CHAIN and %KDS from IBM material

The CHAIN operation retrieves a record from a full procedural file (F in position 18 of the file description specifications), sets a record identifying indicator on (if specified on the input specifications), and places the data from the record into the input fields.

The search argument, *search-arg*, must be the key or relative record number used to retrieve the record. If access is by key, *search-arg* can be a a single key in the form of a field name, a named constant, a figurative constant, or a literal.

If the file is an externally-described file, *search-arg* can also be a composite key in the form of a %KDS. If access is by relative record number, *search-arg* must be an integer literal or a numeric field with zero decimal positions.

The *name* operand specifies the file or record format name that is to be read. A record format name is valid with an externally described file. If a file name is specified in *name* and access is by key, the CHAIN operation retrieves the first record that matches the search argument.

If *name* is a record format name and access is by key, the CHAIN operation retrieves the first record of the specified record type whose key matches the search argument. If no record is found of the specified record type that matches the search argument, a no-record-found condition exists.

If the *data-structure* operand is specified, the record is read directly into the data structure. If *name* refers to a program-described file (identified by an F in position 22 of the file description specification), the data structure can be any data structure of the same length as the file's declared record length. If *name* refers to an externally-described file or a record format from an externally described file, the data structure must be a data structure defined with EXTNAME(...:*INPUT) or LIKEREC(...:*INPUT). See File Operations for information on how to define the data structure and how data is transferred between the file and the data structure.

For a WORKSTN file, the CHAIN operation retrieves a subfile record.

For a multiple device file, you must specify a record format in the *name* operand. Data is read from the program device identified by the field name specified in the DEVID(fieldname) keyword in the file specifications for the device file. If the keyword is not specified, data is read from the device for the last successful input operation to the file.

If the file is specified as an input DISK file, all records are read without locks and so no operation extender can be specified. If the file is specified as update, all records are locked if the N operation extender is not specified.

If you are reading from an update disk file, you can specify an N operation extender to indicate that no lock should be placed on the record when it is read (e.g. CHAIN (N)). See the *WebSphere Development Studio: ILE RPG Programmer's Guide* for more information.

You can specify an indicator in positions 71-72 that is set on if no record in the file matches the search argument. This information can also be obtained from the %FOUND built-in function, which returns '0' if no record is found, and '1' if a record is found.

To handle CHAIN exceptions (file status codes greater than 1000), either the operation code extender 'E' or an error indicator ER can be specified, but not both. For more information on error handling, see File Exception/Errors.

Positions 75 and 76 must be blank.

When the CHAIN operation is successful, the file specified in *name* is positioned such that a subsequent read operation retrieves the record logically following or preceding the retrieved record. When the CHAIN operation is not completed successfully (for example, an error occurs or no record is found), the file specified in *name* must be repositioned (for example, by a CHAIN or SETLL operation) before a subsequent read operation can be done on that file.

If an update (on the calculation or output specifications) is done on the file specified in *name* immediately after a successful CHAIN operation to that file, the last record retrieved is updated.

Note:

Operation code extenders H, M, and R are allowed only when the search argument is a list or is %KDS().

Figure 275. CHAIN Operation with a File Name

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
*
*  The CHAIN operation retrieves the first record from the file,
*  FILEX, that has a key field with the same value as the search
*  argument KEY (factor 1).

/FREE
    CHAIN  KEY  FILEX;
  //  If a record with a key value equal to the search argument is
 //  not found, %FOUND returns '0' and the EXSR operation is
 //  processed. If a record is found with a key value equal
 //  to the search argument, the program continues with
 //  the calculations after the EXSR operation.
     IF  NOT %FOUND;
       EXSR  Not_Found;
    ENDIF;
/END-FREE
```

Figure 276. CHAIN Operation Using a List of Key Fields

```
FFilename++IPEASF.....L.....A.Device+.Keywords++++++++++++++++++++++++++
FCUSTFILE   IF   E           K DISK
```

```
 /free
        // Specify the search keys directly in a list
        chain ('abc' : 'AB') custrec;
        // Expressions can be used in the list of keys
        chain (%xlate(custname : LO : UP) : companyCode + partCode)
              custrec;
        return;
```

Figure 277. CHAIN Operation Using a Data Structure with an Externally-Described File

```
FFilename++IPEASF.....L.....A.Device+.Keywords++++++++++++++++++++++++++
FCUSTFILE   IF   E           K DISK
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D custRecDs       ds                  likerec(custRec)

 /free
        // Read the record directly into the data structure
        chain ('abc' : 'AB') custRec custRecDs;
        // Use the data structure fields
        if (custRecDs.code = *BLANKS);
           custRecDs.code = getCompanyCode (custRecDs);
           update custRec custRecDs;
        endif;
```

%KDS (Search Arguments in Data Structure)

```
%KDS(data-structure-name{:num-keys})
```

%KDS is allowed as the search argument for any keyed Input/Output operation (CHAIN, DELETE, READE, READPE, SETGT, SETLL) coded in a free-form group. The search argument is specified by the subfields of the data structure name coded as the first argument of the built-in function. The key data structure may be (but is not limited to), an externally described data structure with keyword EXTNAME(...:*KEY) or LIKEREC(...:*KEY)..

Notes:

1. The first argument must be the name of a data structure. This includes any subfield defined with keyword LIKEDS or LIKEREC.
2. The second argument specifies how many of the subfields to use as the search argument.
3. The individual key values in the compound key are taken from the top level subfields of the data structure. Subfields defined with LIKEDS are considered character data.
4. Subfields used to form the compound key must not be arrays.
5. The types of all subfields (up to the number specified by "num-keys") must match the types of the actual keys. Where lengths and formats differ, the value is converted to the proper length and format.

6. If the data structure is defined as an array data structure (using keyword DIM), an index must be supplied for the data structure.
7. Opcode extenders H, M, or R specified on the keyed Input/Output operations code affect the moving of the search argument to the corresponding position in the key build area.

Example:

Figure 218. Example of Search on Keyed Input/Output Operations

```
A.........T.Name++++++RLen++TDpB......Functions++++++++++++++++++
A           R CUSTR
A             NAME          100A
A             ZIP            10A
A             ADDR          100A
A           K NAME
A           K ZIP
FFilename++IPEASF.....L.....A.Device+.Keywords++++++++++++++++++++++++
Fcustfile  if   e            k disk    rename(CUSTR:custRec)
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++
D custRecKeys     ds                   likerec(custRec : *key)
 ...
 /free
         // custRecKeys is a qualified data structure
         custRecKeys.name = customer;
         custRecKeys.zip = zipcode;
         // the *KEY data structure is used as the search argument for CHAIN
         chain %kds(custRecKeys) custRec;
 /end-free
```