What's New and Exciting in RPG

(AKA Scott Goes Crazy.)

Presented by

Scott Klement

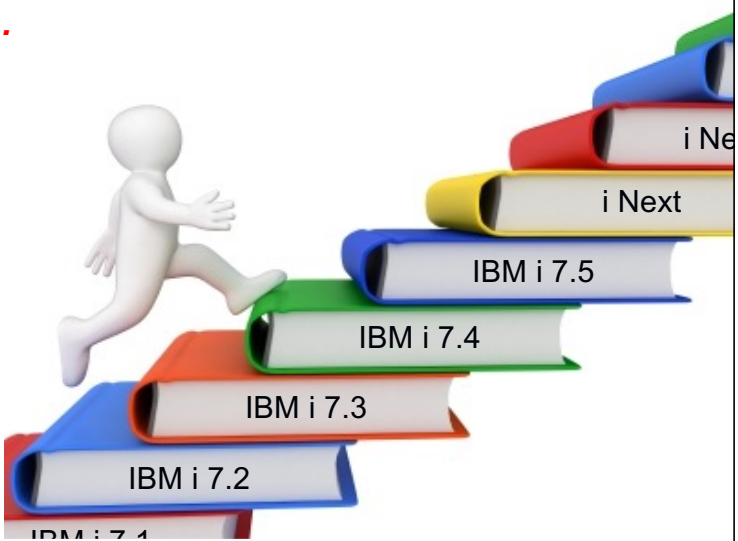
http://www.scottklement.com

There's a band called 1023 MB.
They haven't had any gigs yet.

This Session Is Organized by Feature

This session is organized by *date…*

…not by release.



A chart at the end will detail the releases.

IBM No Longer Waits For Next Release

This is really, really cool!

- Prior to IBM i 7.1 ("V7R1") to get new RPG feature, you waited for next release.
- Couldn't install it right away? Had to wait longer.
- Needed to support both newer and older releases? Wait longer.
- Could be 5+ years before you could use a feature you wanted.

This is no longer true!

- Starting with Open Access (should've been a 7.2 feature) most new stuff is available on all supported releases!
- Fully free format
- DATA-INTO
- Nested Data Structures

Is This Good Or Bad?

Definitely good for developers?
- especially software companies.

Does it look bad for IBM?
- why should people update releases?
- do people perceive IBM's RPG commitment as "less"?

For this reason, IBM holds back at least some of the updates to the next release.

Support in SEU

There is none.

SEU has not had updates since the release of IBM i 6.1
That was March 21 2008!

Use the current version of RDi, or a 3rd party tool:

IBM Rational Developer for i (RDi)
- https://www.ibm.com/support/pages/rational-developer-i-download

MiWorkplace (Remain Software):
- https://remainsoftware.com/miworkplace-development-environment-ibm-i

VS Code (open source, cross-platform):
- VS Code IDE: https://code.visualstudio.com/
- Main IBM i: https://marketplace.visualstudio.com/items?itemName=HalcyonTech Ltd.code-for-ibmi
- RPGLE Language Tools: https://marketplace.visualstudio.com/items?itemName=HalcyonTech Ltd.vscode-rpgle

https://ibm.biz/rpg_cafe

%LEFT and %RIGHT

(Fall 2023) PTF for 7.4, 7.5
- %LEFT takes a substring of the leftmost portion of a string
- %RIGHT takes a substring of the rightmost portion of a string
- Supports the CharCount *NATURAL options (coming up soon)

New

```
leftChars = %LEFT('abcdefg': 3);

// leftChars = 'abc'

rightChars = %RIGHT('abcdefg': 3);

// rightChars = 'efg'
```

Old

```
leftChars = %SUBST('abcdefg': 1: 3);

// leftChars = 'abc'

rightChars = %SUBST('abcdefg': %len('abcdefg') - (3-1): 3);

// rightChars = 'efg'
```

Simple Enumerations

(Fall 2023) PTF for 7.4, 7.5

- DCL-ENUM / END-ENUM declares simple enumerations.
- Basically is a grouping of related named constants, list possible values for an item
- Can be QUALIFIED like a data structure

New

```
dcl-enum STATUS;
  SUCCESS 1;
  FAIL    0;
end-enum;

status = InterestingProc();
if status = SUCCESS;
// yay!
endif;
```

Old

```
dcl-c SUCCESS 1;
dcl-c FAIL    0;

status = InterestingProc();
if status = SUCCESS;
// yay!
endif;
```

```
dcl-enum DB2NULL qualified;
  NULL    -1;
  MAPERR  -2;
  NOTNULL  0;
end-enum;

if ndesc = DB2NULL.notNull;
// use 'desc'
endif;
```


Simple Enumerations

(Fall 2023) PTF for 7.4, 7.5

```
exec SQL
  select name, dept, type
    into :name :nname, :dept, :type
    from EMPMAST
   where id = :myEmpNo;

if nname = DB2NULL.notNull;
  // use 'desc'
endif;

select type;
when-is employee_type.regular;
  // handle regular employee
when-is employee_type.manager;
  // handle manager
when-is employee_type.executive;
  // handle the execs
end-sl;
```

%PASSED

(Spring 2023) PTF for 7.5, 7.4

- %PASSED checks both if a given parameter was passed and wasn't *OMIT-ed.
- Previously would need to check %PARMS, %PARMNUM and %ADDR

New

```
dcl-pi *n;
  p1 int(10)           options(*omit);
  p2 varchar(20) const options(*nopass);
  p3 varchar(20) const options(*nopass : *omit);
end-pi;

// p1 is required, but *OMIT could be passed
if %passed(p1);
  dsply ('Parameter p1 = ' + %char(p1));
else;
  dsply ('Parameter p1 is not available');
endif;

// p2 is optional
if %passed(p2);
  dsply ('Parameter p2 = ' + p2);
else;
  dsply ('Parameter p2 is not available');
endif;

// p3 is optional, and *OMIT could be passed
if %passed(p3);
  dsply ('Parameter p3 = ' + p3);
else;
  dsply ('Parameter p3 is not available');
endif;
```

Old

```
dcl-pi *n;
  p1 int(10)           options(*omit);
  p2 varchar(20) const options(*nopass);
  p3 varchar(20) const options(*nopass : *omit);
end-pi;

// p1 is required, but *OMIT could be passed
if %addr(p1) <> *null;
  dsply ('Parameter p1 = ' + %char(p1));
else;
  dsply ('Parameter p1 is not available');
endif;

// p2 is optional
if %parms >= %parmnum(p2);
  dsply ('Parameter p2 = ' + p2);
else;
  dsply ('Parameter p2 is not available');
endif;

// p3 is optional, and *OMIT could be passed
if %parms >= %parmnum(p3) and %addr(p3) <> *null;
  dsply ('Parameter p3 = ' + p3);
else;
  dsply ('Parameter p3 is not available');
endif;
```


%OMITTED

(Spring 2023) PTF for 7.5, 7.4

- %OMITTED checks if a given parameter was omitted (vs. not passed, etc.)
- Primarily useful to distinguish between no parameter passed vs. *OMIT passed.
- Equivalent to checking `%parms() >= %parmnum(p2) and %addr(p2) = *null`

```
dcl-pi *n;
  p1 varchar(20) const options(*nopass : *omit);
end-pi;

// p1 is optional, and *OMIT could be passed
if %passed(p1);
  dsply ('Parameter p1 = ' + p1);
elseif %omitted(p1);
  dsply ('*OMIT was passed for parameter p1');
else;
  dsply ('No parameter was passed for p1');
endif;
```

SELECT/WHEN-IS/WHEN-IN

(Spring 2023) PTF for 7.5, 7.4

- The SELECT opcode now allows an expression to be provided after the opcode.
- WHEN-IN can be used to check if that expression is IN a list, range, array, etc.
- WHEN-IS can be used to check if it is equal to a value.
- Similar to the switch/case you see in some other languages.

```
dcl-s Price packed(9: 2);

select Price;
when-is 0;
  // No charge item
when-in %range(2.75 : 20.00);
  // Price is allowed
other;
  // Invalid price
endsl;
```

```
dcl-s UOM        char(2);
dcl-s casesArray char(2) dim(2);

casesArray(1) = 'CA';
casesArray(2) = 'BX';

select UOM;
when-in %list('EA': 'PC': 'IT');
  desc = 'Item';
when-is 'PL';
  desc = 'Pallet';
when-in casesArray;
  desc = 'Case';
other;
  desc = '** INVALID UOM **';
endsl;
```


PCML V8 w/Boolean

(Spring 2023) PTF for 7.5, 7.4, 7.3 (!!)

- When generating PCML with the PGMINFO option, you can now specify *V8
- Alternately, you can set the QIBM_RPG_PCML_VERSION environment variable.
- Generated PCML will now supply boolean="true" to the PCML document for any RPG indicator parameters.

```
ADDENVVAR ENVVAR(QIBM_RPG_PCML_VERSION) VALUE('8.0')
```

```
ctl-opt pgminfo(*pcml : *dclcase: *v8 );

dcl-pi *n;
  customer packed(4: 0);
  active   ind;
end-pi;
```

```
<pcml version="8.0">
  <program name="MYPGM" entrypoint="MYPGM">
    <data name="customer" type="packed" length="4" precision="0" usage="inputoutput" />
    <data name="active" type="char" length="1" boolean="true" usage="inputoutput" />
  </program>
</pcml>
```

%SPLIT w/*ALLSEP

(Spring 2023) PTF for 7.5, 7.4

- The %SPLIT built-in function now supports all separators via the *ALLSEP option.
- Will discuss this when discussing the %SPLIT BIF, later.

- %SPLIT was added in Spring 2021, but the *ALLSEP option is newer, Spring 2023!

RPGPPOPT w/Long Records

(January 2023) PTF for 7.5, 7.4, 7.3 (!!)

- The SQL preprocessor (CRTRPGSQLI) now lets you control the record length of the preprocessed source member that it outputs.
- Only when using RPGPPOPT(*LVLx)
- You can specify the PPMINOUTLN parameter on CRTBNDRPG/CRTRPGMOD
- Alternately, use the QIBM_RPG_PPSRCFILE_LENGTH environment variable.
- Previously, long lines could fail with RNF0733 when generating the output. Source lines were limited to 100 characters by the preprocessor.

For example, suppose you compile from the IFS, and you have records that are long (maybe 200 characters long – there is no limit in the IFS!)

```
CRTSQLRPGI OBJ(MYPGM) SRCSTMF('/home/scott/mypgm.sqlrpgle') RPGPPOPT(*LVL2)
RNF0733 The record length of the output file is too small.
```

This is because it preprocesses it into QTEMP/QSQLPRE – a source file, which has a limit of 100 characters per line.

```
ADDENVVAR ENVVAR(QIBM_RPG_PPSRCFILE_LENGTH) VALUE('250')
CRTSQLRPGI OBJ(MYPGM) SRCSTMF('/home/scott/mypgm.sqlrpgle') RPGPPOPT(*LVL2)
```

Now QTEMP/QSQLPRE is created with a record length of 250, so no errors.

OPTIONS(*CONVERT)

(Fall 2022) PTF for 7.5, 7.4

- OPTIONS(*CONVERT) on a prototyped parameter with CONST or VALUE will automatically convert numbers, dates, times and timestamps to character. Pointers will be treated like options(*string).
- Useful when a subprocedure may want data passed from different sources, and you don't want to go to the extra level of implementing overloading.

```
WriteMsg(123);
WriteMsg(%date());
WriteMsg('Scott is cool!');
return;

dcl-proc WriteMsg;
  dcl-pi *n;
    msg varucs2(100) const options(*convert);
  end-pi;

  snd-msg 'The parameter is ' + msg;
end-proc;
```


%CONCAT

(Fall 2022) in PTF for 7.5, 7.4

- %CONCAT will concatenate a list of items, joining them with a given separator.
- Alternately, you can specify *NONE if you do not wish to use a separator.

```
dcl-s csv varchar(200);
dcl-s string varchar(10);

csv = %concat(',' : '1001' : 'Acme Inc' : 'Scott Klement' : '123 Main St');
// csv = '1001,Acme Inc,Scott Klement,123 Main St'

string = %concat(*none : 'A' : 'B' : 'C');
// string = 'ABC'
```

%CONCATARR

(Fall 2022) PTF for 7.5, 7.4

- Similar to %CONCAT, except %CONCATARR will concatenate the elements of an array.

```
dcl-s csv varchar(200);
dcl-s data varchar(20) dim(4);

data(1) = '1001';
data(2) = 'Acme Inc';
data(3) = 'Scott Klement';
data(4) = '123 Main St';

csv = %concatarr(',': data);
// csv = '1001,Acme Inc,Scott Klement,123 Main St'
```


CHARCOUNT NATURAL

(Fall 2022) PTF for 7.5, 7.4

- When coding UTF-8 or UTF-16 data, the length of a character can vary. (UTF-8 characters are 1-4 bytes long, UTF-16 are 2 or 4 bytes.)
- With CHARCOUNT(*NATURAL) BIFs like %SUBST, %SCAN, %XLATE and %SPLIT will work with the number of characters (regardless of the byte size of each.)
- The old mode is called CHARCOUNT(*STDCHARSIZE) and the preceding BIFs will treat the numbers as bytes (for CHAR) or double-bytes (for UCS-2) regardless of the length of the individual character.
- Can specify ctl-opt (or h-spec) CHARCOUNT and CHARCOUNTTYPES
- Or /CHARCOUNT directive
- Or CHARCOUNT file keyword (dcl-f or f-spec)
- Or specify *NATURAL or *STDCHARSIZE on the BIF itself.
- The %CHARCOUNT BIF can be used to get the length in natural characters

```
dcl-s string varchar(20) ccsid(*utf8);
dcl-s len int(10);

string = 'ábcdë';

len = %len(string);
// len = 7 (á and ë are 2-byte characters)
len = %charcount(string);
// len = 5
```

CHARCOUNT NATURAL

```
dcl-s string varchar(20) ccsid(*utf8);
dcl-s string2 varchar(20);

string = 'ábcdë';

string2 = %subst(string : 1 : 3);
// string2 = "áb"

string2 = %subst(string : 1 : 3 : *natural);
// string2 = "ábc"
```

```
ctl-opt charcounttypes(*utf8) charcount(*natural);

dcl-s string varchar(20) ccsid(*utf8);
dcl-s string2 varchar(20);

string = 'ábcdë';

string2 = %subst(string : 1 : 3);
// string2 = "ábc"

/charcount stdcharsize

string2 = %subst(string : 1 : 3);
// string2 = "áb"
```


SND-MSG

(Spring 2022) in 7.5, PTF for 7.4, 7.3

- SND-MSG opcode lets you send program messages from RPG (like SNDPGMMSG from CL)
- Defaults to an *INFO message but can also be *ESCAPE.
- Other types (*DIAG, *STATUS, *COMP) are not currently supported.
- The %MSG BIF is used to control the msg id/msg file.
  - Default for *INFO is no message id, *ESCAPE is CPF9898
- The %TARGET BIF can be used to control which call stack entry the message is sent to. It allows *SELF or *CALLER, plus an optional stack count.

```
// these send *INFO messages that will appear in the job log

SND-MSG 'Program MSGSFL1 has started.';
SND-MSG *INFO 'This works the same, *INFO is the default.';
SND-MSG *INFO %MSG( 'CPF9898'
                  : '*LIBL/QCPFMSG'
                  : 'This uses a message from a MSGF');


// This sends an *ESCAPE message to 1 call stack prior to the caller
// (aka the "caller's caller")

SND-MSG *ESCAPE %MSG('CPF2105': 'QCPFMSG': 'SNDMSG1   *LIBL     PGM')
                %TARGET(*CALLER:1);
```

SND-MSG with MSGSFL

One common use of sending messages in RPG (aside from reporting errors and putting information in the job log) is to load a "message subfile" (DDS MSGSFL)

```
**free
ctl-opt DFTACTGRP(*NO) ACTGRP(*NEW);

dcl-f MSGSFL4D workstn;

dcl-ds sds PSDS;
  PgmQ *proc;
end-ds;

dcl-s num packed(4: 0);

snd-msg 'This is a test message'
        %target(PGMQ);

write msgctl;
exfmt msgscn1;

*inlr = *on;
```

```
A                                      DSPSIZ(*DS3)
A          R MSGSFL
A                                      SFL
A                                      SFLMSGRCD(15)
A            MSGKEY                    SFLMSGKEY
A            PGMQ                      SFLPGMQ(10)
A*
A          R MSGCTL
A                                      SFLCTL(MSGSFL)
A                                      SFLDSP
A                                      SFLDSPCTL
A                                      SFLINZ
A N99                                  SFLEND
A                                      SFLSIZ(20)
A                                      SFLPAG(5)
A            PGMQ                      SFLPGMQ(10)
A          R MSGSCN1
A                                      OVERLAY
```


ON-EXCP

(Spring 2022) in 7.5, PTF for 7.4, 7.3

- ON-EXCP opcode works together with MONITOR to let you monitor for specific exception messages (similar to MONMSG in CL)

```
dcl-pr QCMDEXC extpgm;
  command char(32767)   const;
  length  packed(15: 5) const;
  igc     char(3)       const options(*nopass);
end-pr;

dcl-s cmd varchar(200);

cmd = 'CHKOBJ OBJ(LLLL/FFFF) OBJTYPE(*FILE)';
cmd = %scanrpl('LLLL': %trim(Library): cmd);
cmd = %scanrpl('FFFF': %trim(File)   : cmd);

monitor;
  QCMDEXC(cmd: %len(cmd));
on-excp 'CPF9801';
  // file wasn't found
on-excp 'CPF9810';
  // library wasn't found
endmon;
```

```
monitor;
  QCMDEXC(cmd: %len(cmd));
on-excp 'CPF9801': 'CPF9810';
  // Either the library or the file wasn't found
endmon;
```

SND-MSG and ON-EXCP Together

Naturally, you can use SND-MSG to send messages that you trap with ON-EXCP. For example, you can use SND-MSG to report errors from a subprocedure.

```
dcl-proc getCustomerName;

  dcl-pi *n varchar(30);
    CustNo packed(7: 0) value;
  end-pi;

  dcl-f custmastp disk usage(*input) static;
  dcl-ds custrec likerec(custmastr:*input);

  chain CustNo custmastp custrec;
  if not %found;
    snd-msg *ESCAPE 'Customer not found';   // Default for *ESCAPE is CPF9898
  else;
    return %trim(custrec.csName);
  endif;

end-proc;
```

```
monitor;
  name = getCustomerName(custNo);
  // got the name!
on-excp 'CPF9898';
  // customer wasn't found
endmon;
```


%MAXARR / %MINARR

(Fall 2021) in 7.5, PTF for 7.4, 7.3

- %MAXARR and %MINARR BIFs return the index to the maximum/minimum array elements

```
dcl-s arr     char(1) dim(5);
dcl-s highest int(10);

arr(1) = 'A';
arr(2) = 'Z';
arr(3) = 'B';
arr(4) = 'Y';
arr(5) = 'C';

highest = %MAXARR(arr);  // highest = 2

msg = 'The index with the highest value: '
    + %char(highest);
```

```
ctl-opt datfmt(*usa);

dcl-s birth_date date dim(4);
dcl-s lowest     int(10);

birth_date(1) = d'07/04/1999';
birth_date(2) = d'01/02/2000';
birth_date(3) = d'05/19/1967';
birth_date(4) = d'09/30/2021';

lowest = %MINARR(birth_date);
msg = 'Oldest birth: '
    + %char(birth_date(lowest));
```

```
dcl-s inv_amt packed(9: 2) dim(3);
dcl-s lowest  int(10);
dcl-s highest int(10);

inv_amt(1) = 27700.95;
inv_amt(2) = 12345.67;
inv_amt(3) = -4;

lowest = %MINARR(inv_amt);
msg = 'Lowest total invoice: ' + %char(inv_amt(lowest));

highest = %MAXARR(inv_amt);
msg = 'Highest total invoice: ' + %char(inv_amt(highest));
```

SORTA With Multiple Fields

(Fall 2021) in 7.5, PTF for 7.4, 7.3

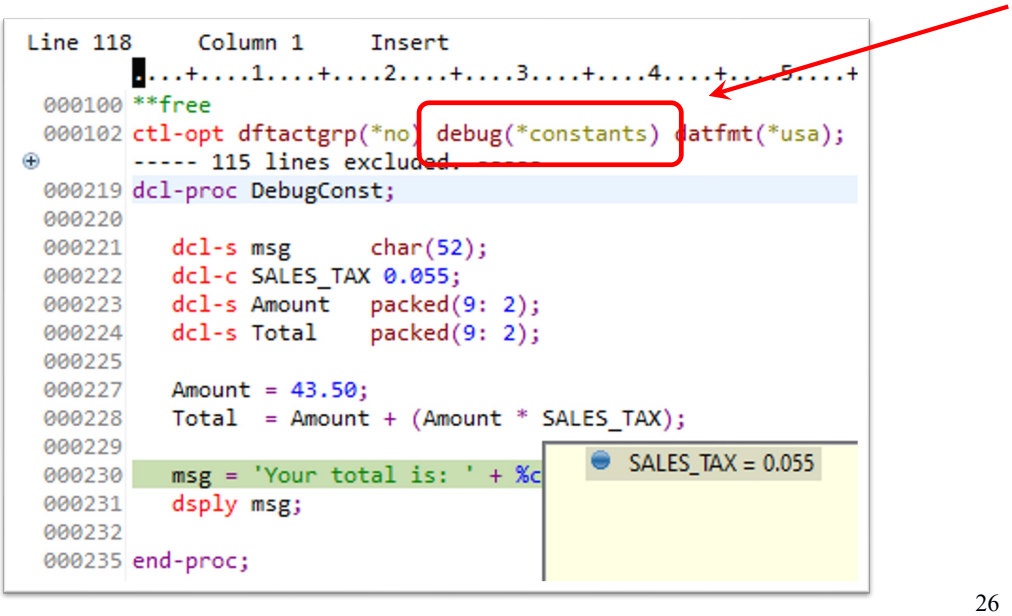
- %FIELDS BIF allows you to sort an array by multiple subfields with SORTA

```
dcl-ds Employee qualified dim(3);
  name  char(20);
  dept  char(12);
  state char(2);
end-ds;

Employee(1).name  = 'Klement, Scott';     // will be second (Development, WI)
Employee(1).dept  = 'Development';
Employee(1).state = 'WI';

Employee(2).name  = 'Russo, David';       // will be first (Development, OH)
Employee(2).dept  = 'Development';
Employee(2).state = 'OH';

Employee(3).name  = 'Seage, Emily';       // will be third (Support, MS)
Employee(3).dept  = 'Support';
Employee(3).state = 'MS';

sorta Employee %fields(dept: state: name);
```


DEBUG(*CONSTANTS)

(Fall 2021) in 7.5, PTF for 7.4, 7.3

- DEBUG(*CONSTANTS) is a CTL-OPT (or H-spec) keyword that lets you view the values of constants while debugging.



%UPPER and %LOWER

(Spring 2021) in 7.5, PTF for 7.4, 7.3

- %UPPER built-in function converts a string to uppercase
- %LOWER built-in function converts a string to lowercase
- Optional second parameter specifies the start position within the string
- Works with international characters, too. (So is better than %XLATE!)

```
myString = 'mIxEdcaSe';
myString = %lower(myString);

// myString is now 'mixedcase'

upperCase = %upper(myString);

// upperCase is now 'MIXEDCASE'

titleCase = %lower(upperCase:2);

// titleCase is now 'Mixedcase'
```


%SPLIT

(Spring 2021) in 7.5, PTF for 7.4, 7.3

- %SPLIT built-in function splits a string when a given substring is found.
- The result is an array of strings

```
dcl-ds invoice qualified;
  name    varchar(30)  inz('unused');
  amount  packed(9: 2) inz(-1);
end-ds;

dcl-s record varchar(200) inz('Acme Foods Inc | 4502.60');
dcl-s array  varchar(50) dim(2);

array = %split(record:'|');

invoice.name = array(1);
invoice.amount = %dec(array(2):9:2);
```

In this example, two fields are stored in one string, separated by the pipe character. %Split is used to split them into two array elements, which can then be assigned to separate fields in a data structure.

FOR-EACH

(Fall 2020) in 7.5, PTF for 7.4, 7.3

- FOR-EACH loop opcode

Loops through all of the elements in an array. Can be used together with %SUBARR

Old way (FOR loop)

```
total = 0;
for x = 1 to %elem(invoices);
  total += invoices(x).amount;
endfor;
```

New way (FOR-EACH loop)

```
total = 0;
for-each invoice in invoices;
  total += invoice.amount;
endfor;
```

Old way (FOR loop)

```
total = 0;
for x = 1 to numLoaded;
  total += invoices(x).amount;
endfor;
```

New way (FOR-EACH loop)

```
total = 0;
for-each invoice in %subarr(invoices: 1: numLoaded);
  total += invoice.amount;
endfor;
```


FOR-EACH with %SPLIT

- Since %SPLIT returns an array, it can be used with a FOR-EACH loop
- In this example, "shelves" is a list of shelves separated by commas
- %SPLIT could be used in a similar manner to the way %LIST was used before.

```
dcl-s shelves varchar(20) inz('TOP,A,B,C,FLOOR');
dcl-s shelf char(5);

for-each shelf in %split(shelves:',');

  chain (whsloc: shelf) INVENP;
  if %found and ItemCode = Item;
    leave;
  endif

endfor;
```

Split is very useful when reading IFS files in CSV, Tab or Pipe-Delimited format, or when a database field or user input contains a delimited list of data.

%SPLIT w/*ALLSEP

(Spring 2023) PTF for 7.5, 7.4

- The %SPLIT built-in function now supports all separators via the *ALLSEP option.
- Without this option %SPLIT would ignore leading, trailing or consecutive separators.

Without *ALLSEP

```
dcl-s csv varchar(1000);
dcl-s fld varchar(20);

csv = '1001,Acme Inc,Scott Klement,123 Main St';
for-each fld in %split(csv : ',' );
  dsply fld;
  // values for fld in the for-each loop:
  //    1  1001
  //    2  Acme Inc
  //    3  Scott Klement
  //    4  123 Main St
endfor;

csv = '1002,Industrial Supply,,500 Superior Ln';
for-each fld in %split(csv : ',' );
  dsply fld;
  // values for fld in the for-each loop:
  //    1  1002
  //    2  Industrial Supply
  //    3  500 Superior Ln
endfor;
```

With *ALLSEP

```
dcl-s csv varchar(1000);
dcl-s fld varchar(20);


// First example works the same, since there
// are no leading, trailing or consecutive
// separators


// Second example however, no longer ignores the
// "empty field" (consecutive separators)

csv = '1002,Industrial Supply,,500 Superior Ln';
for-each fld in %split(csv : ',' : *ALLSEP);
  dsply fld;
  // values for fld in the for-each loop:
  //    1  1002
  //    2  Industrial Supply
  //    3  (empty string)
  //    4  500 Superior Ln
endfor;
```


IN, %RANGE, %LIST

(Fall 2020) in 7.5, PTF for 7.4, 7.3

- IN operator used in comparisons with %RANGE or %LIST
- %RANGE used to compare data in a range
- %LIST returns a temporary array from a list of values

```
// OLD: if Qty >= 1 and Qty <= 999;

if Qty IN %range(1:999);
  msg = 'Quantity acceptable.';
endif;
```

```
// OLD: if shelf = 'TOP' or shelf = 'A' or shelf = 'B'
//         or shelf = 'C' or shelf = 'FLOOR';

if shelf in %list('TOP': 'A': 'B': 'C': 'FLOOR');
  msg = 'Shelf level is acceptable';
endif;
```

FOR-EACH with %LIST

Since %LIST returns a temporary array, it can also be used with the FOR-EACH loop I discussed earlier.

```
// Search shelves in warehouse location for ItemCode

for-each shelf in %list('TOP': 'A': 'B': 'C': 'FLOOR');
  chain (whsloc: shelf) INVENP;
  if %found and ItemCode = Item;
    leave;
  endif;
endfor;
```


EXPROPTS(*STRICTKEYS)

(Spring 2021) in 7.5, PTF for 7.4, 7.3

- EXPROPTS(*STRICTKEYS) CTL-OPT (H-spec) keyword requires the length, ccsid, and decimal places to match (or be smaller than) that of the database

```
ctl-opt EXPROPTS(*STRICTKEYS);

dcl-f PRODP disk keyed;       // key is packed(7, 0)

dcl-s itemno packed(7: 1);

itemno = 953.5;
chain (itemno) PRODP;

RNF0203: KEY NUMBER 1 DOES NOT MEET THE RULES FOR
EXPROPTS(*STRICTKEYS). REASON: DECIMALS.
```

Without EXPROPTS(*STRICTKEYS) the above would compile and would retrieve record with key=953 despite that the key shouldn't have decimal places.

With *STRICTKEYS, it won't compile.

DEBUG(*RETVAL)

(Fall 2020) in 7.5, PTF for 7.4, 7.3

- DEBUG(*RETVAL) control option and _QRNU_RETVAL

```
dcl-proc MyProc;

  dcl-s x int(10) inz(99);
  return (x * 43);

end-proc;
```

You can check the return value in the debugger. To do that:

- Set breakpoint on the end-proc
- View variable _QRNU_RETVAL

```
total = readNext(InvNo: MoreData);
dow MoreData;
  total += readNext(InvNo: MoreData);
enddo;
. . .
dcl-proc ReadNext;

  dcl-pi *n packed(9: 2);
    InvoiceNo packed(7: 0) const;
    MoreData ind;
  end-pi;

  reade (InvoiceNo) INVOICEP;
  If %EOF;
    MoreData = *off;
    return 0;
  else;
    ModeData = *on;
    return (QTY * PRICE);
  endif;

end-proc;
```


EXPROPTS(*ALWBLANKNUM:*USEDECEDIT)

(Fall 2020) in 7.5, PTF for 7.4, 7.3

- EXPROPTS(*ALWBLANKNUM) allows blanks to be converted to 0.
- EXPROPTS(*USEDECEDIT) allows DECEDIT to specify the decimal separator, and thousands separators are ignored
- When DECEDIT specifies comma as decimal separator, period is thousands separator, and vice-versa

```
ctl-opt EXPROPTS(*ALWBLANKNUM:*USEDECEDIT) DECEDIT('0,');   // European-style

dcl-s myChar varchar(10);
dcl-s myDec  packed(7: 2);

myChar = '';
myDec = %dec(myChar: 7: 2);
// With *ALWBLANKNUM, myDec = 0
// Without: RNX0105: A character representation of a numeric value is in error.

myChar = '0.12';
myDec  = %dec(myChar: 7: 2);
// With *USEDECEDIT, myDec = 12.00
// Without: myDec = 0.12

myChar = '10.000,05';
myDec  = %dec(myChar: 7: 2);
// With *USEDECEDIT, myDec = 10000.05
// Without: RNX0105: A character representation of a numeric value is in error.
```

Microsecond Timestamps & *UNIQUE

(Spring 2020) in 7.5, PTF for 7.4, 7.3

- %TIMESTAMP will now return accurate microseconds (6-digit fractions of a second)
- %TIMESTAMP(*UNIQUE) returns 12-digit fractions that are guaranteed to be unique (but are not accurate times)

```
dcl-s ts1 timestamp(12);
dcl-s ts2 timestamp(12);
dcl-s ts3 timestamp(12);
dcl-s ts4 timestamp(12);

// If computer is fast enough, these two could potentially be
// the same -- but they are an accurate measure of time.

ts1 = %timestamp();         // 2022-03-10-09.45.12.990317000000
ts2 = %timestamp();         // 2022-03-10-09.45.12.990338000000

// These two will never be the same, but digits after the 6th
// are not accurate times.

ts3 = %timestamp(*unique); // 2022-03-10-09.45.12.990342000244
ts4 = %timestamp(*unique); // 2022-03-10-09.45.12.990344000244
```


LIKEDS(qualified.name)

(Spring 2020) in 7.5, PTF for 7.4, 7.3

- LIKEDS prior to this update did not allow qualified names (names with data structure, followed by period, followed by a subfield name)
- Now it does!

```
dcl-ds Customer qualified;
  id   packed(5: 0);
  name varchar(30);
  dcl-ds Address;
    street varchar(30);
    city   varchar(20);
    state  char(2);
    postal varchar(10);
  end-ds;
end-ds;

dcl-ds oldAddress likeds(Customer.Address);

// Prior to the update, the LIKEDS above would not
// compile. Now it works fine.
```

%KDS with variable keys

(Spring 2020) in 7.5, PTF for 7.4, 7.3

- %KDS now allows a variable in it's second parameter.
- Prior to this update, you had to use a constant number instead of a variable.

```
dcl-f POSTP disk keyed;     // keyed by PostZip, PostState & PostCity

dcl-ds REC likerec(POSTR:*ALL);
dcl-ds KEY likerec(POSTR:*KEY);
dcl-s  num_keys int(10);

KEY.PostZip   = 53207;
KEY.PostCity  = 'SAINT FRANCIS';
KEY.PostState = 'WI';

// retrieves the first record that matches all 3 keys
num_keys = 3;
chain %kds(KEY:num_keys) POSTP REC;

// retrieves the first record that matches the first key (PostZip)
num_keys = 1;
chain %kds(KEY:num_keys) POSTP REC;
```


OVERLOAD

(Fall 2019) in 7.5, PTF for 7.4, 7.3

- OVERLOAD keyword allows multiple "candidate prototypes" to be considered
- RPG automatically picks the right prototype for the parameter types passed

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

    DCL-PR format_date VARCHAR(100);
       dateParm DATE(*ISO) CONST;
    END-PR;
    DCL-PR format_time VARCHAR(100);
       timeParm TIME(*ISO) CONST;
    END-PR;
    DCL-PR format_message VARCHAR(100);
       msgid CHAR(7) CONST;
       replacement_text VARCHAR(100) CONST OPTIONS(*NOPASS);
    END-PR;
    DCL-PR format VARCHAR(100) OVERLOAD( format_time
                                       : format_date
                                       : format_message);
    DCL-S result varchar(50);


    result = format(%date());
    result = format(%time());
    result = format('MSG0100' : filename);
```

DATA-GEN

(Fall 2019) in 7.5, PTF for 7.4, 7.3

- DATA-GEN opcode to generate JSON, XML, etc documents.

DATA-GEN opcode can be used to generate a structured file format from an RPG variable.

- Most often, used with a data structure
- Can be a DS array, and can have other DSes or arrays embedded/nested
- Works with a 3rd-party generator program (like DATA-INTO)
- You need a generator program that understands how to create the format that you wish to create (such as XML or JSON)

```
dcl-ds req qualified;
   source char(2);
   target char(2);
   text varchar(1000) dim(2);
end-ds;
```

```
{
  "source": "{string}",
  "target": "{string}",
  "text": ["{string}", "{string}"]
}
```

```
  myFile = '/tmp/example.json';
  DATA-GEN req %DATA(myFile: 'doc=file output=clear')
               %GEN('YAJLDTAGEN');
```


OPTIONS(*EXACT)

(Fall 2019) in 7.5, PTF for 7.4, 7.3

- OPTIONS(*EXACT) option for a prototyped parameter.
- Previously, RPG has allows any character variable if it is larger than the parameter in the prototype
- With this feature, the parameter must match exactly,

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

    dcl-pr p1;
       parm5 char(5);
    end-pr;
    dcl-pr p2;
       parm5Exact char(5) OPTIONS(*EXACT);
    end-pr;
    dcl-s fld10 char(10) inz('abcdefghij');

    p1 (fld10);
    p2 (fld10);  // Error, fld10 is not char(5)!!
```

Variable-Dimension Arrays

(Spring 2019) Only in 7.4, 7.5 (and later)

- DIM(*VAR: max-size) lets you create an array with a variable-length. (*VAR is to arrays what VARCHAR is to CHAR).
- DIM(*AUTO: max-size) lets you create a variable-length array that increases automatically.
- %ELEM can be used to return the current length of the array, or to increase/decrease the length
- %ELEM(array:*KEEP) can be used to increase the length without initializing the new elements
- %ELEM(array:*ALLOC) can be used to get/set the allocated size of the array
- *NEXT can be used to assign a value to the next available array element

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

// myArray starts with 0 elements.
DCL-S myArray CHAR(10) DIM(*VAR:100);

// myArray now has 5 elements (blank)
%ELEM(myArray) = 5;

// x will be set to 5
x = %ELEM(myArray);
```


The *VAR Keyword on DIM

- *VAR keyword means that
  - The length of the array isn't always at max
  - With *VAR, the size doesn't increase automatically, you control it with %ELEM
  - RPG utilizes less memory for the array to begin with
  - When the size is changed with %ELEM, RPG will automatically increase the amount of memory used

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

// myArray starts with 0 elements.
// also doesn't use any memory, since it's empty
DCL-S myArray CHAR(10) DIM(*VAR:5000);

// myArray now has 100 blank elements
// uses 1000 bytes of memory
%ELEM(myArray) = 100;

// %ELEM returns the current number of elements
//  (Assuming you knew that there were products 1-100 in PRODTABL)
for x = 1 to %ELEM(myArray);
  chain x PRODTABL;
  myArray(x) = PRODNAME;
endfor;
```

Controlling *VAR Size With %ELEM

- *VAR arrays do not automatically increase in size.
  - You must use %ELEM

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

DCL-S myArray CHAR(10) DIM(*VAR:5000);

// This produces an error; there are only 100 elements in myArray
// even though the max=5000, it is currently only 100
%ELEM(myArray) = 100;
myArray(101) = 'xyz';  <-- ERROR!

// This works; the array now allows 200 elements
%ELEM(myArray) = 200;
myArray(101) = 'xyz';

// Shrink the size back to only 5
%ELEM(myArray) = 5;

// Return length to 200, without resetting values.  (101 will still be 'xyz')
%ELEM(myArray:*KEEP) = 200;

// If you remove *KEEP in the last example, it would set elements 6-200 to
// the default value (blank in this case, or whatever was set with INZ)
```


The *AUTO Keyword on DIM

- *AUTO keyword allows RPG to automatically increase the array
  - Similar to *VAR, except when you access an element beyond the current length
  - If beyond the current length, it is automatically increased
  - Array does not shrink when you access a lower number
  - %ELEM can still be used, and %ELEM will shrink the size

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

DCL-S array2 CHAR(10) DIM(*AUTO:100);

// This is not an error with *AUTO.  It will be extended to 6 elements
%ELEM(array2) = 5;
array2(6) = 'xyz';  <-- WORKS!

// However, this does not shrink the array; it remains 6 long
array2(1) = 'abc';

// We can shrink it with %ELEM
%ELEM(array2) = 1;

// If you access a higher element again, it will extend it again
// but you can never exceed the maximum size from the DIM keyword
array2(100) = 'def';
array2(101) = 'err';  <-- ERROR!  DIM only allowed up to 100 elements
```

Using *NEXT with *AUTO

- *NEXT can be used to specify the next available array element
- Works really well with *AUTO

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

DCL-S array3 CHAR(10) DIM(*AUTO:9999);

setll *start PRODTABL
read PRODTABL;

// *NEXT is the next available array index
// Since this is *AUTO, it automatically increases the array size, too.
dow not %eof(PRODTABL);
  array3(*NEXT) = PRODDESC;
  read PRODTABL;
enddo;

// The array size should be the number of products in PRODTABL
numProds = %elem(array3);
```


Variable-Dimension Restrictions

There are some critical restrictions on varying-dimension arrays:

- Must be either a standalone array, or the top-level definition of a DS (not a subfield)
- Cannot be used in fixed format calculations
- Cannot be based on a pointer. Cannot be imported/exported.
- Cannot be null-capable
- Cannot be used with CTDATA or FROMFILE (compile-time or pre-runtime data)
- You cannot use *VAR or *AUTO in a prototype.
  - You can pass a variable-dimension array with options(*VARSIZE), CONST or VALUE
  - However, it will not be a variable-length in the procedure you're calling
  - You will need to track the length yourself by passing the current length in a separate parameter
- XML-INTO and DATA-INTO do not set the length automatically.

More restrictions can be found in the IBM Knowledge Center:
https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_74/rzasd/vardimarray.htm

DIM(*CTDATA)

(Spring 2019) Only in 7.4, 7.5 (and later)

- DIM(*CTDATA) = array has as many elements as CTDATA records
- Requires you to code **CTDATA at end of module
- Requires one element per record
- If using ALT, the alternating array must also be defined with DIM(*CTDATA)

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

dcl-s DaysOfWeek char(10) dim(*CTDATA) ctdata;
   .
   .
**CTDATA DaysOfWeek
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
```

DaysOfWeek will be DIM(7) because there are 7 records in the CTDATA

SAMEPOS DS Keyword

(Spring 2019) Released in 7.4, PTF back to 7.3

- SAMEPOS keyword for data-structure subfield
- works like POS, but bases position on another field's position

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
dcl-ds dateList qualified;
  date01 zoned(8: 0);
  date02 zoned(8: 0);
  date03 zoned(8: 0);
  date   zoned(8: 0) dim(3) overlay(date01);
  RNF7303: Subfield DATE defined with keyword OVERLAY is too big; specification ignored.
end-ds;
```

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
  .
  .
  date   zoned(8: 0) dim(3) pos(1);  // Old! It works, but hard-coded position
```

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
  .
  .
  date   zoned(8: 0) dim(3) samepos(date01); // New!
```

ON-EXIT

Released in 7.4, PTF for 7.2, 7.3

- ON-EXIT begins a section of code that is run when a procedure ends
- Code is always run, even if procedure "crashes" (ends abnormally)
- Typically used to "clean things up"
- Must be placed in a subprocedure or linear main procedure
- Will not work in a cycle main procedure

```
dcl-proc MYPROC;

// Set the 'in progress' data area to Y
  in *lock INPROGRESS;
  INPROGRESS = 'Y';
  out INPROGRESS;

// do whatever here
// For example, update a file, do calculations, etc.

// Set 'in progress' back to 'N', even if the above has an error.
on-exit;
  in *lock INPROGRESS;
  INPROGRESS = 'N';
  out INPROGRESS;
end-proc;
```


Nested Data Structures

Released in 7.4, PTF for 7.2, 7.3

- Qualified data structures can now be nested inside each other
- Previously had to use LIKEDS, and this wasn't always as intuitive

Before

```
....+....1....+....2....+....3....+....

dcl-ds address_t qualified template;
  Street char(30);
  City   char(30);
  State  char(2);
  Postal char(10);
end-ds;

dcl-ds Customer qualified;
  id   packed(4: 0);
  name char(30);
  address likeds(address_t);
end-ds;
```

After

```
....+....1....+....2....+....3....+....

dcl-ds Customer qualified;
  id   packed(4: 0);
  name char(30);
  dcl-ds address;
    Street char(30);
    City   char(30);
    State  char(2);
    Postal char(10);
  end-ds;
end-ds;
```

This is especially helpful with XML-INTO or DATA-INTO since it makes it easier to line up the DS fields with the document's fields.

%MIN and %MAX BIFs

Released in 7.4, PTF for 7.2, 7.3

- %MIN returns the smallest of its operands
- %MAX returns the largest of its operands
- All operands must be compatible (all numbers, all alpha, all dates, etc)
- You must have at least two operands, there's no maximum limit, however.

```
dcl-s result packed(9: 2);
dcl-s result2 char(3);
dcl-s p1 packed(9: 2)     inz(40.56);
dcl-s p2 packed(9: 2)     inz(80.21);
dcl-s p3 packed(9: 2)     inz(-1);

dcl-s arr1 char(10) dim(3);
dcl-s arr2 char(10) dim(5);
dcl-s arr3 char(10) dim(%max(%elem(arr1) : %elem(arr2))); // will be DIM(5)

// result will be 80.21
result = %max(p1: p2: p3);

// result will be -1.00
result = %min(p1: p2: p3);

// result2 will be 'zzz'
result2 = %max('aaa': 'bbb': 'qqq': 'zzz');
```


%PROC BIF

Released in 7.4, PTF for 7.2, 7.3

- %PROC returns the name of the current procedure
- For a cycle-main procedure, returns the name of the module
- For a subprocedure without extproc, or a linear main procedure, it will return the procedure name in all uppercase
- For a subprocedure with extproc, it will return the case used in the extproc

```
name = %PROC(); // name of module (I used 'PROCBIFRPG' for this example)

proc1();
proc2();

. . .

dcl-proc proc1;
  name = %PROC(); // will be 'PROC1', uppercase because there's no extproc
end-proc;

dcl-proc proc2;
  dcl-pi *n extproc(*dclcase);
  end-pi;
  name = %PROC(); // will be 'proc2', because of extproc(*dclcase)
end-proc;
```

System Name & Internal Job ID

Released in 7.4, PTF for 7.2, 7.3

- Program Status Data Structure (PSDS) enhanced to include two new fields
  - Internal Job ID at positions 380-395
  - System Name at positions 396-403

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

dcl-ds psds qualified psds;
   .
   .
  other PSDS fields here if needed
   .
   .
  InternalJobId char(16) pos(380);
  SystemName    char(8)  pos(396);
end-ds;
```

- Internal Job ID is a hex value used by some APIs.
- System Name is the one set by `CHGNETA SYSNAME(the-name-here)`
- This eliminates the need to call an API to retrieve these fields.

DATA-INTO

Released in 7.4, PTF for 7.2, 7.3

- DATA-INTO opcode allows similar functionality to XML-INTO
- DATA-INTO works with any data format that has a "parser" available.

Big limitation to XML-INTO

- only works with XML!
- JSON has overtaken XML as most popular
- many (thousands) of other document types exist
- other formats used in business today include: YAML, CSV, JSON, XDR, Property List, Pickle, OpenDDL, protobuf, OGDL, KMIP, FHIR, Feather, Arrow, EDN, CDR, Coifer, CBOR, Candle, Bond, Bencode, D-Bus, ASN.1, HOCON, MessagePack, SCaViS, Smile, Thrift, VPack

DATA-INTO

- RPG won't try to understand the document
- Calls 3rd-party tool ("parser") which interprets the document
- ...but, DATA-INTO maps result into RPG variable
- *...all you need is the right parser to read any format!*

DATA-INTO

```
dcl-ds result qualified;
  dcl-ds translations dim(1);
    translation varchar(1000);
  end-ds;
  word_count int(10);
  character_count int(10);
end-ds;
```

```
{
  "translations": [{
    "translation": "{string}"
  }],
  "word_count": {number},
  "character_count": {number}
}
```

Concept: Think of your document (such as JSON) as an RPG variable!

- { } JSON objects match RPG DS (dcl-ds)
- [ ] JSON arrays match RPG arrays (DIM)
- Field names must be the same
- JSON Strings (quoted) match RPG char, varchar, ucs2, etc
- JSON Numbers (unquoted) match RPG packed, zoned, integer, etc.

Then use DATA-INTO opcode:

```
DATA-INTO result %DATA(response) %PARSER('YAJLINTO');
```

DATA-INTO will copy the document (as interpreted by %PARSER) into matching RPG fields.

TGTCCSID keyword

Released 7.4, PTF for 7.3, 7.2, 7.1

- We've had the ability to put source in the IFS for a long time.
- Have always supported EBCDIC
- IFS code could be ASCII by translating to a "matching" EBCDIC (one that supports the same characters.)
- Prior to this update, IFS code could not be Unicode because this would confuse the compiler – there is no flavor of EBCDIC that supports (all of) the same characters as Unicode.
- With the TGTCCSID compile keyword, you can specify the EBCDIC to use, so source can be stored in Unicode now.
- Code is still converted to EBCDIC – still have EBCDIC's limitations.
- Not that exciting? But maybe useful for compatibility with editors, change management, version control systems, etc.

```
CRTRPGMOD TGTCCSID(37)            // or any valid EBCDIC CCSID
CRTBNDRPG TGTCCSID(*JOB)          // or special value *JOB

CRTRPGSQLI RPGPPOPT(*LVL2) COMPILEOPT('TGTCCSID(*JOB)')
```

Misc Other Features

Released in 7.4, PTF for 7.2, 7.3

- ALIGN(*FULL) keyword was added to improve alignment with C programs. It relates to padding data structures to the alignment value.
- PCML 7.0 support now available via RPG's PGMINFO parameter.
  - Adds support for varying length arrays
  - Adds support for DSes that contain varying-length character (varchar) subfields.
  - Note that not all callers (such as Java's ProgramCallDocument) have been updated to support PCML 7.0.

Reverse Scanning

Released in 7.3

- %SCANR is like scan, but scans right-to-left
- it will find the last occurrence of a string rather than the first

```
Pathname = '/home/sklement/test/sales2015.csv';

pos = %scanr('/': Pathname);

// pos = 20

Stmf = %subst(Pathname:pos+1);

// Stmf = sales2015.csv

DirName = %subst(Pathname:1:pos-1);

// dirname = /home/sklement/test

*inlr = *on;
```

Limit Scanning

Released in 7.3

- %SCAN now has an optional length parameter
- previously had start position, but not length
- works with %SCANR as well.

```
// from last slide:
//   Pathname = /home/sklement/test/sales2015.csv
//   pos = 20

if %scan('test': PathName: 1: pos) > 0;
   // "test" was found in the directory portion
   dsply 'test found';
endif;

if %scan('sales': PathName: 1: pos) = 0;
   // "sales" was not found in the directory portion
   dsply 'sales not found';
endif;

x = %scanr('/': PathName: 1: pos-1);
// x = 15
```


ALIAS Support on Files/Tables

Consider this physical file:

- short field names were hard to understand
- ALIAS allows longer names, but prior to 7.1, we couldn't use them in RPG!

```
....+....1....+....2....+....3....+....4....+....5....+....6....+..
.
 A          R CUSTREC
 A            CUSTNO          4S 0       ALIAS(CUST_NUM)
 A            CUBLAD         30A         ALIAS(CUST_BILLING_ADDRESS)
 A            CUBLCT         20A         ALIAS(CUST_BILLING_CITY)
 A            CUBLST          2A         ALIAS(CUST_BILLING_STATE)
 A            CUBLZP         10A         ALIAS(CUST_BILLING_ZIP)
 A            CUSHNM         30A         ALIAS(CUST_SHIPPING_ADDRESS)
 A            CUSHCT         20A         ALIAS(CUST_SHIPPING_CITY)
 A            CUSHST          2A         ALIAS(CUST_SHIPPING_STATE)
 A            CUSHZP         10A         ALIAS(CUST_SHIPPING_ZIP)
 A          K CUSTNO
```

ALIAS Support in SQL

Long names are enabled by default in SQL

- You can use "for" to also give a short name (optional)
- If you don't use "for", SQL will generate a short name like CUS00001

```
....+....1....+....2....+....3....+....4....+....5....+....6....+...
Create Table CUST (
  CUST_NUM                for CUSTNO numeric(4, 0) not null,
  CUST_BILLING_ADDRESS    for CUBLAD char(30) not null,
  CUST_BILLING_CITY       for CUBLCT char(20) not null,
  CUST_BILLING_STATE      for CUBLST char(2)  not null,
  CUST_BIllING_ZIP        for CUBLZP char(10) not null,
  CUST_SHIPPING_ADDRESS   for CUSHAD char(30) not null,
  CUST_SHIPPING_CITY      for CUSHCT char(20) not null,
  CUST_SHIPPING_STATE     for CUSHST char(2)  not null,
  CUST_SHIPPING_ZIP       for CUSHZP char(10) not null,
  primary key (CUST_NUM)
)
rcdfmt CUSTREC;
```


Original 7.1 Alias Support

ALIAS keyword would generate long names

- but only worked with data structure I/O
- required "qualified" keyword on file to avoid I-specs and O-specs

```
....+....1....+....2....+....3....+....4....+....5....+....6....+...
     FCUST      UF   E           K DISK    QUALIFIED ALIAS

     D CUST_IN       E DS                  extname(CUST:*input)
     D                                     qualified alias
     D CUST_OUT      E DS                  extname(CUST:*output)
     D                                     qualified alias

     D Key             s                   like(CUST_IN.CUST_NUM)

      /free
       key = 1000;
       chain key CUST CUST_IN;
       if %found;
          eval-corr CUST_OUT = CUST_IN;
          if cust_out.cust_billing_address = *blanks;
             cust_out.cust_billing_address = cust_out.cust_shipping_address;
             cust_out.cust_billing_city    = cust_out.cust_shipping_city;
             cust_out.cust_billing_state   = cust_out.cust_shipping_state;
             cust_out.cust_billing_zip     = cust_out.cust_shipping_zip;
             update CUST.CUSTREC CUST_OUT;
          endif;
       endif;
```

Improved Alias Support

Released 7.3, PTF for 7.1, 7.2

- ALIAS now works without data structures
- Compiler-generated I-specs/O-specs now support the longer names

```
....+....1....+....2....+....3....+....4....+....5....+....6....+...

     FCUST      UF   E           K DISK    ALIAS

     D Key               s                  like(CUST_NUM)

      /free
       key = 1000;
       chain key CUST;
       if %found;
          if cust_billing_address = *blanks;
             cust_billing_address = cust_shipping_address;
             cust_billing_city    = cust_shipping_city;
             cust_billing_state   = cust_shipping_state;
             cust_billing_zip     = cust_shipping_zip;
             update CUSTREC;
          endif;
       endif;
```


Relaxed DS I/O Rules

Prior to this update, if you use data structures for I/O, you must have separate ones for *INPUT and *OUTPUT.

Released 7.3, PTF for 7.2, 7.1

- Can use EXTNAME(MYFILE:*ALL) for any type of I/O
- Or can use LIKEREC(MYFILE) (with no usage) for any type of I/O

```
**FREE
dcl-f CUST disk keyed usage(*input: *output: *update);
dcl-ds rec extname('CUST':*ALL) qualified alias end-ds;

chain 1000 CUST rec;

if %found;
   rec.cust_billing_address = '123 Main Street';
   update CUSTREC rec;
else;
   rec.cust_num = 1000;
   rec.cust_shipping_address = '123 Main Street';
   write CUSTREC rec;
endif;
```

Same DS on chain, update and write operations

Improved Null Support

For some time, RPG has had support for database nulls in it's native I/O (i.e. "F-spec files") using ALWNULL(*USRCTL) and the %NULLIND BIF.

In version 7.3 (no PTFs for older versions), RPG has extended this support with the NULLIND keyword. This keyword enables you to:

- Define your own (standalone) fields as null-capable.
- Define your own indicators to replace the %NULLIND BIF
- Associate a null map data structure with a record data structure to handle nulls in data structure I/O

Like the %NULLIND BIF, the NULLIND keyword requires ALWNULL(*USRCTL) to be specified on the CTL-OPT or H-spec.

Standalone Null-Capable Field

I've wanted this for a long time!!

```
     H alwnull(*usrctl)

     D ShipDate        s                D   nullind
```

Also free format:

```
**FREE
ctl-opt alwnull(*usrctl);

dcl-s ShipDate date NULLIND;
```

Use %NULLIND to check/set the null indicator (same as a database field)

```
if %nullind(ShipDate) = *OFF;
   msg = 'Order was shipped on ' + %char(ShipDate:*USA);
else;
   msg = 'Order has not been shipped';
endif;

%nullind(ShipDate) = *ON;
```

Works with OPTION(*NULLIND)

Null capable fields can be passed between subprocedures as well

```
dcl-s ShipDate date NULLIND;
  .
  .
TestNull(ShipDate);
  .
  .
dcl-proc TestNull;
   dcl-pi *N;
      TheDate Date options(*nullind);
   end-pi;
   if %nullind(TheDate);
      dsply 'is null';
   else;
      dsply 'is not null';
   endif;
end-proc;
```


Use Your Own Indicator for Nulls

```
ctl-opt alwnull(*usrctl);

dcl-s NullShipDate ind;
dcl-s ShipDate date NULLIND(NullShipDate);

// This is the same as %nullind(ShipDate) = *ON
NullShipDate = *ON;

// This is the same as IF %NULLIND(ShipDate)=*ON
if NullShipDate = *ON;
   // not shipped
endif;
```

Also works in fixed format (in case you were wondering)

```
     H alwnull(*usrctl)

     D NullShipDate    s                 N
     D ShipDate        s                 D   nullind(NullShipDate)
```

Using NULLIND for Files

Released in 7.3, LIKEREC/EXTNAME can be used with *NULL to build a data structure that has the null indicators for a database record.

For example, consider this file

```
Create Table NULLTEST (
   CustNo numeric(4, 0) not null,
   FirstOrd date,
   AmtOwed decimal(9, 2),
   SalesRep char(30)
)
rcdfmt NULLTESTF;
```

FirstOrd, AmtOwed and SalesRep are null capable (but CustNo is not)

RPG code example on next slide

LIKEREC/EXTNAME *NULL

Released in 7.3

- Requires ALWNULL(*USRCTL)
- Still need to say *INPUT/*OUTPUT/*ALL so it knows which fields to include
- Add *NULL to make it the null map for the record
- Add NULLIND to the "normal" record DS to associate the null map to it.

```
**FREE
ctl-opt alwnull(*usrctl);

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

dcl-ds NULL likerec(NULLTESTF:*ALL:*NULL);
dcl-ds REC  likerec(NULLTESTF:*ALL) nullind(NULL);

read NULLTEST rec;

if null.FirstOrd = *ON;
   null.FirstOrd = *off;
   Rec.FirstOrd = %date();
   update NULLTESTF rec;
endif;
```

Use NULLIND in Prototypes

You can use NULLIND on prototypes/procedure interfaces…

- must specify an indicator parameter, NULLIND(field) not just NULLIND
- Indicator parameter must be passed in the same parameter list
- works with both standalone fields and data structures

```
CheckFields(rec : null);
  .
  .
dcl-proc CheckFields;
   dcl-pi *n;
      r likeds(rec) nullind(n);
      n likeds(null);
   end-pi;

   if n.AmtOwed = *off;
      dsply (%char(r.CustNo) + ' owes ' + %char(r.AmtOwed));
   endif;

   if n.SalesRep = *ON;
      dsply (%char(r.CustNo) + ' has no sales rep');
   endif;
end-proc;
```


Free Format (Definitions)

Released in 7.2, PTF back to 7.1

- CTL-OPT replaces H-spec
- DCL-S replaces D-spec for standalone variables
- DCL-F replaces F-spec
- DCL-DS replaces D-spec for data structures.
- New POS keyword sometimes nicer than OVERLAY(var:pos)
- Sequence of F and D specs no longer matters.

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
      ctl-opt dftactgrp(*no) option(*srcstmt:*nodebugio);

      dcl-s rrn packed(4: 0);

      dcl-f MYFILE workstn sfile(SFL01:rrn)
                           indds(inds);

      dcl-ds inds qualified;
         Exit     ind pos(3);
         Cancel   ind pos(12);
         ClearSFL ind pos(50);
         ShowSFL  ind pos(51);
      end-ds;
```

Free Format (More DS Options)

Released in 7.2, PTF back to 7.1

- Use *N for the DS name for an "unnamed" DS
- END-DS can go on the same line if you have no subfields.
- EXTNAME or EXT for externally defined structures.

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
      dcl-ds *N;
         field1 char(10);
         field2 packed(9: 2);
         field3 zoned(7: 1);
      end-ds;

      dcl-ds CUSTREC ExtName('CUSTMAS') end-ds;

      dcl-f PRINTFILE PRINTER(132);
      dcl-ds DATA len(132) end-ds;
       .
       .
      DATA = 'Print this';
      write PRINTFILE DATA;
```


Free Format (Procedures)

Released 7.2, PTF back to 7.1

- DCL-PROC and END-PROC replaces P-spec.
- DCL-PI and END-PI replace D-spec with PI.
- You can use *N on the DCL-PI for same name as the DCL-PROC.
- Prototypes are only needed when there's no matching PI in the same module

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

      dcl-proc myProcedure;

         dcl-pi myProcedure int(10);  ( --or--   dcl-pi *N int(10); )
            parm1 char(10) const;
            parm2 packed(7: 2) const;
         end-pi;

         dcl-s myVar like(parm2);

         return myVar;

      end-proc;
```

EXPORT(*DCLCASE)

Previously, when procedure name was case-sensitive, you had to repeat it in ExtProc:

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     D MiXeDcAsEnaMe   PR                  ExtProc('MiXeDcAsEnaMe')
     D   parm1                       10a   const

     P MiXeDcAsEnaMe   B                   Export
     D MiXeDcAsEnaMe   PI
     D   parm1                       10a   const
      /free
        ... whatever procedure does ...
      /end-free
     P                 E
```

*DCLCASE makes DCL-PROC case-sensitive. Released in 7.2, PTF back to 7.1

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
      dcl-proc MiXeDcAsEnaMe export(*dclcase);
         dcl-pi *n;
            parm1 char(10) const;
         end-pi;
         // whatever procedure does.
      end-proc;
```


EXTPROC(*DCLCASE)

Can be used to call procedures with case-sensitive names without ExtProc.
Released in 7.2, PTF back to 7.1

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

       dcl-pr MiXeDcAsEnaMe int(10) extproc(*dclcase);
         parm1 char(10) const;
       end-pr;

       MiXeDcAsEnaMe(myVariable);
```

... can also be used with APIs, such as the Unix-type ones ...

```
       dcl-pr unlink int(10) extproc(*dclcase);
         path pointer value options(*string);
       end-pr;

       unlink('/tmp/deleteMe.txt');
```

Mix Free Format With Fixed

Released in 7.2, PTF back to 7.1

- The /free and /end-free options are no longer required

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

      dcl-s Y packed(9: 2);
    D X               s              9p 2

      x = 27.50;
    C                   eval      Y = 25.50
```

- D/F spec sequence doesn't matter, even in fixed format

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
    D DATA            ds             132
    FPRINTFILE O    F  132        PRINTER
    C                   eval      DATA = 'Print Me'
    C                   write     PRINTFILE     DATA
```


"Fully Free" Columns Support

Released in 7.3, PTF for 7.1 and 7.2

- Start with **FREE keyword in col 1 of line 1 of your source
- Start with column 1, no limit in line length (aside from record length)
- No length limit on line at all when using IFS files
- cannot use fixed format statements (except via copy book)
- copy books that want **FREE also code it on col 1, line 1 of copy book.
- After line 1 starting with ** is for CTDATA

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
**FREE
dcl-s states char(12) dim(4) ctdata;
dcl-s x      int(10);

for x = 1 to %elem(states);
   dsply states(x);
endfor;

**
California
Ohio
Mississippi
Wisconsin
```

Free Format Is Awesome!

Free format for H, F, D and P specs
- Available in 7.2+, PTF available for 7.1 – no waiting!
- Much easier for new programmers to learn
- Adds new features like *DCLCASE and POS
- Fixed "bad-old" features with F/D spec order and /free /end-free
- Already extremely widely used in the RPG industry!!

The main complaint at the time this was released was
- Still limited to using columns 8 – 80
- But this was fixed later with *FREE support
- So many cool new things, I just can't handle it!

CCSIDs Are Really Important

Funny how many programmers, especially in the USA, seem to ignore CCSIDs.

You use CCSIDs every time you use text.
- Letters, symbols, punctuation, etc.
- Everything that isn't raw binary data like images, sound, or binary numbers!!
- That's the vast majority of what we do in business programming!

If you don't think about the CCSID of your data, the computer *assumes* the default.
- Shouldn't you, the programmer know what's going on?
- When the CCSID is different, don't you want to be in control of what's happening?

CCSIDs Are Not Just For Foreign Countries!!
- In the USA, we typically use CCSID 37, which is one flavor of EBCDIC
- There are many others – even the UK's is different!
- France, Germany, Italy, Spain, Mexico, French Canada, China, Korea, etc
- There are different EBCDICs for each, different ASCIIs for each

Unicode is the best!
- Most modern – today's solution, not yesterdays!
- Supports all characters for all languages all in one CCSID!

Char/Alpha CCSID Support

Released in 7.2 (Not available in 7.1)

- CCSID keyword when defining your fields
- Can be EBCDIC, ASCII or UTF-8 Unicode
- RPG now natively works with all of these character sets!!
- /SET and /RESTORE can be used to set the defaults for a group of definitions

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
dcl-s var1 char(100);                  // default is job CCSID (37 for me)
dcl-s var2 char(100) ccsid(*utf8);     // special value *UTF8 = CCSID 1208

/set ccsid(*char: *utf8)

dcl-s var3 varchar(100);
dcl-s var4 char(200);

/restore ccsid(*char)

var1 = 'MEEEE SCOTT!';                 // EBCDIC, default
var2 = var1;                           // converted to UTF-8
var3 = 'HIIIM SCOTT!';                 // also converted.
var4 = var3;                           // no conversion needed
var1 = var3;                           // converted to EBCDIC
```


More Special Values

Released in 7.2 (Not available in 7.1)

- Already saw CCSID(*UTF8) is UTF-8, same as CCSID 1208
- CCSID(*UTF16) is UTF-16 same as CCSID 1200 (for double-byte fields)
- External data structures can use CCSID(*EXACT) to match the CCSIDs of the file the definition came from.
- CCSID(*HEX) when you never want conversion between CCSIDs
- CCSID(*JOBRUN) uses the job's default CCSID
- CCSID(*JOBRUNMIX) uses the mixed CCSID related to job CCSID

```
Create Table PSNFILE (
  Name    char(30) ccsid 1208,
  Address char(40) ccsid 37
)
rcdfmt PSNFILEF;
```

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
       dcl-s var1 ucs2(10) ccsid(*utf16);
       dcl-s var2 char(10) ccsid(*hex);
       dcl-s var3 char(10) ccsid(*jobrun);

       // Name will have CCSID 1208, Address will have CCSID 37
       dcl-ds PSNFILE ext ccsid(*exact) end-ds;
```

CTL-OPT CCSID Keywords

Released in 7.2 (Not available in 7.1)

- CTL-OPT (H-Spec) CCSID keyword sets default for whole module
- You can also set CCSID(*UCS2:1200), but we've had that for a long time.

```
....+....1....+....2....+....3....+....4....+....5....+....6
     H ccsid(*char: *utf8)
      -or-
     ctl-opt ccsid(*char: *utf8);
```

Released in 7.2, PTF for 6.1, 7.1

CTL-OPT (H-Spec) CCSIDCVT keyword has two functions

- *EXCP causes RPG to send an exception (error) when a character would be lost by CCSID conversion
- *LIST puts a listing of all of the automatic CCSID conversions that are taking place into your compile listing, so it's easy to see where stuff is being translated automatically.

```
....+....1....+....2....+....3....+....4....+..
     H ccsidcvt(*list)
      -or-
     ctl-opt ccsidcvt(*excp);
```


CCSIDCVT(*EXCP)

What should happen if you convert a character that cannot exist in the result CCSID?

Default: A substitution character is inserted that shows that some character is missing.

With *EXCP: An exception/error is sent to your RPG program

```
    ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+
01  **FREE
02  ctl-opt ccsidcvt(*excp: *list);
03  ctl-opt ccsid(*CHAR: *UTF8) ccsid(*ucs2: *utf16);
04
05  dcl-s var1 ucs2(2);
06  dcl-s var2 char(10);
07  dcl-s var3 char(10) ccsid(*jobrun);
08
09  var1 = u'4f605978';     // Chinese characters for "Ni Hao" in UTF-16
10  var2 = var1;            // converted to UTF-8
11  var3 = var2;            // converted to US-EBCDIC

    RNQ0452 Some characters could not be converted from
            CCSID(1208) to CCSID(37)
```

CCSIDCVT(*LIST)

```
    ....+....1....+....2....+....3....+....4....+....5....+....6....+....7...
01  **FREE
02  ctl-opt ccsidcvt(*excp: *list);
03  ctl-opt ccsid(*CHAR: *UTF8) ccsid(*ucs2: *utf16);
04
05  dcl-s var1 ucs2(2);
06  dcl-s var2 char(10);
07  dcl-s var3 char(10) ccsid(*jobrun);
08
09  var1 = u'4f605978';     // Chinese characters for "Ni Hao" in UTF-16
10  var2 = var1;            // converted to UTF-8
11  var3 = var2;            // converted to US-EBCDIC
```

CCSIDCVT(*LIST) adds this to compile listing:

```
                  C C S I D   C o n v e r s i o n s
    From CCSID        To CCSID          References
    1200              1208                10
    1208              *JOBRUN             11
* * * * *   E N D   O F   C C S I D   C O N V E R S I O N S   * * * * *
```


Disabling Database CCSID Conversion

Historically, RPG didn't work with multiple CCSIDs in the same program.

- but the database did!
- therefore, RPG converted all database CCSIDs to/from the job's CCSID
- for backward compatibility, it still does!

Now that it does work with lot of CCSIDs (as we just discussed)

- It's often better to keep the original database CCSIDs in RPG
- We can use RPG's features to convert them if needed.

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7...
**FREE
ctl-opt openopt(*nocvtdata);           // Default for whole program

dcl-f MYFILE workstn sfile(SFL01:rrn)
02  ctl-opt ccsidcvt(*excp: *list);
03  ctl-opt ccsid(*CHAR: *UTF8) ccsid(*ucs2: *utf16);
04
05  dcl-s var1 ucs2(2);
06  dcl-s var2 char(10);
07  dcl-s var3 char(10) ccsid(*jobrun);
08
09  var1 = u'4f605978';     // Chinese characters for "Ni Hao" in UTF-16
10  var2 = var1;            // converted to UTF-8
11  var3 = var2;            // converted to US-EBCDIC
```

Expanded Timestamps

Released 7.2

You can now specify how many fractional seconds you want. You can have up to 12 digits of fractional seconds in a timestamp field.

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8

dcl-s ts   timestamp     inz(*sys);  // YYYY-MM-DD-hh.mm.ss.ffffff (default)

dcl-s ts0  timestamp(0)  inz(*sys);  // YYYY-MM-DD-hh.mm.ss
dcl-s ts1  timestamp(1)  inz(*sys);  // YYYY-MM-DD-hh.mm.ss.f
dcl-s ts3  timestamp(3)  inz(*sys);  // YYYY-MM-DD-hh.mm.ss.fff
dcl-s ts12 timestamp(12) inz(*sys);  // YYYY-MM-DD-hh.mm.ss.fffffffffffff
```

The `INZ()` keyword, `TIME` opcode and `%timestamp()` BIF still only set the first 3 fractional digits. The remainder of the timestamp is set to zeroes unless you initialize it yourself with an API or similar.

%SUBDT Digits/Decimals

Released in 7.2
The %SUBDT BIF can specify

- a number of digits returned in 3rd parameter
- a number of decimal places when working with *SECONDS

```
....+....1....+....2....+....3....+....4....+....5....+....6....+...
**FREE
dcl-s ts    timestamp inz(z'2016-09-16-18.09.33.123456');
dcl-s year4 packed(4: 0);
dcl-s year2 packed(2: 0);
dcl-s secs  packed(4: 2);

year4 = %subdt(ts:*years:4);      //  year4 = 2016
year2 = %subdt(ts:*years:2);      //  year2 = 16
secs  = %subdt(ts:*seconds:4:2); //  secs = 33.12
```

Other Misc 7.1-7.2 Enhancements

There are so many enhancements, I don't even have time to cover them all in-depth! Here are some that I didn't cover in this talk:

Not covered, because these are already well known:

- XML-INTO was enhanced with countprefix, datasubf (7.1, ptf for 6.1)
- XML-INTO support for case=convert and namespaces (7.2 ptf for 7.1, 6.1)
- Open Access and the HANDLER keyword (7.2 ptf for 7.1, 6.1)

Not covered because "too trivial"

- PGMINFO can be used to control which procedures are included in PCML
- VALIDATE(*NODATETIME) slightly improves date/time performance by eliminating error checking (don't use this, please)
- DCLOPT(*NOCHGDSLEN) allows %SIZE to figure out the size of a data structure in some definitions by guaranteeing that input, output and calcs won't change the length.

Soooo Many Enhancements!

There have been 30 enhancements to RPG the past 5 years

…and around 82 since SEU stopped being enhanced (yet, people still use it!)

That does not include other enhancements that we all use:

- RDi, Db2, SQL Services, HTTP server, Integrated Web Services, etc, etc.

That's a lot of new stuff!

During 7.1 Development Cycle

	6.1	7.1
Sort and Search Qualified DS Arrays		X
Sort in Ascending or Descending		X
%SCANRPL BIF		X
%LEN(*MAX)		X
ALIAS Support		X
RTNPARM		X
%PARMNUM		X
Prototypes Are Optional		X
Procedures Convert CCSIDs when CONST		X
Teraspace Storage model		X
Default ACTGRP is now *STGMDL		X
H-spec option to change ALLOC to teraspace		X
Encrypted Debugging View		X
XML-INTO datasubf and countprefix	ptf	X

All features on this chart are available in 7.1+

During 7.2 Development Cycle

	6.1	7.1	7.2
XML-INTO namespaces options	ptf	ptf	X
XML-INTO case=convert	ptf	ptf	X
CCSIDCVT keyword to list or give exceptions during auto-converts	ptf	ptf	X
Date/Time efficiency VALIDATE(*NODATETIME)	ptf	ptf	X
Open Access	ptf	ptf	X
CCSID support on A data type (and UTF-8)			X
CCSID(*EXACT)			X
CCSID(*HEX) and hex literals			X
Conversion During Concatenation			X
OPENOPT(*NOCVTDATA) and DATA(*NOCVT)			X
/SET and /RESTORE			X
Control %SUBDT Length			X
Timestamps up to 12 fractional digits (why??)			X
Free-Format H, F, D and P specs		ptf	X
No more need for /free and /end-free		ptf	X
File/Definitions Can be In Any Sequence (Fixed/Free)		ptf	X
EXPORT(*DCLCASE)			X

All features on this chart are available in 7.2+

During 7.3 Development Cycle

	6.1	7.1	7.2	7.3
**FREE ("fully free form")		ptf	ptf	X
%SCANR BIF				X
Length Parameter to %SCAN				X
Improved ALIAS support		ptf	ptf	X
Relaxed DS I/O rules		ptf	ptf	X
Improvements to DB Null Support -- NULLIND				X
Improvements to PCML generation		ptf	ptf	X
DCLOPT(*NOCHGDSLEN)		ptf	ptf	X

All features on this chart are available in 7.3+

During 7.4 Development Cycle

	7.1	7.2	7.3	7.4
Compile from Unicode Source (TGTCCSID compile keyword)	ptf	ptf	ptf	X
PCML 7.0 Support		ptf	ptf	X
ALIGN(*FULL)		ptf	ptf	X
%MIN and %MAX BIFs		ptf	ptf	X
Support for Complex Qualified Names in %ELEM, %SIZE, DEALLOC and RESET		ptf	ptf	X
%PROC BIF		ptf	ptf	X
DATA-INTO opcode		ptf	ptf	X
Nested Data Structures		ptf	ptf	X
ON-EXIT section		ptf	ptf	X
PSDS subfields for internal job id and system name		ptf	ptf	X
SAMEPOS DS keyword			ptf	X
DIM(*CTDATA)				X
Varying-dimension arrays				X

All features on this chart are available in 7.4+

During 7.5 Development Cycle

	7.3	7.4	7.5
DATA-GEN	ptf	ptf	X
OVERLOAD	ptf	ptf	X
OPTIONS(*EXACT)	ptf	ptf	X
LIKEDS(qualified.name.here)	ptf	ptf	X
%KDS with variable key count	ptf	ptf	X
%TIMESTAMP with microseconds and *UNIQUE	ptf	ptf	X
DEBUG(*RETVAL)	ptf	ptf	X
REQPREXP(*REQUIRE | *WARN)	ptf	ptf	X
%RANGE	ptf	ptf	X
%LIST	ptf	ptf	X
FOR-EACH loops	ptf	ptf	X
EXPROPTS(*ALWBLANKNUM or *USEDECEDIT) for %DEC, %INT, etc	ptf	ptf	X
EXPROPTS(*STRICTKEY)	ptf	ptf	X
%UPPER and %LOWER	ptf	ptf	X
%SPLIT	ptf	ptf	X
%MAXARR and %MINARR	ptf	ptf	X
SORTA with multiple %FIELDS	ptf	ptf	X
DEBUG(*CONSTANTS)	ptf	ptf	X
SND-MSG	ptf	ptf	X
ON-EXCP	ptf	ptf	X

During i+1 (7.6?) Development Cycle

	7.3	7.4	7.5
OPTIONS(*CONVERT)		ptf	ptf
%CONCAT and %CONCATARR		ptf	ptf
CHARCOUNT, CHARCOUNTTYPES and *NATURAL mode		ptf	ptf
PCML 8.0 (with boolean=true)	ptf	ptf	ptf
CRTSQLRPGI RPGPPOPT with long lines (PPMINOUTLN)	ptf	ptf	ptf
%PASSED and %OMITTED		ptf	ptf
SELECT with expression, WHEN-IS, WHEN-IN		ptf	ptf
*ALLSEP option for %SPLIT		ptf	ptf

Get The Latest PTFs

You can install any of the features (if available for your release of IBM i) by installing these PTFs.

There is no need to install separate PTFs for each feature, these PTFs (and their pre-requisites) include it all.

Check RPG Café for the latest features:
https://ibm.biz/rpg_cafe

7.3	
Compiler TGTRLS(*CURRENT)	SI83429
QOAR source files	SI71517
Runtime	SI83428
Debugger	SI81625
7.4	
Compiler TGTRLS(*CURRENT)	SI84886
Compiler TGTRLS(V7R3M0)	SI83483
QOAR source files	SI71518
Runtime	SI83471
Debugger	SI81626
7.5	
Compiler TGTRLS(*CURRENT)	SI85009
Compiler TGTRLS(V7R4M0)	SI85043
Compiler TGTRLS(V7R3M0)	SI83494
Runtime	SI83468

This Presentation

You can download a PDF copy of this presentation:

http://www.scottklement.com/presentations/

Thank you!