RPG is free at last!

Barbara Morris
IBM Rational

TR7? (How to get this support)

How does TR7 relate to this enhancement?

- The RPG free-form support was announced with TR7 on November 15, 2013

How do I get the new free-form support?

- You need RPG compiler PTF SI51094 (or its latest supersede) to compile RPGLE source
- You also need DB2 group PTF SF99701 level 26 to compile SQLRPGLE source
- You don't need TR7

Do I need the PTFs on the system where I run the program?

- No, you only need the PTFs for compiling

What about RDI?

- Fixpack 9.0.1 has support for free-form

Another big step forward for RPG – a totally free-form program

```
ctl-opt bnddir('ACCRCV');

dcl-f custfile usage(*update);
dcl-ds custDs likerec(custRec);
dcl-f report printer;

read custfile custDs;
dow not %eof;
   if dueDate > %date(); // overdue?
      sendOverdueNotice();
      write reportFmt;
      exec sql insert :name, :duedate into
               mylib/myfile;
   endif;
   read custfile custDs;
enddo;
*inlr = '1';

dcl-proc sendOverdueNotice;
   sendInvoice (custDs : %date());
end-proc;
```

RPG programmers will find this new syntax easy to learn

Non-RPG programmers will find this new syntax much easier to learn than fixed form

How far RPG has come

Let's take a look at the last 25 years of RPG syntax

RPG III (OPM RPG) System-38 – V2R3

```
FCUSTFILEIF  E                    DISK
FREPORT  O   E                    PRINTER
ICUSTDS    E DSCUSTFILE
 /COPY GETCURDAT
 /COPY INVOICE
C                     READ CUSTFILE
C           *INLR     DOWNE*ON
C           DUEDAT    IFGT CURDAT
C                     EXSR SNOVDU
C                     WRITEREPORTFM
C/EXEC SQL INSERT :NAME, :DUEDATE INTO
C+              MYLIB/MYFILE
C/END-EXEC
C                     ENDIF
C                     READ CUSTFILE                 LR
C                     ENDDO
C*
C           SNOVDU    BEGSR
C                     CALL 'SNDINVCE'
C                     PARM           CUSTDS
C                     PARM ISOVDU    OVERDU 10
C                     ENDSR
```

Limit of 6 character names. "Send overdue notice" = SNOVDU

All code is upper case

V3R1

```
H bnddir('ACCRCV') dftactgrp(*no)
Fcustfile  uf   e             disk
Freport    o    e             printer
D custDs         e ds                    extname(custfile)
D today            s                 d   datfmt(*iso)
 /copy invoices
C                   time                        today
C                   read      custfile
C                   dow       not %eof
C                   if        dueDate > today
C                   exsr      sendOvrNtc
C                   read      custfile
C                   write     reportFmt
C/exec sql insert :name, :duedate into
C+              mylib/myfile
C/end-exec
C                   endif
C                   enddo
C                   eval      *inlr = '1'

C     sndOvrNtc     begsr
C                   call      'SNDINVCE'
C                   parm                  custDs
C                   parm      IS_OVERDUE  overdue       10
C                   endsr
```

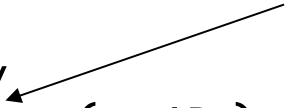
Mixed case

Up to 10 characters for names. "Send overdue notice" = SendOvrNtc

Date/time support

V3R2 – V4R4

```
H bnddir('ACCRCV') dftactgrp(*no)
Fcustfile  uf   e             disk
Freport    o    e             printer
D custDs         e ds                    extname(custfile)
D today            s                 d   datfmt(*iso)
D sendOverdueNotice...
D                 pr
C                   time                        today
C                   read      custfile
C                   dow       not %eof
C                   if        dueDate > today
C                   callp     sendOverdueNotice (custDs)
C                   write     reportFmt
C/exec sql insert :name, :duedate into
C+              mylib/myfile
C/end-exec
C                   endif
C                   read      custfile
C                   enddo
C                   eval      *inlr = '1'

P sendOverdueNotice...
P                 b
  ...
```



Subprocedures

Long names. "Send overdue notice" = SendOverdueNotice

V5R1 – V5R2

```
H bnddir('ACCRCV') dftactgrp(*no)
Fcustfile  uf   e             disk
Freport    o    e             printer
D custDs          ds                  extname(custfile)

D sendOverdueNotice...
D                 pr
 /free
    read custfile custDs;
    dow not %eof;
       if dueDate > %date(); // overdue?
          sendOverdueNotice ();
          write reportFmt;
 /end-free
C/exec sql insert :name, :duedate into
C+             mylib/myfile
C/end-exec
 /free
       endif;
       read custfile custDs;
    enddo;
    *inlr = '1';
 /end-free
...
```

Free form calculations

Indentation!

Many new built-in functions

V5R3 – 7.1

```
H bnddir('ACCRCV') dftactgrp(*no)
Fcustfile  uf   e             disk
Freport    o    e             printer
D custDs        e ds                    extname(custfile)
D sendOverdueNotice...
D                 pr
/free
    read custfile custDs;
    dow not %eof;
       if dueDate > %date(); // overdue?
          sendOverdueNotice ();
          write reportFmt;
          exec sql insert :name, :duedate into
                   mylib/myfile;
       endif;
       read custfile custDs;
    enddo;
    *inlr = '1';
 /end-free

P sendOverdueNotice...
P                 b
 /copy invoices
...
```

Free-form SQL

7.1 TR7 and RDI 9.0.1

```
ctl-opt bnddir('ACCRCV');

dcl-f custfile usage(*update);
dcl-ds custDs likerec(custRec);
dcl-f report printer;

read custfile custDs;
dow not %eof;
   if dueDate > %date(); // overdue?
      sendOverdueNotice ();
      write reportFmt;
      exec sql insert :name, :duedate into
               mylib/myfile;
   endif;
   read custfile custDs;
enddo;
inlr = '1';

dcl-proc sendOverdueNotice;
   /copy invoices
   sendInvoice (custDs : IS_OVERDUE);
end-proc;
```

No /FREE, /END-FREE

All free-form statements

Better colorization options in the editor

What is wrong with fixed-form code?

- Most programmers today have never seen fixed form code
- When they see RPG code like this, it looks like gibberish

```
H bnddir('ACCRCV') dftactgrp(*no)
Fcustfile  if   e             disk
Freports   o    e             printer
```

- Here's what happens when a non-RPG programmer tries to make a change

```
H bnddir('ACCRCV')
Fcustfile  if   e             disk
Freport    o    e            printer
RNF0289E Entry contains data that is not valid; only valid data is
     used.
RNF2013E The Device entry is not PRINTER, DISK, SEQ, WORKSTN or
     SPECIAL; defaults to DISK.
RNF2003E The File Type is not I, O, U, or C; defaults to O if File
     Designation is blank, otherwise to I.
RNF2005E The Sequence entry is not blank, A, or D; defaults to blank.
... more error messages
```

RPG is still not 100% free

There are still some areas where RPG is not yet free

- **Free-form code is still restricted to columns 8 – 80**
- **I specs and O specs must still be coded in fixed-form**
  - **I and O specs are considered deprecated by many RPG programmers in favor of externally-described files**
- **Code related to the RPG cycle must be coded in fixed-form**
  - **The cycle is considered deprecated by many RPG programmers in favor of using SQL for scenarios where the cycle formerly shone**

What does an all-free RPG mean?

- Fewer "secret codes" to remember ("E in column 19 means externally-described")
- Indented code is more maintainable
- Better token-colorization in the RDI editor, allowing programmers to have the same look-and-feel for RPG code as for other languages like Java or PHP
- New programmers will only have to learn how to use RPG, without having to struggle with how it is coded

Removal of many frustrations

- /FREE and /END-FREE in every procedure
- Two lines for many definitions in fixed-form

```
D getNextCustomer...
D                 pr
```

vs

```
dcl-pr getNextCustomer;
```

- Insufficient room in D-spec keywords for long strings

```
D HSSFCellStyle   c
D                                 'org.apache.poi.hssf.-
D                                  usermodel.HSSFCellStyle'
```

vs

```
dcl-c HSSFCellStyle 'org.apache.poi.hssf.usermodel.HSSFCellStyle';
```

More information

Documentation

- There is a new PDF in the 7.1 Info Center with full documentation for the new free-form syntax
  - http://pic.dhe.ibm.com/infocenter/iseries/v7r1m0/topic/books/sc092508a.pdf
  - In the PDF, start at "What's New Since 7.1" in the "What's New" section

RPG Café wiki page https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/We13116a562db_467e_bcd4_882013aec57a

Conversion

- RDI will not do any conversion from H F D P to free-form
- ARCAD announced a free-form conversion at the same time as TR7
- Linoma has a version that supports conversion of H, F, D and P specs

Let's look at the details

- General features
- Control (H)
- File declaration (F)
- Data declaration (D)
- Procedure (P)

Some general features

The new statements all

- **Start with an "opcode"**
- **End with a semicolon**

Just like calculation statements in RPG:

```
if duedate > today;
   sendAngryLetter (customer);
endif;
```

Some general features

Unlike free-form calculations, can have /IF, /ELSEIF, /ELSE, /ENDIF within a statement

```
dcl-s salary
  /if defined(large_vals)
     packed(13 : 3)
  /else
     packed(7 : 3)
  /endif
     ;
```

Some general features

Can mix fixed-form and free-form without /FREE and /END-FREE

Example: Defining the TAG for SQL "whenever"

```
     exec sql whenever sqlerror goto err;
     ...
     return;
C         err           tag
     ok = *off;
     reportSqlError ();
```

Control statements

CTL-OPT (Control Option) statement

- Start with CTL-OPT
- Zero or more keywords
- End with semicolon

```
ctl-opt option(*srcstmt : *nodebugio)
        dftactgrp(*no);
```

Control statements

- Can have multiple CTL-OPT statements
- The rules about not repeating keywords apply across all statements

```
   ctl-opt; // no keywords
   ctl-opt option(*srcstmt : *nodebugio)
           dftactgrp(*no); // two keywords
 H datfmt(*iso) text('My Program')
   ctl-opt alwnull(*usrctl); // free again
```

Control statements

One little enhancement for free-form H:

- **If there is at least one free-form control statement, you don't need DFTACTGRP(*NO) if you have one of the ACTGRP, BNDDIR, or STGMDL keywords**

File statements

DCL-F (Declare file) statement

- Start with DCL-F
- File name
- Keywords
- End with semicolon

File statements

- Only full-procedural and output – no cycle, RAF or table files
- The name can be longer than 10 as long as there's an EXTFILE keyword (and an EXTDESC keyword if externally-described)

```
dcl-f year_end_report printer
          oflind(overflow)
          extdesc('YERPT')
          extfile(*extdesc);
```

File statements – the device

Device keyword or LIKEFILE must be the first keyword

DISK, PRINTER, SEQ, SPECIAL, WORKSTN

- **Defaults to DISK**

Externally-described: *EXT (default)

Program-described: record-length

```
dcl-f orders; // defaults to DISK(*EXT)
dcl-f qprint printer(132);
dcl-f screen workstn; // defaults to *EXT
```

File statements – the usage

USAGE keyword

***INPUT, *OUTPUT, *UPDATE, *DELETE**

Equivalent of fixed-form File Type (I, O, U, C) and File-Addition

Default for USAGE depends on the device

```
dcl-f orders disk;      // *INPUT
dcl-f report printer;   // *OUTPUT
dcl-f screens workstn; // *INPUT : *OUTPUT
```

- **SEQ and SPECIAL default to USAGE(*INPUT)**

File statements – the usage

Some usage values imply other values

- *UPDATE implies *INPUT
- *DELETE implies *UPDATE and *INPUT

```
// USAGE(*INPUT : *UPDATE)
dcl-f orders disk usage(*update);

// USAGE(*INPUT : *UPDATE : *DELETE)
dcl-f arrears disk usage(*delete);
```

Can specify implied values explicitly too

```
dcl-f orders disk usage(*update : *input);
```

File statements – the usage

If you specify the USAGE keyword, the defaults are not considered

```
// output only
dcl-f f1 disk usage(*output);

// input and output
dcl-f f2 disk usage(*input : *output);
```

File statements – difference for *DELETE

In fixed form, U enables update and delete

In free form, *UPDATE does not enable delete

- *DELETE must be coded explicitly

File statements – Keyed files

For externally-described files, KEYED keyword

```
dcl-f orders disk keyed;
```

For program-described files, KEYED(*CHAR:len)

```
dcl-f generic disk(2000) keyed(*CHAR:100);
```

File statements – Program-described keyed files

Only character keys supported for program-described

For other types, use a data structure

```
dcl-f generic disk(2000) keyed(*CHAR:7);

dcl-ds key len(7) qualified;
   item_num packed(12);
end-ds;

key.item_num = 14;
chain key generic;
```

File statements

F specs can be mixed with D specs (even in fixed form)

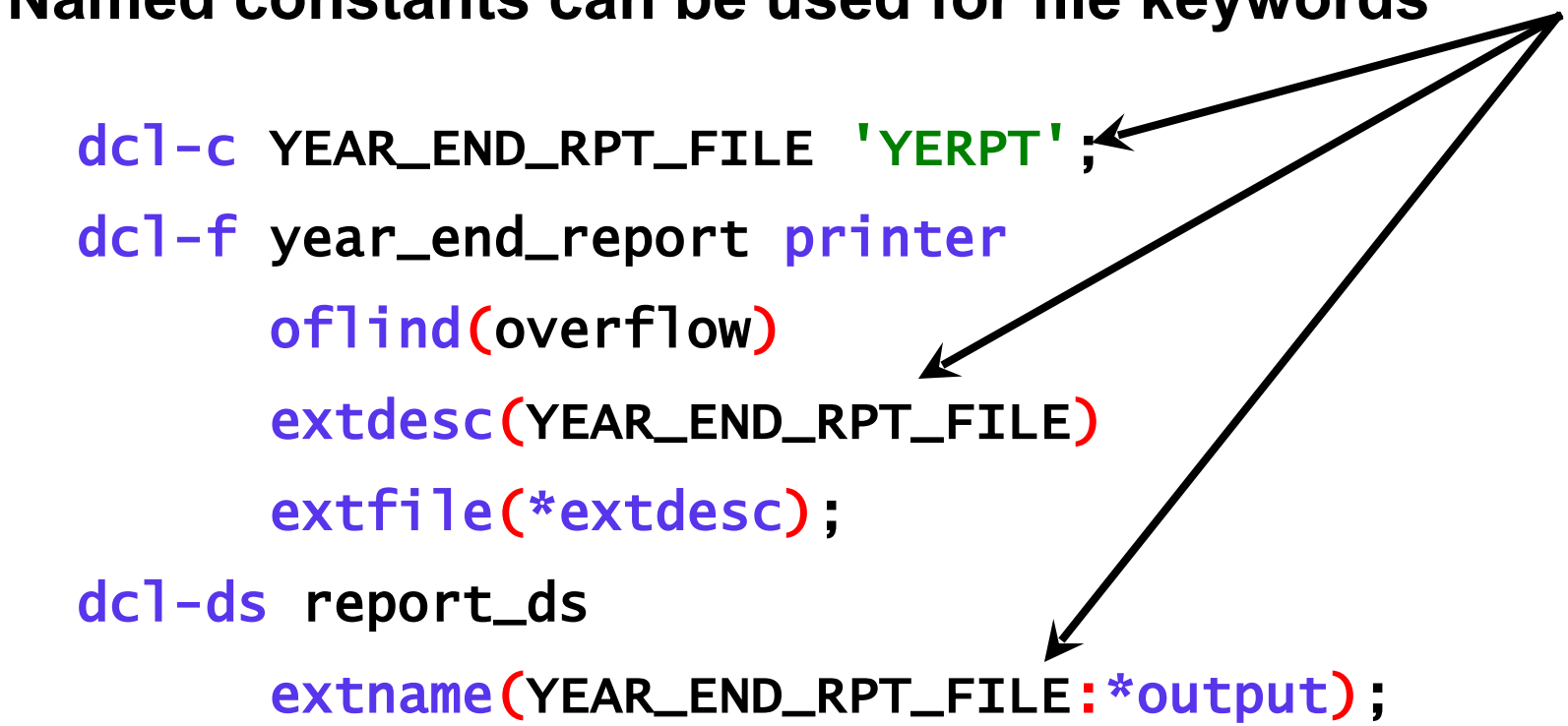
Group related items together

```
dcl-f orders
      usage (*update : *output) keyed;
dcl-ds orders_dsi
      likerec (ordersR:*input);
dcl-ds orders_dso
      likerec (ordersR:*output);
dcl-s num_orders int(10);
```

```
dcl-f report printer;
dcl-ds report_ds
      likerec (reportR:*output);
```

File statements

Named constants can be used for file keywords



```
dcl-c YEAR_END_RPT_FILE 'YERPT';
dcl-f year_end_report printer
      oflind(overflow)
      extdesc(YEAR_END_RPT_FILE)
      extfile(*extdesc);
dcl-ds report_ds
      extname(YEAR_END_RPT_FILE:*output);
```

Data definition statements

- Start with DCL-x
- Item name – can be *N if not named
- Keywords
- End with semicolon

```
dcl-s name like(other_name);
```

Standalone fields

The first keyword must be a data-type keyword.

```
dcl-s salary packed(9:2) inz(0);
```

If you are using the LIKE keyword, it doesn't have to be first.

```
dcl-s annual_salary inz(0)
            like(salary : +2);
```

Data-type keywords

Some data-type keywords match the Data-Type entry exactly

```
CHAR, INT, POINTER ...
```

Some merge the Data-Type entry with another keyword

```
VARCHAR = A + VARYING
DATE = D + DATFMT
OBJECT = O + CLASS
```

Data-type keywords

String data types

Fixed length	CHAR(characters) GRAPH(characters) UCS2(characters)
Varying length	VARCHAR(characters) VARGRAPH(characters) VARUCS2(characters)
Varying length with specific prefix-size	VARCHAR(characters : 4) VARGRAPH(characters : 4) VARUCS2(characters : 4)
Indicator	IND

Data-type keywords

Numeric data types

("BINDEC" is explained on the next slide)

Decimal types with default zero decimal positions	PACKED(digits) ZONED(digits) BINDEC(digits)
Decimal types with specific decimal positions	PACKED(digits : decimals) ZONED(digits : decimals) BINDEC(digits : decimals)
Integer, Unsigned	INT(digits) 3,5,10,20 UNS(digits) 3,5,10,20
Float	FLOAT(bytes) 4,8

BINDEC keyword – reduce confusion over RPG's "binary" type

RPG's "binary" type is a decimal type stored in binary form, not a "true binary".

```
D binfld          S                   9B 3
```

- **Values between -999999.999 and 999999.999**

RPG programmers see "binary" in API documention and think they should code B in their RPG programs

Non-RPG programmers see "binary" as the RPG data type, and think it means true binary

- **When they want an 4 byte binary, they code 4B which is a 2-byte binary with 4 digits**

Other data types

Date, time, timestamp	DATE TIME TIMESTAMP
Date, time with format	DATE(*YMD-) TIME(*HMS-)
Pointer	POINTER
Procedure pointer	POINTER(*PROC)
Object	OBJECT(*JAVA:class) (parameters not needed for a constructor prototype)

Tip for remembering the data-type keywords

If there is a related built-in function, the data-type keyword has the same name:

%CHAR	- CHAR and VARCHAR
%GRAPH	- GRAPH and VARGRAPH
%UCS2	- UCS2 and VARUCS2
%DATE	- DATE
%TIME	- TIME
%TIMESTAMP	- TIMESTAMP
%INT	- INT
%UNS	- UNS
%FLOAT	- FLOAT

Exception: %DEC. The decimal data types are PACKED, ZONED, BINDEC.

Data structures

Data-structures end the subfield list with END-DS

- not used for LIKEDS or LIKEREC data structures

END-DS is optionally followed by the DS name

```
dcl-ds info;
   name varchar(25);
   price packed(4 : 2);
end-ds info;
```

If no subfields, code END-DS on the DCL-DS line

```
dcl-ds prt_ds len(132) end-ds;
```

Data structures

END-DS is not used if LIKEREC or LIKEDS is used

(because you can't code additional subfields)

```
dcl-ds info likeds(info_t);
dcl-ds custInDs likerec(custrec : *input);
```

END-DS is needed for an externally-described DS

```
dcl-ds custDs extname('CUSTFILE') end-ds;
```

Prototypes and procedure interfaces

Prototypes and procedure interfaces are similar

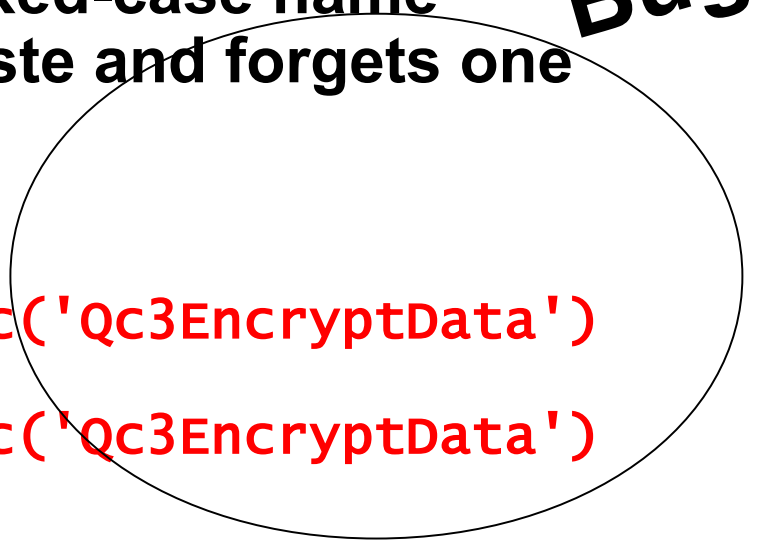
```
dcl-pr qcmdexc extpgm;
   cmd char(3000);
   cmd_len packed(15 : 5);
end-pr;

dcl-pr init end-pr; // no parameters

dcl-pr init;
end-pr;       // can be a separate
   statement

dcl-pi *n varchar(25); // name not needed
   id int(10);
end-pi;
```

Bonus feature: EXTPGM parameter is optional

*DCLCASE for external procedure names

A common bug:

- **EXTPROC is needed for the mixed-case name**
- **The programmer uses copy-paste and forgets one change**




```
D Qc3EncryptData...
D                 pr     extproc('Qc3EncryptData')
D Qc3DecryptData...
D                 pr     extproc('Qc3EncryptData')
```

Use *DCLCASE to avoid retyping the name:

```
dcl-pr Qc3EncryptData extproc(*dclcase);
dcl-pr Qc3DecryptData extproc(*dclcase);
```

- **Less error prone when coding**
- **Easier for code reviewers to see that it's correct**

Subfields

Subfields officially start with the DCL-SUBF opcode

The opcode is optional unless the name is the same as a free-form opcode

```
dcl-ds info;
   name char(25);
   dcl-subf select int(10);
end-ds info;
```

DCL-SUBF must be used because "select" is an opcode supported in free-form

Same as the rule for EVAL and CALLP

```
name = 'Sally';
eval select = 5;
```

Subfields

The POS keyword replaces

- From-and-to positions
- OVERLAY(dsname)

```
D info     DS
D  sub1        25    34A
D  sub2                D     OVERLAY(info:100)
D  sub3                5P 2  OVERLAY(info)
```

```
dcl-ds info;
   sub1 char(10) pos(25);
   sub2 date pos(100);
   sub3 packed(5 : 2) pos(1);
end-ds info;
```

Subfields

Free-form OVERLAY only overlays subfields

- No free-form equivalent for OVERLAY(ds:*NEXT)
- OVERLAY(ds:*NEXT) means "after all previous subfields" which is the same as not having the OVERLAY keyword at all
- SUB3 starts at position 101, after all previous subfields.

```
D info     DS
D  sub1       1  100A
D  sub2      11   20A
D  sub3            5A   OVERLAY(info:*next)
```

Equivalent:

```
dcl-ds info;
   sub1 char(100) pos(1);   // 1-100
   sub2 char(10) pos(11);   // 11-20
   sub3 char(5);            // 101-105
```

Parameters

Parameters officially start with DCL-PARM

DCL-PARM is optional. Same rule as for subfields

```
dcl-pr proc;
   name char(25) const;
   dcl-parm clear ind value;
end-pr;
```

Can use named constants for keywords

```
dcl-c SYS_NAME_LEN 10;

dcl-ds sys_obj qualified;
   obj char(SYS_NAME_LEN);
   lib char(SYS_NAME_LEN);
end-ds;
```

Can use named constants for keywords

In fixed form, some keywords allow literals to be specified without quotes: DTAARA, EXTNAME, EXTFLD

What data area is used for fld1?

```
D fld1        S        10A   DTAARA(dta1)
```

What about fld2?

```
D dta2        C                'MYLIB/DTAARA2'
D fld2        S        10A   DTAARA(dta2)
```

DTAARA keyword change

In free-form, an unquoted name is always a variable or named constant

```
D dta1      C                   'MYLIB/DTAARA1'
```

```
D fld1a     S     10A   DTAARA(dta1)
dcl-s fld1b char(10) dtaara('DTA1');
```

*LIBL/DTA1

```
dcl-s fld1c char(10) dtaara(dta1);
```

MYLIB/DTAARA1

```
D fld2a     S     10A   DTAARA(*VAR:nameFld)
  dcl-s fld2b char(10) dtaara(nameFld);
```

Value of nameFld

Procedure statements

Begin a procedure

- DCL-PROC
- Procedure name
- Keywords
- End with semicolon

```
dcl-proc myProc export;
```

End a procedure

- END-PROC
- Optional procedure name
- End with semicolon

```
end-proc myProc;
```

or

```
end-proc;
```

Procedure example

```
dcl-proc getCurUser export;
   dcl-pi *n char(10) end-pi;

   dcl-s curUser char(10) inz(*user);

   return curUser;
end-proc;
```

- The PI uses the place-holder *N for the name
- END-PI is specified as a keyword at the end of the DCL-PI statement

Gotchas

- Update does not imply delete
- END-DS, END-PR, END-PI needed at the end of a subfield or parameter list (even when there are no subfields or parameters)
- Keywords like DTAARA and EXTNAME that assume unquoted names are named constants or variables

(These have already been discussed)

Another gotcha

If you are in the habit of using ellipsis at the end of D and P spec names

```
P customerName...
P           S                   50A
```

That will not work for free-form declarations

```
dcl-s customerName...
        char(50);
```

The name is customerNamechar, and "(50)" is found where the compiler expects to find the data type.

```
dcl-s customerName
        char(50);
```

Colorization in RDI

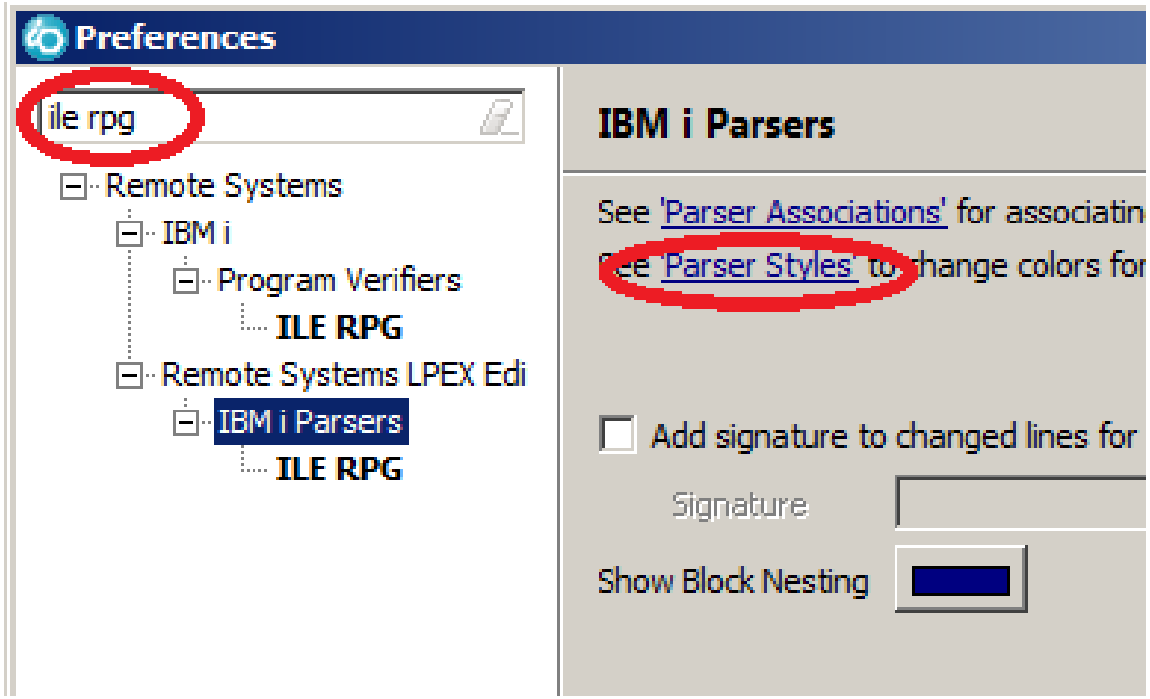
Much more control for colorizing your code

Here is some code with the default colors

```
000101
000102       dcl-f custfile usage(*update);
000103
000104       dcl-ds myDs likerec(custrec : *input);
000105       /if defined(debug)
000106          dcl-s debugMsg varchar(100);
000107       /endif
000108
000109       read custfile myDs;
000110       if myDs.duedate > %date();
000111          handleOverdue (myDs);
000112       endif;
```

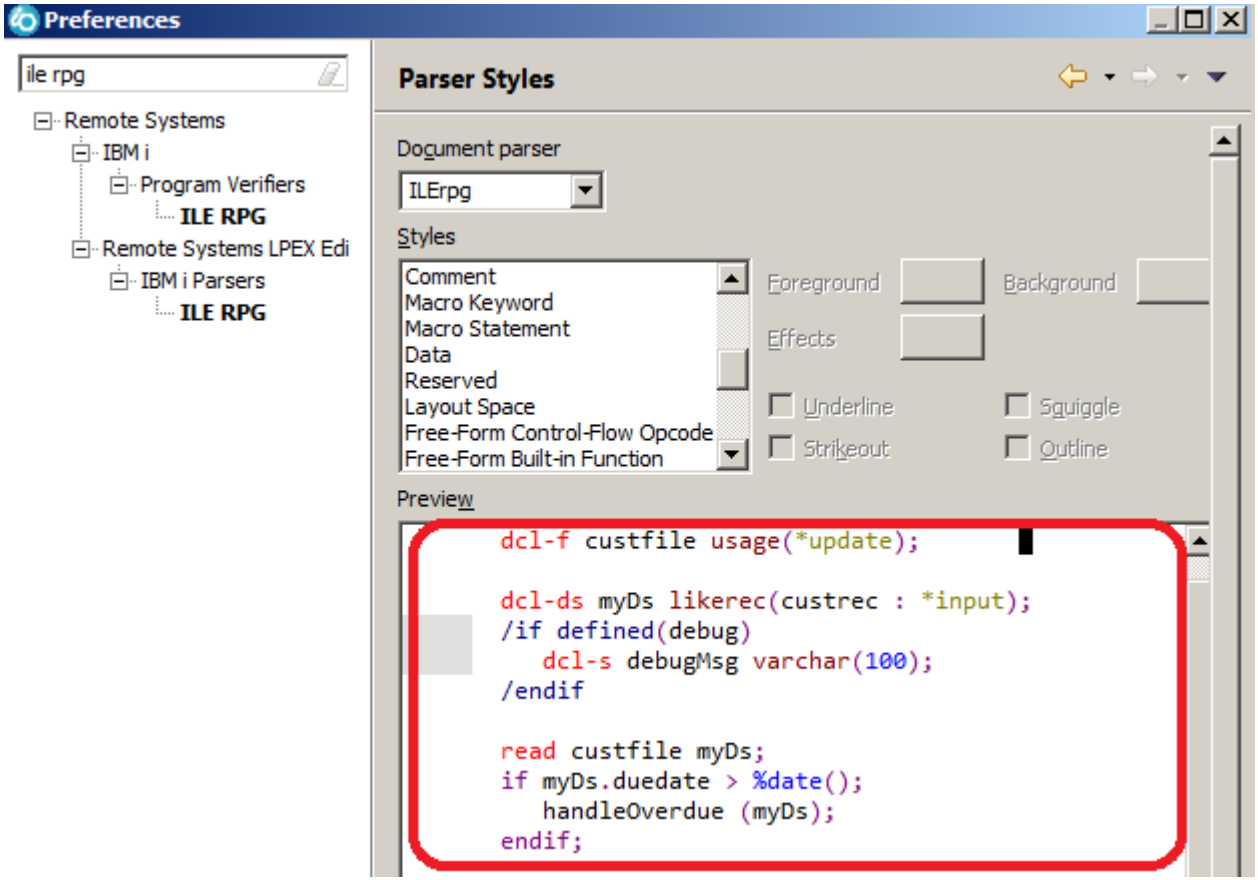
Navigate to the color preferences

- Window > Preferences
- Search for ILE RPG
- Click on Parser Styles



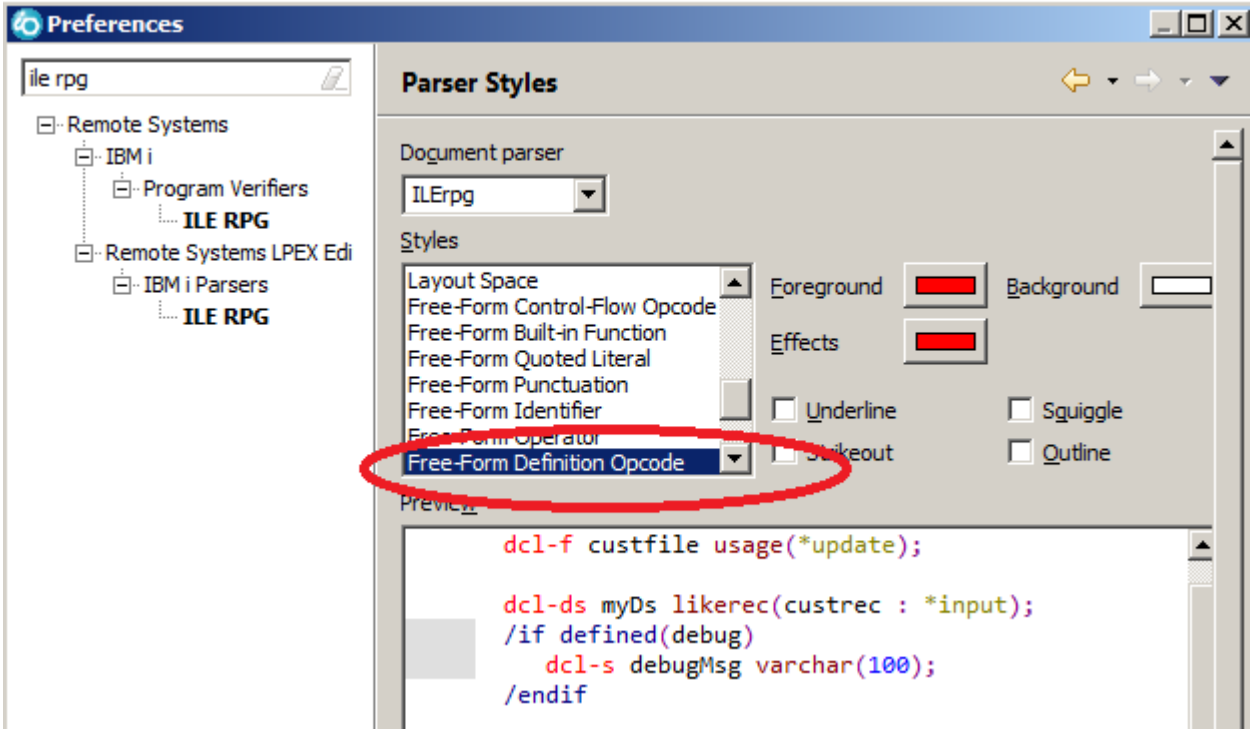
You can change the code to work with

- **In the code section, I like to paste in a bit of my own code at the top**



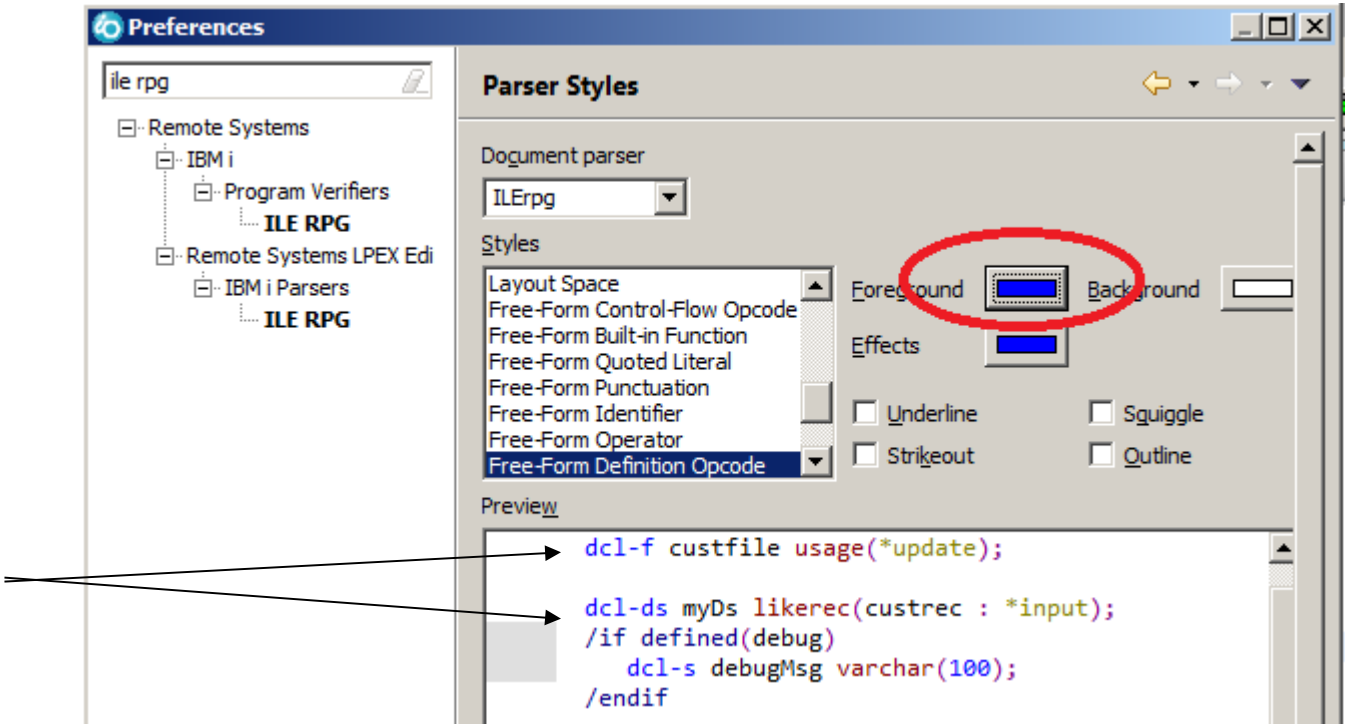
Choose which style you want to change

- Then click on the code you want to change the color for
- The top section will automatically position to the relevant style



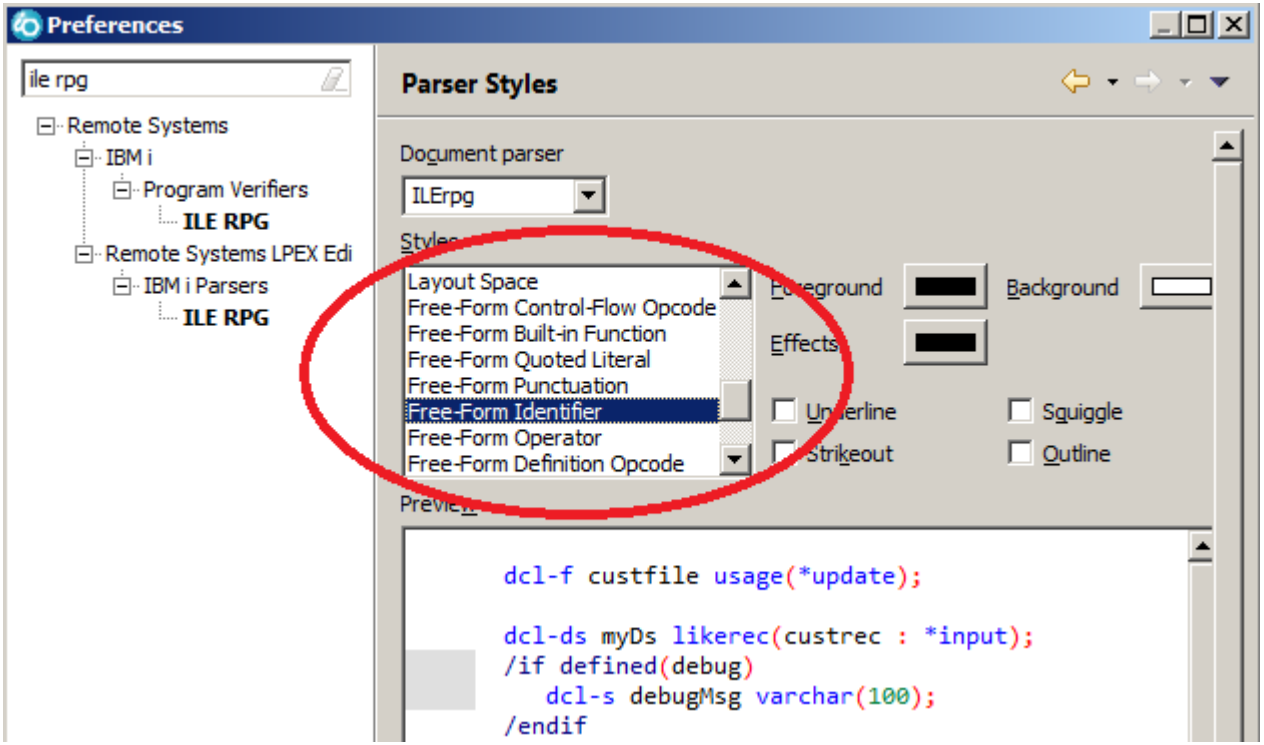
Customize your colors

- Choose the color you want
- It will automatically be colored in the code section so you can see the effect it has



Another way to choose the style

- For most free-form code, the styles are listed together
- You can select them one-by-one, adjusting the colors



Here's how I like it

- **The non-free-form styles I had to change were Operation and Numeric**

```
000101
000102        dcl-f custfile usage(*update);
000103
000104        dcl-ds myDs likerec(custrec : *input);
000105        /if defined(debug)
000106           dcl-s debugMsg varchar(100);
000107        /endif
000108
000109        read custfile myDs;
000110        if myDs.duedate > %date();
000111           handleOverdue (myDs);
000112        endif;
```

Summary

We had two goals when designing the new free-form syntax

- **Easy for non-RPG programmers to learn**
- **Easy for existing RPG programmers to learn**

We hope we have accomplished those goals!

www.ibm.com/software/rational

Special notices

This document was developed for IBM offerings in the United States as of the date of publication. IBM may not make these offerings available in other countries, and the information is subject to change without notice. Consult your local IBM business contact for information on the IBM offerings available in your area.

Information in this document concerning non-IBM products was obtained from the suppliers of these products or other public sources. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. Send license inquires, in writing, to IBM Director of Licensing, IBM Corporation, New Castle Drive, Armonk, NY 10504-1785 USA.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

The information contained in this document has not been submitted to any formal IBM test and is provided "AS IS" with no warranties or guarantees either expressed or implied.

All examples cited or described in this document are presented as illustrations of the manner in which some IBM products can be used and the results that may be achieved. Actual environmental costs and performance characteristics will vary depending on individual client configurations and conditions.

IBM Global Financing offerings are provided through IBM Credit Corporation in the United States and other IBM subsidiaries and divisions worldwide to qualified commercial and government clients. Rates are based on a client's credit rating, financing terms, offering type, equipment type and options, and may vary by country. Other restrictions may apply. Rates and offerings are subject to change, extension or withdrawal without notice.

IBM is not responsible for printing errors in this document that result in pricing or information inaccuracies.

All prices shown are IBM's United States suggested list prices and are subject to change without notice; reseller prices may vary.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

Any performance data contained in this document was determined in a controlled environment. Actual results may vary significantly and are dependent on many factors including system hardware configuration and software design and configuration. Some measurements quoted in this document may have been made on development-level systems. There is no guarantee these measurements will be the same on generally-available systems. Some measurements quoted in this document may have been estimated through extrapolation. Users of this document should verify the applicable data for their specific environment.

Revised September 26, 2006

Special notices (cont.)