Defining variables in RPG all free

Prior to the new *all* free RPG variables (fields) would have been coded in the Definition specification, D-spec. With the new version of RPG the fixed format D-spec has gone. It has been replaced by new free form definition statements. At the time I am writing this post I have not found any other articles giving examples of how to code these new definition statements, therefore, I am going to give examples of how I have used them as I am starting to become this new version of the language.

The definitions I am going to discuss here are:

- Define a standalone variable - `dcl-s`
- Define a data structure - `dcl-ds` and `end-ds`
- Define a data structure subfield - `dcl-subf`
- Define a constant - `dcl-c`

I am going to give examples of the way I would have coded various type of variables using the fixed formal D-spec and how I am coding the same thing using the new free format definition statements. I am not going to go into too much detail as I am assuming that you, the reader, already has a basic knowledge of how to define variables.

Define standalone variable

The free form definition statement to define a standalone variable is `dcl-s`. This is followed by the variable's name, its type and size, and then any relevant keywords which are the same as keywords used with the fixed format D-spec.

This is how I could define an alphanumeric field, and then define another using the `like` keyword. In the first free format statement, line 2, you can see that there is a `char` keyword, this defines the field as alphanumeric, and the number following is its size:

```
    DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
01  D Alpha1          S             10A
02   dcl-s Alpha1 char(10) ;

03  D Alpha2          S                   like(Alpha1)
    D                                       inz(*all'*')
04   dcl-s Alpha2 like(Alpha1) inz(*all'*') ;
```

I would say that packed numeric fields are the most common form of numeric fields in the environments I work in. I am sure it will come as no surprise to you that the keyword `packed` is used to define a packed field. Notice that in the first free statement, line 2, the number of decimal places is not given, while it is in the other free statement, line 4. If the number of decimal places is not given zero is assumed. Also notice that the length of the field is separated from the number of decimal places by a colon, `:`.

```
    DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
01  D Packed1         S              7P 0
02     dcl-s Packed1 packed(7) ;

03  D Packed2         S              5  2
04     dcl-s Packed2 packed(5:2) ;
```

Signed numeric variables are now defined using the `zoned` keyword. Otherwise their definition is the same as packed numeric fields.

```
    DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
01  D Signed1         S              7S 2
02     dcl-s Signed1 zoned(7:2) ;
```

To define integer, unsigned integer, and float numeric variables the `int`, `uns`, and `float` keywords are used.

```
    DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
01  D Int1            S              3I 0
02     dcl-s Int1 int(3) ;

03  D Uns1            S              3U 0
04     dcl-s Uns1 uns(3) ;

05  D Float1          S              8F
06     dcl-s Float1 float(8) ;
```

As you would expect date variables defined using the `date` keyword, see line 2. I can define the date format, if needed, following the `date` keyword. A time variable is defined with the `time` keyword, line 4, if I do not want to use the default time format I can give it following the `time` keyword. Timestamp variables are defined using the `timestamp` keyword.

```
    DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
01  D Date1           S               D   datfmt(*iso)
02     dcl-s Date1 date(*iso) ;

03  D Time1           S               T
04     dcl-s Time1 time ;

05  D TimeStamp1      S               Z
06     dcl-s TimeStamp1 timestamp ;
```

Indicator variables are defined using the `ind` keyword.

```
    DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
01  D Ind1            S               N

02     dcl-s Ind1 ind ;
```

Very occasionally I have to use a binary variable, this would be defined using the `bindec` keyword.

```
    DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
01  D Bin1            S              5B 0

02     dcl-s Bin1 bindec(5) ;
```

And pointers are defined using the `pointer` keyword.

```
    DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
01  D Pointer1        S               *
```

```
   dcl-s Pointer1 pointer ;
```

Arrays are defined pretty much the same way you did with the fixed format. You define your variable and then follow it with the `dim` keyword.

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
D Array1          S              3    dim(100)

   dcl-s Array1 char(3) dim(100) ;
```

Tables are more troublesome. When using the free format you **must** start the name of the table with "tab".

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
D Table1          S              7    dim(3) ctdata

   dcl-s TabTable1 char(7) dim(3) ctdata ;
```

One last example show how to define a variable to receive the data from a data area.

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
D TestDa1         S             10    dtaara('TESTDA')

   dcl-s TestDa1 char(10) dtaara('TESTDA') ;
```

Define data structure and data structure subfield

I was not surprised to learn that the free form definition statement to define a standalone variable is `dcl-ds`. One new feature is that all data structures either require a name or `*N` to denote that the data structure is not named. The `qualified` keyword can be used, and I do use it. A semicolon is required at the end of the `dcl-ds`.

The declare subfield statement, `dcl-subf` is optional, and I do not use it.

After all of the data structure subfields have been defined a `end-ds` statement is needed.

Below is a data structure below is not named and I have define the equivalent twice, one with the `dcl-subf` and other without.

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
D                 DS
D   Telephone1                  12A
D   AreaCode1                    3A   overlay(Telephone1:2)

   dcl-ds *N ;
      dcl-subf Telephone1 char(12) ;
      dcl-subf AreaCode1 char(3) overlay(Telephone1:2) ;
   end-ds ;

   dcl-ds *N ;
```

```
09       Telephone2 char(12) ;
10       AreaCode2 char(3) overlay(Telephone2:2) ;
11     end-ds ;
```

If I defined the data structure with subfield starting and ending positions, lines 1 – 3, I would need to use the `pos` keyword to say where the subfield starts, lines 5 and 6.

```
    DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
01  D                 DS
02  D  Telephone1             1    12
03  D  AreaCode1              2     4

04     dcl-ds *N ;
05        Telephone2 char(12) pos(1) ;
06       AreaCode2 char(3) pos(2) ;
07     end-ds ;
```

Those of you who are regular reader of this blog know that I use an externally described data structure to define the Program Status data structure (PSDS), see Externally described Data Structures. If you notice that the `dcl-ds` does not end with a semicolon as there are no subfields defined.

```
    DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
01  D PgmDs         ESDS                  extname(RPG4DS) qualified

02     dcl-ds PgmDs
03        extname('RPG4DS') psds qualified
04     end-ds ;
```

For those who code their PDS the other way would code it like this.

```
    DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
01  D PgmDs          SDS                  qualified
02  D PgmName           *proc
03  D Status            *status

04     dcl-ds PgmDs psds qualified ;
05        PgmName *proc ;
06        Status *status ;
07     end-ds ;
```

Regular readers will also know that I use an Indicator data structure to communicate (define the indicators) for display and printer files, see No More Number Indicators.

```
    DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
01  D IndDs           DS                  qualified
02  D   Exit                  3      3N
03  D   Errors               50     59A
04  D   ErrControlGroupNotFound...
05  D                        50     50N
06  D   ErrStrRangeGreaterThanEndRange...
07  D                        51     51N

08     dcl-ds IndDs qualified ;
09        Exit ind pos(3) ;
10        Errors char(10) pos(50) ;
11        ErrControlGroupNotFound ind pos(50) ;
```

```
      ErrStrRangeGreaterThanEndRange ind pos(51) ;
    end-ds ;
```

I have to admit I do not use data structures that much to split or put data together. I use the `%subst` to break data apart into "subfields", and concatenate with the `+`.

Define a constant

The free form definition statement `dcl-c` is used to define a constant. As with the fixed format version I do not have to use the `const` keyword, I can just enter the values.

```
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords
D Constant1       C                   const('ABCDEFG')
D Constant2       C                   'ABCDEFG'

   dcl-c Constant1 const('ABCDEFG') ;
   dcl-c Constant2 'ABCDEFG' ;
```