System i Developer

The Latest in RPG: It just keeps growing and growing and …

Jon Paris

Jon.Paris @ Partner400.com
www.Partner400.com
www.SystemiDeveloper.com
authory.com/JonParisAndSusanGantner

Partner400

Notes

In recent years RPG has continued to add more and more new features including the latest 7.4 enhancements

Jon Paris is the co-founder, along with his partner Susan Gantner, of Partner400, a firm specializing in customized education and mentoring services for IBM i developers. Together with Paul Tuohy, they also operate the System i Developer education consortium which runs the RPG & DB2 Summit conferences. See systemideveloper.com/Summit/conferences.html.

Jon's career includes a number of years with IBM, including working at both the Rochester and Toronto laboratories.During his time at IBM Jon was one of the "Fathers" of RPG and a major contributor to its design.

Together with Susan, Jon regularly authors regular technical articles for the IBM publication, IBM Systems Magazine and the companion electronic newsletter Extra. You may view their past articles at ibmsystemsmag.com/authors/jon-paris/

You can also find all of the articles that Jon and Susan have published in recent years at their Authory site. Check it out here: https://authory.com/JonParisAndSusanGantner

Most Recent RPG Enhancements

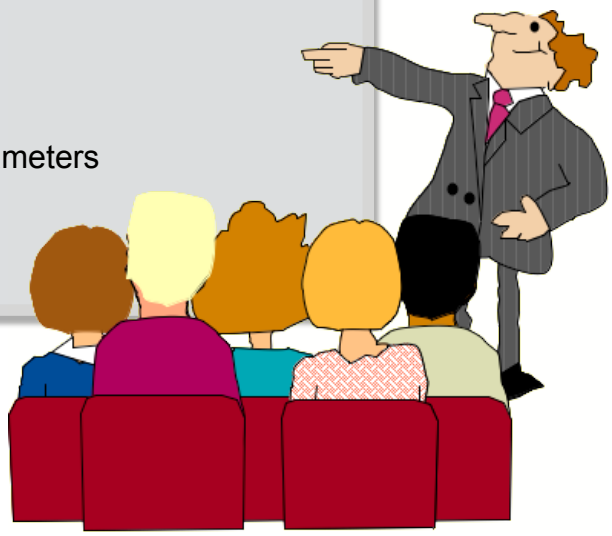
Partner400

Latest Enhancements for 7.4

- Increased accuracy for timestamps
- Added flexibility for %KDS and LikeDS

Earlier 7.4 features

- Variable Dimension Arrays
- New DS keyword SAMEPOS
- PSDS enhancements
- Procedure Overloading
- New Options(*Exact) for parameters
- DATA-GEN



Notes

Partner400

We don't have time to go into all of these features in depth but we have written articles on many of them and of course you can find more about any that I skip by checking out the 7.4 RPG manual in the Information Center.

Enhancements for Timestamps

Previously RPG timestamps were limited to milliseconds

- Not nearly granular enough for today's powerful systems

Now they default to microseconds

- But you can restrict them to the millisecond level
  - %Timestamp(*Sys : 3)

Microseconds still not sufficient to ensure a unique value ?

- Specify %Timestamp(*Unique) and RPG will add an additional 6 digits to create a unique value
- DO NOT use such values directly in duration calculations
  - Use %Timestamp(*timestampField* : 6) to restrict the calculation to the microseconds and ignore the "uniqueness digits"

Notes

SQL has provided microsecond precision for a while now and RPG has now caught up.

Do note that the default has changed - something that rarely happens in RPG. If you want the old millisecond precision you must ask for it.

Enhancement for %KDS

Partner400

%KDS can now use a variable for the number of keys to be used

- Previously you had to hard code a value
  - See example below

Syntax is %KDS(keyDSName : numberOfKeys)

```
// This code:
Select;
  When numberOfKeys = 1;
     Chain %KDS( keyDSName : 1 ) PartMaster;
  When numberOfKeys = 2;
     Chain %KDS( keyDSName : 2 ) PartMaster;
  When numberOfKeys = 3;
     Chain %KDS( keyDSName : 3 ) PartMaster;
EndSl;

// Can now be replaced by this:
Chain %KDS( keyDSName : numberOfKeys ) PartMaster;
```

Notes

Partner400

Personally I don't use %KDS a lot - I prefer to simply specify the required key fields in parentheses. I may use it more now with this added flexibility.

Enhancement for LIKEDS

7.3 and 7.4

Partner400

Until now LIKEDS could only reference a simple DS name

- This sometimes forced you to code additional TEMPLATES

You can now use a qualified DS name

- Much simpler when passing a DS array element as a parameter

```
// Assume this DS was filled in by XML-INTO or DATA-INTO
Dcl-DS  Orders  Qualified;
   num_CustOrder  int(5);
   Dcl-DS  CustOrder  Dim(50);
      CustomerId  Char(8);
      num_Item  int(5);
      Dcl-DS  Item  Dim(99);
         productCode  char(10):
         quantity     packed(7);
      End-DS;
   ...

// We can now easily define Item as a parm to the validation routine
Dcl-Pr  validItem  ind;
   itemToCheck  LikeDS(Orders.CustOrder.Item);
End-Pr;
```

Notes

Partner400

This version of the example - includes a call to the validation routine

```
// Assume this DS was filled in by XML-INTO or DATA-INTO
Dcl-DS  Orders  Qualified;
  num_CustOrder  int(5);
  Dcl-DS  CustOrder  Dim(50);
    CustomerId  Char(8);
    num_Item  int(5);
    Dcl-DS  Item  Dim(99);
      productCode  char(10);
      quantity     packed(7);
    End-DS;
  End-Ds;
End-DS;

Dcl-Pr  validItem  ind;
  itemToCheck  LikeDS(Orders.CustOrder.Item);
End-Pr;

// Validate an individual item
If validItem( Orders.CustOrder(c).Item(i) );
  // Process the item
Else;
  //  Report validation failure.
EndIf;
```

DATA-GEN

Partner400

IBM's response to requests for a "Reverse XML-INTO"

- i.e. Something that could convert a data structure into an XML document

But that would be a short term solution

- JSON has already overtaken XML for many uses
  - And who is to say what the next big thing will be?

So IBM made DATA-GEN far more flexible

- It can generate any kind of data
  - CSVs, JSON, HTML, XML, etc.
- Whatever the generator chooses to produce

In this respect it is like DATA-INTO

- You supply a generator to produce exactly what you need

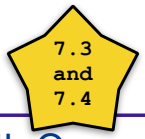
Notes

Partner400

One of the most requested features for RPG in recent years was for IBM to provide an operation that would reverse functionality of XML-INTO. XML-FROM was often suggested as the opcode name. The idea being that whereas XML-INTO decomposes an XML document and unloads it into a DS, "XML-FROM" would do the opposite and take the data in a DS and format it as an XML document. There are a number of potential problems in providing such a facility, not the least being that XML is just one of many potential output formats. What about JSON? Or HTML, or CSVs?

So IBM have chosen not to provide "XML-FROM" - rather they have given us something far more valuable. DATA-GEN. DATA-GEN follows the same path that IBM took with DATA-INTO. i.e. They have provided an opcode framework into which users can plug their own generator. This opens the door to generating any document type you like - including company specific data interchange formats, or for interchange standards that have not even been invented yet!

DATA-GEN



Partner400

DATA-GEN has some similarities with Open Access

- RPG feeds the generator a field at a time
  - Giving it the name, value, data type and other essential information such as whether it is a DS, an array element, etc.
- The generator "wraps" the data and passes it back to RPG
- RPG then adds it to the file or variable being built

DATA-GEN can work in an additive mode

- Allowing you to build the data in pieces
  - And adding them to the file or field being constructed

IBM supply sample generators for XML and HTML

- But they should not be considered production ready code

Scott Klement has added a JSON generator to his YAJL toolkit

- It is blazingly fast and offers a number of useful options

Notes

Partner400

Among the generator samples that IBM is supplying are a simple generator that produces HTML, and one that generates XML. As usual these are intended primarily as demonstrations and are not production level code. In time more complex generators will be forthcoming from third parties. Scott Klement has already added an excellent JSON generator to his YAJL toolkit, just as he added a JSON parser for DATA-INTO.

We wrote an article on this topic and other 7.4 features which you can read here: https://ibmsystemsmag.com/Power-Systems/10/2019/Procedure-Overloading-DATA-GEN

In addition I recently published two articles which demonstrate the use of the capabilities of DATA-INTO and DATA-GEN when combined with the YAJL toolkit. Find them here https://ibmsystemsmag.com/Power-Systems/04/2020/data-gen-web-services and here https://ibmsystemsmag.com/Power-Systems/06/2020/diving-deeper-data-into

DATA-GEN - Example using YAJLDTAGEN

The objective is to generate the web service request shown on the left

- The DS needed to achieve this is shown on the right

This is the DATA-INTO operation to generate the JSON

```
Data-Gen  Request
          %Data( JSONRequestText : 'countprefix=num_' )
          %Gen('YAJL/YAJLDTAGEN');
```

```
{
  "QuotationId": "A12345",
  "CustomerNumber":123456,
  "CustomerClass":"D2",
  "LineItems": [ {
           "ItemNumber":"1",
           "Quantity":2
                 },
          ... repeat item as
              needed ...
                ]
}
```

```
dcl-ds Request qualified  Inz;

   QuotationId      char(6);
   CustomerNumber   char(6);
   CustomerClass    char(2);
   num_LineItems    int(5);

   dcl-ds  LineItems Dim(50);
      ItemNumber         char(10);
      Quantity           int(5);
   end-ds  LineItems;

end-ds  Request;
```

Notes

This example is taken from an article I wrote for IBM Systems Magazine - you can find it here:

https://ibmsystemsmag.com/Power-Systems/04/2020/data-gen-web-services

The code is downloadable - the link is in the article.

The point of the num_LineItems element is that, in combination with the %Data option countprefix=num_, it controls how many instances of the lineItem elements are generated. Without it DATA-GEN would output a full 50 elements (i.e. the DIM(50) value), many of those of course would be empty and not make the receiving web service at all happy!

DATA-GEN - Building The DS

As with XML-INTO and DATA-INTO the hardest part can be designing the DS

- Scott Klement has made this easier by supplying YAJLGEN

This program generates a DS from a sample of the JSON

- It generates a whole skeleton program including the required DATA-GEN

```
// FIXME:
//   - The field lengths (varchar/packed) are guesses
//       and should be adjusted based on your business rules.
//   - The array lengths (dim keywords) are also guesses
//       and should be adjusted based on your business rules

dcl-ds jsonDoc qualified;
   QUOTATIONID varchar(6) inz('');
   CUSTOMERNUMBER packed(6) inz(0);
   CUSTOMERCLASS varchar(2) inz('');
   num_LINEITEMS int(10) inz(0);
   dcl-ds LINEITEMS dim(3);
      ITEMNUMBER varchar(1) inz('');
      QUANTITY packed(1) inz(0);
   end-ds;
end-ds;
```

Notes

This is only part of what is generated by YAJLGEN - the full source is a complete program that processes the JSON file and populates the DS.

As you can see all field names are generated in upper-case which makes it a little ugly but ...

You also need to check the assumptions re data type and size made by the tool. You will probably need to change the Dim() sizes too. The assumptions made by the program are based on the first element of a specific name encountered so sizes will always need to be checked.

SAMEPOS - Redefine DS Fields

Partner400

Until now if you wanted to redefine part of a DS

- You had to use the keyword OVERLAY
  - But ... the new field had to be contained within the overlaid field
- You could of course use absolute positioning - POS keyword
  - But we'd have to fire you as using a hard coded position is bad practice

Now you can use SAMEPOS (fieldname)

- It simply defines where in the DS the field starts
- It does not constrain the definition to the size of the original

Lots of interesting uses - for example:

- Creating S/36 style record layouts without I-Specs
- Redefining contiguous sets of values as arrays
  - Yes I know you could do this before but not this easily
- Creating a single field across set of fields
  - For example a date field from individual Day, Month and Year fields

Notes

Partner400

SAMEPOS is another step on the road to eliminating the need for hard coded positions in Data Structures.

RPG IV did away with the need for the old From/To style notations but there have always been circumstances when absolute positioning was still needed. I don't have a problem with POS(1) to indicate that the field definition starts at the first position in the DS or indeed POS(95) when I am giving a name to *IN95. Those are obvious and, more importantly will never change. But using Pos to position within a DS when the position may have to change in the future due to field expansion or addition is just an accident waiting to happen. The fact that, unlike Overlay, the new field can be larger that the one it s based on is an added advantage.

SAMEPOS - Examples

7.4 and 7.3

First example redefines non-normalized record data as an array

- Nearly all of us have some legacy records like this

Second example groups a date's components into a single field

- Also useful when date components are separate in external descriptions

See the Notes page for the SALESDATA record layout

```
Dcl-Ds  SalesData  ExtName('PARTNER400/SALESDATA');
   MonthlySales  Like(JanSales)  Dim(12)  SamePos(JanSales);
End-Ds;

Dcl-Ds  EmployeeData;
   FirstName    Varchar(20);
   LastName     Varchar(20);
   BirthMonth   Zoned(2);
   BirthDay     Zoned(2);
   BirthYear    Zoned(4);
   BirthDate    Char(8)  SamePos(BirthMonth);  // Format MMDDYYYY
End-Ds;
```

Notes

This is the record layout for the SALESDATA. Note that there are two fields preceding the sales data "array" which means that POS(1) - which is about the only absolute position number I use - cannot be used to locate the start of the array. SAMEPOS(JANSALES) deals with this nicely and has the added advantage that it will continue to work if the size of DIVISION or PRODCODE were ever to change.

```
A          R SALESDATAR
A            DIVISION       2A
A            PRODCODE       7A
A            JANSALES       9P 2
A            FEBSALES       9P 2
A            MARSALES       9P 2
A            APRSALES       9P 2
A            MAYSALES       9P 2
A            JUNSALES       9P 2
A            JULSALES       9P 2
A            AUGSALES       9P 2
A            SEPSALES       9P 2
A            OCTSALES       9P 2
A            NOVSALES       9P 2
A            DECSALES       9P 2
```

Over the years a number of techniques have been devised to allow non-normalized data such as this to be accessed as an array. The advent of SAMEPOS is probably the cleanest and easiest method to date.

The second example would also work if the component fields were externally described. You could achieve the same effect by coding BirthDate as a group field and defining the components as OVERLAYing that field - but that is a lot more typing.

SAMEPOS - Examples

7.4 and 7.3

Partner400

This example shows how an S/36 style record can be mapped

- Because I-specs are so 1900s and can't be used in free-format !

```
Dcl-F  S36File  Disk(40);  // Program described input file

Dcl-Ds RecordLayout  Qualified;
   // Common fields
   CustNo     char(5);
   RecordId   char(1);
   OrderNumber  zoned(5);
   // Layout for header
   Dcl-Ds  Header;
      OrderDate     date(*USA);
      ItemCount     int(5);
      OrderTotal    packed(7:2);
   End-Ds;
   // Layout for Detail
   Dcl-Ds  Detail  SamePos(Header);
      ItemCode      char(7);
      ItemQty       packed(5);
      ItemPrice     packed(7:3);
   End-Ds;
End-Ds;

Read S36File  RecordLayout;  // Load record into DS
```

Fields common to all layouts

Invoice Header record layout

Detail record layout

Notes

Partner400

I recently encountered a customer who still had a lot of S/36 style files in the shop. For that matter I have also met many shops with mainframe heritages that have multi-format files around the place. These folks are disappointed when told that I-specs are not supported in free-form declarations as they would like to code in a more modern style but don't have the resource to rebuild the database and everything else tomorrow. There are other techniques that can be used to avoid I-specs, but they can get messy and/or verbose.

SAMEPOS in combination with nested data structures to the rescue! As you can see in the example it provides a simple, clean way of defining the variable portion on the record layout. An added advantage of using this approach (although those who hate typing more than 6 character field names will disagree) is that the section of the record being accessed is always obvious. e.g. S36Record.Detail.ItemCode or S36Record.OrderNumber.

Think it is too much to type? That just tells me you're not using RDi where the content assist feature would do most of the typing for you!

Variable Dimension Arrays

Partner400

Something we have wanted for a long, long time

- Though, there are still some "gaps"

There are two "flavors"

- Variable and dynamic

Variable Length - programmer sets the maximum length as required

- Specified as DIM (*VAR : maxentries)
  - e.g. DIM (*VAR : 1000);

Dynamic Length - RPG sets the maximum as needed

- DIM (*AUTO : maxentries)
  - e.g DIM (*AUTO : 1000)

maxentries specifies the maximum that the array can ever hold

There is also a third option - *CTDATA

- For compile time data arrays
  - But I dislike compile time data!

Notes

Partner400

Maybe it is just me, but whenever I set up an array I always find myself conflicted as to whether to make the array as big as it could ever conceivably need to be (or even bigger) but in doing so "waste" memory, or to set a more practical maximum. Of course no matter what choice I make, one of the days it is just not going to be big enough!

Those days are now behind me - I can now make the array massive without using up more memory than needed. An added benefit is that RPG will now keep track of the current capacity of the array and so I can use SORTA and/or %LOOKUP without have to consider using %SUBARR to restrict the operation to active elements. That alone is good enough reason to use this support.

Variable Dimension Arrays ... more

Partner400

DIM (*VAR : maxentries)

- Starts off with a maximum capacity of Zero
- You can increase or decrease the capacity by using %ELEM

DIM (*AUTO : maxentries)

- Starts off with a maximum capacity of Zero
- Automagically increases the %ELEM value to accommodate the highest index used to-date

DIM (*CTDATA)

- Automatically ensures that the array is large enough to contain the supplied data.

SORTA and %LOOKUPxx areconstrained by the current maximum

- Not by the maximum array size as with traditional RPG arrays
- Provides a performance boost without the added complexity of using %SUBARR

Notes

Partner400

Most modern programming languages do not force you to pre-declare the size of an array. Until now RPG has been somewhat of an outlier in that regard. But no more!

IBM have added this feature while still maintaining the existing array support. So there's nothing new to learn, just some nice additional capabilities to the things we are already familiar with.

Limitations of New Array Support

Partner400

Can only be used on a top-level definition

- i.e a stand-alone array or a DS array.

Can only be used as a parameter if the prototype specifies Options(*VarSize)

Cannot be ...

- Used for procedure return values
- Used in prototype parameter definitions
- Used on fixed-form calculations (i.e. the old C-specs)
- Based on a pointer
- Imported or Exported.
- Of type Object
- Null-capable
- Used on Output specs
  - Unless an index is specified
  - The same restriction applies to Input specs but they are so rarely used
- And there are a few more esoteric restrictions ...
  - See the Notes page for the full list

Notes

Partner400

The manual lists the following as the full set of restrictions. As you'll see, many are things that you probably would never attempt to do anyway but ...

A varying-dimension array can only be a top-level definition, either a standalone array or a data structure array. Tables, multiple-occurrence data structures, subfields, procedure return values, and procedure parameters are not allowed.

A varying-dimension array cannot be used in fixed-form calculations.

A varying-dimension array cannot be based on a pointer.

A varying-dimension array cannot be imported or exported.

When a varying-dimension array is passed as a parameter to a prototyped procedure or program, the parameter must be defined with OPTIONS(*VARSIZE).

A varying-dimension array cannot have type Object.

A varying-dimension array cannot be null-capable.

A subfield of a varying-dimension data structure array cannot be null-capable unless the null-indicator is also a subfield in the same data structure.

A varying-dimension array cannot appear on an Input specification unless an index is specified.

A varying-dimension array cannot appear on an Output specification unless an index is specified.

A varying-dimension array element cannot appear on a Lookahead Input specification.

A varying-dimension array cannot be a compile-time array or a pre-run array. The CTDATA and FROMFILE keywords cannot be used.

Getting/Setting Element Limits

Partner400

%ELEM (ArrayName : Option)

Available Options are:

- no entry
  - Used on the Right Hand Side (RHS) returns the number of active elements
  - Used on the Left Hand Side (LHS) sets the number of active elements
- *KEEP
  - When increasing maximum elements preserve existing values
    - The default is to initialize new elements to the default value
- *ALLOC
  - On the RHS returns the number of elements for which storage is allocated
  - On the LHS requests allocation of storage for the specified number of elements
- *MAX
  - Returns the maximum number of elements

Examples of *ALLOC and *KEEP follow

Notes

Partner400

The idea that %ELEM can be used on the left or right hand side of an expression takes a bit of getting used to.
For example if we wanted to increase the maximum number of array elements by 50% we could do this:

```
%Elem ( MyArray ) += %Int ( %Elem ( MyArray ) * 0.5 );
```

%ELEM() - Changing The Array Size

7.4 ONLY

Partner400

%ELEM is used to increase or reduce current array size

- Added elements are initialized to the default value
  - Unless *KEEP is used in which case no initialization takes place
    - As demonstrated by the sample below
- If the array is reduced in size the memory is not cleared
  - The existing values will remain but be inaccessible via array operations

```
Dcl-S  VaryArray3  Int(5)  Dim(*Var: 100);
Dcl-S  i           Int(5);

%Elem(VaryArray3) = 10; // Set to capacity of 10

For i = 1 to %Elem(VaryArray3);
   VaryArray3(i) = i;
EndFor;

Dsply ('Element 5 has the value ' + %Char(VaryArray3(5)) );

%Elem(VaryArray3) = 1; // Set to 1 element

%Elem(VaryArray3 : *KEEP ) = 5; // And now to 5

Dsply ('Element 5 should still be 5 and is ' +
       %Char(VaryArray3(5)) );
```

Notes

Partner400

When an array is defined as *VAR (Variable) in size, %Elem must be used to control the current maximum size. The maximum starts at zero so %Elem must be used before any elements are stored in the array.

If we start the program by coding:

MyArray(1) = 'New Value';

and have not set the %Elem value to 1 (or greater) the attempt to store the value will fail in the same way as the old RPG fails if you try to store a value in the 11th element of a 10 element array. Similarly

MyArray(11) = 'New Value';

will fail if the current limit is set to 10.

Note that %ELEM cannot increase the maximum number of elements for an array. So if the array is defined as:

Dcl-S MYArray Char(12) Dim(*Var : 1000);

Then

%ELem(MyArray : 1001);

Will also fail because it attempts to increase the current size of the array beyond its declared maximum.

Special Indexing Feature - *Next

7.4 ONLY

Dynamic Arrays defined with *AUTO offer an addition feature

You don't need to keep track of the index when adding new elements

Simply use *NEXT as the index

- Then retrieve the active count with %ELEM when needed

```
Dcl-S  VaryArray2  Int(5)  Dim(*Auto:9999);
Dsply ('Start: Active elements = ' + %Char(%Elem(VaryArray2)) );

For i = 1 to 50;
   VaryArray2(*Next) = i;
EndFor;

Dsply ('End: Active elements = ' + %Char(%Elem(VaryArray2)) );
Dsply ('Maximum capacity of array is: '
       + %Char(%Elem(VaryArray2 : *Max)) );
```

Notes

I really like this feature - simply use *NEXT as the index and the compiler takes care of the index number and updates the active element count. If you ever actually need the count, simply use %Elem(arrayName). Like all BIFs, it can be used anywhere that a variable of that type can be used. So you can return it from a subprocedure, or assign if to a variable, or use it as a parameter, or ...

%ELEM() - Allocate Memory

Partner400

Suppose you want to pass a Variable Length Array as a parameter

- The called routine will fill in the values - be sure enough memory is allocated !

Once again %ELEM is your friend

- Use *ALLOC to ensure that enough memory is available
- Call the subprocedure/program
- The called routine has to notify you of the updated element count
- Use that number to set the current count preserving the content

```
Dcl-Pr  UpdateArray  Int(5);
   Array     Like(VaryArray3)  Dim(100)  Options(*Varsize);
   Elements  Int(5);
End-Pr;

Dcl-S  ActiveElements  Int(5);

Dcl-S  VaryArray3      Int(5)  Dim(*Var: 100);

%Elem(VaryArray3 : *ALLOC) = 100; // Allocate memory for 100 elements

ActiveElements = %Elem(VaryArray3);

ActiveElements = updateArray( VaryArray3 : ActiveElements );

%Elem(VaryArray3 : *Keep) = ActiveElements;
```

Notes

Partner400

The critical thing to remember is that the compiler will only have associated enough memory to the array to accommodate its current size. But when you pass it as a parameter the called routine doesn't know that - it expects to hav access to the full amount of memory needed for the maximum number of elements.

Because of this, when you pass such arrays as parameters, you need to make sure the full amount of memory is available or nasty things may happen.

At some point in the future this restriction will hopefully be lifted bu for now this is what we need to do.

P.S. On the main chart I showed the setup and the call of the subprocedure over three lines. In fact due to the flexibility of BIFs I could have done the whole thing as one line - like this:

```
%Elem(VaryArray3) = UpdateArray( VaryArray3 : %Elem(VaryArray3) );
```

That is pretty straightforward but doesn't make what is going on quite as obvious as the version on the chart.

Danger Will Robinson ... Caution Needed ...

If the current size of the array changes it may move in memory

- If you have used %Addr to obtain the array's address you must refresh that address after each change in size

Debugger is blissfully unaware that the array size may change

- So you must avoid trying to access elements beyond the current size
- The special variable _QRNU_VARDIM_ELEMS_arrayname contains the current maximum and can be queried
  - But changing its value will have zero effect

Notes

It is important to understand that, unlike conventional RPG arrays, dynamic arrays are just that - dynamic! And that means that as their size changes they may move around in memory. For this reason it is critical that if you capture the address of the array for any reason, you must use %Addr() to refresh that address prior to attempting to actually use the pointer.

PSDS Enhancements

Two new fields have been added to the PSDS

The 16 byte Internal Job Id.

- It can be found in positions 380 to 395
- It will look like "random hex garbage" to quote Barbara Morris
- Previously the QUSRJOBI API would have had to be called to get the information

The 8 character System Name

- Located in positions 396 to 403

```
Dcl-Ds  PSDS;
   JobId       Char(16)   Pos(380);
   SystemName  Char(8)    Pos(396);
End-Ds;

Dsply ( 'Job Id is: ' + JobId );
Dsply ( 'System name is: ' + SystemName );
```

Notes

Just another small enhancement - but useful nonetheless

Print to the addition one the Internal Job Id. to the PSDS it would have been necessary to call the QUSRJOBI API to obtain this information. This makes life much simpler when the Id is needed for tasks such as socket programming.

Procedure Overloading

Almost (but not quite) Polymorphism

- Allows the use of a single procedure name
- But invokes different logic based on the parameter(s) supplied

The keyword OVERLOAD supplies the list of candidate procedures

- All must share the same definition for the return value

The compiler determines which actual routine to call

```
Dcl-Pr DOW_Date  int(5);
   inpDate  date(*ISO)  Const;
End-Pr;

Dcl-Pr DOW_Num  int(5);
   inpDate  packed(7)  Const;
End-Pr;

Dcl-Pr  DayOfWeek  int(5)
        OVERLOAD( DOW_Date : DOW_Num );
```

Notes

If you are familiar with any Object Oriented programming languages, then the concept of procedure overloading should already be familiar to you. For those of you who do not have that experience, the best way to explain it is by way of an example.

Suppose we currently have a simple subprocedure that accepts a date as its input and returns a number representing the day of the week in the range 1 through 7. Now suppose that we get asked to create a new subprocedure that performs the same task but accepts one of our old 7 digit packed decimal "dates" as its input. One way of dealing With this request would be to tell the programmer in question to "go away" and use the %DATE built-in function (BIF) as the parameter to the existing routine.

Another, more polite, approach would be to create a new subprocedure with a different name and parameter list and tell the programer to use that routine instead. Under the covers of course the new routine would probably simply format a date and then call the original routine.

This is all well and good until someone else points out that the company also has some 8 character fields that contain "dates" and it would be really nice if your routine to could handle those as well. That would make three routines, all performing the same basic function but with different names. Confusing at best.

Enter procedure overloading. With this capability we can now use a single function name and have the compiler work out which routine to actually call! As you can imagine, such a facility goes well beyond what we can do with existing parameter options such as CONST and VALUE.

Procedure Overloading - Continued

Will not always easily do what you want

- Because the compiler wants to choose ONLY fit - not BEST fit
  - And will fail compile if there are multiple "fits"

Suppose our date routine should also accept 6 or 7 digit numbers

- You could not use CONST or VALUE on the parameters
  - That would result in the compiler thinking they were all the same
- So you end up needing four extra routines to handle both packed and zoned values!

```
Dcl-Pr DOW_Date  int(5);
   inpDate  date(*ISO)  Const;
End-Pr;

Dcl-Pr DOW_NumP6  int(5);
   inpDate  packed(6);
End-Pr;
Dcl-Pr DOW_NumP7  int(5);
   inpDate  packed(7);
End-Pr;

Dcl-Pr  DayOfWeek  int(5)
        OVERLOAD( DOW_Date : DOW_NumP6 : DOW_NumP7 );
```

Notes

As you can imagine, this can quickly get messy. In the example we have only shown three options (a date and two sizes of numeric). But now suppose that we want to add options to handle 8 digit numerics, and 6 and 8 character options. It is going to get a little out of hand.

The thing to remember is that this is just the first phase of this support. As IBM sees how programmers are using it, and the challenges that they face, we can expect them to make changes and enhancements down to road to meet those needs.

But this is a great first step and one I've wanted to see for a long time.

Procedure Overloading - Example

Here we are showing three procedures all being mapped to a single name

- What is the potential problem with DOW_Num ?

7.4 and 7.3

```
Dcl-s aDate     date(*YMD)  inz(D'2020-03-01');
Dcl-s numDate   zoned(7)    inz(1200301);
Dcl-s charDate  char(6)     inz('200301');

Dcl-Pr DOW_Date  int(5);
   inpDate  date(*ISO)  Const;
End-Pr;

Dcl-Pr DOW_Num  int(5);
   inpDate  packed(7)  Const;
End-Pr;

Dcl-Pr DOW_Char  int(5);
   inpDate  char(6);
End-Pr;

Dcl-Pr  DayOfWeek  int(5)
        OVERLOAD( DOW_Date : DOW_Num : DOW_char);

dayNumber = dayOfWeek(aDate);
dayNumber = dayOfWeek(numDate);
dayNumber = dayOfWeek(charDate);
```

Notes

This example is based on the one we used in the article referenced below.

So what is wrong with DOW_Num?

The use of the CONST keyword means that it would, among other things, accept a 6 digit number such as 200301. That would result in the value 0200301 being passed. But the leading zero would be taken to mean the data was in the 1900s i.e. the 1st of March 1920 - not 2020!

So - exercise caution when using Const or Value and be sure that you are not creating problems for yourself.

On the next chart we'll look at how the three routines are implemented.

For a fully detailed description of this capability see our article https://ibmsystemsmag.com/Power-Systems/10/2019/Procedure-Overloading-DATA-GEN

Procedure Overloading - Example - Contd.

Here we are showing the code behind one of the procedures

- Notice that it includes validation of the input parameter before attempting to process it with the base Day of Week routine

7.4 and 7.3

```
Dcl-Proc DOW_Char;
  Dcl-Pi DOW_Num  int(5);
    inpDate  char(6);
  End-Pi;

  Dcl-S workDate   date(*ISO);

  Dcl-S dayNumber  int(5);

  Test(DE) *YMD& inpDate;
  If %Error;
    dayNumber = -1;
  Else;
    workDate = %Date( inpDate : *YMD& );
    dayNumber = DOW_Date( workDate );
  EndIf;

  return dayNumber;
End-proc;
```

New *EXACT Option for Parameters

7.3 and 7.4

New option to improve parameter passing safety

- Prior to this you could pass a longer parameter than specified
  - And that can produce unexpected and often undesirable results
- It would have been nice to go back and change the default but ...

When *EXACT is specified ...

- The parameter's definition must match the prototyped parameter
  - "Definition" includes integer and decimal places, number of array elements, etc.
  - More in a moment ...

```
Dcl-S  bigField  char(60);

Dcl-Pr  originalWay;
   myParm  char(40);
End-Pr;

Dcl-Pr  exactWay;
   myParm  char(40) Options(*Exact);
End-Pr;

originalWay(bigField);  // Accepted but ...

exactWay(bigField);  // Compiler error - field does not match
```

Notes

*Exact can be used to ensure that the parameter passed exactly matches what is required.

The chart demonstrates my favorite use of this feature - preventing fields that are longer than the definition from being passed.

New *EXACT Option for Parameters

Not just the length of the item must match

- Arrays must have the same dimensions
- DS parameters must be related by LIKEDS/LIKEREC
  - Similar rules apply to parameters defined with LIKE
- Fields must have the same CCSID

Tightening the rules ensures that data is never accidentally lost

- Something that can easily happen when VALUE or CONST is used

Use *EXACT to remove ambiguity when procedure overloading

- The compiler is very picky about what it considers ambiguous

Notes

In addition to the items specified on the chart there are also similar requirements for Java method call prototypes. i.e. The Java class for Object parameters must match.

Also when passing arrays, if ASCEND or DESCEND is specified it must also match

With character, UCS-2 and graphic (double-byte) fields the CCSIDs must also match

*EXACT can also be used with VALUE and CONST. With numerics, for example, it ensures that numeric overflow cannot occur on the call by preventing a field with more integer digits than specified in the prototype from being passed.

Check out the relevant manual pages for full details. For the 7.4 release you will find them here:
https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_74/rzasd/doptns.htm

Want New Features in RPG ?

RPG is now part of the RFE (Request for Enhancements) process.

- You can submit requirements
- You can vote on requirements that others have requested

Check out the current RFEs for the RPG compiler: ibm.biz/rpg_rfe

When searching or creating you may need to specify:

- Brand: ..... Servers and Systems Software
- Product family: Power Systems
- Product: ..... IBM i
- Component:.... Languages - RPG

To vote or submit an RFE you will need to have an IBM Id.

- Just register with an email address and away you go !

Notes

Please, please, please participate in the process. Even if you don't have any ideas of your own vote for the ones that others have submitted to help IBM prioritize.

Resources

Articles by Jon Paris and Susan Gantner

Easiest way to find them is via our Authory site

authory.com/JonParisAndSusanGantner

You can subscribe to the site so that you will always know when new articles are published.

No spam - we promise!

Any Questions ?

Partner400

?

Please e-mail me at:
Jon.Paris @ Partner400.com
for any questions