What's new for RPG in 7.2

Barbara Morris
IBM

Agenda

- Code free-form in any column
- **Free-form H F D and P specs**
- Other 7.2 enhancements (some available as PTFs for 7.1)

Agenda

- Code free-form in any column
- Free-form H F D and P specs
- Other 7.2 enhancements (some available as PTFs for 7.1)

Fully free-form RPG

7.1 and 7.2 PTFs

Soon* it will be possible to code free-form RPG starting in column 1 and going to the end of the line.

* With 7.1 TR11 and 7.2 TR3, coming very soon.

There is no practical limit on the length of a source line.

- CRTSRCPF has a limit of 32766
- IFS files have no limit

Fully free-form RPG – how long should your lines be?

Various style-guides for other languages recommend a maximum line length of 80, 132, 120 etc.

The “80” comes from IBM punch cards.

Google [maximum length of a code line] to see some of discussions about line length.

If you create your RPG source files with RCDLEN(112), then that gives you 100 characters, which is probably ideal.

Fully free-form RPG – source must start with **FREE

Any source member that contains fully-free code must have **FREE in column 1 of the first line of the source.

```
**FREE
ctl-opt main(greeting);

dcl-proc greeting;
   dsply ‘Hello’;
end-proc;
```

Fully free-form RPG

- All code in a **FREE source member must be free-form. If you need any fixed-form code, you can put it in a /COPY file
- Source lines must not begin with ** unless they are the special directives for compile-time data, file-translation, or alternate collating sequence.
- /FREE and /END-FREE are not allowed in a **FREE source member

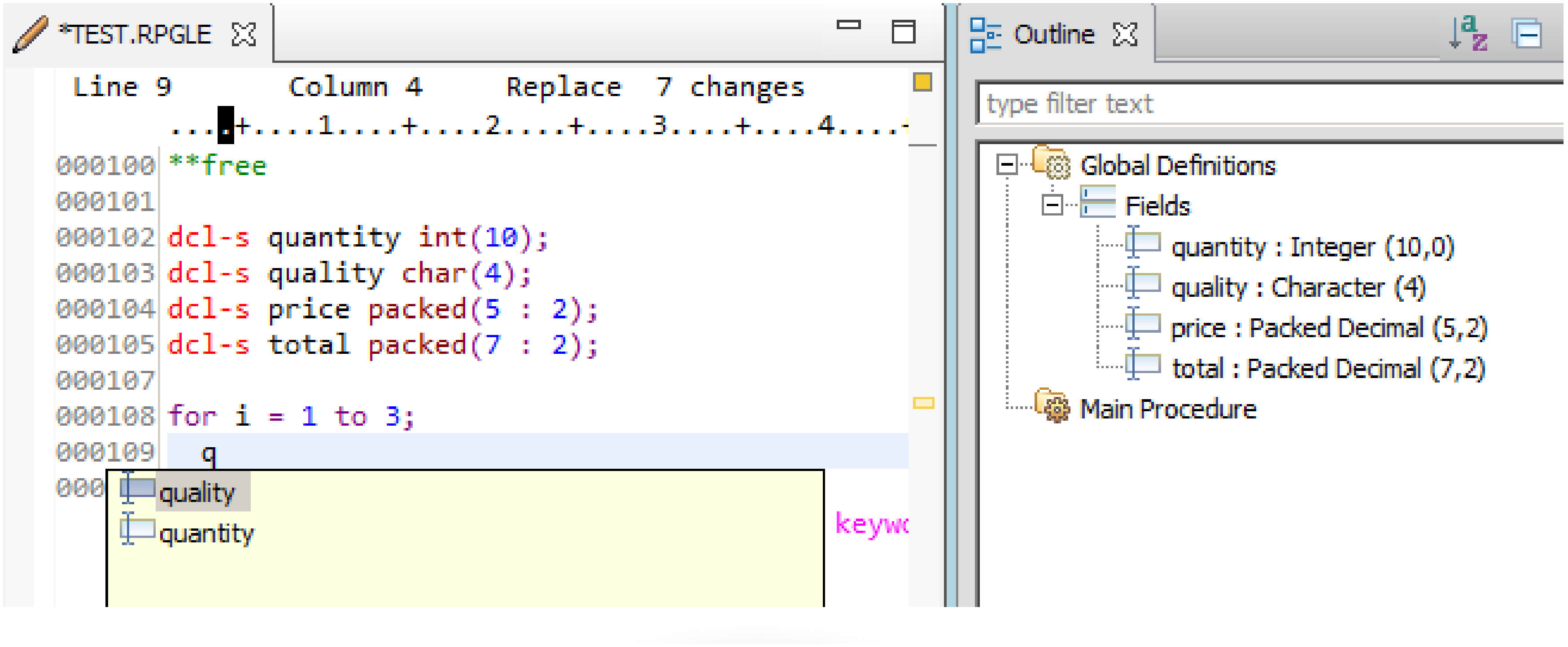
Fully free-form RPG – copy files

- Each copy file has its own source mode

- A copy file is always assumed to have column-limited source mode unless it has **FREE in line 1

Fully free-form RPG – RDI

RDI V9.5 already supports fully-free RPG code



```
Line 9        Column 4      Replace   7 changes
...+....1....+....2....+....3....+....4....+
**free

dcl-s quantity int(10);
dcl-s quality char(4);
dcl-s price packed(5 : 2);
dcl-s total packed(7 : 2);

for i = 1 to 3;
  q
```

Fully free-form RPG – Embedded SQL

The SQL precompiler will support fully-free RPG code at the same time as the RPG compiler

```
**FREE
dcl-s greeting char(10);

exec sql set :greeting = 'Hello';
dsply greeting;
return;
```

7.1 Free-form H F D P

7.1 PTF

```
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

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
- Some code still has to use fixed-form specs
- I specs and O specs

•I and O specs are considered deprecated by many RPG programmers in favor of externally-described files

- Code related to the RPG cycle

•Cycle files (primary, secondary, RAF, table)
•The cycle is considered deprecated by many RPG programmers in favor of using SQL for scenarios where the cycle formerly shone

Some general features

Can mix fixed-form and free-form

- Defining the TAG for SQL "whenever"

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

- Renaming fields on I specs

```
  dcl-f custfile disk usage(*update);
Icustrec       01
I              XYZ                 NEWNAME
   read custrec;
```

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);
```

Fixed form equivalent:

```
H option(*srcstmt : *nodebugio)
H dftactgrp(*no)
```

File statements

DCL-F (Declare file) statement

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

File statements

The file name can be longer than 10 in free form

```
    dcl-f new_customers
                   extdesc('NEWCST')
                   extfile(*extdesc);

    read new_customers;
```

Fixed form equivalent:

```
Fnewcustmr IF    e     disk    extdesc('NEWCST')
F                              extfile(*extdesc)

    read newcustmr;
```

File statements – keywords have useful defaults

- Most common device type is DISK
- Most common is externally-described
- The most common usage depends on the device

```
dcl-f orders;
dcl-f report printer;
dcl-f screens workstn;
```

The two sets of definitions mean the same thing.

```
dcl-f orders disk(*ext) usage(*input);
dcl-f report printer(*ext) usage(*output);
dcl-f screens workstn(*ext) usage(*input:*output);
```

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);
```

Data definition statements

- Start with DCL-x
- Item name – can be *N if not named
- Data type keyword for fields and parameters
  CHAR, VARCHAR, INT, DATE, POINTER etc
- Other keywords
- End with semicolon

```
dcl-s full_name char(10); // standalone
dcl-c MAX_ELEMS 1000;     // constant
dcl-ds info qualified;    // data structure
   name varchar (30);     // subfield
   salary packed(7 : 2);
end-ds info;
```

Procedure

- Start with DCL-PROC
- End with END-PROC

```
dcl-proc getNextOrder export;
   dcl-pi *n ind;
      orders likefile(orderFile_t);
      data likerec(orderFile_t.ord);
   end-pi;

   read orders data;
   return %eof();
end-proc;
```

Can use named constants for keywords

Named constants for keywords

- reduce hard-coding
- makes code more self-explanatory

```
dcl-c SYS_NAME_LEN 10;
dcl-ds sys_obj qualified;
   obj char(SYS_NAME_LEN);
   lib char(SYS_NAME_LEN);
end-ds;

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

*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')
```

Bug!

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

Agenda

- Code free-form in any column
- Free-form H F D and P specs

- **7.2 enhancements (some available as 7.1 PTFs)**
  - Extended ALIAS support
  - Easier to use data structures for I/O
  - Support for UTF-8 and other alpha CCSIDs
  - Avoid unnecessary CCSID conversions for database file I/O
  - Date/Time/Timestamp enhancements
  - Improved PCML generation

Support for ALIAS names

7.1 and 7.2 PTFs

Background

- Fields in externally described files can have a standard name up to 10 characters and an alternate (ALIAS) name up to 128 characters.
- RPG III only allowed 6 characters, so many databases have files with cryptic names like CUSNAM, CUSADR. The files often have alternate names such as CUSTOMER_NAME and CUSTOMER_ADDRESS, that can be used in SQL queries.
- RPG programmers would like to use the alternate names in their RPG programs.

Support for ALIAS names

7.1: New ALIAS keyword for RPG

- When ALIAS is specified, RPG will use the alternate name <u>instead of</u> the 10-character standard name.
- Supported on D specs for any externally-described data structure.
- **New with 7.1 and 7.2 PTFs**: Supported on F specs for all externally-described files
  - Affects program fields and LIKEREC data structures
  - The PTF for 7.1 is SI54502
  - The PTF for 7.2 *CURRENT is SI54155
  - The PTF for 7.2 *PRV is SI54521

7.1 & 7.2 PTF: Support for ALIAS names

```
A    R CUSTREC
A      CUSTNM            25A          ALIAS(CUSTOMER_NAME)
A      CUSTAD            25A          ALIAS(CUSTOMER_ADDRESS)
A      ID                10P 0
```

```
Fmyfile   o      e    DISK        ALIAS

   customer_name = 'John Smith';
   customer_address = '123 Mockingbird Lane';
   id = 12345;
   Write custRec;
```

When there is no alternate name for a field, the short name is used.

7.1 & 7.2 PTF: Support for ALIAS names – one limitation

Limitation:

If you code ALIAS for a file, you can't code I specs or O specs for that file.

The compiler will still generate I and O specs into the listing. If the name is too long to fit in the generated spec, the name will be listed on the next line.

```
12=O                                COMPANY                     6A CHAR
13=O                                *ALIAS                     31A CHAR
          MAILING_ADDRESS
14=O                                STATUS                      2A CHAR
```

7.1 & 7.2 PTF : Easier to use data structures for I/O

7.1 and 7.2 PTFs

The problem:

RPG was very strict about which data structures could be use for I/O:

- For an input operation, it had to be defined with *INPUT (the default for LIKEREC)
- For a WRITE operation, it had to be defined with *OUTPUT
- For an UPDATE operation, it could be defined with either *INPUT or *OUTPUT

7.1 PTF: Easier to use data structures for I/O

Before:

For a file that allowed both READ and WRITE, it could be very awkward to use data structures for I/O to the file.

```
Fmyfile   if   a    e     disk

D inDs          ds              likerec(myfmt : *input)
D outDs         ds              likerec(myfmt : *output)

     read myfmt inDs;
     ...
     eval-corr outDs = inDs;  // copy over the fields
     write outDs;
```

7.1 PTF: Easier to use data structures for I/O

Now, with new 7.1 and 7.2 PTFs

Now, if you use LIKEREC with no type parameter for a DISK record, you can use the data structure for any operation.

```
Fmyfile  if  a   e     disk

D ds             ds              likerec(myfmt)

     read myfmt ds;
     ...
     write ds;
```

(The same PTFs as the ALIAS PTFs)

7.1 PTF: Easier to use data structures for I/O

You can also use a LIKEREC data structure with no type parameter for a PRINTER file.

```
Fmyprtf  o     e    printer
D ds            ds              likerec(myprtfmt)
     Write myprtfmt ds;
```

7.1 PTF: Easier to use data structures for I/O

If the data structure is defined with *ALL (E-DS or LIKEREC), you can use it for any I/O operation.

```
Fdiskf     UF   A    E   DISK
Fprtf      O         E   PRINTER

D diskDs       e ds                extname(diskf : *all)
D prtDs        e ds                extname(prtf : *all)

     read diskfmt diskDs;
     write diskfmt diskDs;
     update diskfmt diskDs;

     write prtfmt prtDs;
```

(*ALL was already supported in 6.1 for WORKSTN files, including subfile formats)

6.1 & 7.1 PTFs: New XML-INTO options: namespace option

```
<emp employee:type=“regular” employee:id=“13573”>
  <standard:name>John Smith</standard:name>
</emp>
```

6.1 and 7.1 PTFs

Problem: This XML document uses “namespaces” to qualify the XML tag names. This causes a problem for XML-INTO because the name “employee:type” cannot match an RPG subfield name.

Solution: The ns (namespace) option

`ns=remove:` remove the namespace part of the name for subfield matching. Matches with subfield “type”.

`ns=merge:` merge the namespace with the rest of the name using underscore. Matches with subfield “employee_type”

6.1 & 7.1 PTFs: New XML-INTO options: ns=remove

```
<emp employee:type="regular" id="13573">
  <standard:name>John Smith</standard:name>
</emp>
```

The RPG code for ns=remove.

```
D emp                   DS             qualified
D   type                      25A
D   id                        10I 0
D   name                      25A

xml-into emp %xml('emp.xml' : 'ns=remove');
// emp.type = 'regular'
// emp.id = 13573
// emp.name = 'John Smith'
```

6.1 & 7.1 PTFs: New XML-INTO options: ns=merge

```
<emp employee:type=“regular” id=“13573”>
  <standard:name>John Smith</standard:name>
</emp>
```

The RPG code for ns=merge.

```
D emp                   DS            qualified
D   employee_type                  25A
D   id                             10I 0
D   standard_name                  25A

xml-into emp %xml('emp.xml' : 'ns=merge');
// emp.employee_type = 'regular'
// emp.id = 13573
// emp.standard_name = 'John Smith'
```

6.1 & 7.1 PTFs: New XML-INTO options: nsprefix

Problem: If the namespace might be different in different XML documents, the ns=remove option must be used. But the RPG programmer may want to know what the namespace was.

Solution: Define subfields to receive the namespace that was removed. nsprefix gives the prefix for the subfield names that will receive the namespace that was removed from the XML tag.

```
<emp>
   <standard:type>manager</standard:type>
</emp>

D emp                   DS            qualified
D   type                        25A
D   ns_type                     25A

xml-into emp %xml('emp.xml'
                : 'ns=remove nsprefix=ns_');
// emp.type = ‘manager‘
// emp.ns_type = 'standard'
```

6.1 & 7.1 PTFs: New XML-INTO options: case=convert

New value for the case option lets you tell XML-INTO how to handle characters in the XML name that can't appear in RPG names

```
<Étudiant Pre-nom="Élise" Âge="12">
  <École>Collège Saint-Merri</École>
</Étudiant>
```

With option case=convert, the tag names are converted before being compared to the subfield names:

– Alphabetic characters like 'Â' are mapped to the matching A-Z (using the job's *LANGIDSHR table).

– Other characters other than 0-9 and underscore are mapped to underscore.

– Then, all underscores are merged to a single underscore.

6.1 & 7.1 PTFs: New XML-INTO options: case=convert

```
<Étudiant Pre-nom="Élise" Âge="12">
  <École>Collège Saint-Merri</École>
</Étudiant>


D etudiant     ds                   qualified
D  age                        3p 0
D  pre_nom                   25a   varying
D  ecole                     50a   varying

   xml-into etudiant %xml('info.xml'
                          : 'case=convert');
   // etudiant.age = 12
   // etudiant.pre_nom = 'Élise'
   // etudiant.ecole = 'Collège Saint-Merri'
```

6.1 & 7.1 PTFs: Warnings or exceptions for CCSID conversions

6.1 and 7.1 PTFs

Sometimes a CCSID conversion will result in a "substitution" character being placed in the result.

Unicode source data:

```
The Thai word for "house" is "บ้าน".
```

The target is an alphanumeric variable with CCSID 37:

```
The Thai word for "house" is "■■■".
```

CCSID 37 uses the "Latin" character set, and there are no matching characters for the Thai characters that are in the Unicode variable. Substitution characters are placed in the alphanumeric result.

The original Thai characters are all converted to the same substitution characters, so their value is lost.

6.1 & 7.1 PTFs: Warnings or exceptions for CCSID conversions

Non-error RPG status code 50 is set when the conversion has to use substitution characters.

You have to add code to check whether %status = 50

```
alphaText = unicodeText;
if %status() = 50;
   ... there was loss of data
```

Two problems:

- It's too awkward to check for status code 50 after every statement with a CCSID conversion

- It's not always easy to tell which statements have CCSID conversions

6.1 & 7.1 PTFs: Get an exception when substitution occurs

CCSIDCVT(*EXCP)

Code new H spec keyword CCSIDCVT(*EXCP) to get an exception when a CCSID conversion results in a substitution character.

- New status code 00452

In 6.1 and 7.1, you will need to add messages RNX0452 and RNQ0452 to your message file. The cover letter of the PTF for the RPG runtime has CLP code for adding the messages.

6.1 & 7.1 PTFs: Get an list of CCSID conversions

CCSIDCVT(*LIST)

Code new H spec keyword CCSIDCVT(*LIST) to get a list of all the CCSID conversions in the module.

For each conversion, it shows

- The source statements using that conversion
- Whether the conversion might result in substitution characters

If you want both options, code CCSIDCVT(*EXCP:*LIST) or CCSIDCVT(*LIST:*EXCP)

6.1 & 7.1 PTFs: Sample CCSIDCVT summary

```
                  C C S I D   C o n v e r s i o n s
           From CCSID     To CCSID       References
RNF7361    834            *JOBRUN         15      25
RNF7357    1200           *JOBRUN         27     921     1073
           *JOBRUN        1200            28      12      321      426
                                         552     631
RNF7359    835            834             41     302      302
RNF7360    *JOBRUN        834            242     304      305
 * * * *   E N D   O F    C C S I D   C O N V E R S I O N S    * * * *
```

- RNF7357 Conversion from UCS-2 to Alpha might not convert all data.
- RNF7358 Conversion from UCS-2 to DBCS might not convert all data.
- RNF7359 Conversion from DBCS to DBCS might not convert all data.
- RNF7360 Conversion from Alpha to DBCS might not convert all data.
- RNF7361 Conversion from DBCS to Alpha might not convert all data.

6.1 & 7.1 PTFs: How to use the CCSIDCVT summary

You can use this information for two purposes:

- You can improve performance: Reduce the number of conversions by changing the data types of some of your variables.
- You can improve the reliability of your program by eliminating some of the conversions that have the potential to result in substitution characters. For example, if you have conversion from UCS-2 to an alphanumeric variable, and that alphanumeric data is later converted back to UCS-2, you may be able to change the type of the alphanumeric variable to UCS-2, to avoid the potential data loss.

7.2 Support for other CCSIDs for character data

Starting in 7.2, you can code the CCSID keyword for character fields:

- All EBCDIC CCSIDs
- ASCII CCSIDs
- The Unicode CCSID UTF-8 (1208, or *UTF8)

7.2 Support for other CCSIDs for character data

Assume that the getData procedure returns UTF-8 data (CCSID 1208).

Prior to 7.2, you would call an API to convert the data to UCS-2:

```
D stringA      S          10000A
D stringC      S          10000C   VARYING
D getData      PR         10000A   VARYING

     stringA = getData ();
     stringC = convert(stringA: %len(stringA): 1208: 13488);
```

Remember not to use the data for ordinary RPG statements because RPG thinks the data is in the job CCSID.

```
     if stringA = *blanks;   // BUG!
      ...
```

7.2 Support for other CCSIDs for character data

In 7.2, you can say that the data is UTF-8.

```
D stringA      S          10000A  CCSID(*UTF8)
D getData      PR         10000A  VARYING CCSID(*UTF8)

   stringA = getData ();
```

You can use the data in ordinary RPG statements because RPG knows that it is UTF-8 data.

```
   if stringA = *blanks;   // OK!
    ...
```

7.2 Avoid CCSID conversions for database files

By default, for a database file

- When you read a record, database converts the alphanumeric data from the field CCSID to the job CCSID
- When you write or update a record, database converts the alphanumeric data from the job CCSID to the field CCSID

Use DATA(*NOCVT) for a file to open the file so these conversions do not happen at the database level. Any CCSID conversions will be performed in the RPG program if necessary.

Use H spec OPENOPT(*NOCVTDATA) to default this behaviour for all database files.

7.2 More support for CCSIDs for character data

There are several other CCSID-related enhancements in 7.2 for RPG, such as defining a data structure to pick up the CCSIDs from the file.

I won't discuss them here, but if you are interested in CCSIDs you can read about them in the What's New section of the ILE RPG Reference.

http://www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzasd/rpgrelv7r2.htm

7.2 Control the length for %SUBDT

By default, %SUBDT returns an integer value. This makes it inconvenient if you want to edit it into a string.

```
s = %editc(%subdt(t : *hours) : 'X')
     + ':'
     + %editc(%subdt(t : *minutes) : 'X');

// s = '0000000013:0000000047'
```

Use the third parameter of %SUBDT to specify the number of digits

```
s = %editc(%subdt(t : *hours : 2) : 'X')
    + ':'
    + %editc(%subdt(t : *minutes : 2) : 'X');

// s = '13:47'
```

7.2 Timestamps with 0 – 12 fractional seconds

Starting in 7.2, timestamps can have zero to twelve fractional seconds.

They still default to 6 (microseconds).

RPG still only sets milliseconds with %TIMESTAMP.

Free-form syntax:

```
    dcl-s datetime timestamp(0);
    dcl-s max timestamp(12);
```

Fixed-form syntax:

```
   D datetime        s              z 0
   D max             S              Z12
```

Power Systems

7.2 Handle seconds and microseconds together

You can get the difference between timestamps as a number of seconds with decimal places by specifying the number of decimal places you want.

```
elapsed = %diff(finish : start : *SECONDS : 3);

// elapsed = 1922.483
```

You can add or subtract seconds and microseconds together with %SECONDS by specifying a value with decimals

```
start = %timestamp();
target = start + %seconds(1.5);

// start  = '2014-01-15-14.25.03.123000'
// target = '2014-01-15-14.25.04.623000'
```

7.1 & 72 PTF: PCML with mixed-case names

7.1 and 7.2 PTFs

By default, the RPG compiler generates PCML with the names in uppercase.

An RPG procedure:

```
dcl-proc placeOrder export;
   dcl-pi *n;
      qty packed(15) const;
      itemName car(30) const; ...
```

The generated PCML:

```
<pcml version="4.0">
   <program name=“PLACEORDER" entrypoint=“PLACEORDER">
      <data name=“QTY" type=“packed" length="15” ... />
      <data name=“ITEMNAME" type="char" length=“30" ... /> ...
```

Anything using the PCML must also use the uppercase names:

```
pcd.setValue (“PLACEORDER.QTY", … );
pcd.setValue (“PLACEORDER.ITEMNAME”, … );
pcd.callProgram (“PLACEORDER”);
```

7.1 & 72 PTF: PCML with mixed-case names

New: Specify PGMINFO(*DCLCASE) in the H spec to have the PCML generated with the same case as the RPG source

The generated PCML:

```
    <pcml version="4.0">
       <program name="placeOrder" entrypoint="PLACEORDER">
          <data name="qty" type="packed" length="15" ... />
          <data name="itemName" type="char" length="30" ... /> ...
```

The Java code using the PCML can use the same mixed-case names:

```
     pcd.setValue ("placeOrder.qty ", … );
     pcd.setValue ("placeOrder.itemName", … );
     pcd.callProgram ("placeOrder);
```

7.1 & 72 PTF: More granular PCML

7.1 and 7.2 PTFs

By default, the RPG compiler generates PCML for all exported procedures.

Some procedures have parameter or return value types that make it impossible to generate PCML.

This causes the compile to fail.

7.1 & 72 PTF: More granular PCML

New: P-spec keyword PGMINFO(*YES | *NO)

Either

- Specify PGMINFO(*YES) for all the procedures that should have PCML generated

Or

- Specify PGMINFO(*NO) for all the procedures that should not have PCML generated

These PGMINFO keywords are ignored if PCML is not being generated.

Where to find out about PTF enhancements for RPG

- ❑ Subscribe to the blog in the RPG Cafe
  - Whenever we provide an enhancement through PTFs, we post a blog entry in the Cafe blog
  - The blog entry will usually point to a new page in the Cafe wiki

- ❑ Regularly check the “welcome” page in the RPG Cafe wiki. There is a section for “Enhancements delivered through PTFs”, and the “Announcement” section at the top will have information about the most recent enhancement.

- ❑ Regularly check the “What’s New Since …” section in the ILE RPG Reference
  - Starting in 7.2, the ILE RPG manuals are updated with enhancements delivered through PTFs. If there are PTFs for earlier releases, the 7.2 documentation for that feature applies to earlier releases too.

Go to IBM

Special notices

Special notices