Table of Contents

An **indexed file** is a file where the sequence of records is determined by a data element known as a *key,* recorded in a separate file. Iron Spring PL/I support for indexed files is similar to IBM mainframe ISAM files or VSAM Key Sequenced Data Sets (KSDS). Indexed file support is currently available only for Linux.

Indexed file support is a recent enhancement, and should be considered beta in the current release

Declaring INDEXED Files

You can declare an indexed file by coding the attribute
`ENVIRONMENT(INDEXED).`
Unless otherwise specified the file defaults to:

```
    KEYED ENVIRONMENT(V RECSIZE(1024))
```

You could also declare the file with record format F or U instead of V, and any RECSIZE up to 1024 bytes. Additive attributes you can specify are DIRECT or SEQUENTIAL, and INPUT, OUTPUT, or UPDATE. The BACKWARDS and EXCLUSIVE attributes are currently not supported, and other environment options are not applicable to indexed files. You can code the KEYED attribute,but it is implied. `ENVIRONMENT(VSAM)` is a synonym for `ENVIRONMENT(INDEXED)`.

An example of an indexed file declaration is:

```
    DECLARE DIREC FILE RECORD KEYED ENVIRONMENT(INDEXED);
```

Opening INDEXED Files

When you open an indexed file you can specify the attributes INPUT, OUTPUT, or UPDATE, and DIRECT or SEQUENTIAL, only if you didn’t code them in the file declaration – conflicting attributes will raise the UNDEFINEDFILE condition at open time. You can code the TITLE option on the open statement to supply a name for the files. If you don’t provide a title the declared file name, translated to upper-case, is used. Each indexed “data set” consists of two separate files, xxx.**`I`** and xxx.D for the index and data components respectively, where *xxx* is the value of the title or filename.

If you open an existing file as OUTPUT, records you write will be appended to the existing data. You have to delete the existing data and index files in order to start with a new, empty file.

If you open a file as INPUT or UPDATE the file must exist, or UNDEFINEDFILE is raised.

An example of an open statement for an indexed file is:

```
    OPEN FILE(DIREC) DIRECT UPDATE;
```

Error conditions:

- UNDEFINEDFILE - conflicting attributes between declaration and open
  INPUT or UPDATE file does not exist.
- ERROR - pblIsam signaled error

Closing INDEXED Files

To close an indexed file code a standard close statement. This closes both the data and index components.

```
    CLOSE FILE(DIREC);
```

If you don't close a file it is normally closed when the program finishes execution. Some execution errors may prevent files from being properly closed. It's good practice to close all files when you're finished with them.

Error conditions:

- none

Reading INDEXED Files

You can read indexed files sequentially, in key order, or randomly by supplying the key of the record you want. You can read input or update files, but not output.

To read a **sequential** file you must either indicate where the record is to be placed, using the INTO or SET options, or code the IGNORE option to skip a number of records. The optional KEYTO option will cause the key of the record read to be stored. The EVENT option is ignored.

Here are examples of read statements for sequential indexed files:

```
    1.READ FILE(DIREC) INTO(OUTREC) KEYTO(NAME);
    2.READ FILE(DIREC) SET(REC_PTR) KEYTO(NAME);
    3.READ FILE(DIREC) INTO(OUTREC);
    4.READ FILE(DIREC) IGNORE(2);
```

Statement (1) reads the next record from the file (in key order), stores the record into OUTREC, and stores the record key in NAME. When all the records have been read, the next read raises the ENDFILE condition. Statement (2) is similar except that the read returns the address, in an internal buffer, of the record read in the pointer REC_PTR.

Statement (3) is identical to statement (1), except that the key is not returned to the caller.

Statement (4) will skip the next two records in the file and return no data. A subsequent read will retrieve the next record after those two. IGNORE, without the number in parentheses, will skip one record.

To read a **direct** file you need to specify where the record is to be placed, using the INTO option, and the key of the record you want to read, using the KEY option. The EVENT option is ignored. Here is an example of a read for a direct file:

```
    READ FILE(DIREC) INTO(OUTREC) KEY(NAME);
```

Error conditions:

- `UNDEFINEDFILE` – file not open INPUT or UPDATE
  READ IGNORE specified for a DIRECT file
- `ENDFILE` - Last record read for a SEQUENTIAL file
- `KEY` - No record with matching key, or key length is zero or greater than 255
- `RECORD` - Wrong length record for ENV(F) file
  Record length exceeds declared RECL for ENV(V)
- `ERROR` - pblIsam signaled error

Writing INDEXED FILES

You can write records to an indexed file in any order. You can write to output or update files. If you open an existing file any records you write will be added to the file. A write statement has the following format

```
    WRITE FILE(DIREC) FROM(OUTREC) KEYFROM(NAME);
```

The record you want to write, OUTREC, should be the declared length (RECL) for files declared as ENVIRONMENT(F), and between 1 byte and RECL for ENVIRONMENT(V). If you don't specify this information RECL(1024) is the default.

The key, NAME, can be between 1 and 255 bytes. Keys don't all have to be the same length, although this is recommended.

The LOCATE statement, an alternate form for sequential buffered output, is not implemented in this release.

Error conditions:

- `UNDEFINEDFILE` – file not open OUTPUT or UPDATE
- KEY - record with matching key already exists
- RECORD - Wrong length record for ENV(F) file
  Record length exceeds declared RECL for ENV(V)
- `ERROR` - pblIsam signaled error

Updating INDEXED Files

You use the REWRITE statement to update existing records in an indexed file. The file must be opened for UPDATE; the KEY condition is raised if the record does not exist. Here are examples of rewrite statements for sequential indexed files:

```
1. REWRITE FILE(DIREC);
2. REWRITE FILE(DIREC) FROM(OUTREC);
3. REWRITE FILE(DIREC) KEY(NAME);
4. REWRITE FILE(DIREC) FROM(OUTREC) KEY(NAME);
```

Statement (1) rewrites the last record read if it was read by READ SET, otherwise it is a no-op,

Statement (2) rewrites the last record read using the contents of OUTREC.

Statements (3) and (4) rewrite the specific record identified by the key NAME. (3) rewrites the last record read if it was read by READ SET, and (4) rewrites the record using the contents of OUTREC.

Error conditions:

- UNDEFINEDFILE - file not open UPDATEKEY - No record with matching key, or key length is zero or greater than 255
- RECORD - Wrong length record for ENV(F) file
  Record length exceeds declared RECL for ENV(V)
  Previous read not READ SET
- ERROR - pblIsam signaled error

Linking Programs that use INDEXED Files

The interface module is included in the Iron Spring PL/I library, and will automatically be linked if needed. The required library “libpbl.a” (see Technical Details below) must be specifically included when you link a program that uses indexed files.

The easiest way to link a PL/I program with C functions is to compile the PL/I source to generate an object module, and then link with gcc.

The sample makefile “makefile.isam” is an example of compiling and linking the two sample programs.

First compile the PL/I program and generate an object:

```
%.o: %.pli
     ${PLI} -C ${PLIFLGS}  $^ -o $*.o
```

Next link the PL/I object with the required libraries:

```
%:   %.o
     gcc -o $@ $^ ${ALTDIR}/fhs.o ${ALTDIR}/ghs.o           \
     -lprf -lpbl -static -zmuldefs -m32 -Wl,-M  >$@.map
```

The readme file for Linux section “Linking PL/I programs” includes additional detail on linking.

Limitations

INDEXED files can have a record length from 1 to 1024 bytes. A single key is allowed, with a length from 1 to 255 bytes. The key does not have to be contained within the record. Records and keys are inherently variable length, although a uniform record size can be enforced by declaring the file as `ENVIRONMENT(F)`. All characters are valid in both records and keys.

An INDEXED dataset consists of two separate components (files). These are named xxx.I and xxx.D, where *xxx* is the value of the title or filename.

Sample Code

The following sample programs are taken from the IBM PL/I for MVS & VM Programming Guide Release 1.1 (SC26-3113-01) pp.235-238.

The programs “loadsamp” and “updtsamp”, together with the makefile and data to compile and run them are in the “samples” directory.

The following program loads a sample indexed dataset.

```
      TELNOS: PROC OPTIONS(MAIN);

               DCL DIREC FILE RECORD SEQUENTIAL KEYED
               ENV(INDEXED),
                   CARD CHAR(80),
                   NAME CHAR(20) DEF CARD POS(1),
                   NUMBER CHAR(3) DEF CARD POS(21),
                   OUTREC CHAR(23) DEF CARD POS(1),
                   EOF BIT(1) INIT('0'B);
               ON ENDFILE(SYSIN) EOF='1'B;
               OPEN FILE(DIREC) OUTPUT;
               GET FILE(SYSIN) EDIT(CARD)(A(80));
               DO WHILE (^EOF);
               PUT FILE(SYSPRINT) SKIP EDIT (CARD) (A);
               WRITE FILE(DIREC) FROM(OUTREC) KEYFROM(NAME);
               GET FILE(SYSIN) EDIT(CARD)(A(80));
               END;
               CLOSE FILE(DIREC);
               END TELNOS;
```

Sample program “loadsamp”

```
ACTON,G.            162
BAKER,R.            152
BRAMLEY,O.H.        248
CHEESEMAN,O.        141
CORY,G.             336
ELLIOTT,D.          875
FIGGINS,S.          413
HARVEY,C.D.W.       205
HASTINGS,G,M.       391
KENDALL,J.G.        294
LANCASTER,W.R.      624
MILES,R.            233
NEWMAN,M.W.         450
PITT,W.H.           515
ROLF,D.E.           114
SHEERS,C.D.         241
SUTCLIFFE,M.        472
TAYLOR,G.C.         407
WILTON,L.W.         404
WINSTONE.E.M.       307
```

Sample input data, sample output for “loadsamp”

The following program updates the sample dataset. The highlighted statement had to be added due to different handling of ONCODE values.

```
DIRUPDT: PROC OPTIONS(MAIN);
        DCL DIREC FILE RECORD KEYED ENV(INDEXED),
            ONCODE BUILTIN,
            OUTREC CHAR(23),
            NUMBER CHAR(3) DEF OUTREC POS(21),
            NAME CHAR(20) DEF OUTREC,
            CODE CHAR(1),
            EOF BIT(1) INIT('0'B);
        ON ENDFILE(SYSIN) EOF='1'B;
        ON KEY(DIREC) BEGIN;
         IF ONCODE=51 THEN PUT FILE(SYSPRINT) SKIP EDIT
                           ('NOT FOUND:',NAME)(A(15),A);
         IF ONCODE=52 THEN PUT FILE(SYSPRINT) SKIP EDIT
                           ('DUPLICATE:',NAME)(A(15),A);
         /* Following stmt added to sample program */
         IF ONCODE=50 THEN PUT FILE(SYSPRINT) SKIP EDIT
                           ('KEY ERROR:',NAME)(A(15),A);
         END;
        OPEN FILE(DIREC) DIRECT UPDATE;

        GET FILE(SYSIN) EDIT(NAME,NUMBER,CODE)
          (COLUMN(1),A(20),A(3),A(1));
        DO WHILE (^EOF);
        PUT FILE(SYSPRINT) SKIP EDIT ('
         ',NAME,'#',NUMBER,' ',CODE)
           (A(1),A(20),A(1),A(3),A(1),A(1));
```

Sample program “updtsamp”
(1 of 2)

```
           SELECT (CODE);
           WHEN('A') WRITE FILE(DIREC) FROM(OUTREC)
              KEYFROM(NAME);
           WHEN('C') REWRITE FILE(DIREC) FROM(OUTREC)
              KEY(NAME);
           WHEN('D') DELETE FILE(DIREC) KEY(NAME);
           OTHERWISE PUT FILE(SYSPRINT) SKIP
             EDIT('INVALID CODE:',NAME)(A(15),A);
        END;
        GET FILE(SYSIN) EDIT(NAME,NUMBER,CODE)
        (COLUMN(1),A(20),A(3),A(1));
        END;
        CLOSE FILE(DIREC);
        PUT FILE(SYSPRINT) PAGE;
        OPEN FILE(DIREC) SEQUENTIAL INPUT;
        ON ENDFILE(DIREC) EOF='1'B;
        READ FILE(DIREC) INTO(OUTREC) KEYTO(NAME);
        DO WHILE (^EOF);
        PUT FILE(SYSPRINT) SKIP EDIT(NAME,NUMBER)(A);
        READ FILE(DIREC) INTO(OUTREC) KEYTO(NAME);
        END;
        CLOSE FILE(DIREC);

END DIRUPDT;
```

Sample program “updtsamp” (2 of 2)

```
NEWMAN,M.W.         516C
GOODFELLOW,D.T.     889A
MILES,R.               D
HARVEY,C.D.W.       289A
BARTLETT,S.G.       183A
CORY,G.                D
READ,K.M.           001A
PITT,W.H.
ROLF,D.E.              D
ELLIOTT,D.          291C
HASTINGS,G,M.          D
BRAMLEY,O.H.        439C
```

Sample input for “updtsamp”

```
 NEWMAN,M.W.          #516 C
 GOODFELLOW,D.T.      #889 A
 MILES,R.             #    D
 HARVEY,C.D.W.        #289 A
KEY ERROR:     HARVEY,C.D.W.
 BARTLETT,S.G.        #183 A
 CORY,G.              #    D
 READ,K.M.            #001 A
 PITT,W.H.            #
INVALID CODE:  PITT,W.H.
 ROLF,D.E.            #    D
 ELLIOTT,D.           #291 C
 HASTINGS,G,M.        #    D
 BRAMLEY,O.H.         #439 C
```

Sample output of “updtsamp”

The following expected errors occurred:

1. HARVEY,C.D.W. could not be added as there was already a record on file.
2. PITT,W.H. had an invalid code (not A,C, or D)

```
BAKER,R.            152
BARTLETT,S.G.       183
BRAMLEY,O.H.        439
CHEESEMAN,O.        141
ELLIOTT,D.          291
FIGGINS,S.          413
GOODFELLOW,D.T.     889
HARVEY,C.D.W.       205
KENDALL,J.G.        294
LANCASTER,W.R.      624
NEWMAN,M.W.         516
PITT,W.H.           515
READ,K.M.           001
SHEERS,C.D.         241
SUTCLIFFE,M.        472
TAYLOR,G.C.         407
WILTON,L.W.         404
WINSTONE,E.M.       307
```

Sample file after running “updtsamp”

Technical notes

Iron Spring PL/I INDEXED file support uses an open-source (LGPL) package “pblIsam,” by Peter Graf, written in the C language. Documentation for pblIsam can be found at http://www.mission-base.com/peter/source/pbl/doc/isamfile.html. Source for pblIsam is available on GitHub. A 32-bit version of pblisam is included with the Iron Spring PL/I library in file “libpbl.a”.

pblIsam provides many more options than are used in the current PL/I implementation including alternate indexes and duplicate keys.

A PL/I library procedure “isam” provides the mapping between PL/I statements and options and pblIsam functions. This procedure will be linked with any program that declares an INDEXED file.