RPG Procedure Tips

Susan Gantner
Partner400 & System i Developer
susan.gantner@partner400.com

Check out Jon & Susan's library of technical articles for RPGers:
authory.com/JonParisAndSusanGantner

In this session, Susan provides some tips and best practices about writing RPG procedures/subprocedures.

She'll cover such topics as:

- Why you should structure your code with procedures
- Best practices with parameters and return values
- Why CONST can be an RPGer's best friend
- How to return large or complex values
- Prototypes - Why use them even when not required?

Topics

Structuring your code via procedures - how and why

Best practices for coding procedures for code structure

What if I need to return more than one value?

Are prototypes still required?

- Should I use them even when they are not required?
- Why are they in a separate source member?

Frustrations when testing with service programs?

- Tips to help with a common issue

Subprocedures: For Structuring Code

Use RPG subprocedures as a better way to structure code

- Regardless of whether coded in the same or a separate member
- For the moment, we won't worry about where the code is
  - Internal or externalized into a Service Program

Better than what?

- Certainly better than just inline code
- Better than subroutines
- Better than called programs

Why?

- Makes your code more obvious
- Makes your code more flexible
- Makes your code easier to test

Note: For purposes of this session,
"**Procedure**" = "**RPG Subprocedure**"

Structuring Code with Procedures

```
Exsr TaxCalc;                                  // Subroutine
     vs
TaxCalc( TAmt : PAmt : PRAll : Loc);           // Program Call
     vs
TAmt = CalcTax( PAmt : PRAll : Loc);           // Procedure call
```

Subroutine

- What data goes into / comes out of TaxCalc?
- How do I find out? Read/understand the code
- What else may be impacted?
- Did I have to move data around to simulate parameters?

Program call

- I know the data potentially impacted from the parameters
- But which are changed?

Procedure call

- Clearly, TAmt contains the result of the tax calculation
- If following best practices, no other data outside CalcTax is affected

Structuring Code with Procedures

```
Exsr ValidCust;
If Stat = 1;
       vs
ValidateCust( CustNo : Stat );
If Stat = 1;
       vs
If ValidCustomer( CustNo );
```

Procedure calls can reduce logic required

- And can make the intent of the code more obvious
  - e.g. Stat - does 1 = valid customer or invalid customer?
- ValidCustomer returns Indicator
- Another example of reducing logic coming up shortly

Best Practices for Procedures

The result of calling the procedure is the return value

- And nothing else outside the procedure should be impacted
  - No parameters updated
  - The procedure should use local data & input parameters only
    - Avoid using or updating global data
      - Rare exceptions, often related to native file I/O operations
- Think of the procedure as a stand-alone "black box"
  - Whether it is coded in the same or different member from its caller

Parameters should be coded with CONST keyword

- CONST = Constant
  - If following the best practice above, the value will never be changed
- Why is this important?
  - Flexibility and ease of use
  - Reduce excess code "noise"
    - Extra code to change data type, calculate values to be passed, etc
- Examples coming up

Best practices for handling prototypes coming up later

CONST Keyword in Practice

With this Prototype (or PI) and this data in the calling code:

```
Dcl-PR  CalcTax  Zoned(8:2);
  GrossPay     Packed(9:2) Const;
  Allowances   Packed(9:2) Const;
  State        Char(2) Const;
End-PR;
```

```
Dcl-S    PAmt       Zoned(7:2);
Dcl-S    Base       Packed(6:2);
Dcl-S    Allow      Packed(5:2);
Dcl-S    Tax        Packed(8:2);
```

Then a call like this works:

```
  Tax = CalcTax( PAmt : Base + Allow : GetEmpState(Empno));
```

But how?

- First parameter defined as packed but we're passing zoned
- 2nd parameter is an expression
- 3rd parameter is a function/procedure call
  - or it could have been an RPG built-in function

The power of Const ...

The Power of CONST

```
Dcl-PR  CalcTax  Zoned(8:2);
  GrossPay    Packed(9:2) Const;
  Allowances  Packed(9:2) Const;
  State       Char(2) Const;
End-PR;
```

```
Dcl-S    PAmt      Zoned(7:2);
Dcl-S    Base      Packed(6:2);
Dcl-S    Allow     Packed(5:2);
Dcl-S    Tax       Packed(8:2);
```

```
  Tax = CalcTax( PAmt : Base + Allow : GetEmpState(Empno));
```

Under the covers, the compiler generates code equivalent to this:

```
Temp1 = PAmt; // Temp1 is Packed(9:2) - convert data type & size
Temp2 = Base + PersAllow; // Temp2 is Packed(9:2) - eliminate EVAL
Temp3 = GetEmpState(Empno) // Temp3 is Char(2) - another EVAL gone

Tax = CalcTax ( Temp1 : Temp2 : Temp3 );
```

And calls like this work, too:

```
  Tax = CalcTax( 1000 : Base + Allow : 'GA');
```

More on RPG Procedures

What if I need to return more than one value?

- Or a very large value?

Are prototypes still required?

- Should I include them even when they are not required?
- Why are they in a separate source member?

Frustrations when testing with service programs?

Returning Complex Values

Need your procedure to return more than a simple variable?

- It can return a Data Structure (or array)
- In place of the definition of a simple variable type on the PI / PR
  - Use LikeDS referencing a DS Template
  - Same template will be used to define the return value in the caller
    - DS template definition typically included where the prototype (if any) is defined
- Tip: DS may also be passed as a parameter using LikeDS

```
/Copy QRPGLESRC,PRODINFOPR

Dcl-PI GetProdInfo LikeDS(ProdInfo_T);
  ProdCode Char(7) Const;
End-PI;

Dcl-Ds  ProdInfo LikeDS(ProdInfo_T);

 ProdInfo.ProdCode = PRCode;
 ProdInfo.ProdName = PRName;
 ProdInfo.Price = PRPrice;

 Return ProdInfo;
```

```
// This is QRPGLESRC,PRODINFOPR

Dcl-PR GetProdInfo  LikeDS(ProdInfo_T);
  ProdCode   Char(7)    Const;
End-PR;

Dcl-Ds ProdInfo_T  Template;
  ProdCode     Char(7);
  ProdName     Char(20);
  Price        Packed(7,2);
End-Ds;
```

Note
Qualified Names

Returning Very Large Values

Potential performance issue for very large return values

- For example, a result set of info for 1000 products
- Or the contents of a large IFS file

Return as Parameter (RTNPARM) performs much better

- When calling from RPG, behaves like a return value
  - Under the covers, it's actually passing an extra parameter
    - So beware if it is ever called from CL or any other non-RPG language
    - It's a plus if calling from Java - no need for wrapper to convert return value

```
Dcl-PI  GetProdByCategory LikeDS(ProdInfo_T) Dim(1000) RtnParm;
  CategoryCode  Packed(5)  Const;
End-PI;

Dcl-PI getFileData  VarChar(1000000) RtnParm;
  FileName  VarChar(500) Const;
End-PI;
```

Procedures defined like this ...

```
Dcl-DS ProdInfo LikeDS(ProdInfo_T) Dim(1000);
Dcl-S  Data  VarChar(1000000);

ProdInfo = GetProdByCategory (CatCode);

Data = getFileData ('/home/mydir/myfile.txt');
```

... can be called like this.

How Big is Too Big for Return Values?

The entire value is moved from one place to another

- Unlike with traditional RPG parameters
  - when the address of the data in the caller (a pointer) is all that is moved

Technically, anything bigger than a pointer is less efficient

- Our pointers are 16 bytes
- In reality, somewhat bigger values won't have any measurable impact on performance

"Too big" in the context of how the function is used

- Is it called 50 times a day?
- 50 times a hour?
- 50 times a minute?
- 50 times a second or more?

There are no absolutes

- ~ 100,000 bytes is likely big enough for concern for many scenarios
- Just be aware of the potential performance impact
  - And keep RTNPARM in your toolbox in case it's needed

Prototypes

Rules for when prototypes are required were relaxed in V7.1

- No real reason to create (or keep) them for internal procedures
  - The compiler will use the PI to get info on how to deal with the call
- They are still required to call an external procedure
- When compiling external procedures
  - The compiler no longer requires prototypes to be present
  - But you should always include them anyway!
  - Why? Because if the compiler can see both PI & PR at the same time, it will be able to validate the prototype
    - That's critical because the PR will be used in all callers
    - It had better be correct!
  - This is also why prototypes should be in separate members
    - To be used with /Copy or /Include

Best Practice - Always Copy/Include prototype member for external procedures:

1. **Into the member with the procedure code, and ...**
2. **Into the members of programs/procedures that call the procedure**

Testing Tip for ILE Applications

ILE Activation Groups "remember" things

- Such as connections between ILE programs and service programs
  - This is a performance benefit, but sometimes has a frustrating side effect
- You will sometimes find that you can't seem to call the latest version of your program/service program procedure right after making a change
  - i.e., the previous version is always called
- This can make it tough to iteratively test new versions of programs and particularly service program procedures

How to resolve this situation?

- Clean up the activation group where the previous code was running
  - Either by starting a new job
    - e.g., sign off & back on - Activation Groups end with the job ends
  - Or - better - Reclaim the Activation Group in your current job
    - RCLACTGRP command specifying a specific AG, e.g., RCLACTGRP QILE
    - Or RCLACTGRP *Eligible - which reclaims all AGs not in the current job stack
  - Important Note:
    - RCLACTGRP *Eligible is often useful for developers to use in their own jobs
    - The *Eligible parameter value should NEVER be used in production code

To Learn More

Partner400

For a deeper dive into RPG Procedures & other IBM i dev topics

- Over 800 articles can be found here:
  - Authory.com/JonParisAndSusanGantner
- Join us for the virtual RPG & DB2 Summit conference in October
  - SystemiDeveloper.com/pages/events/RPGDb2Summit/

Any questions on this topic?

- I'll take as many as time permits here
- Or feel free to send them to:
  - susan.Gantner@Partner400.com