Strategy and best practice for modern RPG

Barbara Morris
IBM

Agenda

- Learn the features of ILE RPG that allow you to write "modern" code
- Learn which old features and customs to avoid
- Learn how using service programs can modernize your development

Use EXTPROC(*DCLCASE) for your procedures

Compare these two call stacks:

- Using RPG's default of uppercasing names:

```
5   QUOCMD      QSYS                        /03B3
      PGMSTK      BMORRIS                          _QRNP_PEP_PGMSTK
      PGMSTK      BMORRIS       6                  HANDLEORDER
      PGMSTK      BMORRIS       10                 CHECKCUSTSTATUS
      PGMSTK      BMORRIS       14                 GETCUSTID
```

- Using a mixed case name:

```
5   QUOCMD      QSYS                        /03B3
      PGMSTK      BMORRIS                          _QRNP_PEP_PGMSTK
      PGMSTK      BMORRIS       6                  handleOrder
      PGMSTK      BMORRIS       10                 checkCustStatus
      PGMSTK      BMORRIS       14                 getCustId
```

Use EXTPROC(*DCLCASE) for your procedures

An easy way to get mixed-case names is to use EXTPROC(*DCLCASE) on your prototype or procedure interface:

```
dcl-proc handleOrder;
   dcl-pi *n extproc(*dclcase) end-pi;
```

Using mixed-case names makes your joblog easier to follow too:

```
MCH1211 Escape        40   17/09/05  13:37:55.210768
        From module . . . . . . . . :   PGMSTK
        From procedure  . . . . . . :   getCustId
        Statement . . . . . . . . . :   15
```

Bullet-proof your /copy files

Problem:

If you have a date, time, character, UCS-2 or graphic definition in a /copy file, there could be a mismatch between the modules using the /copy file.

For example, if there is a prototype with a date parameter, and a calling module has H spec keyword DATFMT(*USA), the call will fail.

A more subtle issue could be a prototype with a character parameter and a calling module has H spec keyword CCSID(500), and the job CCSID is 37. There would be a CCSID mismatch for a few characters that are different in CCSIDs 37 and 500 (for example, exclamation mark)

Bullet-proof your /copy files

Difficult solution:

You **could** add the required keywords to every definition in the copy file.

Easy solution:

Use the /SET directive to set copy-file defaults for DATFMT, TIMFMT, and the character, UCS-2 and graphic CCSIDS.

The defaults set by /SET will stay in effect until the copy file ends.

```
/SET DATFMT(*ISO) CCSID(*CHAR : *JOBRUN)
```

Also see the /RESTORE directive, if you want to use /SET to temporarily set new defaults for just part of a source member.

Linear-main modules

The RPG cycle is rarely needed any more. Most programmers don't really think about how setting on LR or coding a RETURN prevents the calculations from looping, but that's how the cycle works.

```
dsply 'hello';
*inlr = *off;
```

The compiler requires *INLR to be set, but it doesn't care what it is set to.

If I run this program, this is how the joblog looks:

```
DSPLY  hello  (many more before this)
DSPLY  hello
DSPLY  hello
DSPLY  hello
 5 >  *SYSTEM/ENDRQS
Last request at level 4 ended.
```

I had to do a SYS-REQ to stop the program from looping forever

Linear-main modules

Since 6.1, you can designate one subprocedure to be the main procedure of your program.

The calculations for a subprocedure begin at the beginning and end at the end, so the calculations are "linear" rather than "cyclical".

```
ctl-opt main(sayHello);
dcl-proc sayHello;
   dsply 'hello';
end-proc;
```

If I run this program, this is how the joblog looks:

```
DSPLY  hello
```

Partial arrays

Most arrays have a maximum number of elements, but not all elements are actually being used.

In the past, programmers had some difficulty keeping the array sorted.

After sorting the array, blanks go to the top:

```
'               '
'               '
'Adams          '
'Campbell       '
'Jackson        '
```

Partial arrays

The trick that programmers used was to initialize the array to *HIVAL, so the unused elements would sort to the end.

Now, after sorting the array, *HIVAL elements go to the end:

```
'Adams       '
'Campbell    '
'Jackson     '
'************'   (x'FF's)
'************'
```

But that trick isn't needed any more. There's no need to sort the entire array if only the first few elements are being used.

Partial arrays

With %SUBARR, you can limit the sort to only the elements you're using:

```
SORTA %SUBARR(arr : 1 : num_arr_elems);
```

Use %SUBARR to set a subset of the elements:

```
%SUBARR(arr : start : num_per_page) = 'x';
```

To search only some elements of the array, specify the extra parameters for %LOOKUP:

```
index = %LOOKUP('Jack' : arr : 1 : num_arr_elems);
```

Passing a trimmed parameter

To pass a parameter that should always be trimmed

The hard way - always remember to code %TRIM

```
dcl-pr getFileInfo;
   file varchar(101) const;
end-pr;
getFileInfo (%trim(filename));
```

The easy way - let OPTIONS(*TRIM) handle trimming

```
dcl-pr getFileInfo;
   file varchar(101) const options(*trim);
end-pr;
getFileInfo (filename);
```

Null-terminated string parameters

To pass a parameter that ends with x'00' (a null-terminator)

The hard way - manually add the x'00'

```
dcl-pr openIfsfile extproc('open');
   file char(101) const;
   … more parameters
end-pr;
openIfsfile (%trim(filename) + x'00);
```

The easy way - let OPTIONS(*STRING) handle the x'00'

```
dcl-pr openIfsfile extproc('open');
   file pointer value options(*string);
   … more parameters
end-pr;
openIfsfile (%trim(filename));
```

Null-terminated string parameters

Use OPTIONS *STRING and *TRIM together

```
dcl-pr openIfsfile extproc('open');
   file pointer value options(*string : *trim);
   … more parameters
end-pr;
openIfsfile (filename);
```

The passed parameter will be trimmed even if you pass a pointer

```
dcl-s pName pointer inz(%addr(name));
dcl-s name char(100) inz('myfile.txt'); // + 90 blanks

openIfsFile (pName); // procedure receives "myfile.txt"
```

Consider using data structures for I/O

A great feature of RPG is that it's **NOT** necessary to use data structures for I/O.

When you do I/O without a data structure, RPG copies the data from the I/O buffer into your program fields or it copies the data from your program fields into the I/O buffer. One field may be a standalone field, and others may be subfields in various data structures.

This is a central feature of RPG. It can be wonderful when the same field is used in two different files.

Consider using data structures for I/O

But it's not always the best choice.

Sometimes it's better to control where the data is read into or written from.

```
read custfile;          // Where is the data going?

read custfile custDs; // custDs gets the data
```

Use qualified names

Which is clearer?

```
read ordRec;
dow not %eof(ord92);
   ok = checkInventory (city : item_id : quantity);
   ...

   read ordRec;
enddo;
```

Is "city" something that was set by the READ operation? Is "ordRec" really a record in file "ord92"?

```
read ord92.ordRec order;
dow not %eof(ord92);
   ok = checkInventory (cust.city
                      : order.item_id : order.quantity);
   ...

   read ord92.ordRec;
enddo;
```

It's a bit more code, but it's easy to see where everything comes from

A bonus if you use qualified file names

```
dcl-f orders;
dcl-ds orderDs likerec(orderRec);

read orderRec orderDs; // read into the DS
…
if quantity = 0; // oops, should be orderDs.quantity
```

The fields associated with the file are available as standalone fields. It's easy for maintenance programmers to forget that they shouldn’t use those fields if they are using data structures for I/O.

```
dcl-f orders qualified;
dcl-ds orderDs likerec(orders.orderRec);

read orders.orderRec orderDs; // read into the DS
…
if quantity = 0; // Compile error. Field doesn't exist
```

Use alias names

You probably have nice readable alternate names for your files. But you may also have less readable short names.

```
dcl-f orders;
...

if ordqty > 0;
   placeOrder (ordId : cstId : ordqty
               : splcty : cstcty);       // "splcty" ???
```

Using the alternate (alias) names:

```
dcl-f orders alias;
...

if order_quantity > 0;
   placeOrder (order_id : customer_id : order_quantity
               : supplier_city : customer_city);
```

Naming conventions

The most important rule is Be Consistent

- If you use abbreviations, have one standard abbreviation
  - If the standard is that "cvt" is used for "convert", don't name the procedure convertDate or convDate, name it cvtDate
- Use camelCaseNames or underscore_names, but not both

Maximize readability

- Name procedures with verb + noun: placeOrder, terminateAccount.
- Name most variables with noun, or adjective + noun: quantity, yearlyTotal
- Name indicators with conditions: isValid, orderSuccessful, exitKeyPressed …

Defining complex data structures

Until very recently, there was only one way to define a complex data structure:

1. Define a template for the sub-data structures
2. Define the sub-data structures using LIKEDS

```
dcl-ds emp_t qualified template;
   name varchar(25);
   salary packed(7 : 2);
   is_manager ind;
end-ds;

dcl-ds dept qualified;
   num_emps int(10);
   emps likeds(emp_t) dim(30);
end-ds;
```

The more levels of nesting, the more difficult to understand.

Defining complex data structures

Now, it's possible to directly define the sub data structures.

```
dcl-ds dept qualified;
   num_emps int(10);
   dcl-ds emps dim(30);
      name varchar(25);
      salary packed(7 : 2);
      is_manager ind;
   end-ds;
end-ds;
```

Procedures to handle numeric values of any length

Historically, OPM RPG supported a maximum of 30 digits. ILE RPG supported 31 digits.

Since V5R3, ILE RPG supports 63 digits.

But many programmers still define their "generic" numeric procedures with 31 digits. For example, defining a procedure with a packed(31:9) parameter.

Better:

Take advantage of the entire 63 digit range. If 9 decimal places will always be adequate, define the parameter as packed(63:9).

Or sacrifice a few integer places and increase the accuracy by defining the parameter as packed(63:15).

How to define a 4-byte binary for an API?

Historically, OPM RPG only supported a somewhat bizarre form of binary. ILE also supports this data type.

When a 4-byte "binary" field with 9 digits and 0 decimal positions is used, it is first copied to a packed(9:0) temporary.

This limits the value for the binary field to a range of -999,999,999 to 999,999,999.

This means that the "binary" field is basically being treated as a decimal value.

But the true range of a 4-byte binary is -2,147,483,648 - 2,147,483,647.

How to define a 4-byte binary for an API?

Since V3R2/V3R6, ILE RPG has supported true integers, both signed and unsigned.

The "binary" type should almost always be avoided.

One possible exception is a programmer actually wants a decimal value, possibly with decimal places, but wants to save space by using binary storage. (A 9-digit packed value requires 5 bytes, but a 9-digit binary value only requires 4 bytes.)

Free-form definitions:

- The INT and UNS data types define true binary values.
- The BINDEC data type defines a "binary decimal" value.

Binary fields in externally described files and DS

By default, RPG treats binary fields in externally-described files and data structures as BINDEC fields.

- Fields defined with type 'B' in DDS
- Fields defined as SMALLINT, BIGINT etc in SQL

To have RPG treat these fields as true integer, code EXTBININT(*YES) in your H spec.

Recommendation: Add EXTBININT(*YES) to the set of H spec keywords that are added to every module

RPG's bizarre default CCSID for character fields

By default, RPG assumes that alphanumeric fields have the job CCSID.

Actually, that's not quite true.

RPG assumes that the fields have the **mixed-byte CCSID related to the job CCSID**.

If your job CCSID is 37, RPG assumes that your alphanumeric fields have CCSID 937.

Why does this matter?

RPG's bizarre default CCSID for character fields

Normally, this doesn't matter.

But if you have x'0E' in your field, and that field gets assigned to a UCS-2 field, the x'0E' would be interpreted as a "shift-out" character, and all the data following it would be interpreted as double byte characters. The UCS-2 field would not have the correct value.

Solution: Add CCSID(*CHAR:*JOBRUN) to your H spec.

Recommendation: Add CCSID(*CHAR:*JOBRUN) to the set of H spec keywords that are added to every module.

Assigning data structures

RPG considers a data structure to be also a character string.

You can assign one data structure to another using EVAL.

```
eval ds1 = ds2;
```

This is fine as long as

- The data structures have identical subfields
- The data structures don't have any null-capable subfields

Assigning data structures

Rather than using EVAL, use EVAL-CORR ("corresponding").

EVAL-CORR assigns subfield by subfield.

- Subfields that have the same name and compatible data types are assigned. Null indicators are also assigned for null-capable subfields.
- Other subfields are ignored.

Use the EVAL-CORR Summary in the listing to see exactly what is happening for an EVAL-CORR operation.

If two data structures are related by LIKEDS, EVAL-CORR will just copy all the data at once, so there is no need to worry about performance.

Handling cleanup tasks

To ensure that cleanup tasks are done at the end of a procedure, careful programmers have historically defined a "cleanup" procedure. The cleanup procedure is called

- just before a procedure returns
- from a cancel handler enabled by the CEERTX API

This can be awkward and error-prone

- The cleanup procedure may need access to several variables from the procedure needing the cleanup
- A maintenance programmer may add an early return and forget to add the call to the cleanup procedure

Handling cleanup tasks

The solution

Put the cleanup tasks in the ON-EXIT section of the procedure.

```
dcl-proc myproc;
   ...
   p = %alloc(1000);
   ...
   if not %found;
      return;
   endif;
   ...
on-exit;
   dealloc p;
end-proc;
```

The ON-EXIT section is **always** run, no matter how the procedure ends.

Privacy

There are 5 levels of privacy available to an ILE programmer.

- **Local to a procedure**: the file or variable can only be used within the procedure
- **Global in the module**: the procedure, file, or variable can be used by any procedure in the module
- **Exported from the module**: the variable or procedure can be used by any other module in the same program or service program that imports the variable or calls the procedure
- **Exported from the service program**: the variable or procedure can be used by anything that binds to the service program and imports the variable or calls the procedure
- **Public**: anyone or any program can call a program

Privacy

The more private something is, the easier it is to change how it is defined or used.

Rules of thumb:

- Within a module, avoid global variables and global files when possible
- Think carefully about which procedures you export from a service program. If you have a utility module within the service program, export the utility procedures the module, but if they are specific to the service program, don't export them from the service program
- Only use programs for things that need to be programs. Otherwise, use procedures in service programs to restrict them from being called from the command line.

Modernize development with service programs

The goal is to be able to easily

- reuse your code without having to copy it to make small modifications
- modify your code without being worried about the impact of your changes

Both goals can be achieved by having many small procedures which only do one thing

Other non-general procedures can combine calls to these procedures to do application-specific things

Small procedures which do one specific thing

Example:

- You want to calculate the full price for an order
- You could have one procedure that reads the order, looks up the customer info, calculates the price and then calculates the tax
- Or you could have several procedures

Small procedures which do one specific thing

Example:

You want to calculate the full price for an order

You could have one procedure that reads the order, looks up the customer info, calculates the price and then calculates the tax

Or you could have several general-purpose procedures

- Calculate price based on order, item price, item category, customer information such as discounts
- Calculate tax based on item price, item category

These procedures would not need to read the order file or the customer

Small procedures which do one specific thing

If the procedures do their calculations just based on their parameters

- The procedures can be called by any other procedure that obtained price, customer info by any means
  - **Including a testcase that doesn't actually work with any actual files**
- The procedures can be modified and tested without needing any complex setup of file data

Why service programs?

Instead of having separate procedures, you **could** use separate programs, but …

Why are service programs better?

- Better control of privacy
- Fewer objects in your libraries
  - If you have zillions of small routines, it's "nicer" to have a few service programs than a zillion little programs
- If you have several related procedures, they can be in the same module
  - reduce module initialization time at runtime
  - easier maintenance if similar changes have to be made to several procedures

Special notices

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

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

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

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

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

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

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

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

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

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

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

Special notices