally an error code structure is passed to the API. This structure is initialized with a zero, indicating that the API is not to fill in any error code data. Instead, the API issues an exception if an error occurs. No errors are expected, so this should not matter.

Before QteRetrieveModuleViews is called again, a buffer large enough to hold all of the information is created. The API is called again with the same parameters, but this time the entire information will be stored by the API in the allocated buffer.

If the API does not return any elements, this means that none of the modules has debug data. In this case, the program cannot be debugged, so the debug session is ended.

Now that a list of views has been retrieved, it is time to register all of the views to the system, making it possible to do debug operations against them. In a real debugger, only the views requested to be seen by the user would be registered to save processing time, but in this example, all views will be registered at once.

Not all of the fields in the VEWL0100 structure are needed by this debugger. However, they will be described here. The API returns one element for each view in the program. Each module in the program may have several views. All views for a particular module are contiguous in the list.

ModuleName This is the name of the module in the program which this particular view is for.

ViewType	This indicates the type of view. A *TEXT view contains text retrieved from source files residing on the iSeries server. The text contains sequence information from these files that the debugger may not want to display. A *LISTING view contains text that is stored with the program object itself. A *STATEMENT view contains information about HLL statements in the module, and this information is not generally displayed to the user but is used by the debugger. In the case of this debugger, all views are displayed exactly as the text for the views are retrieved.
CompilerID	This indicates the language that the particular module is written in. This is not used by the example debugger.
MainIndicator	Only one module in a program is the module with the program entry procedure (main() in the case of ILE C programs). If a particular view in the list comes from this module, then this field indicates that the module contains this procedure. This field is not used by the example debugger.
TimeStamp	This indicates when the view was created. This is useful in allowing the debugger to detect if a program has been recompiled and the debugger has down-level view information. This field is not used by the example debugger.
ViewDescription	This is text given to the view by the compiler creating the view. It is a description of the view which can be displayed by the debugger.
ViewNumber	This is a sequence number of the view in a particular module. When registering a view, the program name, module name, and view number must be provided.
NumViews	This is how many views are in the module. All elements for views in a given module have the same value for this field. This field is not used by the example debugger.

A loop through all the views returned by QteRetrieveModuleViews is done, registering the view using the QteRegisterDebugView API. The program name, program type, module name, and view number of the module are passed as inputs to the API. The API returns the library of the program (in case *LIBL) is passed in as the program library), the timestamp of the view (in case the program has been recompiled between the time the view information was retrieved and the time the view was registered), the number of lines of text in the view, and a view ID. The view ID is a handle, and it is used in identifying the registered view to various APIs. For example, when retrieving text for a particular view, the view must be registered, and the view ID returned when registering the view is passed to the QteRetrieveViewText API.

The structure that held the views retrieved by QteRetrieveModuleViews is also used by the debugger. The view number is no longer needed, since it is just a sequence number passed to QteRegisterDebugView. Thus, this number is overwritten and will hold the view ID, which is needed by other debugger APIs.

ProcessCommands()

```
void ProcessCommands(void) {
  char InputBuffer[80];
  char *token;
  int i;
  int step=0;                       /* do an exit for step when 1    */

  if (pgm_dbg_dta == NULL) {        /* if no debug data              */
    printf("Debug session has ended.\n");
    exit(0);                        /* end the debugger              */
  }

  while(!step) {                    /* read until step or quit cmd   */
    ReadLine(InputBuffer,sizeof(InputBuffer));
    token = strtok(InputBuffer," ");

    if (token==NULL) continue;      /* ignore blank lines            */
    else if (strcmp(token,"quit") == 0) /* the quit command?         */
      return;                       /* exit debugger                 */
    else if (strcmp(token,"list") == 0) /* the list command?         */
      ProcessListCommand();         /* process command               */
    else if (strcmp(token,"switch") == 0) { /* switch command?       */
      token = strtok(NULL," ");     /* get view number token         */
      if (token == NULL)
        printf("'switch' must be followed by a view number.\n");
      else
        current_view = atoi(token);   /* switch current view         */
    }
    else if (strcmp(token,"help") == 0) {
      printf("The following are the allowed debugger commands:\n");
      printf("  list views - lists all views in the program\n");
      printf("  list text - lists the text of the current view\n");
```

```
      printf("  switch n - changes current view to view n\n");
      printf("  help - displays this help text\n");
      printf("  quit - ends the debug session\n");
      printf("Other commands are interpreted by the debug support.\n");
    }
    else {                          /* pass command to API           */
      /* Undo modifications that strtok did                          */
      InputBuffer[strlen(InputBuffer)] = ' ';

      step = ProcessDbgCommand(InputBuffer);
    }
  }
}
```

This function reads an input line from the user and processes it. If it is a command recognized by the debugger, it process it. If not, it calls ProcessDebugCommand which lets QteSubmitDebugCommand process the command.

The first test is to make sure that the pointer to the debug data is not null. This is here for safety reasons. If program DEBUG is compiled with the wrong activation group name or no name at all, its global variables can be destroyed when the program exits, causing problems when the program is called again. This test prevents debug commands from being entered if the activation group has been destroyed, wiping out the global view data.

The function loops until the quit command is entered or until a step is done. It calls the appropriate function based on the command entered, or displays an error message if a syntax error is detected. If the command is unknown, it is processed by ProcessDbgCommand.

The switch command is processed directly by the function. It changes the current view to a number provided. There is no error checking in this sample debugger.

ReadLine()

```
void ReadLine(char *Buffer, int length) {
  int i;                            /* loop counter                  */

  printf("Enter a debugger command or 'help'.\n");
  fgets(Buffer,length,stdin);       /* read line of text             */

  /* Blank out line from \n to the end of the string.               */
  for (i=0; i<length; i++) {        /* loop, searching for newline   */
    if (Buffer[i] == '\n') {        /* if newline character found
*/
      break;                        /* end loop searching for newline*/
    }
  }

  memset(Buffer+i,' ',length-i);     /* blank remainder of line      */
}
```

This function reads a line of text from the user and fills the input buffer with trailing blanks.

ProcessListCommand()

```
void ProcessListCommand(void) {
  char *token;                      /* pointer to next token of input*/

  token = strtok(NULL," ");         /* get next token in input buffer*/

  if (token==NULL)                  /* list not followed by anything */
    printf("'list' must be followed by 'views' or 'text'.\n");
  else if (strcmp(token,"views") == 0)/* if list views               */
    PrintViews();
  else if (strcmp(token,"text") == 0) /* if list text                */
    PrintText();
  else                              /* list <something-else>         */
    printf("'list' must be followed by 'views' or 'text'.\n");
}
```

This routine process the list command. There are two versions of the list command, list views and list text. The appropriate function is called depending on the type of list command entered, or a syntax error message is issued.

PrintViews

```
void PrintViews(void) {
  int k;

  /* loop through views printing view#, module, and view desc. text  */
  for (k=0; k< pgm_dbg_dta->NumberElements; k++) {
    printf("%d) %.10s:%.50s",
           k,
           pgm_dbg_dta->Element[k].ModuleName,
           pgm_dbg_dta->Element[k].ViewDescription);
    if (current_view == k)         /* indicate if view is current   */
      printf("<---Current\n");
    else
      printf("\n");
  }
}
```

This routine lists all of the views available in the program being debugged. The information about the views is stored in the buffer that was passed to QteRetrieveModuleViews.

The module name and view descriptive text is printed for each view. If the current view being printed is also the current view, this is noted by printing this fact next to the view information.

A view number is printed next to each view. This is not the view ID returned by the QteRegisterDebugView. It is a number allowing the user to change the current view to one of the views in the list.

PrintText()

```
void PrintText(void) {

  long LineLength = 92;             /* length of lines of text       */
  long NumberOfLines = 0;           /* lines to retrieve - 0 = all   */
  long StartLine=1;                 /* retrieve from line 1 (first)  */
  long bufferLength = 100000;       /* size of retrieved text buffer */
  long viewID;                      /* view ID of text to retrieve   */
  _TE_TEXT_BUFFER_T *buffer;        /* text retrieved by API         */
  _TE_ERROR_CODE_T errorCode = {0}; /* Exceptions will be signaled   */
  int i;                            /* points to start of each line  */
  int line_number;                  /* line number counter for loop  */

  /* Get View ID of current view                                     */
  viewID = pgm_dbg_dta->Element[current_view].ViewNumber;

  buffer = malloc(bufferLength);    /* malloc space for big text buf */

  /* Call Retrieve_View_Text for the current view.                   */
  QteRetrieveViewText((char *)buffer, &bufferLength, &viewID,
                      &StartLine, &NumberOfLines, &LineLength,
                      &errorCode);

  /* Print out the text                                              */
  for (i=0,line_number=1;
       line_number <= buffer->NumLines;
       line_number++,i+=LineLength) {
     printf("%3d) %.70s\n", line_number, buffer->Text+i);
  }

  free(buffer);                     /* free memory for buffer        */
}
```

This function retrieves the text associated with the current view and prints it. This text is the source of the program and is the heart of a source debugger screen.

The text of the current view is retrieved, so the view ID of that view is determined. It is this view that is passed to QteRetrieveViewText.

In the sample debugger, a large buffer is allocated, and as much text as will fit in this buffer is retrieved. The QteRetrieveViewText API returns the text and the number of lines that fit in the buffer.

Once the text is retrieved, it is printed out along with the line number. The line number is needed when setting breakpoints based on the view.

ProcessDbgCommand()

```
int ProcessDbgCommand(char InputBuffer[80]) {
  _TE_ERROR_CODE_T errorCode = {64}; /* fill in bytes provided       */
  char OutputBuffer[4096];
  struct _TE_RESULT_BUFFER_T *Results;
  long InputBufferLength = 80;
  long OutputBufferLength = sizeof(OutputBuffer);
  long view_ID;
  _TE_COMPILER_ID_T *CompilerID;
  int i;
  int return_value = 0;

  view_ID = pgm_dbg_dta->Element[current_view].ViewNumber;
  CompilerID = &pgm_dbg_dta->Element[current_view].CompilerID;

  /* Give command to QteSubmitDebugCommand                          */
  QteSubmitDebugCommand(OutputBuffer, &OutputBufferLength,
                        &view_ID, InputBuffer, &InputBufferLength,
                        *CompilerID, &errorCode);

  if (errorCode.BytesAvailable != 0) {
    printf("Error = %.7s\n",errorCode.ExceptionID);
    return return_value;
  }

  /* Process results from QteSubmitDebugCommand                     */
  Results = (_TE_RESULT_BUFFER_T *) OutputBuffer;

  /* Loop through Results array                                     */
  for (i=0; i<Results->Header.EntryCount; i++) {
    switch (Results->Data[i].ResultKind)
    {
      case _TE_kStepR            :
        printf("Step set\n");
        return_value=1;             /* indicate step is to be done   */
        break;
      case _TE_kBreakR           :
        printf("Breakpoint set");
        break;
      case _TE_kBreakPositionR   :
        printf(" at line %d\n",
               Results->Data[i].V.BreakPosition.Line);
        break;
      case _TE_kExpressionTextR  :
        printf("%s",
               ((char *)Results) + Results->Data[i].V.
               ExpressionText.oExpressionText);
        break;
      case _TE_kExpressionValueR :
        printf(" = %s\n",
               ((char *)Results) + Results->Data[i].V.
               ExpressionValue.oExpressionValue);
        break;
      case _TE_kQualifyR         :
        printf("Qual set\n");
        break;
      case _TE_kClearBreakpointR :
```

```
        printf("Breakpoint cleared\n");
        break;
      case _TE_kClearPgmR :
        printf("All breakpoints cleared\n");
        break;
      default:                      /* ignore all other record types */
        break;
    }                               /* switch                        */
  }                                 /* loop through results array    */
  return return_value;
}
```

This function is called to process all commands not known by the debugger. It calls the QteSubmitDebugCommand API which is passed a view ID, compiler ID, and a command. The API needs the compiler ID because each programming language used in compiling a particular module has different debug commands or command syntax, and the API needs to know which language was used when compiling the module.

The API returns back a series of result records which indicate what was done by the API. Most of this function reads the results of the records returned and prints an appropriate response message.

Some results records indicate that a particular function has been performed. These include:

_TE_kStepR	The step command was successfully done.
_TE_kBreakR	The break command was successfully done.
_TE_kQualifyR	The qual command was successfully done.
_TE_kClearBreakpointR	The clear breakpoint command was successfully done.
_TE_kClearPgmR	The clear pgm command was successfully done.

Other results records contain numeric data useful by the debugger.

_TE_kBreakPositionR	Contains the line number where a breakpoint was set. It is possible that a breakpoint set on two different lines will correspond to the same HLL statement. In this case, only one breakpoint is really set. To determine if this is the case, it is necessary to map the position in the view where the breakpoint is set to a position in the statement view.

Still other results records contain string data. In this case, the record contains an offset into the string space returned by the API as well as a string length.

_TE_kExpressionTextR	This points to the expression entered in the eval command.
_TE_kExpressionValueR	This points to the value of the evaluated expression.

There are other kinds of results records than processed by the sample debugger. The QteSubmitDebugCommand API discusses in detail each result record and the data it contains.

The API description also discusses the syntax of the debug command that must be passed to it. The commands and their syntax will not be discussed in depth here, but a few example commands will be shown:

- break 5 when x == 3

  This is a conditional breakpoint. The debugger will stop the program indicated by the view ID passed to the API when it reaches line 5 of the view and when the expression "x == 3" is true. The "when" part of the break statement is optional, in which case an unconditional breakpoint is set.

- step 1 into

  The step command instructs the debug support to stop the a program when it has executed one or more statements. In this example, the program is stopped after 1 statement has been executed. The "into" means that statements in procedures are counted when stepping. "over" means that statements in called procedures are skipped over and not counted. The default step type is "into", and the default step count is 1.

- qual 13

  The qual command is necessary when there are blocks of code with the same variable name. In this case, the user indicates where the variable is searched for in the program. Normally, this command is not used.

- clear 8

  A conditional or unconditional breakpoint is removed from line 8 of the view indicated by the view ID parameter.

HandleStop()

```
void HandleStop(_TE_OBJLIB_T *ProgramLib,
                _TE_NAME_T ProgramType,
                _TE_NAME_T Module,
                char reason[10],
                long Statements[],
                int StatementsCount,
                char *message) {
  int i;
  _TE_MAPP0100_T Map_Return_Structure;
  long Column = 1;
  long MapLength = sizeof(Map_Return_Structure);
  _TE_ERROR_CODE_T errorCode = {64};
  long stmt_view;

  /* If current view is for a different module than the one that is  */
  /* stopped, change current view to first view in the stopped module*/
  if (memcmp(Module,
             pgm_dbg_dta->Element[current_view].ModuleName,
             sizeof(_TE_NAME_T)) != 0) { /* a different module?      */
    for (i=0; i<pgm_dbg_dta->NumberElements; i++) {
      if (memcmp(Module,
                 pgm_dbg_dta->Element[i].ModuleName,
                 sizeof(_TE_NAME_T)) == 0) { /* found module         */
        current_view = i;           /* change current view to module */
        printf("Current view changed to %d.\n",current_view);
        break;                      /* exit search loop             */
      }                             /* module found                 */
    }                               /* loop through views           */
  }                                 /* current view to be changed    */

  /* Get number of statement view for module stopped                 */
  for (i=0; i<pgm_dbg_dta->NumberElements; i++) {
    if ((memcmp(Module,
                pgm_dbg_dta->Element[i].ModuleName,
                sizeof(_TE_NAME_T)) == 0) &&
        (memcmp("*STATEMENT",
                  pgm_dbg_dta->Element[i].ViewType,
                  sizeof(_TE_NAME_T)) == 0))
      stmt_view = i;
  }


  /* Call QteMapViewPosition to map the stopped location (which      */
  /* is in terms of the *STATEMENT view) to the current view of      */
  /* the module                                                      */
  QteMapViewPosition((char *)&Map_Return_Structure, &MapLength,
                     &pgm_dbg_dta-> Element[stmt_view].ViewNumber,
                     &Statements[0], &Column,
                     &pgm_dbg_dta->Element[current_view].ViewNumber,
                     &errorCode);

  /* Tell the user about the program that stopped.                   */
  for (i=0;i<4;i++) {               /* See why program stopped       */
    if (reason[i] == '1') {
      switch(i) {
        case 0: printf("Unmonitored exception");
          break;
        case 1: printf("Breakpoint");
          break;
        case 2: printf("Step completed");
          break;
        case 3: printf("Breakpoint condition error");
          break;
      }
```

```
    }
  }
  printf(" in module %.10s at line %d.\n",
         Module,
         Map_Return_Structure.MapElem[0].LineNumber);

  ProcessCommands();               /* put user into debugger        */
}
```

This function is called when program DEBUG is called as a Program Stop Handler. It is passed the name, library, and type of the program stopped, the line number in the statement view where it has stopped, a count of line numbers stopped in, if the system cannot determine exactly where the program has stopped (this is the case for optimized code), and an array of character flags indicating why the program was stopped.

The first thing the function does is determine if the current view is set to the module where the program stopped. If not, then it needs to be reset to the first view in the module where the program has stopped.

Next, the statement view ID for the module stopped needs to be determined. This is necessary because the stopped position is given in terms of the statement view, and this position needs to be converted to a position in the current view.

The QteMapViewPosition API maps a position in the statement view to a statement in another view in that module. This allows the debugger to determine the source line of the current view where the program has stopped, even though the program is only told the line number in the statement view.

Finally, the character flags are checked to see why the program was stopped. Note that the program can be stopped for more than one reason, so every flag is checked, and if it is on, a message for that flag is printed.

Finally, the ProcessCommands function is called, allowing the user to enter debug commands.

Other APIs

This section discusses other APIs not covered in this example debugger. Some or all of these APIs could be used in a real ILE source-level debugger. All of them are used in the debugger shipped with OS/400.

QteRetrieveDebugAttributes

This API allows a debugger to retrieve information about the debug session. This includes the value of the Update Production Files, set on the Start Debug command, as well as an indication of whether the job where the debugger is running is servicing and debugging another job.

QteSetDebugAttributes

The only attribute that can be set is the value of the Update Production Files. This can also be accomplished using the Change Debug (CHGDBG) CL command.

QteRemoveDebugView

Views that are registered can be removed from debug. This is desirable if a program is to be removed from debug so that it can be recompiled and added again. It is not necessary to remove views from debug when ending the debug session, as QteEndSourceDebug will do this automatically.

QteRetrieveStoppedPosition

This indicates if a program is currently stopped and on the stack, and whether this stopped position is anywhere in a given view. This is useful whenever a source debugger is about to put up a source screen. If the program is stopped somewhere within the source to be displayed, this can be indicated to the user.

This is necessary because a program can be stopped by other means than the debugger. For example, an ILE program could have put up a command entry screen, and the debugger could be displayed from there. In this case, it is nice to indicate to the user that the program being debugged is stopped.

QteAddBreakpoint

This and the following APIs are not really needed, as their function can be done with the QteSubmitDebugCommand. However, this API is much faster, since a debug language command does not need to be parsed and interpreted. In cases where the debugger knows the information without needing to specify a debug command to the API, these "shortcut" APIs should be used.

This API performs the same function as the break n debug language command.

QteRemoveBreakpoint

This API performs the same function as the clear n debug language command.

QteRemoveAllBreakpoints

This API performs the same function as the clear pgm debug language command.

QteStep

This API performs the same function as the step n into and step n over debug language commands.

Debugger Code--Sample

Following is the entire program listing for the ILE C program containing the example debugger discussed in the preceding sections.

```
/*****************************************************************/
/*****************************************************************/
/*                                                               */
/* FUNCTION:  The entire program listing for the program         */
/*            containing the example debugger discussed in the   */
/*            preceding sections.                                */
/*                                                               */
/* LANGUAGE:  ILE C for OS/400                                   */
/*                                                               */
/* APIs USED: QteRetrieveViewText, QteSubmitDebugCommand,        */
/*            QteEndSourceDebug, QteRetrieveModuleViews,         */
/*            QteRegisterDebugView, QteStartSourceDebug,         */
/*            QteMapViewPosition                                 */
/*                                                               */
/*****************************************************************/
/*****************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <qtedbgs.h>

/* Global variables holding information about a program in debug mode*/
static _TE_VEWL0100_T *pgm_dbg_dta = NULL;
static long current_view = 0;       /* current view - defaults to 1st*/
static _TE_OBJLIB_T program_lib;    /* name and lib of pgm debugged  */

/* ReadLine: Reads a line of input and stores it in a string.        */
void ReadLine(char *Buffer, int length) {
  int i;                            /* loop counter                */

  printf("Enter a debugger command or 'help'.\n");
  fgets(Buffer,length,stdin);       /* read line of text           */

  /* Blank out line from \n to the end of the string.              */
  for (i=0; i<length; i++) {        /* loop, searching for newline   */
    if (Buffer[i] == '\n') {        /* if newline character found
*/
      break;                        /* end loop searching for newline*/
    }
  }

  memset(Buffer+i,' ',length-i);     /* blank remainder of line      */
}

/* PrintText: This function will print the text for the current view */
void PrintText(void) {
```

```
  long LineLength = 92;             /* length of lines of text       */
  long NumberOfLines = 0;           /* lines to retrieve - 0 = all   */
  long StartLine=1;                 /* retrieve from line 1 (first)  */
  long bufferLength = 100000;       /* size of retrieved text buffer */
  long viewID;                      /* view ID of text to retrieve   */
  _TE_TEXT_BUFFER_T *buffer;        /* text retrieved by API         */
  _TE_ERROR_CODE_T errorCode = {0}; /* Exceptions will be signaled   */
  int i;                            /* points to start of each line  */
  int line_number;                  /* line number counter for loop  */

  /* Get View ID of current view                                     */
  viewID = pgm_dbg_dta->Element[current_view].ViewNumber;

  buffer = malloc(bufferLength);    /* malloc space for big text buf */

  /* Call Retrieve_View_Text for the current view.                   */
  QteRetrieveViewText((char *)buffer, &bufferLength, &viewID,
                      &StartLine, &NumberOfLines, &LineLength,
                      &errorCode);

  /* Print out the text                                              */
  for (i=0,line_number=1;
       line_number <= buffer->NumLines;
       line_number++,i+=LineLength) {
     printf("%3d) %.70s\n", line_number, buffer->Text+i);
  }

  free(buffer);                     /* free memory for buffer        */
}

/* PrintViews: Prints all the views of the program being debugged.   */
void PrintViews(void) {
  int k;

  /* loop through views printing view#, module, and view desc. text  */
  for (k=0; k< pgm_dbg_dta->NumberElements; k++) {
    printf("%d) %.10s:%.50s",
           k,
           pgm_dbg_dta->Element[k].ModuleName,
           pgm_dbg_dta->Element[k].ViewDescription);
    if (current_view == k)         /* indicate if view is current   */
      printf("<---Current\n");
    else
      printf("\n");
  }
}

/* ProcessListCommand:  Process list command to list views or text   */
void ProcessListCommand(void) {
  char *token;                      /* pointer to next token of input*/

  token = strtok(NULL," ");         /* get next token in input buffer*/

  if (token==NULL)                  /* list not followed by anything */
    printf("'list' must be followed by 'views' or 'text'.\n");
  else if (strcmp(token,"views") == 0)/* if list views               */
    PrintViews();
  else if (strcmp(token,"text") == 0) /* if list text                */
    PrintText();
  else                              /* list <something-else>         */
    printf("'list' must be followed by 'views' or 'text'.\n");
}

/* ProcessDbgCommand:  This function will process commands sent to   */
/*                     the QteSubmitDebugCommand API.                */
int ProcessDbgCommand(char InputBuffer[80]) {
```

```
  _TE_ERROR_CODE_T errorCode = {64}; /* fill in bytes provided       */
  char OutputBuffer[4096];
  struct _TE_RESULT_BUFFER_T *Results;
  long InputBufferLength = 80;
  long OutputBufferLength = sizeof(OutputBuffer);
  long view_ID;
  _TE_COMPILER_ID_T *CompilerID;
  int i;
  int return_value = 0;

  view_ID = pgm_dbg_dta->Element[current_view].ViewNumber;
  CompilerID = &pgm_dbg_dta->Element[current_view].CompilerID;

  /* Give command to QteSubmitDebugCommand                           */
  QteSubmitDebugCommand(OutputBuffer, &OutputBufferLength,
                        &view_ID, InputBuffer, &InputBufferLength,
                        *CompilerID, &errorCode);

  if (errorCode.BytesAvailable != 0) {
    printf("Error = %.7s\n",errorCode.ExceptionID);
    return return_value;
  }

  /* Process results from QteSubmitDebugCommand                      */
  Results = (_TE_RESULT_BUFFER_T *) OutputBuffer;

  /* Loop through Results array                                      */
  for (i=0; i<Results->Header.EntryCount; i++) {
    switch (Results->Data[i].ResultKind)
    {
      case _TE_kStepR           :
        printf("Step set\n");
        return_value=1;             /* indicate step is to be done   */
        break;
      case _TE_kBreakR          :
        printf("Breakpoint set");
        break;
      case _TE_kBreakPositionR  :
        printf(" at line %d\n",
               Results->Data[i].V.BreakPosition.Line);
        break;
      case _TE_kExpressionTextR  :
        printf("%s",
               ((char *)Results) + Results->Data[i].V.
               ExpressionText.oExpressionText);
        break;
      case _TE_kExpressionValueR :
        printf(" = %s\n",
               ((char *)Results) + Results->Data[i].V.
               ExpressionValue.oExpressionValue);
        break;
      case _TE_kQualifyR        :
        printf("Qual set\n");
        break;
      case _TE_kClearBreakpointR :
        printf("Breakpoint cleared\n");
        break;
      case _TE_kClearPgmR :
        printf("All breakpoints cleared\n");
        break;
      default:                      /* ignore all other record types */
        break;
    }                               /* switch                        */
  }                                 /* loop through results array    */
  return return_value;
}
```

```
/* ProcessCommands:  Read input from user and process commands.      */
void ProcessCommands(void) {
  char InputBuffer[80];
  char *token;
  int i;
  int step=0;                       /* do an exit for step when 1    */

  if (pgm_dbg_dta == NULL) {        /* if no debug data              */
    printf("Debug session has ended.\n");
    exit(0);                        /* end the debugger              */
  }

  while(!step) {                    /* read until step or quit cmd   */
    ReadLine(InputBuffer,sizeof(InputBuffer));
    token = strtok(InputBuffer," ");

    if (token==NULL) continue;      /* ignore blank lines            */
    else if (strcmp(token,"quit") == 0) /* the quit command?         */
      return;                       /* exit debugger                 */
    else if (strcmp(token,"list") == 0) /* the list command?         */
      ProcessListCommand();         /* process command               */
    else if (strcmp(token,"switch") == 0) { /* switch command?       */
      token = strtok(NULL," ");     /* get view number token         */
      if (token == NULL)
        printf("'switch' must be followed by a view number.\n");
      else
        current_view = atoi(token);   /* switch current view         */
    }
    else if (strcmp(token,"help") == 0) {
      printf("The following are the allowed debugger commands:\n");
      printf("  list views - lists all views in the program\n");
      printf("  list text - lists the text of the current view\n");
      printf("  switch n - changes current view to view n\n");
      printf("  help - displays this help text\n");
      printf("  quit - ends the debug session\n");
      printf("Other commands are interpreted by the debug support.\n");
    }
    else {                          /* pass command to API           */
      /* Undo modifications that strtok did                          */
      InputBuffer[strlen(InputBuffer)] = ' ';

      step = ProcessDbgCommand(InputBuffer);
    }
  }
}

/* TearDownDebugger: End the debugger.                               */
void TearDownDebugger(void) {
  _TE_ERROR_CODE_T errorCode = {8}; /* errors will be ignored        */

  /* Call EndSourceDebug to get out of ILE debug mode               */
  QteEndSourceDebug(&errorCode);

  exit(0);                          /* destroy activation group      */
}

/* AddProgram: Add a program to debug mode.                          */
void AddProgram(void) {

  _TE_ERROR_CODE_T errorCode = {0};    /* Signal exceptions on error */
  _TE_NAME_T       Library;            /* Lib returned               */
  _TE_TIMESTAMP_T  TimeStamp;          /* TimeStamp returned         */
  int viewIndex;
  long int         iViewID;
  long int         iViewLines;
  long rtvModViewDataLength = 8;       /* size of receiver buffer    */
  char tempBuffer[8];                  /* Temp storage               */
```

```
  int i, tempModuleCount;

  /* Call QteRetrieveModuleViews to determine the number of bytes    */
  /* the receiver variable needs to be to hold all of the views for  */
  /* the program.                                                    */
  pgm_dbg_dta = (_TE_VEWL0100_T *)tempBuffer;
  QteRetrieveModuleViews((char *)pgm_dbg_dta, &rtvModViewDataLength,
                         "VEWL0100", &program_lib,
                         "*PGM      ", "*ALL      ", Library,
                         &errorCode);

  /* Get a buffer large enough to hold all view information          */
  rtvModViewDataLength = pgm_dbg_dta->BytesAvailable;
  pgm_dbg_dta = (_TE_VEWL0100_T *)malloc(rtvModViewDataLength);

  /* Call QteRetrieveModuleViews again, passing a big enough buffer. */
  QteRetrieveModuleViews((char *)pgm_dbg_dta, &rtvModViewDataLength,
                         "VEWL0100", &program_lib,
                         "*PGM      ", "*ALL      ", Library,
                         &errorCode);

  /* If number of elements is zero, program is not debuggable.       */
  if (pgm_dbg_dta->NumberElements == 0) {
    printf("Program %.10s in Library %.10s cannot be debugged.",
           program_lib.obj, program_lib.lib);
    TearDownDebugger();
  }

  /* Put the library returned by Retrieve Module Views in PgmLib     */
  memcpy(program_lib.lib, Library, sizeof(_TE_NAME_T));

  /* Register all views in the program                               */
  for (i=0; i < pgm_dbg_dta->NumberElements; i++) {
    QteRegisterDebugView(&iViewID, &iViewLines, Library, TimeStamp,
                         &program_lib, "*PGM      ",
                         pgm_dbg_dta->Element[i].ModuleName,
                         &pgm_dbg_dta->Element[i].ViewNumber,
                         &errorCode);

    /* overwrite unneeded ViewNumber with obtained view id           */
    pgm_dbg_dta->Element[i].ViewNumber = iViewID;
  }
}

/* Typedef for program list passed to this program at STRDBG time    */
typedef struct {
    _TE_OBJLIB_T PgmLib;            /* Name and Library of program   */
    _TE_NAME_T PgmType;             /* program type, *PGM or *SRVPGM */
} PgmList_t;

/* StartUpDebugger: Initialize the debugger.                         */
void StartUpDebugger(PgmList_t ProgramList[],
                     int ProgramListCount) {

  _TE_ERROR_CODE_T errorCode = {0}; /* exceptions are generated      */
  _TE_OBJLIB_T StopHandler = {"DEBUG     ", "*LIBL     "};
  int i;

  if (ProgramListCount!=1) {        /* is only 1 pgm passed on STRDBG*/
    printf("Exactly ONE program must be specified on STRDBG.\n");
    TearDownDebugger();             /* end debugger                  */
  }

  /* Copy program name to global variables                           */
  memcpy(&program_lib, &ProgramList->PgmLib, 20);

  /* Call StartSourceDebug: giving the name and library of the       */
```

```
  /* stop handler.  This will start ILE debug mode                   */
  QteStartSourceDebug(&StopHandler, &errorCode);

  AddProgram();                     /* add program to debug          */
}

/* HandleSession:  This function is called to handle the session     */
/* events STRDBG, DSPMODSRC and ENDDBG.                              */
void HandleSession(char reason[10],
                   PgmList_t ProgramList[],
                   int ProgramListCount) {

  if (memcmp(reason,"*START    ",10) == 0) /* reason is *START       */
    StartUpDebugger(ProgramList, ProgramListCount);
  else if ( memcmp(reason,"*STOP     ",10) == 0) /* reason is *STOP  */
    TearDownDebugger();
  else if ( memcmp(reason,"*DISPLAY  ",10) == 0) /* reason *DISPLAY  */
    ProcessCommands();
}

/* HandleStop: This function is called to handle stop events like    */
/* breakpoint, step, unmonitored exception, etc.                     */
void HandleStop(_TE_OBJLIB_T *ProgramLib,
                _TE_NAME_T ProgramType,
                _TE_NAME_T Module,
                char reason[10],
                long Statements[],
                int StatementsCount,
                char *message) {
  int i;
  _TE_MAPP0100_T Map_Return_Structure;
  long Column = 1;
  long MapLength = sizeof(Map_Return_Structure);
  _TE_ERROR_CODE_T errorCode = {64};
  long stmt_view;

  /* If current view is for a different module than the one that is  */
  /* stopped, change current view to first view in the stopped module*/
  if (memcmp(Module,
             pgm_dbg_dta->Element[current_view].ModuleName,
             sizeof(_TE_NAME_T)) != 0) { /* a different module?      */
    for (i=0; i<pgm_dbg_dta->NumberElements; i++) {
      if (memcmp(Module,
                 pgm_dbg_dta->Element[i].ModuleName,
                 sizeof(_TE_NAME_T)) == 0) { /* found module         */
        current_view = i;           /* change current view to module */
        printf("Current view changed to %d.\n",current_view);
        break;                      /* exit search loop              */
      }                             /* module found                  */
    }                               /* loop through views            */
  }                                 /* current view to be changed    */

  /* Get number of statement view for module stopped                 */
  for (i=0; i<pgm_dbg_dta->NumberElements; i++) {
    if ((memcmp(Module,
                pgm_dbg_dta->Element[i].ModuleName,
                sizeof(_TE_NAME_T)) == 0) &&
        (memcmp("*STATEMENT",
                 pgm_dbg_dta->Element[i].ViewType,
                 sizeof(_TE_NAME_T)) == 0))
      stmt_view = i;
  }


  /* Call QteMapViewPosition to map the stopped location (which      */
  /* is in terms of the *STATEMENT view) to the current view of      */
  /* the module                                                      */
```

```
  QteMapViewPosition((char *)&Map_Return_Structure, &MapLength,
                     &pgm_dbg_dta-> Element[stmt_view].ViewNumber,
                     &Statements[0], &Column,
                     &pgm_dbg_dta->Element[current_view].ViewNumber,
                     &errorCode);

  /* Tell the user about the program that stopped.                   */
  for (i=0;i<4;i++) {              /* See why program stopped       */
    if (reason[i] == '1') {
      switch(i) {
        case 0: printf("Unmonitored exception");
          break;
        case 1: printf("Breakpoint");
          break;
        case 2: printf("Step completed");
          break;
        case 3: printf("Breakpoint condition error");
          break;
      }
    }
  }
  printf(" in module %.10s at line %d.\n",
         Module,
         Map_Return_Structure.MapElem[0].LineNumber);

  ProcessCommands();               /* put user into debugger        */
}

/* main: Entry point for the debugger (session or stop handler)      */
main (int argc, char *argv[]) {
  if (argc == 4)                   /* called as source debug program*/
    HandleSession(argv[1], (PgmList_t *)argv[2], *(int
*)argv[3]);
  else if (argc == 8)             /* called as program stop handler */
    HandleStop((_TE_OBJLIB_T *)argv[1], argv[2],
argv[3], argv[4],
               (long *)argv[5], *(int *)argv[6],
argv[7]);
}
```

Using the Spawn Process and Wait for Child Process APIs

The following two examples illustrate programs that use a parent/child relationship.

See the QlgSpawn--Spawn Process (using NLS-enabled path name) API for an example of supplying parameters in any CCSID.

Parent Program

This program acts as a parent to a child program (see Child Program).

This program demonstrates the use of the spawn() function and the wait() and waitpid() functions in a parent/child relationship. The use of file descriptors, the creation of a new process group, arguments passed from parent to child, and environment variables are demonstrated. The parent program uses spawn() in three different ways.

Use the Create C Module (CRTCMOD) and the Create Program (CRTPGM) commands to create this program (see Creating the Parent and Child Program).

Call this program with no parameters (see Calling the Parent Program).

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:  This program acts as a parent to a child program.      */
```

```
/*                                                                   */
/* LANGUAGE:  ILE C for OS/400                                       */
/*                                                                   */
/* APIs USED: putenv(), spawn(), wait(), waitpid()                   */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/

#include <errno.h>
#include <fcntl.h>
#include <spawn.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

#define MAP_NUM    5
#define ARGV_NUM   6
#define ENVP_NUM   2
#define CHILD_PGM  "QGPL/CHILD"

extern char **environ;

/* This is a parent program that will use spawn() in 3 different     */
/* ways for 3 different children.  A file is created that is         */
/* written to, both by the parent and the 3 children.  The end result*/
/* of the file will look something like the following:               */
/*     Parent writes    Child writes                                 */
/*     -------------    ---------------------------------------      */
/*           1          argv[0]  getppid()  getpgrp()  getpid()      */
/*           2          argv[0]  getppid()  getpgrp()  getpid()      */
/*           3          argv[0]  getppid()  getpgrp()  getpid()      */
/* The parent uses wait() or waitpid() to wait for a given child to  */
/* return and to retrieve the resulting status of the child when it  */
/* does return.                                                      */
int main(int argc, char *argv[])
{
    int    rc;                          /* API return code           */
    int    fd, fd_read;                 /* parent file descriptors   */
    char   fd_str[4];                  /* file descriptor string     */
    char   f_path_name[] = "A_File";            /* file pathname    */
    int    buf_int;                     /* write(), read() buffer    */
    char   buf_pgm_name[22];           /* read() program name buffer */
    char   spw_path[] = "/QSYS.LIB/QGPL.LIB/CHILD.PGM";
                                        /* spawn() *path             */
    int    spw_fd_count;                /* spawn() fd_count          */
    int    spw_fd_map[MAP_NUM];
                                         /* spawn() fd_map[]         */
    struct inheritance spw_inherit;     /* spawn() *inherit          */
    char  *spw_argv[ARGV_NUM];
                                       /* spawn() *argv[]            */
    char  *spw_envp[ENVP_NUM];
                                       /* spawn() *envp[]            */
    int    seq_num;                       /* sequence number        */
    char   seq_num_str[4];              /* sequence number string    */
    pid_t  pid;                           /* parent pid              */
    char   pid_str[11];                  /* parent pid string        */
    pid_t  pgrp;                          /* parent process group    */
    char   pgrp_str[11];               /* parent process group string */
    pid_t  spw_child_pid[3];             /* 3 spawn() child pid       */
    pid_t  wt_child_pid[3];            /* 3 wait()/waitpid() child pid */
    int    wt_stat_loc[3];
                                        /* 3 wait()/waitpid() *stat_loc*/
    int    wt_pid_opt = 0;               /* waitpid() option         */
```

```
char   env_return_val[16];
                                /* environ var "return_val="   */

memset(&spw_inherit,0x00,sizeof(spw_inherit));

/* Get the pid and pgrp for the parent.                         */
pid = getpid();
pgrp = getpgrp();

/* Format the pid and pgrp value into null-terminated strings.  */
sprintf(pid_str, "%d", pid);
sprintf(pgrp_str, "%d", pgrp);

/* Create a file and maintain the file descriptor.              */
fd = creat(f_path_name, S_IRWXU);
if (fd == -1)
  {
    printf("FAILURE: creat() with errno = %d\n",errno);
    return -1;
  }

/* Format the file descriptor into null-terminated string.      */
sprintf(fd_str, "%d", fd);

/* Set the spawn() child arguments that are common for each     */
/* child.                                                       */
/* NOTE: The child will always get argv[0] in the               */
/* LIBRARY/PROGRAM notation, but the                            */
/* spawn() argv[0] (spw_argv[0]                                 */
/* in this case) must be non-NULL in order to allow additional  */
/* arguments.  For this example, the character pointer spw_path */
/* was chosen.                                                  */
/* NOTE: The parent pid and the parent process group are passed */
/* to the child for demonstration purposes only.                */
spw_argv[0] = spw_path;
spw_argv[1] = pid_str;
spw_argv[2] = pgrp_str;
spw_argv[4] = fd_str;
spw_argv[5] = NULL;

/* Write a '1' out to the file.                                 */
buf_int = 1;
write(fd, &buf_int, sizeof(int));

/* The 1st spawn() will use simple inheritance for file         */
/* descriptors (fd_map[] value is NULL).                        */
spw_fd_count = 0;
spw_inherit.pgroup = 0;
seq_num = 1;
sprintf(seq_num_str, "%d", seq_num);
spw_argv[3] = seq_num_str;
spw_envp[0] = NULL;
spw_child_pid[0] = spawn(spw_path, spw_fd_count, NULL, &spw_inherit,
                         spw_argv, spw_envp);
if (spw_child_pid[0] == -1)
  {
    printf("FAILURE: spawn() #1 with errno = %d\n",errno);
    close(fd);
    unlink(f_path_name);
    return -1;
  }

/* NOTE: The parent can continue processing while the child is  */
/* also processing.  In this example, though, the parent will   */
/* simply /* wait() until the child finishes processing.        */

/* Issue wait() in order to wait for the child to return.       */
```

```
wt_child_pid[0] = wait(&wt_stat_loc[0]);
if (wt_child_pid[0] == -1)
  {
    printf("FAILURE: wait() #1 with errno = %d\n",errno);
    close(fd);
    unlink(f_path_name);
    return -1;
  }

/* Check to ensure the child's pid returned from spawn() is the  */
/* same as the child's pid returned from wait(), for which       */
/* status was returned.                                          */
if ( (spw_child_pid[0] != wt_child_pid[0]) )
    printf("FAILURE: spawn() #1 and wait() #1 pid not the same\n");

/* Check to ensure the child did not encounter an error          */
/* condition.                                                    */
if (WIFEXITED(wt_stat_loc[0]))
  {
    if (WEXITSTATUS(wt_stat_loc[0]) != 1)
        printf("FAILURE: wait() exit status = %d\n",
               WEXITSTATUS(wt_stat_loc[0]));
  }
else
    printf("FAILURE: unknown child #1 status\n");

/* Write a '2' out to the file.                                  */
buf_int = 2;
write(fd, &buf_int, sizeof(int));

/* The 2nd spawn() will use mapping for the file descriptor,     */
/* along with the inheritance option to create a new process     */
/* group for the child.                                          */
spw_fd_count = 1;
spw_fd_map[0] = fd;
spw_inherit.pgroup = SPAWN_NEWPGROUP;
seq_num = 2;
sprintf(seq_num_str, "%d", seq_num);
spw_argv[3] = seq_num_str;
spw_envp[0] = NULL;
spw_child_pid[1] = spawn(spw_path, spw_fd_count, spw_fd_map,
                         &spw_inherit, spw_argv, spw_envp);
if (spw_child_pid[1] == -1)
  {
    printf("FAILURE: spawn() #2 with errno = %d\n",errno);
    close(fd);
    unlink(f_path_name);
    return -1;
  }

/* NOTE: The parent can continue processing while the child is   */
/* also processing.  In this example, though, the parent will    */
/* simply waitpid() until the child finishes processing.         */

/* Issue waitpid() in order to wait for the child to return.     */
wt_child_pid[1] = waitpid(spw_child_pid[1], &wt_stat_loc[1],
                          wt_pid_opt);
if (wt_child_pid[1] == -1)
  {
    printf("FAILURE: waitpid() #2 with errno = %d\n",errno);
    close(fd);
    unlink(f_path_name);
    return -1;
  }

/* Check to ensure the child's pid returned from spawn() is the  */
/* same as the child's pid returned from waitpid(), for which    */
```

```
/* status was returned.                                          */
if ( (spw_child_pid[1] != wt_child_pid[1]) )
    printf("FAILURE: spawn() #2 and waitpid() #2 pid not same\n");

/* Check to ensure the child did not encounter an error          */
/* condition.                                                    */
if (WIFEXITED(wt_stat_loc[1]))
  {
    if (WEXITSTATUS(wt_stat_loc[1]) != 2)
        printf("FAILURE: waitpid() exit status = %d\n",
               WEXITSTATUS(wt_stat_loc[1]));
  }
else
    printf("FAILURE: unknown child #2 status\n");

/* Write a '3' out to the file.                                  */
buf_int = 3;
write(fd, &buf_int, sizeof(int));

/* The 3rd spawn() will use mapping for the file descriptors     */
/* with some file descriptors designated as being closed         */
/* (SPAWN_FDCLOSED) and the same parent file descriptor mapped   */
/* to more than one child file descriptor.  In addition, an      */
/* environment variable will be set and used by the child.       */
spw_fd_count = 5;
spw_fd_map[0] = SPAWN_FDCLOSED;
spw_fd_map[1] = SPAWN_FDCLOSED;
spw_fd_map[2] = fd;
spw_fd_map[3] = SPAWN_FDCLOSED;
spw_fd_map[4] = fd;
spw_inherit.pgroup = 0;
seq_num = 3;
sprintf(seq_num_str, "%d", seq_num);
spw_argv[3] = seq_num_str;
strcpy(env_return_val,"return_val=3");
rc = putenv(env_return_val);
if (rc < 0)
  {
    printf("FAILURE: putenv() with errno = %d\n",errno);
    close(fd);
    unlink(f_path_name);
    return -1;
  }
spw_child_pid[2] = spawn(spw_path, spw_fd_count, spw_fd_map,
                         &spw_inherit, spw_argv, environ);
if (spw_child_pid[2] == -1)
  {
    printf("FAILURE: spawn() #3 with errno = %d\n",errno);
    close(fd);
    unlink(f_path_name);
    return -1;
  }

/* The parent no longer needs to use the file descriptor, so it  */
/* can close it, now that it has issued spawn().                 */
rc = close(fd);
if (rc != 0)
    printf("FAILURE: close(fd) with errno = %d\n",errno);

/* NOTE: The parent can continue processing while the child is   */
/* also processing.  In this example, though, the parent will    */
/* simply wait() until the child finishes processing.            */

/* Issue wait() in order to wait for the child to return.        */
wt_child_pid[2] = wait(&wt_stat_loc[2]);
if (wt_child_pid[2] == -1)
  {
```

```
    printf("FAILURE: wait() #3 with errno = %d\n",errno);
    unlink(f_path_name);
    return -1;
  }

/* Check to ensure the child's pid returned from spawn() is the  */
/* same as the child's pid returned from wait(), for which       */
/* status was returned.                                          */
if ( (spw_child_pid[2] != wt_child_pid[2]) )
    printf("FAILURE: spawn() #3 and wait() #3 pid not the same\n");

/* Check to ensure the child did not encounter an error          */
/* condition.                                                    */
if (WIFEXITED(wt_stat_loc[2]))
  {
    if (WEXITSTATUS(wt_stat_loc[2]) != 3)
        printf("FAILURE: wait() exit status = %d\n",
               WEXITSTATUS(wt_stat_loc[2]));
  }
else
    printf("FAILURE: unknown child #3 status\n");

/* Open the file for read to verify what the child wrote.        */
fd_read = open(f_path_name, O_RDONLY);
if (fd_read == -1)
  {
    printf("FAILURE: open() for read with errno = %d\n",errno);
    unlink(f_path_name);
    return -1;
  }

/* Verify what child #1 wrote.                                   */
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != 1) )
    printf("FAILURE: read() #1\n");
memset(buf_pgm_name,0x00,sizeof(buf_pgm_name));
rc = read(fd_read, buf_pgm_name, strlen(CHILD_PGM));
if ( (rc != strlen(CHILD_PGM)) ||
     (strcmp(buf_pgm_name,CHILD_PGM) != 0) )
    printf("FAILURE: read() child #1 argv[0]\n");
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != pid) )
    printf("FAILURE: read() child #1 getppid()\n");
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != pgrp) )
    printf("FAILURE: read() child #1 getpgrp()\n");
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != spw_child_pid[0]) ||
     (buf_int != wt_child_pid[0]) )
    printf("FAILURE: read() child #1 getpid()\n");

/* Verify what child #2 wrote.                                   */
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != 2) )
    printf("FAILURE: read() #2\n");
memset(buf_pgm_name,0x00,sizeof(buf_pgm_name));
rc = read(fd_read, buf_pgm_name, strlen(CHILD_PGM));
if ( (rc != strlen(CHILD_PGM)) ||
     (strcmp(buf_pgm_name,CHILD_PGM) != 0) )
    printf("FAILURE: read() child #2 argv[0]\n");
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != pid) )
    printf("FAILURE: read() child #2 getppid()\n");
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int == pgrp) )
    printf("FAILURE: read() child #2 getpgrp()\n");
rc = read(fd_read, &buf_int, sizeof(int));
```

```
    if ( (rc != sizeof(int)) || (buf_int != spw_child_pid[1]) ||
         (buf_int != wt_child_pid[1]) )
        printf("FAILURE: read() child #2 getpid()\n");

    /* Verify what child #3 wrote.                                  */
    rc = read(fd_read, &buf_int, sizeof(int));
    if ( (rc != sizeof(int)) || (buf_int != 3) )
        printf("FAILURE: read() #3\n");
    memset(buf_pgm_name,0x00,sizeof(buf_pgm_name));
    rc = read(fd_read, buf_pgm_name, strlen(CHILD_PGM));
    if ( (rc != strlen(CHILD_PGM)) ||
         (strcmp(buf_pgm_name,CHILD_PGM) != 0) )
        printf("FAILURE: read() child #3 argv[0]\n");
    rc = read(fd_read, &buf_int, sizeof(int));
    if ( (rc != sizeof(int)) || (buf_int != pid) )
        printf("FAILURE: read() child #3 getppid()\n");
    rc = read(fd_read, &buf_int, sizeof(int));
    if ( (rc != sizeof(int)) || (buf_int != pgrp) )
        printf("FAILURE: read() child #3 getpgrp()\n");
    rc = read(fd_read, &buf_int, sizeof(int));
    if ( (rc != sizeof(int)) || (buf_int != spw_child_pid[2]) ||
         (buf_int != wt_child_pid[2]) )
        printf("FAILURE: read() child #3 getpid()\n");

    /* Attempt one more read() to ensure there is no more data.     */
    rc = read(fd_read, &buf_int, sizeof(int));
    if (rc != 0)
        printf("FAILURE: read() past end of data\n");

    /* The parent no longer needs to use the read() file descriptor, */
    /* so it can close it.                                          */
    rc = close(fd_read);
    if (rc != 0)
        printf("FAILURE: close(fd_read) with errno = %d\n",errno);

    /* Attempt one more wait() to ensure there are no more children. */
    wt_child_pid[0] = wait(&wt_stat_loc[0]);
    if ( (wt_child_pid[0] != -1) || (errno != ECHILD) )
        printf("FAILURE: ECHILD wait()\n");

    /* Clean up by performing unlink().                             */
    rc = unlink(f_path_name);
    if (rc != 0)
      {
        printf("FAILURE: unlink() with errno = %d\n",errno);
        return -1;
      }
    printf("completed successfully\n");
    return 0;
}
```

Child Program

This program acts as a child to a parent program (see Parent Program). This program demonstrates how a child program uses characteristics expressed through the use of spawn() in the parent program. The use of file descriptors, the creation of a new process group, arguments passed from the parent, and environment variables are demonstrated. The child program handles three distinct calls through the use of one of its arguments.

Use the CRTCMOD and CRTPGM commands to create this program (see Creating the Parent and Child Program).

This program is called by the spawn() function from the parent program. The program name must be CHILD and must be created into library QGPL, as indicated by the parent program. This program is not to be called directly.

```
/*********************************************************************/
/*********************************************************************/
```

```
/*                                                                  */
/* FUNCTION:  This program acts as a child to a parent program.     */
/*                                                                  */
/* LANGUAGE:  ILE C for OS/400                                      */
/*                                                                  */
/* APIs USED: getenv(), getpid(), getppid(), getpgrp()              */
/*                                                                  */
/********************************************************************/
/********************************************************************/

#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>

/* This is a child program that gets control from a parent program   */
/* that issues spawn().  This particular child program expects the   */
/* following 5 arguments (all are null-terminated strings):          */
/*     argv[0] - child program name                                  */
/*     argv[1] - parent pid (for demonstration only)                 */
/*     argv[2] - parent process group (for demonstration only)       */
/*     argv[3] - sequence number                                     */
/*     argv[4] - parent file descriptor                              */
/* If the child program encounters an error, it returns with a value */
/* greater than 50.  If the parent uses wait() or waitpid(), this    */
/* return value can be interrogated using the WIFEXITED and          */
/* WEXITSTATUS macros on the resulting wait() or waitpid()           */
/* *stat_loc field.                                                  */
int main(int argc, char *argv[])
{
    pid_t  p_pid;                   /* parent pid argv[1]           */
    pid_t  p_pgrp;                  /* parent process group argv[2] */
    int    seq_num;                 /* parent sequence num argv[3]  */
    int    fd;                      /* parent file desc argv[4]     */
    int    rc;                         /* API return code           */
    pid_t  pid;                        /* getpid() - child pid      */
    pid_t  ppid;                       /* getppid() - parent pid    */
    pid_t  pgrp;                       /* getpgrp() - process group */
    char  *env_return_val;          /* environ var for "return_val" */

    /* Get the pid, ppid, and pgrp for the child.                    */
    pid = getpid();
    ppid = getppid();
    pgrp = getpgrp();

    /* Verify 5 parameters were passed to the child.                 */
    if (argc != 5)
        return 60;

    /* Since the parameters passed to the child using spawn() are    */
    /* pointers to strings, convert the parent pid, parent process   */
    /* group, sequence number, and the file descriptor from strings  */
    /* to integers.                                                  */
    p_pid = atoi(argv[1]);
    p_pgrp = atoi(argv[2]);
    seq_num = atoi(argv[3]);
    fd = atoi(argv[4]);

    /* Verify the getpid() value of the parent is the same as the    */
    /* getppid() value of the child.                                 */
    if (p_pid != ppid)
        return 61;

    /* If the sequence number is 1, simple inheritance was used in   */
    /* this case.  First, verify the getpgrp() value of the parent   */
    /* is the same as the getpgrp() value of the child.  Next, the   */
    /* child will use the file descriptor passed in to write the     */
```

```
/* child's values for argv[0], getppid(), getpgrp(),            */
/* and getpid().  Finally, the child returns, which will satisfy */
/* the parent's wait() or waitpid().                              */
if (seq_num == 1)
  {
    if (p_pgrp != pgrp)
        return 70;
    rc = write(fd, argv[0], strlen(argv[0]));
    if (rc != strlen(argv[0]))
        return 71;
    rc = write(fd, &ppid, sizeof(pid_t));
    if (rc != sizeof(pid_t))
        return 72;
    rc = write(fd, &pgrp, sizeof(pid_t));
    if (rc != sizeof(pid_t))
        return 73;
    rc = write(fd, &pid, sizeof(pid_t));
    if (rc != sizeof(pid_t))
        return 74;
    return seq_num;
  }

/* If the sequence number is 2, file descriptor mapping was used */
/* in this case.  In addition, an inheritance option was used to */
/* indicate this child will create a new process group.  First,  */
/* verify the getpgrp() value of the parent is different than    */
/* the getpgrp() value of the child.  Next, the child will use   */
/* a literal value of '0' as the file descriptor (instead of the */
/* parent's file descriptor passed in) since a known mapping was */
/* performed by the parent.  This literal is used to write the   */
/* child's values for argv[0], getppid(), getpgrp(),             */
/* and getpid().  Finally, the child returns, which will satisfy */
/* the parent's wait() or waitpid().                             */
else if (seq_num == 2)
  {
    if (p_pgrp == pgrp)
        return 80;
    rc = write(0, argv[0], strlen(argv[0]));
    if (rc != strlen(argv[0]))
        return 81;
    rc = write(0, &ppid, sizeof(pid_t));
    if (rc != sizeof(pid_t))
        return 82;
    rc = write(0, &pgrp, sizeof(pid_t));
    if (rc != sizeof(pid_t))
        return 83;
    rc = write(0, &pid, sizeof(pid_t));
    if (rc != sizeof(pid_t))
        return 84;
    return seq_num;
  }

/* If the sequence number is 3, file descriptor mapping was used */
/* in this case.  In addition, an environment variable by the    */
/* name of "return_val" was set with the desired return value.   */
/* First, verify the getpgrp() value of the parent is the same   */
/* as the getpgrp() value of the child.  Next, the child will    */
/* use literal values of '2' and '4' as the file descriptor      */
/* (instead of the parent's file descriptor passed in) since a   */
/* known mapping was performed by the parent.  These literals    */
/* are used to write the child's values for argv[0], getppid(),  */
/* getpgrp(), and getpid().  Finally, getenv() is performed to   */
/* retrieve the desired value to use on return, which will       */
/* satisfy the parent's wait() or waitpid().                     */
else if (seq_num == 3)
  {
    if (p_pgrp != pgrp)
```

```
            return 90;
        rc = write(4, argv[0], strlen(argv[0]));
        if (rc != strlen(argv[0]))
            return 91;
        rc = write(2, &ppid, sizeof(pid_t));
        if (rc != sizeof(pid_t))
            return 92;
        rc = write(4, &pgrp, sizeof(pid_t));
        if (rc != sizeof(pid_t))
            return 93;
        rc = write(2, &pid, sizeof(pid_t));
        if (rc != sizeof(pid_t))
            return 94;
        env_return_val = getenv("return_val");
        return (atoi(env_return_val));
      }

    /* If the sequence number is an unexpected value, return         */
    /* indicating an error.                                          */
    else
        return 99;
}
```

Creating the Parent and Child Program

The following examples show how to create the example programs (Parent Program and Child Program). These examples assume that the source for the parent program is member PARENT in the file QGPL/QCSRC and the source for the child program is member CHILD in the file QGPL/QCSRC.

Create the parent module:

```
CRTCMOD MODULE(QGPL/PARENT)
        SRCFILE(QGPL/QCSRC)
        SRCMBR(PARENT)
        TEXT('Example Parent')
```

Create the child module:

```
CRTCMOD MODULE(QGPL/CHILD)
        SRCFILE(QGPL/QCSRC)
        SRCMBR(CHILD)
        TEXT('Example Child')
```

Create the parent program:

```
CRTPGM PGM(QGPL/PARENT)
```

Create the child program:

```
CRTPGM PGM(QGPL/CHILD)
```

Calling the Parent Program

The following example shows how to start the example programs:

```
CALL PGM(QGPL/PARENT)
```

Working with Stream Files

The following ILE C program performs the following functions:

- Opens an existing stream file
- Creates or replaces a database file
- Reads from the stream file and writes to the database file until end-of-file
- Closes both files

The program uses the following hierarchical file system (HFS) APIs:

- Create Directory (QHFCRTDR)
- Open Stream File (QHFOPNSF)
- Read from Stream File (QHFRDSF)
- Close Stream File (QHFCLOSF)

```
/*********************************************************************/
/* Program Name:  HFSCOPY                                            */
/* Language    :  ILE C for OS/400                                   */
/* Description :  This program will do the following:                */
/*                -- Create or replace a stream file                 */
/*                -- Create or replace a database file               */
/*                -- Read from the stream file and write to the      */
/*                   database file until EOF                         */
/*                -- Close both files when done                      */
/* APIs Used   :  QHFCRTDR, QHFOPNSF, QHFRDSF, QHFCLOSF              */
/*********************************************************************/

/*********************************************************************/
/* Include files                                                     */
/*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <qhfopnsf.h>
#include <qhfrdsf.h>
#include <qhfclosf.h>
#include <qhfcrtdr.h>
#include <qusec.h>

/*********************************************************************/
/* Structure and variable definitions                                */
/*********************************************************************/

#define ON   1
#define OFF  0
typedef struct error_code_struct {
               Qus_EC_t EC;
               char exception_data[256];
              }error_code_struct;
error_code_struct error_code;
char file_handle[16];
char path_name[30];
char open_info[10];
char attrib_info;
char action;
char read_buffer[80];
int  path_length;
```

```
int  attrib_length = 0;
int  bytes_to_read;
int  bytes_read = 0;
int  end_file;
int  cmpgood;
FILE *FP;

/*********************************************************************/
/*printErrCode: Routine to print the error code structure            */
/*********************************************************************/
void printErrCode(error_code_struct *theErrCode)
{
   int i;
   char *tempptr = theErrCode->EC.Exception_Id;
   printf("Bytes Provided -> %d\n",theErrCode->EC.Bytes_Provided);
   printf("Bytes Available -> %d\n",theErrCode->EC.Bytes_Available);
   printf("Exception ID -> ");
   for (i=0;i<7 ;i++,tempptr++ )
     {
       putchar(*tempptr);
     }
   putchar('\n');
}


/*********************************************************************/
/* Start of code                                                     */
/*********************************************************************/
main()
{

 error_code.EC.Bytes_Provided = 116;
/*********************************************************************/
/* Create the directory                                              */
/*********************************************************************/
 strcpy(path_name,"/QDLS/HFSFLR");
 path_length = strlen(path_name);

 QHFCRTDR(path_name,path_length,&attrib_info,attrib_length,&error_code);
 if ( error_code.EC.Bytes_Available != 0 )
   {
    if (!memcmp(error_code.EC.Exception_Id,"CPF1F04",7))
       printf("Directory HFSFLR already created.\n");
    else
       {
        printErrCode(&error_code);
        exit(1);
       }
   }

/*********************************************************************/
/* Open the stream file                                              */
/*********************************************************************/
 strcpy(open_info,"210 120   ");    /* Create or replace the file   */
 strcpy(path_name,"/QDLS/HFSFLR/SAMPLE.HFS");
 path_length = strlen(path_name);
 printf("OPEN STREAM FILE:\n  ");
 QHFOPNSF(&file_handle,
          path_name,
          path_length,
          open_info,
          &attrib_info,
          attrib_length,
          &action,
          &error_code);
 if (error_code.EC.Bytes_Available != 0)
   {
```

```
    printErrCode(&error_code);
    exit(1);
   }

/*******************************************************************/
/* Open a database file                                            */
/*******************************************************************/
 system("CRTLIB LIB(HFSLIB)");
 if (( FP = fopen("HFSLIB/HFSFILE(SAMPLE)","wb")) == NULL)
   {
    printf("Cannot open HFSLIB/HFSFILE(SAMPLE)\n");
    exit(1);
   }

/*******************************************************************/
/* Loop through reading from the stream file and writing to the    */
/* database file.                                                  */
/*******************************************************************/

 end_file = OFF;
 while (end_file == OFF)
   {
     /*************************************************************/
     /* Read 80 bytes from the stream file                        */
     /*************************************************************/
     bytes_to_read = 80;
     printf("READ STREAM FILE:\n  ");
     QHFRDSF(&file_handle,
             read_buffer,
             bytes_to_read,
             &bytes_read,
             &error_code);
     if (error_code.EC.Bytes_Available != 0)
       {
         cmpgood = strncmp("CPF1F33",error_code.EC.Exception_Id,7);
         if (cmpgood != 0)
           printErrCode(&error_code);
         end_file = ON;
       }
     else
       {
         printf("BYTES READ: %d\n  ",bytes_read);
         printf("READ BUFFER: %s\n",read_buffer);
         if (bytes_read < bytes_to_read)
           {
             end_file = ON;
             /*****************************************************/
             /* Write remaining bytes to the database file        */
             /*****************************************************/
             if (bytes_read > 0)
               fwrite(read_buffer,1,bytes_read,FP);
           }
       }
   }

/*******************************************************************/
/* Close the stream file                                           */
/*******************************************************************/
 printf("CLOSE STREAM FILE:\n  ");
 QHFCLOSF(&file_handle,
          &error_code);
 if (error_code.EC.Bytes_Available != 0)
   printErrCode(&error_code);

/*******************************************************************/
/* Close the database file                                         */
/*******************************************************************/
```

```
 fclose(FP);
}
```

To create the program using ILE C, specify the following:

```
CRTBNDC PGM(QGPL/HFSCOPY) SRCFILE(QGPL/QCSRC)
```

Using the Operational Assistant Exit Program for Operational Assistant Backup

The following contains a CL example of a user-written exit program for doing Operational Assistant backup.

```
/*************************************************************/
/*************************************************************/
/*                                                           */
/* FUNCTION:  User-written exit program for doing Operational  */
/*            Assistant backup.                              */
/*                                                           */
/* LANGUAGE:  CL                                             */
/*                                                           */
/* APIs USED: None                                           */
/*                                                           */
/*************************************************************/
/*************************************************************/


PGM   PARM(&PRODID &FLAG &OPTIONS &DEVS &TAPSET &RETCODE)
DCL   VAR(&PRODID) TYPE(*CHAR) LEN(10)     /* Calling product. +
        Will be 'QEZBACKUP' when called from +
        Operational Assistant.                          */
DCL   VAR(&FLAG) TYPE(*CHAR) LEN(10)       /* Indicates whether +
                               before or after backup.      */
DCL   VAR(&DEVS) TYPE(*CHAR) LEN(40)       /* Devices used.  */
DCL   VAR(&TAPSET) TYPE(*CHAR) LEN(4)      /* Tape set name  */
DCL   VAR(&RETCODE) TYPE(*CHAR) LEN(7)     /* Return code    */
DCL   VAR(&OPTIONS) TYPE(*CHAR) LEN(10)    /* Options used   */
DCL   VAR(&MSG) TYPE(*CHAR) LEN(512)       /* Message text   */

IF    COND(&FLAG *EQ '*BEFORE    ') THEN(DO)
/*-----------------------------------------------------------*/
/*  Insert commands to be run before the backup here.     */
/*-----------------------------------------------------------*/
ENDDO

IF    COND(&FLAG *EQ '*AFTER     ') THEN(DO)
/*-----------------------------------------------------------*/
/*  Insert commands to be run after the backup here.      */
/*-----------------------------------------------------------*/
ENDDO
ENDPGM
```

Creating a Program Temporary Fix Exit Program

This example exit program written in CL, covers the following possible changes in the logical state of the PTF:

- Loaded to temporarily applied
- Temporarily applied to temporarily removed

The example program shows where you can add your code. You can write a PTF exit program in any programming language.

Note: This example does not show the apply-temporary to apply-permanent or the not-applied to remove-permanent cases. It is assumed that all action was taken on the moves from loaded to apply-temporary and from apply-temporary to not-applied. If additional actions are necessary, code could be added to handle those transitions as well.

Do not assume the default values for parameters on CL commands or for library lists. Users can change these values. Library lists can vary from one system to another.

```
/****  START OF SPECIFICATIONS  *************************************/
/*                                                                   */
/* LANGUAGE:  CL                                                     */
/*                                                                   */
/* APIs USED: None                                                   */
/*                                                                   */
/* FUNCTION:                                                         */
/*            THIS EXIT PROGRAM IS CALLED DURING ANY                 */
/*            OF THE FOLLOWING CASES.                                */
/*                                                                   */
/*            APPLY TEMPORARILY -  (user defined)                    */
/*                                                                   */
/*            APPLY PERMANENTLY -  (user defined)                    */
/*                                                                   */
/*            REMOVE TEMPORARILY - (user defined)                    */
/*                                                                   */
/*            REMOVE PERMANENTLY - (user defined)                    */
/*                                                                   */
/* Input:     PARM1 - CHAR(7)  - Product ID                          */
/*            PARM2   CHAR(7)  - PTF ID                              */
/*            PARM3 - CHAR(6)  - Product release                     */
/*            PARM4   CHAR(4)  - Product option ID                   */
/*            PARM5   CHAR(4)  - Product load ID                     */
/*            PARM6   CHAR(10) - PTF library                         */
/*            PARM7   CHAR(50) - User data                           */
/*            PARM8 - CHAR(1)  - Current PTF Status                  */
/*                               0 - LOADED BUT NOT APPLIED          */
/*                               1 - APPLIED TEMPORARILY             */
/*            PARM9   CHAR(1)  - PTF Operation                       */
/*                               0 - REMOVE TEMPORARILY              */
/*                               1 - APPLY TEMPORARILY               */
/*                               2 - APPLY PERMANENTLY               */
/*                               3 - REMOVE PERMANENTLY              */
/*                               4 - PRE-REMOVE TEMPORARILY          */
/*                               5 - PRE-APPLY TEMPORARILY           */
/*                               6 - PRE-APPLY PERMANENTLY           */
/*                               7 - PRE-REMOVE PERMANENTLY          */
/*                                                                   */
/*                                                                   */
/*******  END OF SPECIFICATIONS  ***********************************/
PGM PARM(&PARM1 &PARM2 &PARM3 &PARM4 &PARM5 &PARM6 &PARM7 &PARM8 &PARM9)
/*-------------------------------------------------------------------*/
/*                                                                   */
/* DECLARE INPUT PARAMETERS                                          */
/*                                                                   */
/*-------------------------------------------------------------------*/

DCL  &PARM1   TYPE(*CHAR) LEN(7)   /* Product ID                     */
DCL  &PARM2   TYPE(*CHAR) LEN(7)   /* PTF ID                         */
DCL  &PARM3   TYPE(*CHAR) LEN(6)   /* Product release                */
DCL  &PARM4   TYPE(*CHAR) LEN(4)   /* Product option ID              */
DCL  &PARM5   TYPE(*CHAR) LEN(4)   /* Product load ID                */
DCL  &PARM6   TYPE(*CHAR) LEN(10)  /* PTF library                    */
DCL  &PARM7   TYPE(*CHAR) LEN(50)  /* User data                      */
DCL  &PARM8   TYPE(*CHAR) LEN(1)   /* Current PTF status             */
DCL  &PARM9   TYPE(*CHAR) LEN(1)   /* PTF operation                  */
/*-------------------------------------------------------------------*/
```

```
 /*-------------------------------------------------------------------*/
 /*                                                                   */
 /* DECLARE VARIABLES                                                 */
 /*                                                                   */
 /*-------------------------------------------------------------------*/
 DCL &ACTION   TYPE(*CHAR) LEN(1)   /* PTF action to occur            */
 DCL &STATUS   TYPE(*CHAR) LEN(1)   /* PTF current status             */
                                    /* Handle exceptions              */
 MONMSG     MSGID(CPF0000) EXEC(GOTO CMDLBL(HDLERR))


   CHGVAR VAR(&ACTION) VALUE(&PARM9) /* Gets action being performed  */
   CHGVAR VAR(&STATUS) VALUE(&PARM8) /* Gets current PTF status      */

   /*-----------------------------------------------------------------*/
   /*     THE CURRENT STATUS OF THE PTF IS "LOADED (NOT APPLIED)"     */
   /*-----------------------------------------------------------------*/
   IF (&STATUS = '0') THEN(DO)   /* If PTF is loaded but not applied */
     IF (&ACTION = '1') THEN(DO) /* If action is temporarily         */
                                 /* applied then                     */
         /*?---- TEMP APPLY -  ADD YOUR STATEMENTS HERE -----        */
     ENDDO
     IF (&ACTION = '5') THEN(DO)    /* If action will be temporarily */
                                    /* apply then                    */
         /*?---- PRE-TEMP APPLY -  ADD YOUR STATEMENTS HERE -----    */
     ENDDO
   ENDDO                            /* End of loading the PTF        */

   /*-----------------------------------------------------------------*/
   /*     THE CURRENT STATUS OF THE PTF IS "APPLIED TEMPORARILY"      */
   /*-----------------------------------------------------------------*/
   IF (&STATUS = '1') THEN(DO)      /* If PTF is temporarily         */
                                    /* applied then                  */
     IF (&ACTION = '0') THEN(DO)    /* If action is temporarily      */
                                    /* removed then                  */
         /*?---- TEMPORARILY REMOVE - ADD YOUR STATEMENTS HERE ---   */
     ENDDO
     IF (&ACTION = '4') THEN(DO)    /* If action will be temporarily */
                                    /*  remove then                  */
         /*?---- PRE-TEMP REMOVE - ADD YOUR STATEMENTS HERE -----    */
     ENDDO
   ENDDO                            /* End of remove the PTF         */
    /*-----------------------------------------------------------------*/
    /*     PTF HAS BEEN SUCCESSFULLY PROCESSED                         */
    /*-----------------------------------------------------------------*/
       QSYS/SNDPGMMSG  MSGID(CPC1214) MSGF(*LIBL/QCPFMSG) +
                       MSGDTA(*NONE) TOPGMQ(*PRV (* *NONE +
                       *NONE)) TOMSGQ(*TOPGMQ) MSGTYPE(*COMP)
   RETURN
   /*-----------------------------------------------------------------*/
   /*     HANDLE ALL ERROR CONDITIONS                                 */
   /*-----------------------------------------------------------------*/
  HDLERR:
      /* Try to back out any changes already made */
      /* If nothing to back out or back-out operation was successful */
      QSYS/SNDPGMMSG  MSGID(CPF3638) MSGF(*LIBL/QCPFMSG) +
                      MSGDTA(*NONE) TOPGMQ(*PRV (* *NONE +
                      *NONE)) TOMSGQ(*TOPGMQ) MSGTYPE(*ESCAPE)
      /* Else the permanent changes not backed out    */
      QSYS/SNDPGMMSG  MSGID(CPF3639) MSGF(*LIBL/QCPFMSG) +
                      MSGDTA(*NONE) TOPGMQ(*PRV (* *NONE +
                      *NONE)) TOMSGQ(*TOPGMQ) MSGTYPE(*ESCAPE)

ENDPGM                              /* Return to external caller     */
```
