IBM Software Group

What's new for RPG in 7.1

Barbara Morris IBM
Craig Jacquez ServIT

Agenda

- **Overview of what's new for RPG since 6.1, through PTFs**
- Overview of what was new for RPG in 7.1

ILE RPG PTF enhancements for 6.1 and 7.1

- Several new XML-INTO options
- Performance option for date/time/timestamp
- Get warnings or exceptions for failed CCSID conversions

PTFs for 6.1 (RPG runtime and *CURRENT compiler):

The latest supersedes to SI47610 and SI46495

PTFs for 7.1 (RPG runtime and *CURRENT/*PRV compilers):

The latest supersedes to SI47646, SI45902, and SI46563

6.1 & 7.1 PTFs: New XML-INTO options:

The first set of PTFs introduced

- datasubf

```
<emp type="reg" id="13573">John Smith</emp>
```

  Without datasubf, XML-INTO could not get the "John Smith" data from this XML fragment

- countprefix
  - When an RPG array was used to capture repeated XML data, and the DIM was set to the maximum possible
  - When an XML tag may or may not appear in a particular XML document

  Without countprefix, `allowmissing=yes` was necessary. It allowed everything to be missing

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

<emp type="reg" id="13573">John Smith</emp>

Problem : For XML-INTO, "emp" has to be an RPG data structure with subfields "type" and "id" to receive "reg" and "13573". But where does "John Smith" go?

```
D emp                   ds
D    type                       10a   varying
D    id                          5p 0
```

Solution : The datasubf option lets you tell XML-INTO the name for any subfields that are intended to receive text data for a data structure.

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

<emp type="reg" id="13573">John Smith</emp>

Add another subfield to receive the John Smith data.

```
D emp                   ds
D    type                      10a   varying
D    id                         5p 0
D    val                       25a   varying
```

Use the datasubf option to tell XML-INTO the name of the subfields to handle XML data for a data structure

```
xml-into emp %xml(xmldoc
                  : 'doc=file datasubf=val');
```

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

```
<dept>
   <manager>John Smith</manager>
   <emp>Mary Jones</emp>
   <emp>Sam Thompson</emp>
</dept>
```

Problem : The “emp” RPG subfield must be an array big enough to hold the maximum number.

```
D dept                ds
D    manager                25a   varying
D    emp                    25a   varying DIM(20)
     xml-into dept %xml(xmldoc
                        : 'doc=file allowmissing=yes');
```

Option allowmissing=yes is necessary if there are less than 20 in the XML document.

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

But option allowmissing=yes allows everything to be missing. There would be no error for the following XML document.

```
<dept>
</dept>
```

Solution : Option countprefix lets you add “counter” subfields to your RPG data structure. With countprefix, you can remove the allowmissing option and control exactly what you want to be optional.

countprefix gives the prefix for subfield names that are used as counters. With countprefix=n, the counter for “emp” is “nemp”.

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

```
<dept>
   <manager>John Smith</manager>
   <emp>Mary Jones</emp>
   <emp>Sam Thompson</emp>
</dept>
```

```
D dept                ds
D    manager                25a   varying
D    emp                    25a   varying DIM(20)
D    numEmp                 10i 0
     xml-into dept %xml(xmldoc
           : 'doc=file countprefix=num');
```

The “numEmp” subfield will receive the value 2. This makes it easy to know how many array elements to process.

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

You can use countprefix with non-arrays too, to allow the XML document to omit a particular element.

If some documents have a “secretary” tag, you can add a secretary subfield, and a numSecretary to make it optional in the XML document.

```
D dept                ds
D  manager                 25a   varying
D  secretary               25a   varying
D  numSecretary            10i 0
```

```
     xml-into dept %xml(xmldoc
                        : 'doc=file countprefix=num');
     if numSecretary = 1;
        // the XML document had the <secretary> tag
```

6.1 & 7.1 PTFs: New XML-INTO options: PTFs for 6.1 and 7.1

The second set of PTFs introduced

- ns and nsprefix

```
<emp employee:type="regular" employee:id="13573">
```

  Without the ns option, XML-INTO could not match names like "employee:type" and "employee:id" to RPG subfield names

- case=convert

```
<Étudiant Pre-nom="Élise" Âge="12">
```

  Without the case=convert option, XML-INTO could not match names Étudiant and Âge to RPG subfield names

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>
```

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=remove');
// 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, 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: Performance option for date/time/timestamp

Date, time, and timestamp (DTZ) operations can be costly because the value of a DTZ field is validated every time the field is used.

New H spec keyword

```
VALIDATE(*NODATETIME)
```

This keyword allows the RPG compiler to treat date, time, and timestamp data as though it were alphanumeric data when it is convenient for the compiler.

6.1 & 7.1 PTFs: Performance option for date/time/timestamp

If this keyword is coded, the compiler may skip the validation step for some operations.

Warning:

This means that incorrect data will not always be detected, and may be propagated to other date, time, timestamp fields

Recommendation:

Only use this option in modules where you **never** have date, time, or timestamp errors.

Use the TEST operation to check a field before it is used. The TEST operation will always validate, independent of the VALIDATE keyword.

6.1 & 7.1 PTFs: Performance option for date/time/timestamp

Question :

What is currently affected by this keyword?

Answer:

- Moving data between values with the same format and separator
  - Assignments (EVAL, MOVE etc)
  - Moving data between fields and I/O buffers on I and O specs
- Comparison between values with the same format and separate AND where the date is formatted as yyyy mm dd, or where the time is formatted as hh mm ss.

6.1 & 7.1 PTFs: Performance option for date/time/timestamp

Question :

Might other operations and formats be affected by VALIDATE(*NODATETIME) in the future?

Answer:

Yes. By coding this keyword, you give the RPG compiler permission to skip the validation step for any date, time, timestamp operation.

6.1 & 7.1 PTFs: Performance option for date/time/timestamp

Question :

Is the performance benefit significant?

Answer:

Yes, for a single operation.

But it is normally only noticeable if you have a significant proportion of date, time, timestamp operations compared to the number of I/O operations.

6.1 & 7.1 PTFs: Warnings or exceptions for CCSID conversions

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

You will need to add messages RNX0452 and RNQ0452 to you 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.

Agenda

- Overview of what's new for RPG since 7.1, through PTFs
- **Overview of what was new for RPG in 7.1**

ILE RPG enhancements for 7.1

- **Open Access: RPG Edition**
- **Enhancements for arrays**
  - Sort and search a data structure array
  - Sort arrays either descending or ascending
- **Enhancements for defining procedures**
  - Optional prototypes
  - One string procedure to handle any string type
  - Fast return values
  - Soft-code parameter numbers
- **Alias names in data structures**
- **Miscellaneous**
  - Built-in function to scan and replace
  - Encrypted debug view
  - Teraspace storage model
  - SEU syntax checker frozen at the 6.1 level

7.1: Open Access: RPG Edition

Open Access provides a way for RPG programmers to use the simple and well-understood RPG I/O model to access resources and devices that are not directly supported by RPG.

- Web
- Mobile phones
- IFS files
- Data queues
- Etc etc etc

Instead of the system handling the I/O, instead, a “handler” program or procedure handles the I/O requests.

7.1: Open Access: the RPG coding

The RPG coding to use Open Access is simple. Just add the HANDLER keyword to the F spec.

```
FmyScreen CF   E    WORKSTN HANDLER('HDLR')
```

The parameter for the HANDLER keyword can be

- A program

```
HANDLER('MYLIB/MYHANDLER')
HANDLER('MYHANDLER')
```

- A procedure in a service program

```
HANDLER('MYLIB/MYSRVPGM(myHandler)')
HANDLER('MYSRVPGM(myHandler)')
```

- A character variable, with one of those values set at runtime

```
HANDLER(myField)
```

7.1: Open Access: the handler

What is more complex is the handler itself.

The handler must do all the work to perform the required I/O.

For a READ operation, the handler must get the data from the device or resource it is working with, and then transform the data into the form required by the RPG program.

For example, the handler might get the data like this:

```
item: Hammer
cost: 200.51
```

It must transform the data into the Input Buffer subfield of the handler parameter, in the data types used by the RPG program

```
0002005100Hammer
```

7.1: Open Access: the handler

Handlers are not shipped as part of Open Access.

You have to write the handlers yourself, or more likely, purchase them. Here are some companies that provide Open Access handlers

Handlers for modernizing WORKSTN files

- ProfoundLogic
- Look Software
- ASNA Wings

Handlers for working with alternate databases

- RJS Software

If you want to try writing your own Open Access handler, the documentation for writing handlers is in the RPG Café, and in the 7.1 Info Center under the RPG part of the Programming topic.

Licensing change for Open Access: RPG Edition

You may have heard that Open Access is a separate product that you have to buy. That has changed.

On January 31, 2012, IBM announced

- Open Access is available as part of the RPG compiler
- No longer dependent on the 5733-OAR product
- The copy files in library QOAR become part of the WDS product

- See the RPG Café for full details on the PTFs for 6.1 or 7.1 http://tinyurl.com/rpg-oar-ptfs

RPG: Sort and search a data structure array

Sort a data structure array using one subfield as a key

```
// sort the info array by name
SORTA info(*).name;

// sort the info array by dueDate
SORTA info(*).dueDate;
```

Search a data structure array using one subfield as a key

```
// search for 'Jack' in name
pos = %LOOKUP('Jack' : info(*).name);

// search for today's date in dueDate
pos = %LOOKUP(%date() : info(*).dueDate);
```

RPG: Sort and search a data structure array

If you have a complex data structure with nested arrays

An array of family that has sub-arrays of child

```
family(x).child(y)
```

Then all except one of the arrays must have a “real” index.

- The part up to the (*) index indicates which array will be sorted.
- The part after the (*) index indicates the “key” for sorting.

Sort the child array in one of the family elements by the age of the children:

```
family(2).child(*).age
```

Sort the family array by the first child’s age

```
family(*).child(1).age
```

RPG: Example of sorting a complex data structure array

name	numChild	child	
		name	age
Smith	2	Sally	12
		Jimmy	2
Jones	3	Polly	9
		Andy	5
		Mary	11

RPG: Example of sorting a complex data structure array

```
 * A type definition for a child
D child_t      ds             qualified template
D   name                25a   varying
D   age                 5i 0
 * The family array. Each element has a child array.
D family       ds             qualified dim(5)
D   name                25a   varying
D   numChild             5i 0
D   child                      likeds(child_t) dim(10)
```

name	numChild	child	
		name	age
Smith	2	Sally	12
		Jimmy	2
Jones	3	Polly	9
		Andy	5
		Mary	11

RPG: Example of sorting a complex data structure array

```
// sort the child arrays by age, oldest first
for i = 1 to numFamily;
    SORTA(D) %SUBARR (family(i).child(*).age
                     : 1 : family(i).numChild);
endfor;
```

Sort

name	numChild	child	
		name	age
Smith	2	Sally	12
		Jimmy	2
Jones	3	Mary	11
		Polly	9
		Andy	5

RPG: Example of sorting a complex data structure array

```
// sort the family array by age of first
  child
SORTA family(*).child(1).age;
```

Sort `family` by `child(1).age`, ascending

name	numChild	child	
		name	age
Jones	3	Mary	11
		Polly	9
		Andy	5
Smith	2	Sally	12
		Jimmy	2

RPG: Sort ascending or descending

Non-sequenced arrays can be sorted either ascending or descending.

```
D meetings        S         D  DIM(100)
 /free
      // sort descending, with the
      // most recent date first
      sorta(d) meetings;
```

(D) extender indicates a descending sort.

(A) Extender indicates ascending (default).

RPG: Optional prototypes

If a program or procedure is not called by another RPG module, it is optional to specify the prototype.

These are some of the programs and procedures that do not require an RPG prototype

- An exit program, or the command-processing program for a command
- A program or procedure that is never intended to be called from RPG
- A procedure that is not exported from the module

RPG: Optional prototypes

Rules:

1. If an RPG procedure is called from another RPG module, it must have a prototype
2. All modules either calling the procedure and the module that defines the procedure must all use the same prototype (use a /COPY file)

The RPG compiler cannot enforce these rules

But they are simple to follow if you remember one of the purposes of a prototype:

The prototype ensures that the callers of a procedure pass the parameters according to the way the procedure expects them them to be passed.

RPG: Optional prototypes: Example

```
H main(hello)

P hello                 b
 /free
   dsply ('Hello ' + getName());
 /end-free
P hello                 e

P getName               b
D                       pi                10a
D ans                   s                 10
 /free
   dsply ('What is your name?') '' ans;
   return ans;
 /end-free
P getname               e
```

RPG: Implicit CCSID conversion for parameters

Previous support : implicit conversion between the different string types (alpha, unicode, dbcs) for assignment

New support : implicit conversion on parameter passing

- Enables writing a single procedure that can handle any string type.
- The procedure is written to have Unicode parameters and a Unicode return value, and the RPG compiler handles any necessary conversions.

```
// makeTitle() upper-cases and centers the parameter
alphaTitle = makeTitle(alphaValue : 50);
ucs2Title = makeTitle(ucs2Value : 50);
dbcsTitle = makeTitle(dbcsValue : 50);
```

RPG: Performance returning large values

If you have a lot of calls to procedures that return large values, performance can be noticeably poor

```
title = center(getDesc(id));
```

One possible solution is to change the procedures so they use a parameter instead of a return value

```
getDesc(tempDesc : id);
center(tempTitle : tempDesc);
title = tempTitle;
```

But that is awkward. The temp fields have to defined exactly the same as the parameters.

RPG: Performance returning large values

Solution: use the RTNPARM keyword.

The RPG compiler changes the return value to be an extra parameter.

- The speed of using a parameter with the convenience of using a return value
- Especially noticeable when the prototyped return value is a large varying length value

```
D center           pr          100000a   varying
D                                        rtnparm
D   text                        50000a   const varying
D   len                           10i 0  value
D title            s             100a   varying
 /free
   title = center ('Chapter 1' : 60);
```

RPG: Performance returning large values

RTNPARM is internal to RPG.

If you want to call a RTNPARM procedure from another language, you must define it in the other language as though it has an extra parameter, and no return value.

```
D getDesc          pr          1000a   rtnparm
D   id                            9p 0 const
```

To call this from CL, add the “desc” parameter first:

```
dcl &id type(*dec) len(9 0)
dcl &desc type(*char) len(1000)

callprc getDesc parm(&desc &id)
```

RPG: Softcode the parameter number

The %PARMNUM built-in function returns a parameter's position in the parameter list.

```
D  myProc          pi        10A   OPDESC
D    company                 25A
D    city                    25A   OPTIONS(*VARSIZE)
```

Problems solved by %PARMNUM:

- Pass a parameter number to a Parameter-Information API

```
CEEDOD (2 : more parms);     // hard to understand
CEEDOD (%PARMNUM(city) : more parms);  // better
```

- Check to see if the number of passed parameters is high enough for a particular parameter

```
if %parms >= 1;                    // hard to understand
if %parms >= %PARNUM(company);           // better
```

RPG: %PARMNUM is imperative with RTNPARM

When a procedure is defined with RTNPARM

- The return value is handled as an extra parameter under the covers
- The extra parameter is the first parameter
- %PARMS and the parameter APIs use the true number
- The apparent parameter number is off by one

```
D  myProc          pi       10A   RTNPARM
... RTNPARM hidden parameter
D   parm1                   25A
D   parm2                   25A
```

The ‘parm2’ parameter looks like the second parameter, but it is actually the third parameter.

%PARMNUM must be used with %PARMS or the CEE APIs that take parameter numbers.

RPG: Support for ALIAS names

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.

RPG: Support for ALIAS names in data structures

New ALIAS keyword for RPG

- When ALIAS is specified, RPG will use the alternate name instead of the 10-character standard name.
- Supported on D specs for any externally-described data structure.
- Supported on some F specs, and then used for LIKEREC data structures.
  - Supported for qualified files or local files in subprocedures.
  - Not supported for 99.99% of your files (unfortunately).

The subfields of the data structure will have the alternate names instead of the standard name.

RPG: Support for ALIAS names

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


D custDs          e ds          ALIAS
D                               QUALIFIED EXTNAME(custFile)
/free
    custDs.customer_name = 'John Smith';
    custDs.customer_address = '123 Mockingbird Lane';
    custDs.id = 12345;
```

RPG: New built-in function %SCANRPL

The %SCANRPL built-in function replaces all occurrences a string with another string.

```
fileErr = 'File &1 not found. Please create &1.';
msg = %scanrpl ('&1' : filename : fileErr);

// msg = 'File MYFILE not found. Please create MYFILE.'
```

Problem solved by %SCANRPL:

Hand-written versions of scan-and-replace tend to be large, error prone, and difficult to maintain.

Other 7.1 enhancements

A couple of enhancements that are not just for RPG

- Encrypted debug view
- Teraspace storage model

Encrypt the debug listing view (all ILE compilers)

The problem:

- You want to ship a debuggable version of your application to your customers, but you don't want them to be able to read your source code through the debug view

The solution:

- Encrypt the debug view so that the debug view is only visible if the person knows the encryption key.

```
==> CRTBNDRPG MYPGM DBGENCKEY('my secret code')
```

- Then either

```
==> STRDBG MYPGM DBGENCKEY('my secret code')
```

- Or

```
==> STRDBG MYPGM
```

  and wait to be prompted for the encryption key

Teraspace storage model (RPG and COBOL)

The problems:

- 16MB automatic storage limits with the single-level storage model, for a single procedure, and for all the procedures on the call stack
- RPG's %ALLOC and %REALLOC have a 16MB limit

Teraspace storage model (RPG and COBOL)

The solution: use the teraspace storage model

- Much higher limits for automatic storage.
- Compile with STGMDL(*TERASPACE) to always use the teraspace storage model
- Can compile *CALLER and programs and service programs with STGMDL(*INHERIT) so they can be called from either single-level or teraspace programs

Teraspace storage model (RPG and COBOL)

Challenges if you change to use the teraspace storage model

- Any one activation group can only use one storage model.
  - Either change every program and service program that uses an activation group to be STGMLD(*TERASPACE) or STGMDL(*INHERIT)

  Or
  - Split up the programs and service programs into two different activation groups, say MYACTGRP and MYACTGRPTS.
    - This requires careful analysis of overrides, commitment control, shared files etc

Teraspace storage model (RPG and COBOL)

Larger RPG allocations can be used without completely changing to the teraspace storage model:

RPG's %ALLOC and %REALLOC can allocate teraspace with a much higher limit

- Teraspace allocations are the default in the teraspace storage model
- Specify H-spec ALLOC(*TERASPACE) to have teraspace allocations in any storage model

Go to IBM