Programming RPG with Style

Paul Tuohy
ComCon
System *i* Developer
5, Oakton Court,
Ballybrack
Co. Dublin
Ireland

Phone: +353 1 282 6230
e-Mail: paul@systemideveloper.com
Web: www.systemideveloper.com
www.ComConAdvisor.com

IBM CHAMPION

System *i* Developer
The Summit of Education Excellence

ComCon

Paul Tuohy

System *i* Developer
The Summit of Education Excellence
ComCon

Paul Tuohy, author of "Re-engineering RPG Legacy Applications" and "The Programmer's Guide to iSeries Navigator", is one of the most prominent consultants and trainer/educators for application modernization and development technologies on the IBM Midrange. He currently holds positions as CEO of ComCon, a consultancy firm based in Dublin, Ireland, and founding partner of System i Developer, the consortium of top educators who produce the acclaimed RPG & DB2 Summit conference. Previously, he worked as IT Manager for Kodak Ireland Ltd. and Technical Director of Precision Software Ltd.

In addition to hosting and speaking at the RPG & DB2 Summit, Paul is an award-winning speaker at COMMON, COMMON Europe Congress and other conferences throughout the world. His articles frequently appear in iProDeveloper, The Four Hundred Guru, RPG Developer and other leading publications. Paul also hosts the popular *iTalk with Tuohy* podcast interviews.

Agenda

Just Another Programming Language
Make Use of the Tools
RPG is FREE
Modern RPG Programs and Subprocedures
What's in a Name?
Comments
Structuring Code
Use Templates and Qualified Data Structures
Qualify Wherever Possible
Strings
Subroutines
Out With The Old
Embedded SQL
Global Definitions
Parameters, Prototyping and Procedure Interfaces
The Integrated Language Environment
Roll Your Own

Just Another Programming Language

Has a lot in common with most modern languages
- Java, PHP, C, C++ etc.

Common Rules

Learn from other languages
- Use /INCLUDE instead of /COPY
- Named constants should be all upper case

Make Use of the Tools

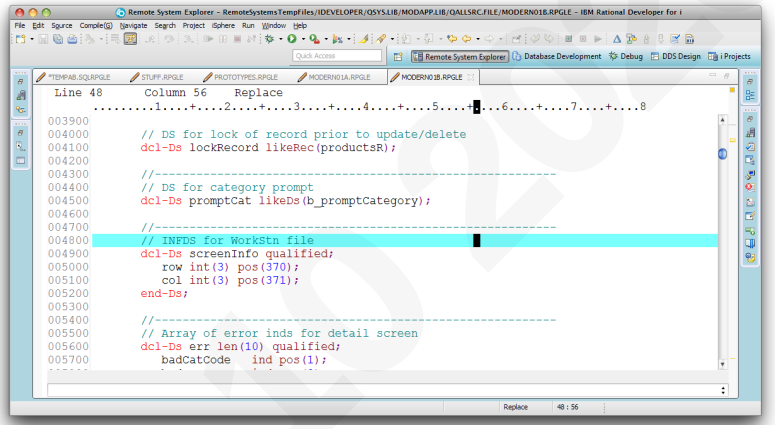
RDi (Rational Developer for i) can help with the implementation of standards

Learn to make use of

- Automatic Indent
- The Outline View
- Content Assist
- Templates
- Snippets

They can all help automate standards and styles

PDM/SEU is NOT an option



RPG is FREE

Complete free form means that a lot of old “bad habits” are no longer available

- MOVE, MOVEL, GOTO etc.

Modern RPG programs should contain only free form code

- No mixture of fixed form and/or extended factor-2 and/or free form

All modifications should be free form only

- Even if existing programs are not converted to free form
  - Which they should be

Only code in free form RPG

- Whether writing new programs or modifying existing programs

Modern RPG Programs

A modern RPG program is

- Written using free form syntax
- Is modular in structure

Modular structure is implemented through subprocedures

Subprocedures may be coded

- Internally in a module
- Externally in a service program or another bound module

Subprocedures

Think of every subprocedure as a “stand alone program”

- Although multiple subprocedures will be coded within a module

The design of a subprocedure is such that it

- Makes copious use of local variables as opposed to global variables
- All required data that is external to the subprocedure is passed as parameters and/or a return value

This approach means that a “useful” subprocedure can simply be

- Removed from it’s current module
- Places in a service program

A subprocedure should be designed to perform one task

- calculate_Pay(), get_customerData() etc.
- And it is OK if the subprocedure has to call other subprocedures to achieve that single task
- Subprocedures should be short and to the point

A good “rule of thumb”

- Should be able to see all the executable code for a subprocedure in a single window in RDi

When you find yourself defining global variables, stop and ask yourself why?

What's in a Name?

System i Developer
The Summit of Education Excellence
ComCon

Names may be mixed case but are not case sensitive

- ***customerID, CustomerID, customerid*** and ***customerId*** all refer to the same variable

Names should be meaningful

- Ten character system names on IBM i has made RPG programmers masters of abbreviation
- It is a habit that needs to be broken
- Names should be meaningful and should not be restricted by a length
  - Although 4096 is the maximum length allowed
  - Names should be meaningful and descriptive
  - Just as a name should not be overly abbreviated it should not be overly verbose
  - The variable should be ***customerID***, not ***cusID*** or ***theIDOfTheCustomer***.

When naming variables, arrays and data structures

- Think of the name as a noun - it simply states what the "item" is
- ***currentAccountNumber, customerID*** or ***customerList***

When naming subroutines, subprocedures or prototype names

- Think of the name as a verb combined with a noun
- There is an action and an item
- ***calculate_Pay(), get_customerData()*** or ***convert_toCelsius()***

Constructing a Name

System i Developer
The Summit of Education Excellence
ComCon

Names should be mixed case

- With the exception of named constants

The usual standard is to use CamelCase

- A name is made up of compound words where each word begins with a capital letter
- The first word may start with a capital letter or with a lower case letter
  - But all following words would start with a capital letter
- ***CurrentAccountNumber*** or ***currentAccountNumber***

But, since RPG is not case sensitive, it is up to the programmer to follow the guideline

Special characters (#, $, £, @) should not be used in names

- With the possible exception of the underscore character
- Special characters are prone to change dependent on CCSID definitions and should be avoided

Underscore

The underscore character can be used to add clarity to a name

There is an inclination to use underscore to separate compound words in a name

- Usually superfluous when CamelCase is being used
- The name ***currentAccountNumber*** is just as legible as ***current_Account_Number***`

But underscore can be useful in subroutine, subprocedure or prototype names

- The underscore is used to separate the action from the item
- ***calculate_Pay()*** or *get_customerData()*

Underscore is also useful in named constants

```
// salary = the 97th element of an array
salary = calculatePay(97);

// salary = the returned value from a call to a subprocedure with a parameter of 97
salary = calculate_Pay(97);
```

Named Constants

Use named constants instead of literals

- Constant names make code self documenting and easier to maintain
- The exception is the use of 0 and 1 in expressions when clearing, incrementing and decrementing

Constant names should be all uppercase

- The convention in most programming languages

Accordingly, underscore should be used to separate compound words within the name

- Since all the characters are uppercase

Common named constants should be placed in a copy member

```
// Compare the use of a literal
if (%status(myFile) = 1218);

// with the use of a named constant
if (%status(myFile) = ERR_RECORD_LOCKED);
```

Naming Conventions

Names should be meaningful

Use naming conventions

- To identify a usage or grouping of variables or named constants
- Correlated names start with the same characters followed by an underscore

If using global variables (in subprocedures)

- Begin with the characters gv_

When naming subprocedures and prototyped program calls

- It is imperative that the naming convention is consistent
- Subprocedures that add information to a database should start with add_ or write_
  - Not a mixture of the two

The proper use of subprocedures and qualified data structures

- Minimize the requirement for naming conventions for variables and named constants
- The naming convention should apply to names that are defined through copy members or globally defined in a program/module

```
dcl-C MSGID_CODE        'ERR0001';
dcl-C MSGID_DESCRIPTION 'ERR0002';
```

Comments

All programs need to be documented

- Free form RPG and proper naming conventions reduce the need for detailed documentation
  - The code is self explanatory
  - Even so, programs need to be documented

Summary Comments

- Should be at the start of every program and subprocedure
- Should contain, at least, the following information:-
  - The title of the program/subprocedure
  - A description of what the program/subprocedure does
  - The name of the author
  - History of changes made to the program

Detailed Commenting

- Only required to explain complex coding techniques or to highlight a technique

Other Commenting

- Use blank lines to group and segment code.
- Use a marker line comment to separate major sections of a program
  - Although the requirement for this is somewhat nullified when using the Filter View feature in RDi

Positions 1 to 5

- Historically used to indicate or flag lines that were changed for a certain modification
- This practice should be avoided. Positions 1 to 5 may now be used for code

Structuring Code

All code should be structured

Need to consider standards and guidelines for

- Declarative code
- Executable code
- Embedded SQL

Structuring Declarative Code (1 of 2)

Declarative code is defined at the start of a module/program/subprocedures

- Definitions should be grouped together by type of declaration
- For example, items are declared in a sequence of:-
  - Files
  - Copy member include of declaratives
  - Procedure Interface
  - Data Structures
  - Stand alone variables
  - Named Constants
- Alternatively, File declarations might be immediately followed by the definition of any stand alone variables or named constants that are used in keywords for a file

Structuring Declarative Code (2 of 2)

System i Developer The Summit of Education Excellence ComCon

Indentation should be used with data structured to identify overlaying structures

- Align definitions so they are easy to read
- When defining stand alone variables, parameters or data structure sub fields, align the data type on each line

```
dcl-Ds APIError qualified;
   bytesprovided int(10) inz(%size(APIError));
   bytesavail    int(10) inz(0);
   msgid         char(7);
   *N            char(1);
   msgdata       char(240);
end-Ds;
```

```
dcl-Ds APIError qualified;
bytesprovided int(10) inz(%size(APIError));
bytesavail int(10) inz(0);
msgid char(7);
*N char(1);
msgdata char(240);
end-Ds;
```

Structuring Executable Code (1 of 2)

System i Developer The Summit of Education Excellence ComCon

All executable code should be indented

- Indentation within loops and groups add to the legibility of the code

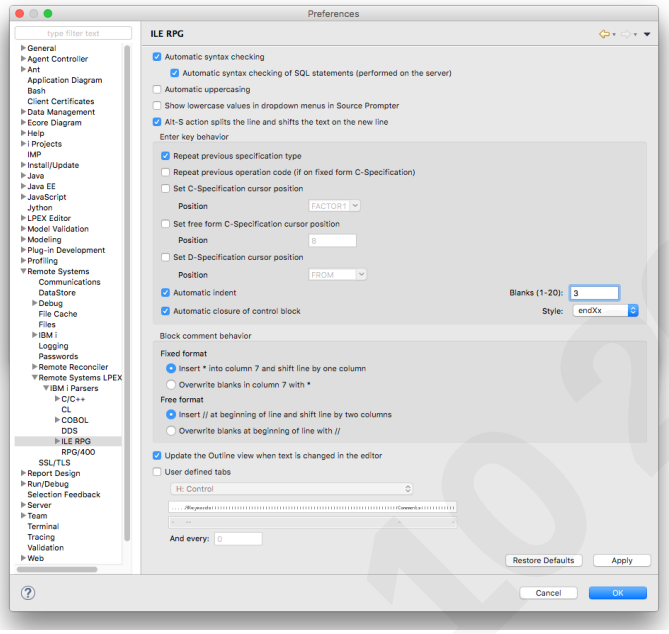
Use alignment for legibility if a statement takes more than one line of code

```
if (messageCount() = 0) ;
   select;
      when CGIOption = 'CANCEL';

      when CGIOption = 'DELETE';
         failed  = delete_Event(persistId);
      other;
         set_eventData(persistId : data);
         if (messageCount() = 0);
            failed  = put_Event(persistId);
         endif;
   endSl;
endIf;

set_days_for_Event(data.event:
                   %date(data.fromdate: *USA):
                   %date(data.todate: *USA):
                   %date(data.wrkshpdate: *USA));
```

Structuring Executable Code (2 of 2)

RDi can help

- ILE RPG preferences for the Remote Systems LPEX Editor in RDi
  - Automatic indent
  - Closure of control block



Structuring Embedded SQL

Use indentation and alignment to make SQL statement legible

- EXEC SQL on separate line
- Align SQL clauses

```
exec sql
   declare C001 scroll cursor for
      select event, daynum, agendaid,
             fromtime, totime, showseq, title
      from AGENDA
      where event = :eventIn
      order by event, daynum, showseq, agendaid
      for read only;
```

Use Templates and Qualified Data Structures

System i Developer The Summit of Education Excellence ComCon

A means of clearly defining and documenting data items in a program

- An excellent means of gathering together related “work” variables in a program
- Providing parameters for a call interface
  - Definition of the template DS can be placed in the same copy member as the prototype for a called program or subprocedure

All references to the subfields must be qualified with the data structure name

- e.g., mailAddress.city, mailAddress.state
- Allows for subfields with the same name to be defined in multiple data structures

Defining a qualified data structure as a template

- The data structure may not be used as a data structure
- Is used as the template for other data structures
  - Which are defined using the LIKEDS keyword

```
dcl-Ds baseAddress template qualified;
   street1 char(30);
   street2 char(30);
   city    varchar(20);
   state   char(2) inz('MN');
   zip     char(5);
   zipplus char(4);
end-Ds;

dcl-Ds mailAddress likeds(baseAddress) inz(*likeDS);
```

Qualify Wherever Possible

System i Developer The Summit of Education Excellence ComCon

Always provide the associated file name when using built in functions

- %EOF(), %FOUND(), %EQUAL() and %STATUS()

Unfortunately, the %ERROR() built in function does not allow for a file name parameter

- Always check the %ERROR() function directly after an operation with an error (E) extender

Strings

The more that you program for the web, the more you work with strings.

For string handling

- Use varying length (VARCHAR)
- As opposed to character (CHAR) fields

Reduces the requirement for string functions

- %TRIM() etc.

Makes the code more legible

Subroutines

Subroutines should not be used for modularization/structure

- Subprocedures should be used instead

Very rarely used in modern RPG

Out With The Old

Free-Form RPG means a lot of the old RPG “functionality” is no longer available

- Operation codes such as MOVE, MOVEL, GOTO, ADD, SUB
- conditioning and resulting indicators etc.

But there are still some old features which are available but should be avoided

- RPG’s Built In Indicators
- Compile Time Arrays
- Multiple Occurrence Data Structures

Out With The Old - RPG’s Built In Indicators

Avoid the use of RPG’s indicators

- *IN01 to *IN99
- Special indicator (*INU1 to *INU8)

They are not self explanatory and are dependent on comments to clarify their usage

Define your own indicators when a boolean condition is required

For externally described display files or print files,

- Define an Indicator Data Structure for the display/print file
- Re-maps the indicators (in the display/print file) to indicators with meaningful names in the program

```
dcl-F Mod30101D workstn(*ext) usage(*input:*output) IndDs(WSI);
dcl-Ds WSI qualified;
   F3Exit       ind pos(3);
   F5Refresh    ind pos(5);
   F12Cancel    ind pos(12);

   errorInds    char(10) pos(31);

   SFLInds      char(3) pos(51);
   SFLDsp       ind pos(51);
   SFLDspCtl    ind pos(52);
   SFLClr       ind pos(53);
   SFLNxtChg    ind pos(54);

   enableMsgSFL ind pos(91) inz(*on);
end-Ds;
```

Out With The Old - Compile Time Arrays

System i Developer The Summit of Education Excellence ComCon

With a compile time array

- Definition of the array is in the data declarations
- Definition of the data is at the end of the program

With a data structure

- Definition of an array in the same location as the data
- The definition of the array overlays the definition of the data

Makes it easier to modularise code into subprocedures, when required

```
 dcl-S monthNames char(9) dim(12) ctData perRcd(3);
   // (lots and lots and lots of code here)
**CTDATA MonthNames
January  February March
April    May      June
July     August   September
October  November December
```

```
dcl-Ds allDays;
   *N          char(27) inz('January  February March');
   *N          char(27) inz('April    May      June');
   *N          char(27) inz('July     August   September');
   *N          char(27) inz('October  November December');
   monthNames char(9) dim(12) pos(1);
end-Ds;
```

Out With The Old - Multiple Occurrence Data Structures

System i Developer The Summit of Education Excellence ComCon

Multiple occurrence data structures

- Only allow access to one occurrence (element) at a time
- Cannot directly compare two occurrences of a multiple occurrence data structure
- Setting the required occurrence (OCCUR/%OCCURS()) is cumbersome

Data structure arrays should be used instead

- Easier using an index to identify an array element as opposed to an occurrence

Embedded SQL

The standard style guidelines apply

Keep SQL statements as simple as possible

- You do not want to have to debug a complex SQL statement in a RPG program
- Create views that “hide” the complexity of joins and casting
- Select from the views in the RPG program

Avoid naming variables that start with SQL

- They might conflict with variable names that are automatically included by the SQL pre-compiler

Use SET OPTIONS

- Ensures that the SQL environment is specified correctly at compile time

Wherever possible, make use of multi-row fetch as opposed to single-row fetch

Global Definitions

Global definitions should be kept to a minimum

Candidates for global definitions are

- File definitions
- Data area definitions.

Consider using qualified data structures for all I/O

If global variables are being defined, they should be qualified

- In a qualified data structure or
- With a prefix (such as gv_)
- Then they are easily identified as global variables within a subprocedure

IMPORT and EXPORT of variables

- Should only be used as a last resort
- Should be well documented when used

If globe variables are being used

- Modularise them within a module with the subprocedures that will be processing them

Parameters, Prototyping and Procedure Interfaces

System i Developer The Summit of Education Excellence ComCon

Prototyping

- Provide a means of validating parameter definition at compile time
- Specify how parameters are used
- Prototypes are only required for external subprocedure or program calls
  - Not required for subprocedures that are coded and only called within the same module/program

Control parameters using the VALUE and CONST keywords

- Stop parameters from being inadvertently changed by a subprocedure or program
- Documents parameter usage

Subprocedures as Procedures or Functions

- Procedures do not return a value
  - A Procedure is code that takes (optional) parameters and performs an action
  - A call to a program would be a Procedure call, since programs cannot return a value
- Functions return a value
  - A Function is code that takes (optional) parameters and "calculates" a return value.

Argument for Procedures always returning a value

- Indicate whether or not the process worked?

Prototyping and Copy Members (1 of 4)

System i Developer The Summit of Education Excellence ComCon

All prototypes should be coded in copy members

- Included in required programs and modules

A prototype should never be coded in more than one source member.

The copy members that contain the definition of the prototypes should also contain

- The definition of any corresponding templates
- The definition of any corresponding named constants

Have one /INCLUDE directive that includes all prototypes for an application

- The included copy member can contain nested /INCLUDE directives
- The actual members containing the prototypes are easy to maintain.

```
/include common,baseInfo
```

```
/include utility,pUtility
/include fileProcs,protoFile
/include common,commproto
/include genstat,pStatGen
/include regFunc,pRegFunc
 //-------------------------------------------------------
 // Include CGI Prototypes, if required
/If Defined(CGIPGM)
/include CGIDEV2/QRPGLESRC,PrototypeB
/include CGIDEV2/QRPGLESRC,Usec
/endIf
```

Prototyping and Copy Members (2 of 4)

System i Developer
The Summit of Education Excellence
ComCon

Each of the included members might contain

- Prototype definitions
- Further nested include directives
  - For example, the PUTILITY member, in turn, contains nested include directives
  - Members PUTILMSGS, PUTILCGI, PUTILSPACE, PUTILIFS and PUTILDATE contain the actual prototypes

There is a difficulty with this approach

- A lot of prototypes (and templates) are being included
- Has no effect on the size of the program/module when it is compiled
  - Prototypes and templates are declarative
- But it can take a long time for the outline view in RDi to refresh

```
/include utility,PUTILMSGS
/include utility,PUTILCGI
/include utility,PUTILSPACE
/include utility,PUTILIFS
/include utility,PUTILDATE
```

Prototyping and Copy Members (3 of 4)

System i Developer
The Summit of Education Excellence
ComCon

Using compiler directives to indicate what should and should not be included

Consider this change to the definition of the PUTILITY member

- A definition name must be set in order for the prototypes to be included
- The definition name UTILITY includes all copy members

```
/if defined(UTILITY)
/define UTIL_MESSAGE
/define UTIL_CGI
/define UTIL_USERSPACE
/define UTIL_IFS
/define UTIL_DATES
/endIf

/if defined(UTIL_MESSAGE)
/include utility,PUTILMSGS
/endIf
/if defined(UTIL_CGI)
/include utility,PUTILCGI
/endIf
/if defined(UTIL_USERSPACE)
/include utility,PUTILSPACE
/endIf
/if defined(UTIL_IFS)
/include utility,PUTILIFS
/endIf
/if defined(UTIL_DATES)
/include utility,PUTILDATE
/endIf
```

Prototyping and Copy Members (4 of 4)

The originating program sets the required definition names

- or can define UTILITY, which would result in all members being included

The "difficulty" with this approach

- The programmer must know the required definition names
- So documentation is required.

The benefit of this approach

- It documents what is being included in the program.

Note

- In this example, definition names are being used to include single members
- But they could just as easily be used to include multiple members
  - As with CGIPGM in the BASEINFO member

```
/define UTIL_MESSAGE
/define UTIL_USERSPACE
/include common,baseInfo
```

The Integrated Language Environment (ILE)

Common questions:-

- How many service programs should you have?
- How many modules should there be in a service program?
- How many modules should there be in a program?
- When should you use bind by copy and bind by reference?
- How many binding directory should you have?
- How should you control signature violations?
- How many activation groups should you have?

Unfortunately, the answer to all of these questions is - it depends

- The structure of an ILE application depends on the application and what it does

A change management system can have a major impact on the methodology

- It may have a preferred technique for managing service programs, binder language etc.

One example of how to structure an ILE application

- "Development Environments"
  http://www.itjungle.com/fhg/fhg051210-story01.html

ILE Programs

System i Developer The Summit of Education Excellence ComCon

The process of creating an ILE program should be as simple as possible

- Although the structure of ILE programs can be more complicated
  - One or more modules
  - Binding to service programs

In order to compile a program

- The programmer should not need to know about activation groups
- What all the service programs are
  - Let alone what subprocedures are in what service programs

This can be achieved through the diligent use of

- Binding directories
- A standard control specification
  - which gets included in every program via an /INCLUDE directive.

```
Ctl-Opt debug datEdit(*MDY/) option(*srcStmt:*noDebugIO) bndDir('MYAPP');
/if defined(*CRTBNDRPG)
Ctl-Opt dftActGrp(*no) actGrp('PGMBND');
/endIf
```

Service Programs (1 of 2)

System i Developer The Summit of Education Excellence ComCon

Service programs are at the core of any ILE application

- If a subprocedure can be used in more than one place, then it belongs in a service program

Management of a service program requires a bit more care than “normal” programs

- Content of a service program can be called from multiple places
- Akin to managing changes to a database
- The minimum of people should have the ability to make changes

Any programmer can develop a subprocedure

- But not any programmer can incorporate it in a service program

How many service programs should an application have?

- There is no magic number
- Each service program should contain groups of related functions
  - Utilities, database processing, API handlers etc.
- For ease of maintenance, some service programs might be split into multiples
  - e.g. database processing

Ease of maintenance is the key

- The number of modules per service program
- The number of subprocedures per module
- Should be determined by related functions and ease of use

Service Programs (2 of 2)

System i Developer
The Summit of Education Excellence
ComCon

Keep the source members for a service program separate from other source members

- Maybe use a source file name that has the same name as the program

Determine how to manage prototype/template members

- One copy member, a copy member per module etc.

Consider having a separate binding directory for each service program

- Bindings for a service program can be different from bindings for a program

Always use binder language

- Never use EXPORT(*ALL) when creating a service program
- Binder language does mean more maintenance
  - When subprocedures are added/removed
- But it also allows the greatest flexibility in managing the interface to the service program

When adding new parameters to a subprocedure

- Add the parameters to the end of the parameter list using OPTIONS(*NOPASS)
- Minimises the requirement of re-creating anything that calls the changed subprocedure

Binding Directories

System i Developer
The Summit of Education Excellence
ComCon

Binding directories provide a means of generically providing a list of objects (modules or service programs) that ***MAY*** be required for binding

Binding directories should be kept to a minimum

A single binding directory should list all service programs (and modules) that might be required when a standard program is created

Each service program might also require a binding directory

A binding directory may list objects that are common to multiple applications

A list of binding directories can be included in the BNDDIR keyword on a Control Spec

Never have a binding directory per program

Activation Groups

Usually, an Activation Group applies to an application

- At times, a separate activation group might be used for scoping
  - Commitment control and/or file overrides

ILE programs should never be run in the default activation group

- Can only be achieved if an ILE program is created with an activation group of *CALLER
- And the program is called by an OPM program (or from a command line)

It is best to use the name of the activation group when creating programs

- This is easily achieved using the ACTGRP keyword in a standard Control Spec in the programs

Service programs should never be run in the default activation group

- Even worse than having an ILE program in the default activation group
- Can only happen if an ILE program is running in the default activation group
- Again, ILE programs should never be run in the default activation group

iTalk with Tuohy

Check it out at ibmsystemsmag.com/ibmi/trends/iTALK-WITH-TUOHY/

"Re-Engineering RPG Legacy Applications"

- ISBN 1-58347-006-9

"The Programmers Guide to iSeries Navigator"

- ISBN 1-58347-047-6
- www.mcpressonline.com
- www.midrange.com
- www.amazon.com
- etc.

iSeries Navigator for Programmers

- A self teach course
- www.lab400.com

Article links at

- www.comconadvisor.com