3

Input/Output Using Free Format

Most input and output functions are the same in free-format RPG IV as in fixed format except for the location of the code within the source line. One more substantial difference introduced in free format is the use of alternatives to a key list in database I/0 for Chain, Set, and similar operations. Also, database update now features the new %Fields built-in function option.

In this chapter, we look at the operations, options, and built-in functions now available for database I/O, as well as for workstation I/O and printer output. You'll find that the free-format approach to input and output varies little from the extended Factor 2 calculation format.

Database Input

Input from database files comes from various operations: Read (Read next), ReadE (Read next equal), ReadP (Read prior), ReadPE (Read prior equal), Chain (Chain), Setll (Set lower limit), and Setgt (Set greater than). The set file pointer operations Setll and Setgt don't provide data from a record, but they can furnish information about a file's key (found or equal) without accessing the record data. Of course, these operations also set the file pointer.

If successful, all the read operations and Chain provide data from an entire record. If unsuccessful, these operations set a condition: "end-of-file" for the read operations or "not found" for Chain. In most programs, we check these conditions to determine what to do next. If end-of-file or not found is determined, the record data remains unchanged from a prior successful operation. If no prior successful operation occurred, fields of the record retain their initial value.

If you're already familiar with the database input operations from your experience with fixed-format RPG, you'll have no problem adjusting to free format. The big difference is that we can't call on resulting indicators in free format, so we must use built-in functions to determine the outcome of attempted input or output functions. These built-in functions — %Eof, %Equal, %Error, and %Found — have been available for extended Factor 2 calculations for several years.

%Eof

The %Eof built-in function tests a specified file for end-of-file. If you specify no file, %Eof checks the last file read for end-of-file. The function returns a value of the indicator data type: either the value '1' to signify that the end-of-file condition was met or '0' otherwise.

You can use the %Eof built-in with all read operations. In the case of read prior operations, %Eof lets you test for beginning-of-file. Listing 3-1 shows examples of read operations used with the %Eof built-in function.

Listing 3-1: Read operations using the %Eof built-in function

```
/free
 Read File_A;           // Read first record
 Dow not %eof(File_A);  // While not at end-of-file
     // Process record
   Read File_A;         // Read next record
 Enddo;
```

```
      Setgt *Hival File_B;   // Set file pointer to eof
      ReadP File_B;          // Read prior - first
      Dow not %eof(File_B);  // While not at beg-of-file
        // Process record
        ReadP File_B;        // Read next prior
      Enddo;

     /end-free
```

%Found

Another built-in function used with database input is %Found. You use this function after a Chain operation to determine whether the record access was successful. Like the %Eof built-in function, %Found returns a value of the indicator data type: value ‘1’ for record found or ‘0’ for no record found. You can optionally specify the name of the file you want to test with %Found. If you specify no file name, the operation checks the most recent operation that sets a %Found condition. In addition to Chain, the following operations can set %Found: Check (Check characters), CheckR (Check reverse), Delete (Delete record), Lookup (Look up a table or array element), Scan (Scan string), Setgt (Set greater than), and Setll (Set lower limit).

In Chapter 2, we reviewed keyed access for Chain and Set operations. Two alternatives are now available that eliminate the need for the Klist (Define a composite key) and Kfld (Define parts of a key) operations. In the first method, a named data structure defined in definition specifications uses a keyword LikeRec with a data file record name as its first parameter and *Key as the second. (You can also use the keyword ExtName with the data file record name and *Key as the second parameter.) The record name you specify should match the record name of the file that will be used in the Chain or similar operation in the calculations. The data structure becomes a qualified data structure, with subfields referenced using the form recordname.fieldname. The subfields of this data structure that are related to the keys of the file will be used as the argument key fields. On the Chain or similar operation, you use the %Kds built-in function. This function has two parameters: the named data structure mentioned above and an optional constant specifying how many key fields to use from the data structure in the operation. If omitted, the second parameter defaults to all key fields.

The second method available as an alternative to the Klist and Kfld operations is the inline composite argument list, which you provide on the calculation operation line. In this approach, you specify fields in a parameter-style list, and together these fields comprise the lookup key argument.

When you use either of these methods, no fixed-format calculations (for Klist and Kfld) are necessary. Listing 3-2 shows sample Chain operations that use the %Found built-in function and the two key list alternatives.

Listing 3-2: Chain operation alternatives with %Found built-in function

```
D Rec_Key          DS             LikeRec(File_C:*key)
 * Assume File_C has key fields CustNo and Invno

 /free
  // Method 1, using the Rec_Key data structure
  Rec_key.Custno = Arg_Cust;
  Rec_key.Invno  = Arg_Inv;
  Chain %kds(Rec_key) File_C;
  If %found(FileC);
    // Process found record here
  Endif;

  // Method 2, using a composite argument list
  // No data structure or key list is needed
  Chain (Arg_Cust:Arg_Inv) File_C;
  If %found(File_C);
    // Process found record here
  Endif;

 /end-free
```

You can also use the %Found built-in function after a Setll or Setgt operation. In this situation, %Found returns the value ‘1’ if there is a key in the file whose value is equal to or greater than the key list argument (for Setll) or whose value is greater than the key list argument (for Setgt).

%Error

Another built-in function available for Read and Chain database operations is the %Error built-in function. To enable this function, you must specify the operation extender (e) on the Read or Chain operation. The (e) extender tells the compiler that you want to handle file errors associated with the Read or Chain. Specifying the (e) disables the RPG default error handler for the operation. The %Error built-in function returns a value of data type indicator: ‘1’ if an error occurred or ‘0’ otherwise.

If you use the (e) extender, it’s important to include some kind of error handling in your program. To do so, code an “If %error” statement after the Read or Chain, followed by the desired error handling. The (e) operation extender provides the same function as placing an indicator in the low position of resulting indicators in the original fixed-format version of Read and Chain. If you place the (e) extender on a Read or Chain operation but don’t test %Error, you are permitting an error to occur without any action being taken. It’s difficult to predict the harm that this omission might cause.

Listing 3-3 shows an example that uses the %Error function.

Listing 3-3: Chain and Read operations with the %Error built-in function

```
/free
 Chain(e) (Arg_Cust:Arg_Inv) File_D;
 If %error;
   Exsr Error_subr;
 Endif;

 Read(e) File_A;
 If %error;
   Exsr Error_subr;
 Endif;
/end-free
```

%Equal

You can use the %Equal built-in function after a Setll operation to determine whether a record whose key matches the key list argument exists in the file. You can use a partial key in the key argument list as well. This combination can provide a valuable utility if the file being accessed has a multiple-field key. For example, say you have an “on-order” file with a two-part key: customer number and order number. To determine whether a certain customer has one or more data records in the file, you need only code a Setll to the file using the customer number as the argument and then check the status of %Equal after the Setll. If the function returns ‘1’, data exists for the customer, and the file pointer is positioned at the first record. If %Equal returns ‘0’, no data records exist for the customer.

Listing 3-4 illustrates using Setll and Setgt with built-in functions %Found and %Equal.

Listing 3-4: Using Setll and Setgt with %Found and %Equal built-in functions

```
/free
 // Assume EmpMast has a three-part key:
 // Company, Dept, and EmpNo.
 // To check for existence of a specific record by
 // key but without chaining:
 Setll (Arg_Co:Arg_Dept:Arg_Empl) EmpMast;
 If %equal(EmpMast);
   // A specific employee record was found.
 Endif;

 // To determine whether at least one employee
 // exists in a particular company and department
 Setll (Arg_Co:Arg_Dept) EmpMast; // Set file pointer
 If %equal(EmpMast);      // Check for existence
   // An employee record was found.
 Endif;

 // To set the file pointer to the next dept
 Setgt (Arg_Co:Arg_Dept) EmpMast;

 // %found will return ‘1’ after Setll or Setgt
 // unless end-of-file is reached

/end-free
```

Data Area Input

Data areas, especially data area objects, could reasonably be called part of the database. You use the same data structures (definition specifications) regardless of the format of the calculations. Input from a data area is accomplished automatically, via the In (Retrieve a data area) operation, or by both methods.

Database Output

The following operations perform output to database files: Write (Add a new record), Update (Modify an existing record), Delete (Delete a record), and, if you're using program-described files, the Except (Exception output) operation.

Write

The Write operation adds a new record to a file. No prior read is necessary, but the file must be opened before you request the Write. The operation writes an entire record, and you must take care to ensure all fields are loaded properly before the Write. If new records are to be added to a file, the file must have either a file type of O or an A in position 20 to denote that you plan to add records to the file being read (file type I) or updated (file type U).

Update

The Update operation works the same in free format as before. However, a new built-in function, %Fields, gives free-format programmers the option to update only specified fields instead of an entire record. You can do this in fixed format by using the Except operation and output specifications, but the %Fields function isn't available to fixed-format users of the Update operation. Without the %Fields option, the entire record is modified using the current values of the fields defined in the record.

For an update to succeed, you must specify U (for update) on the file description specification, and a successful Read (or Chain) of a record must occur. During the time between the successful Read (or Chain) and the Update operation, the record is locked. No other user can access the record for update until the update is performed, another record is read, or an Unlock (Unlock a data area or release a record) operation is performed. Be aware that record locking provides a needed function to maintain data integrity, but unless you program carefully, it can create operational grief.

Delete

The Delete operation removes a record from a file. If you specify no search argument, the operation deletes the record most recently read with a Read or Chain operation. If you specify a search argument (a relative record number or key), the compiler uses the argument to locate the record to delete. For files using a key, free format lets you use the %Kds built-in function or an in-line composite list instead of the fixed-format Klist to specify the search key.

To verify that the Delete operation found and deleted the record, you should use the %Found built-in function after the Delete. Using the search argument, no prior read is needed. After a delete using a key argument, the file pointer is positioned to the next record after the deleted one.

Except

You use the Except operation with a label to specify which output in output specifications to perform. The output specifications have a matching label on Exception output records. The function of Except output in free-format RPG IV is identical to fixed-format calculations.

Listing 3-5 shows some sample database output operations and their built-in functions.

Listing 3-5: Database output operations and built-in functions

```
/free
 // Write a new record in a file
 Write Record_A;

 // Update an entire record
 Update Record_A;
```

```
  // Update only certain fields in the record
  Update Record_A %fields(Name:Zip:Amount_Due);

  // Delete the last record read
  Delete Record_A;

  // Delete a record using a key
  Delete (Arg_1:Arg_2) Record_A;
  If not %found;
    Message = ‘Record was not deleted’;
  Endif;

  // Perform exception output
  Except Label_A;

/end-free
```

Workstation I/O

To perform workstation input and output in free format, you use the same methods as in fixed format. The only difference is where you place the operation code and parameters in the free-format source statement.

Write/Read

The most common operation to a workstation device is the Exfmt (Write/then read format) operation. The write portion of this operation moves the data from the specified record to the buffer of the display device’s open data path. The device function manager checks the option indicators and performs the selected options, such as setting display attributes, displaying error messages, or performing keyword functions.

The read part of the Exfmt operation sends the output buffer to the device and then waits for input from the device. The input occurs when the user presses either Enter or an enabled function key. The operating system handles non-enabled function keys by returning a message informing the user that the key is not available.

Write

The Write operation usually is associated with a display file for which an overlay function is needed — for example, for a trailer record preceding a subfile control record or a message subfile record preceding a regular display format.

Read

We seldom use the Read operation in a display file. You can code it immediately after a Write operation, but programmers usually use the Exfmt operation to perform this combination. Read is more commonly used when the specified file is an Intersystem Communications Facility (ICF) file.

ICF I/O

RPG IV supports ICF files by letting you specify device WORKSTN in the file description. The RPG operation codes that you specify dictate what the communications device will do. The file description includes a device name (Dev) keyword that ICF requires. This keyword's value corresponds to the device entry names in the ICF file.

An Exfmt operation to an ICF record becomes a three-function combination: send a record, send a "turnaround" instruction to the other station, and then receive a record from the other station. The Write operation is simply a send record operation. The Read operation with a record format is a receive operation. If the Invite keyword (DDS in the ICF file) was used previously on a Write operation, a Read operation using the ICF file name becomes a read-from-invited-devices operation. In communications lingo, this means that any device in the device file (that has been invited) may now send to the program. With this kind of Read, you can specify a record wait time limit that, if reached, can cause control to return to the program along with a timeout exception.

Dsply

The Dsply (Display message) operation is available in free format to provide the same functionality as its fixed-format counterpart. Because this operation comes from an original RPG format, you must remember to code the operation Dsply first, followed by Factor 1 information, and then Factor 2 information.

Listing 3-6 shows examples of workstation I/O using free-format RPG IV.

Listing 3-6 Workstation I/O operations available in free format

```
/free

 // The following Exfmt sends the “Prompt” record
 // to the workstation and waits for the response.

 Exfmt Prompt Display_F;


 // The Write operation sends a record to a buffer.
 // The requested functions are performed, but data
 // is not displayed now. In subfile programming, a
 // “Trailer” record is written before issuing a
 // write/read for the “Control” record.

 Write Trailer;
 Exfmt Control;


 // Using Intersystem Communications Facility (ICF),
 // a Read is used (by file name) to access data
 // from any invited communications device.
 // The file is an ICF file and uses the Workstn
 // device in the RPG IV file description.

 Read(e) Comm_File;

 // After the Read, either a data record has been
 // received or a time-out has occurred. You can use
 // the %error built-in to determine which is the
 // case.


 // The Dsply operation lets you communicate with
 // the user. The operation can display a message
 // and accept a response to an interactive user
 // or to the system operator if running in batch.
 // The following question will be sent to the
 // external message queue, and the response
 // will be returned in field Food.
```

```
        Dsply ('What''s for Lunch?') ' ' Food);
       /end-free
```

Printer Output

In free-format RPG IV, you code printer output, whether program-described or externally described, the same way you do in fixed format.

Overflow Indicator

The overflow indicator has been with RPG for a long time. Indicators OA–OG and OV have served us well. In RPG IV, an externally described printer file can use any numeric indicator. As of V5R1, you can also use a named indicator. This new feature can make RPG IV programs that use printer files easier to read and maintain. The named or numbered indicator is automatically set to *On when printing occurs on or after the overflow line specified in the printer file definition. You can change the overflow line permanently by using the CHGPRTF (Change Printer File) CL command or temporarily by using the OVRPRTF (Override with Printer File) command.

Write

The Write operation uses a record name defined in the printer file and causes all output for the record to be printed.

Except

You can use the Except operation with program-described printer files to print using output specifications. Program-described printing provides nearly all the functionality of externally described printing.

Listing 3-7 shows examples of printer output using free-format RPG IV.

Listing 3-7: Output operations in free format

```
      * An externally described printer file PrintFile
     FPrintFile   O   E           Printer   Oflind(Ofl_1)
     D Ofl_1              s            n

      /free
       // To print all lines described by record Headings
       Write Headings;

       // To check for overflow and redo headings
       If Ofl_1;
         Write Headings;
         Clear Ofl_1;
       Endif;

       // To print a line described by record Detail
       Write Detail;

      /end-free

* A program-described printer file Qprint
      FQprint    O   F    120         Printer Oflind(Ofl_2)
      D Ofl_2            s        n


       /free
        // To print using tag Hdgs on output specs
        Except Hdgs;

        // To check for overflow and redo headings
        If Ofl_2;
          Except Hdgs;
          Clear Ofl_2;
        Endif;

        // To print a detail line using tag Detail
        Except Detail;

      /end-free
```