Free-form RPG support on IBM i

Overview of recent RPG compiler free-form changes

Scott Hanson
Jing Li

January 09, 2014

This article explains the free-form RPG function supported on IBM i, its advantages, and how to program for H, F, D and P specs within free-form. The support is intended to allow RPG to be easier to write and understand for programmers who are familiar with other high-level languages.

Overview

RPG programming on IBM i was greatly improved on IBM i 7.1 in the Technology Refresh 7 (TR7) time frame. Additional free-form support for the RPG language and embedded SQL precompiler was provided for H, F, D, and P specs. Free-form C specs were previously supported, and now all RPG specs have free-form support except I and O specs.

Each free-form statement begins with an operation code and ends with a semicolon. Here is a list of the new operation codes:

- `CTL-OPT` for control specs (H)
- `DCL-F` for file specs (F)
- `DCL-S`, `DCL-DS`, `DCL-SUBF`, `DCL-C`, `DCL-PR`, `DCL-PI`, `DCL-PARM` for data specs (D)
- `DCL-PROC` for procedure specs (P)

Advantages

RPG syntax in free form is similar to other modern languages and can be understood easily. It allows programmers who are familiar with other languages such as Microsoft® Visual Basic, Java™, and PHP to be trained more easily in RPG and with lower cost. The new syntax also makes programming easier for existing RPG programmers. As such, the free-form syntax is beneficial for both existing RPG programmers and new RPG programmers.

- Free-form RPG allows code to be specified as free-form statements rather than in specific fixed columns.

- Free-form code is still restricted to columns 8 – 80.
- The `/FREE` and `/END-FREE` compiler directives are tolerated, but are no longer required for free-form.
- Multiple line definitions are easier to code, easier to maintain, and easier to understand.
- Defining data items with long symbol names is no longer problematic. Table 1 contains a long symbol name definition in fixed form and free form.

Table 1. Long symbol name definition

Fixed-form entry	Free-form equivalent

`D thisisalong...`
`D name S 10A`
	`DCL-S thisisalongname CHAR(10);`
	or
	`DCL-S thisisalong...`
	`name CHAR(10);`

- Free-form definitions can have `/IF`, `/ELSEIF`, `/ELSE`, and `/ENDIF` within the statement. Example 1 illustrates the compiler directives within a declaration statement.

Example 1. Compiler directives within a declaration statement

```
DCL-S salary
    /IF DEFINED(large_vals)
        PACKED(13:3)
    /ELSE
        PACKED(7:3)
    /ENDIF
    ;
```

Changes for control specifications (H spec)

This section describes the changes for a control specification (H spec). The free-form keyword is `CTL-OPT` (Control Option). The free-form control statement starts with `CTL-OPT`, followed by zero or more keywords, and ends with a semicolon. Allowed keywords are the same keywords as an H spec. Multiple control statements are allowed and free-form control statements can be intermixed with fixed-form H specs. Each statement must be a complete statement. Example 2 illustrates an empty control statement.

Example 2. Empty control statement

```
// no keywords
CTL-OPT;
```

The only effect of an empty control statement is to prevent the ILE RPG compiler from using the default H spec data area. Example 3 illustrates additional control statements.

Example 3. Additional control statements

```
    // two keywords
    CTL-OPT OPTION(*SRCSTMT:*NODEBUGIO)
        DFTACTGRP(*No);

    // intermixed free-form and fixed-form
    CTL-OPT DATFMT(*ISO);
H TIMFMT(*ISO)
    CTL-OPT CCSID(*UCS2 :
                  1200);
    // another intermixed free-form and fixed-form
H DATFMT(*ISO)
    CTL-OPT TIMFMT(*ISO);
H CCSID(*UCS2 :
H       1200)
```

One enhancement related to control statements is that if the module contains at least one free-form control statement and one of the `ACTGRP`, `BNDDIR` or `STGMDL` keywords is specified, `DFTACTGRP(*NO)` is implicitly specified and is not required.

Changes for declaration statements (D spec)

This section describes the changes for declaration statements (D spec). A free-form data definition statement starts with `DCL-<x>`, and is followed by the data item name (which can be `*N` if the data item is unnamed), then by an optional data type, and then by zero or more keywords, and finally ends with a semicolon. There are several types of declaration statements: named constants, stand-alone fields, data structures, procedure prototypes, and procedure interfaces.

For those who are familiar with fixed-form definitions, there are a few keyword changes related to data types. Fixed-form keywords that modify a data type, such as `VARYING`, `DATFMT`, `TIMFMT`, `PROCPTR`, and `CLASS,` are merged with the data type keywords. This section provides additional details about these keywords.

- `CLASS`
  Instead of defining a field as an object and adding the `CLASS` keyword, the class is specified as the parameter to the `OBJECT` keyword.

- `DATFMT`
  Instead of defining a field as date and adding the `DATFMT` keyword, the date format is specified as the parameter to the `DATE` keyword.

- `PROCPTR`
  Instead of defining a field as a pointer and adding the `PROCPTR` keyword, `*PROC` is specified as a parameter to the `POINTER` keyword.

- `TIMFMT`
  Instead of defining a field as time and adding the `TIMFMT` keyword, the time format is specified as the parameter to the `TIME` keyword.

- `VARYING`

Instead of defining a field as alphanumeric/UCS-2/graphic and adding the `VARYING` keyword, the field is defined using the `VARCHAR`/`VARUCS2`/`VARGRAPH` keyword.

Example 4 illustrates fixed-form and free-form declarations with these keywords.

Example 4. Fixed-form and free-form keywords

```
D* fixed-form declarations
D string        S       50A VARYING
D date          S         D DATFMT(*MDY)
D obj           S         O CLASS(*JAVA:'MyClass')
D ptr           S         * PROCPTR

    // free-form declarations
    DCL-S string VARCHAR(50);
    DCL-S date DATE(*MDY);
    DCL-S obj OBJECT(*JAVA:'MyClass');
    DCL-S ptr POINTER(*PROC);
```

Table 2 contains all of the data types for free form.

Table 2. Free-form data types

Data type	Free-form syntax	Examples
Alphanumeric	CHAR(len) VARCHAR(len {: varying-size})	DCL-S library CHAR(10); DCL-S libfilembr VARCHAR(33); DCL-S cmdparm VARCHAR(10:4);
UCS-2	UCS2(len) VARUCS2(len {: varying-size})	DCL-S firstName UCS2(10); DCL-S filePath VARUCS2(5000);
Graphic	GRAPH(len) VARGRAPH(len {: varying-size}	DCL-S firstName GRAPH(20); DCL-S fullName VARGRAPH(50);
Indicator	IND	DCL-S isValid IND;
Packed	PACKED(digits {:decimals})	DCL-S numRecords PACKED(5); DCL-S salary PACKED(15:2);
Zoned	ZONED(digits {: decimals})	DCL-S numRecords ZONED(5); DCL-S salary ZONED(15:2);
Binary	BINDEC(digits {: decimals})	DCL-S numRecords BINDEC(9); DCL-S bonus BINDEC(9:2);
Integer	INT(digits) Digits can be 3, 5, 10, 20	DCL-S index INT(10);
Unsigned	UNS(digits) Digits can be 3, 5, 10, 20	DCL-S count UNS(20);
Float	FLOAT(bytes) Bytes can be 4, 8	DCL-S variance FLOAT(8);
Date	DATE { (format) }	DCL-S duedate DATE; DCL-S displayDate DATE(*YMD);
Time	TIME { (format) }	DCL-S startTime TIME; DCL-S displayTime TIME(*USA);
Timestamp	TIMESTAMP	DCL-S start TIMESTAMP;
Pointer	POINTER	DCL-S pUserspace POINTER;
Procedure	POINTER(*PROC)	DCL-S pProc POINTER(*PROC);

pointer		
Object	OBJECT{(*JAVA : class)}	DCL-S obj OBJECT(*JAVA:'Cls');

One enhancement related to the declaration statements is that file definitions and data definitions might be intermixed. This new rule also applies to fixed-form F and D specifications.

Named constants

A named constant declaration starts with `DCL-C`, and is then followed by the name, then by the optional keyword `CONST`, and then by the value and finally ends with a semicolon. Example 5 illustrates several named constant declarations.

Example 5. Named constants

```
// without the optional CONST keyword
DCL-C lower_bound -50;
DCL-C max_count 200;
DCL-C start_letter 'A';

// with the optional CONST keyword
DCL-C upper_bound CONST(-50);
DCL-C min_count CONST(200);
DCL-C end_letter CONST('A');
```

Specifying the `CONST` keyword makes no difference in the meaning of the declaration.

Stand-alone fields

A stand-alone field declaration starts with `DCL-S`, and is followed by the name, then by an optional data type keyword, and then by zero or more keywords, and ends with a semicolon. If a data type keyword is specified, it must be the first keyword. Example 6 illustrates several stand-alone field declarations.

Example 6. Stand-alone fields

```
DCL-S first_name CHAR(10) INZ('John');
DCL-S last_name  VARCHAR(20);
DCL-S index PACKED(6);
DCL-S salary PACKED(8:2);
```

If the `LIKE` keyword is used, there is no data type keyword. An optional second parameter can be specified on the `LIKE` keyword to do a length adjustment. Example 7 illustrates stand-alone declarations with the `LIKE` keyword.

Example 7. Stand-alone fields with LIKE

```
// Define using the LIKE keyword
DCL-S cust_index LIKE(index);

//Specify length adjustment with LIKE keyword
DCL-S big_index LIKE(index : +6);
```

Named constants can be used within free-form keyword declarations. The named constant must be defined before the definition statement. Example 8 illustrates using named constants within declarations.

Example 8. Declarations using named constants

```
DCL-C name_len CONST(10);
DCL-S one CHAR(name_len);
DCL-S two VARCHAR(name_len);
DCL-C digits 10;
DCL-C positions 3;
DCL-S value PACKED(digits:positions);
```

Data structures

A data structure declaration starts with `DCL-DS`, and is followed by a name (which can be `*N` if the data structure is unnamed) and then by zero or more keywords, and ends with a semicolon. Externally described data structures must have either the `EXT` keyword or the `EXTNAME` keyword specified as the first keyword. If the data structure can have program-described subfields, an `END-DS` statement (with an optional name) must be specified. Example 9 illustrates several data structure declarations.

Example 9. Data structures

```
// Program described data structure
DCL-DS data_str_1;
    emp_name CHAR(10);
    first_name CHAR(10);
    salary PACKED(8:2);
END-DS;

// Program described data structure
DCL-DS data_str_2;
    value VARCHAR(4);
    index INT(10);
END-DS data_str_2;

//Unnamed data structure
DCL-DS *N;
    item VARCHAR(40);
END-DS;
```

If the data structure does not have a name, the `END-DS` statement must be specified without an operand.

The `END-DS` operation is not used for a data structure defined with the `LIKEDS` or `LIKEREC` keywords, because additional subfields cannot be coded. Example 10 illustrates a data structure declaration using the `LIKEREC` keyword.

Example 10. Data structure using LIKEREC

```
DCL-DS custoutput LIKEREC(custrec);
```

If the subfield list is empty, the `END-DS` statement can be merged with the `DCL-DS` statement by being coded immediately before the semicolon for the `DCL-DS` statement. In this case, the data structure name parameter of the `END-DS` statement is not allowed. Example 11 illustrates a data structure declaration with the `END-DS` statement coded on the data declaration.

Example 11. Data structure with END-DS

```
DCL-DS PRT_DS LEN(132) END-DS;
```

Example 12 contains two declarations that are equivalent. In the first declaration, the `END-DS` statement is coded separately. In the second declaration, the `END-DS` statement is merged with the `DCL-DS` statement.

Example 12. Two equivalent data structures (END-DS)

```
DCL-DS myrecord EXT;
END-DS;

DCL-DS myrecord EXT END-DS;
```

Data structure subfields start with the `DCL-SUBF` keyword, but the keyword is optional unless the name is the same as a free-form operation code (op code). Example 13 illustrates a subfield declaration where the `DCL-SUBF` keyword is required. The `DCL-SUBF` keyword must be used because `read` is an op code supported in free-form.

Example 13. Data structure with the DCL-SUBF keyword

```
DCL-DS record_one;
    buffer CHAR(25);
    DCL-SUBF read INT(3);
END-DS;
```

The new `POS` keyword replaces From/To positions and `OVERLAY(dsname)`. The data structure name is not allowed as a parameter for the `OVERLAY` keyword in free format. Example 14 contains two declarations that are equivalent. In the first declaration, fixed-form is used. In the second declaration, free-form is used.

Example 14. Two equivalent data structures (OVERLAY)

```
D* fixed-form declaration
D myds          DS
D   subf1           11      15A
D   subf2                    5P 2   OVERLAY(myds)
D   subf3                   10A     OVERLAY(myds:100)
D   subf4                   15A     OVERLAY(myds:*NEXT)

    //free-form declaration
    DCL-DS myds;
        subf1 CHAR(5) POS(11);
        subf2 PACKED(5:2) POS(1);
        subf3 CHAR(10) POS(100);
        subf4 CHAR(15);
    END-DS;
```

The `OVERLAY(dsname:*NEXT)` keyword means the same as no `OVERLAY` keyword at all.

Procedure prototypes

A procedure prototype declaration starts with `DCL-PR`, and is followed by a name and then by zero or more keywords, and ends with a semicolon. If the procedure has a return value and a data type keyword is used to specify the type of the return value, the data type keyword must be specified as the first keyword. An `END-PR` statement (with an optional name) must be specified. Example 15 illustrates a procedure prototype named `cosine` which is a prototype for the C function `cos.`

Example 15. Cosine procedure prototype

```
DCL-PR cosine FLOAT(8) EXTPROC('cos');
    angle FLOAT(8) VALUE;
END-PR;
```

Example 16 illustrates a procedure prototype for a bound application programming interface (API) named `Qp0zGetEnvCCSID`. It uses the special name `*DCLCASE` on the `EXTPROC` keyword to have the external procedure name in the same case as the prototype name is coded. The external name of the procedure is `Qp0zGetEnvCCSID`, exactly as specified for the procedure name.

Example 16. Qp0zGetEnvCCSID procedure prototype

```
DCL-PR Qp0zGetEnvCCSID EXTPROC(*DCLCASE);
    envvar POINTER VALUE OPTIONS(*STRING);
    ccsid INT(10) VALUE;
END-PR;
```

As with `DCL-DS` and `END-DS`, the `END-PR` statement can be merged with the `DCL-PR` statement for a procedure that has no parameters. Example 17 illustrates a prototype for a procedure named `getCurTotal` that has no parameters. The `END-PR` operation code is merged with the `DCL-PR` statement.

Example 17. GetCurTotal procedure prototype

```
DCL-PR getCurTotal PACKED(31:3) END-PR;
```

Procedure prototype parameters start with `DCL-PARM` and follow the same rules as that of the data structure subfields. The `DCL-PARM` keyword is optional unless the name is the same as a free-form op code. Example 18 illustrates a parameter declaration where the `DCL-PARM` keyword is required. The `DCL-PARM` keyword must be used because `read` is an op code supported in free form.

Example 18. Procedure prototype parameter with the DCL-PARM keyword

```
DCL-PR proc_one;
    buffer CHAR(25) CONST;
    DCL-PARM read INT(3) VALUE;
END-PR;
```

Procedure interfaces

Procedure interfaces are very similar to procedure prototypes.

A procedure interface declaration starts with `DCL-PI`, and is followed by a name (which can be `*N` if the procedure interface is unnamed) and then by zero or more keywords, and ends with a semicolon. If the procedure has a return value and a data type keyword is used to specify the type of the return value, the data type keyword must be specified as the first keyword. An `END-PI` statement (with an optional name) must be specified. Example 19 illustrates a procedure interface with no name and no parameters.

Example 19. Procedure interface without parameters

```
DCL-PI *N CHAR(10);
END-PI;
```

Procedure interface parameters start with `DCL-PARM` and follow the same rules as that of data structure subfields. The `DCL-PARM` keyword is optional unless the name is the same as that of a free-form op code. Example 20 illustrates a parameter declaration where the `DCL-PARM` keyword is required. The `DCL-PARM` keyword must be used because `read` is an op code supported in free form.

Example 20. Procedure interface parameter with the DCL-PARM keyword

```
DCL-PI proc_one;
    buffer CHAR(25) CONST;
    DCL-PARM read INT(3) VALUE;
END-PI;
```

Changes for procedure specifications (P spec)

This section describes the changes for a procedure specification (P spec). The free-form procedure statement declaration starts with `DCL-PROC`, and is followed by a name, then by an optional return type, and then by zero or more keywords, and ends with a semicolon. Then the optional parameter definitions are specified. An `END-PROC` statement (with an optional name) must be specified.

Example 21 illustrates a procedure named `SubProc1`. It returns two elements in an array of data type `VARCHAR(100)` and includes several parameters with different data types.

Example 21. Procedure statement

```
DCL-PR SubProc1 VARCHAR(100) DIM(2);
    varchar1    VARCHAR(10) CONST;
    ucs1        UCS2(5) CONST;
    varucs1     VARUCS2(5) CONST;
    graph1      GRAPH(20) CONST;
    vgraph1     VARGRAPH(50) CONST;
    packed1     PACKED(10) CONST;
    binary1     BINDEC(2) CONST;
    uns1        UNS(3) CONST;
    float1      FLOAT(4) CONST;
END-PR;
```

Changes for file specifications (F spec)

This section describes the changes for a file specification (F spec). The free-form file definition statement starts with `DCL-F`, and is followed by a file name and then by zero or more keywords, and ends with a semicolon. The usage is determined by default if it is not explicitly declared.

Example 22 illustrates a file definition statement that defines an externally described `WORKSTN` file named `dspf`, which is opened for input and output using the file description from external file `TABF035001`.

Example 22. File definition statement

```
DCL-F dspf WORKSTN
           EXTDESC('TABF035001');
```

Example 23 is an equivalent file definition statement that explicitly lists the file attributes, rather than relying on the compiler default values.

Example 23. File definition statement (with additional keywords)

```
DCL-F dspf WORKSTN(*EXT) USAGE(*INPUT:*OUTPUT)
           EXTDESC('TABF035001');
```

Table 3 illustrates free-form equivalents for fixed-form file entries.

Table 3. Free-form equivalents for fixed-form file entries

Fixed-form entry	Free-form equivalent
File type	`USAGE` keyword
File designation	No free-form syntax is related to the file designation. All free-form files are fully procedural or output
End of file	Not supported in free-form
File addition	USAGE(*OUTPUT)
Sequence	Not supported in free form
Record length	Parameter of device keyword `DISK`, `PRINTER`, `SEQ`, `SPECIAL`, and `WORKSTN`
Limits processing	Not supported in free form
Length of key field	Length parameter of the `KEYED` keyword
Record address type	`KEYED` keyword
File organization	**I -** `KEYED` keyword **T -** Not supported in free-form
Device	Device keyword `DISK`, `PRINTER`, `SEQ`, `SPECIAL`, `WORKSTN`

Summary

The additional free-form RPG support provides free-form programming for H, F, D and P specs and is intended to make RPG programming easier for both existing RPG programmers and to

those trying to learn RPG. The required software to enable the new support is 7.1 RPG Compiler PTF SI51094 and IBM® DB2® PTF Group SF99701 Level 26. Go ahead and give it a try!

Related topic

- IBM i Database Embedded SQL programming