

Application Modernization

One RPG Programmer's Story
(so far...)

Anders Roede


First, a quick Norwegian lesson:

Q: How do you get "Ahnes" from Anders?

- A = "Ah"
- Don't pronounce the D
- RS makes "Sh"

→ So you get "Ahnes" (close enough)
(Note: accent is on first syllable)

What is modernization?



This is what we hear and read about the most.

- Bringing applications to the web.

To me, it's also (at least):

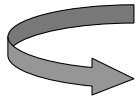
1. Free-format RPG
2. Working in WDSC/RDi/RDp
3. Procedures and service programs
4. Embedded SQL where it makes sense

(Of course there's XML, IFS, web services, smartphones, iPads, etc., etc...)

These four things are kind of a prerequisite to doing other more modern stuff

Stuck in a Rut

- **Old-style techniques**
RPGIII, or RPGIV but with same style and techniques as RPGIII
- **Just need to get the project done**
Doing it in familiar ways is faster than learning something new
- **Not quite sure how to get started using newer stuff**
Where to start, how to organize and name everything, etc.

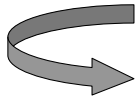


System is so backward compatible that we've been getting away with this for many years!

1. Free-Format RPG

Why convert? A few reasons:

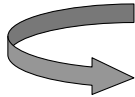
- Easier to read, and faster to enter
- More room for long expressions
- New features are for free-format
- Most examples in articles and forums are in free-format
- Easier to learn for non-RPG programmers



P.S. Learning new stuff may also make you more marketable, plus it makes programming fun again!

Tools to convert to Free-format

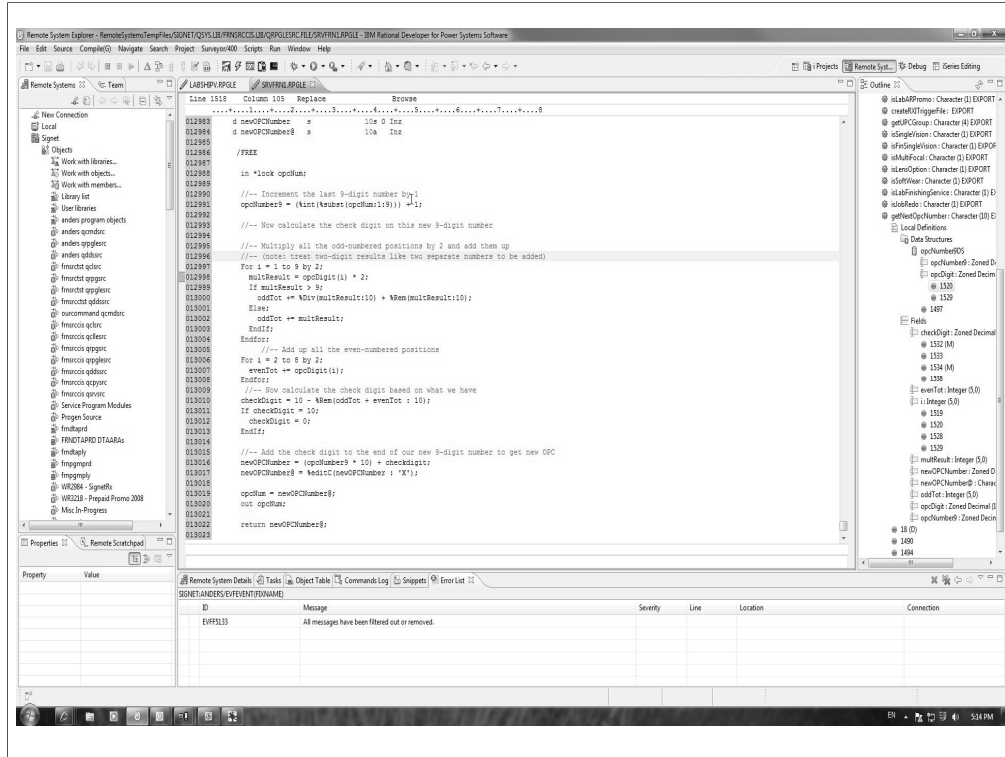
- Linoma's RPG Toolbox
- WDSC/RDi/RDp (Source/Convert All To Free-Form)



These get you most of the way there, then you go through and clean things up

2. Working in WDSC/RDi/RDp


- Increased efficiency
 - ✓ See more code on the screen
 - ✓ Outline view, filters, verify
 - ✓ Multiple members open at once in different tabs
 - ✓ Multiple editable views of the same member open at once
 - ✓ Debugging with Service Entry Points
- New RPG features that are released won't be recognized by SEU



Example of my RDp workspace

3. Procedures and Service Programs

- Modular, re-usable code

A grey thought bubble with a tail pointing towards the left, containing the text "Change things in one place instead of 100's...".

Change things
in one place
instead of
100's...

But where to start?

- Start with local subprocedures, using local variables
- Pick a few similar and frequently used functions and group them together in a service program

It's not an argument between subroutines and procedures... use the right tool for the job! (you can even have subroutines in a procedure)

Service Programs: What's involved and how to organize everything

- A. Service program modules
- B. Prototypes
- C. Binder Source
- D. Binding Directory

I understood sample code, etc., but wasn't sure how to organize things, what to name them, etc.

Read books, articles, forums, etc., at night. May need to invest some of your own time... get your work done at work and learn this new stuff on your own if necessary. But here are some pointers to get you started.

A. Service Program modules

- A service program consists of one or more modules
- A module consists of one or more procedures
- Name the modules the same as the service program, but with a suffix

e.g. SRVUTIL1, SRVUTIL2, etc. all make up service program SRVUTIL (this will enable creating service program SRVUTIL from modules SRVUTIL* instead of naming them all).

Each module has NoMain specified at the H-specs

Compiled object is a *MODULE (don't need to keep it around after the service program is created if you don't want)

I've seen some recommendations to have only one procedure per module, but why?? Group similar procedures together in a module and service program.

B. Prototypes

- RPGLE members in QCPYSRC
- Named to identify the corresponding service program
- Include header documentation (with tags for auto-documentation purposes)
- Bring into programs with /copy or /include

e.g. the SRVUTIL prototype member is named ProtoUTIL

My RPG ProcDoc that runs through the prototype members looking for those header documentation tags, creates an xml and html file on IFS that we can search, view, etc.

Sample Prototypes

```
PROTOFRN.RPGLE 33
Line 58      Column 121  Replace      Browse
.....1.....2.....3.....4.....5.....6.....7.....8.....
004900      //-----
004900      // @Name: getCancelDate
004500      // @Desc: Get order canceled date
004600      // @Author: Anders Roede
004700      // @Return: Canceled date (6,0) in MMDDYY format
004800      // @Parm: Order number (6,0)
004900      // @Parm: Order suffix (2,0), optional (0 assumed)
005000      // @Category: Frontier
005100      // @Module: SRVFRN1
005200      //-----
005300      //-----
005400      d  getCancelDate  pr          6S 0
005500      d  orderNumber   pr          6S 0 Const
005600      d  orderSuffix   pr          2S 0 Const Options(*NoPass)
005700
005800
005900      //-----
006000      // @Name: getCat
006100      // @Desc: Get product category for part number
006200      // @Author: Anders Roede
006300      // @Return: Product Category Code (4A)
006400      // @Parm: Part Number (15A)
006500      // @Category: Frontier
006600      // @Module: SRVFRN2
006700      //-----
006800      d  getCat         pr          4A
006900      d  partNumber    pr          15A  Const
007000
007100
007200      //-----
007300      // @Name: getCatDesc
007400      // @Desc: Get description for product category
007500      // @Author: Anders Roede
007600      // @Return: Product Category Description (50A)
007700      // @Parm: Product Category (4A)
007800      // @Category: Frontier
007900      // @Module: SRVFRN1
008000      //-----
008100      d  getCatDesc    pr          50A
008200      d  category      pr          4A  Const
.....
```

CONST does some type conversion as well as making sure the parm isn't changed by the procedure

Notice the tags in the header documentation... the procedures will automatically get documented by my ProcDoc program.

Sample procedure usage

```
005900 //-----
006000 // @Name: getCat
006100 // @Desc: Get product category for part number
006200 // @Author: Anders Roede
006300 // @Return: Product Category Code (4A)
006400 // @Parm: Part Number (15A)
006500 // @Category: Frontier
006600 // @Module: SRVFRN2
006700 //-----
006800 d getCat      pr      4A
006900 d partNumber 15A  Const
007000
007100
007200 //-----
007300 // @Name: getCatDesc
007400 // @Desc: Get description for product category
007500 // @Author: Anders Roede
007600 // @Return: Product Category Description (50A)
007700 // @Parm: Product Category (4A)
007800 // @Category: Frontier
007900 // @Module: SRVFRN1
008000 //-----
008100 d getCatDesc pr      50A
008200 d category  4A  Const

Pgm's
D-specs 002400 //-- Prototypes
002500 /Copy Fmnrccis/QcopySrc,ProtoFrn

Usage 011800 myPartCategory = getCat(partNumber);
011900 myPartCategoryDesc = getCatDesc(myPartCategory);
011901
011902 myPartCategoryDesc = getCatDesc(getCat(partNumber));
```

C. Binder Source

- List of a service program's exported procedures
- One per service program, named the same as the service program.
- In QSRVSRCL, member type BND.
- Explicitly name the signature the same as the service program

- Never use Export(*ALL)
- Explicitly name the service program in its signature
- New exported procedures will always be added to the bottom of the list

Sample Binder Source

```
*SRVFRN.BND
Line 10      Column 1      Replace 40 changes
-----1-----2-----3-----4-----5-----6-----7-----
000100 /*
000200 /* Name:          SRVFRN
000300 /* Description:    Binder Source for SRVFRN service program
000400 /* Created By:    Anders Roede
000500 /* Date Created:   03/28/2008
000700 /* Narrative:     Binder source for service program SRVFRN
000800 /*                containing all exported procedures from the
000900 /*                service program.
001000 /*
001200 /* Rules:         1. Never change the Signature ('SRVFRN')
001300 /*                2. Never change the order of exports
001400 /*                3. Add new exports at the end of the export list
002100 /*
002200 StrPgmExp PgmLvl(*Current) Signature('SRVFRN')
002600     Export Symbol(getCustName) /* 1 */
002700     Export Symbol(getCustGroup) /* 2 */
002800     Export Symbol(isCustCredHold) /* 3 */
002900     Export Symbol(isCustDeleted) /* 4 */
003200     Export Symbol(getRegion) /* 5 */
003300     Export Symbol(getRegionName) /* 6 */
003400     Export Symbol(getRep) /* 7 */
003500     Export Symbol(getRepName) /* 8 */
003600     Export Symbol(getRepEmail) /* 9 */
003800     Export Symbol(getEnterDate) /* 10 */
003900     Export Symbol(isPrinted) /* 11 */
004000     Export Symbol(getPrintDate) /* 12 */
004100     Export Symbol(isShipped) /* 13 */
004200     Export Symbol(getShipDate) /* 14 */
004300     Export Symbol(isInvoiced) /* 15 */
004400     Export Symbol(getInvoiceDate) /* 16 */
004500     Export Symbol(isCanceled) /* 17 */
004600     Export Symbol(getCancelDate) /* 18 */
004800     Export Symbol(getPartDesc) /* 19 */
004900     Export Symbol(isPartObsolete) /* 20 */
005000     Export Symbol(getPartLensSpec) /* 21 */
005500     Export Symbol(getCat) /* 22 */
005600     Export Symbol(getCatDesc) /* 23 */
015200 Endpgmexp
```

This method vs. the *PRV method, etc. This way, you don't have to worry about signatures, having a very long and confusing binder source, etc.

Also, note the fact that I've not used quotes around my procedure names... so everything will get interpreted at all caps and I don't have to remember the case of each letter.

The comments on the right are sequence numbers I've added to make sure I know the order of the exports, just in case...

Creating the Service Program

Never use Export (*ALL). Specify the Binder Source:

```
                Create Service Program (CRTSRVPGM)

Type choices, press Enter.

Service program . . . . . srvfrr      Name
  Library . . . . . frnpgmprd      Name, *CURLIB
Module . . . . . srvfrr*          Name, generic*, *SRVPGM, *ALL
  Library . . . . . frnpgmprd      Name, *LIBL, *CURLIB...
      + for more values _____

Export . . . . . *SRCFILE          *SRCFILE, *ALL
Export source file . . . . . QSRVSRC      Name, QSRVSRC
  Library . . . . . frnsrcis          Name, *LIBL, *CURLIB
Export source member . . . . . *SRVPGM      Name, *SRVPGM
Text 'description' . . . . . SRVFRN Service Program
_____
```

Updating the Service Program

In this example, I haven't added any procedures, just changed one (or more):

```
Update Service Program (UPDSRVPGM)

Type choices, press Enter.

Service program . . . . . srvfrn      Name
Library . . . . . frnpgmprd      Name, *USRLIBL, *CURLIB
Module . . . . . srvfrn2      Name, generic*, *NONE, *ALL
Library . . . . . frnpgmprd      Name, *LIBL, *CURLIB..
      + for more values
      *LIBL
Export . . . . . *CURRENT      *CURRENT, *SRCFILE, *ALL
Export source file . . . . . QSRVSRC      Name, QSRVSRC
Library . . . . . frnsrccls      Name, *LIBL, *CURLIB
Export source member . . . . . *SRVPGM      Name, *SRVPGM
```

If I had added a new procedure, I would specify *SRCFILE.

D. Binding Directory

- List of service programs to search for procedures
- Specify the binding directory in your H-specs

```
Work with Binding Directory Entries

Binding Directory:  SABNDDIR      Library:  FRNSRCCIS

Type options, press Enter.
  1=Add   4=Remove

Opt  Object      Type      Library      -----Creation-----
-----
  -   SRVUTIL      *SRVPGM   FRNPGMPRD    09/11/08     16:55:24
  -   SRVMATH      *SRVPGM   FRNPGMPRD    10/16/08     14:25:21
  -   SRVSTR       *SRVPGM   FRNPGMPRD    12/22/08     17:30:49
  -   SRVOPTIC    *SRVPGM   FRNPGMPRD    11/18/08     18:13:57
  -   SRVFRN      *SRVPGM   FRNPGMPRD    01/14/09     14:59:59
  -   SRVDATM    *SRVPGM   FRNPGMPRD    08/28/08     17:33:32
  -   SRVMAX      *SRVPGM   FRNPGMPRD    10/20/08     18:01:56
  -   SRVPROMO   *SRVPGM   FRNPGMPRD    08/26/08     12:15:18
                                           More...
```

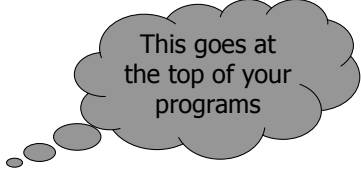
In your H-specs, also specify the binding directory QC2LE (prior to i6.1) for C-runtime library functions (high-level math functions, system(), etc.)

Note: IFS APIs are Unix-type APIs.

Standard H-Specs Sample

```
// Standard H-Specs Copy Module
H Copyright('(c) 2010 My Company, Inc. All rights reserved.')
H BndDir('QC2LE')
H BndDir('MYLIB/MYBNDDIR')
H Option(*NoDebugIO:*SrcStmt) CopyNest(10)
H AlwNull(*Usrctl)
H ExtBinInt(*Yes)
H OpenOpt(*InzOf1)
/If Defined(*CRTBNDRPG)
H DftActGrp(*No) ActGrp('MYACTGRP')
/EndIf

//-- Standard header specifications
/Copy MyLib/QCpySrc,Hspecs
```



This goes at
the top of your
programs

Use standard H-Specs copy member to avoid having to worry about a bunch of stuff at compile time... Just use option 14 like the good ol' days!

Note: Perhaps ActGroup(*CALLER) is right for your shop...? (e.g. if you have ILE being called from OPM RPG)

Activation groups:

1. Default activation group
2. User named activation group (e.g. QILE)
3. System named (*NEW) activation group – only use when calling recursively
4. *CALLER
5. *ENTMOD

Summary Overview – Service Programs

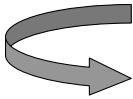
- A. Write a procedure in a module source member
→ e.g. getCat in SRVFRN2
- B. Add prototype to prototype source member
→ e.g. getCat in QCPYSRC/ProtoFRN
- C. Add procedure name to bottom of Binder Source
→ e.g. getCat in QSRVSRC/SRVFRN
- D. Make sure service program is listed in your binding directory (use WRKBNDDIR)

Now compile module source to *MODULE and create/update the service program... voila!

D. WRKBNDDIR SABNDDIR in my case

4. Embedded SQL

- ✓ SQL is great at selecting and manipulating groups of data
- ✓ Can simplify your program code
- ✓ Source member type SQLRPGLE
- ✓ Compile with CRTSQLRPGI, specifying *PGM or *MODULE, and COMMIT(*NONE)



Need Commit(*none) to update/delete files that are not journaled

Eliminates the need to specify the file(s) in the F-specs

But we don't have to choose SQL or native I/O, you can mix and match within a program as needed... use the right tool for the job!

SQL excels at handling groups of data

```
//-- SQL to get customers with active promo contracts
Exec Sql Declare C1 Cursor
  For Select CONTRACTS.PRCODE, Min(CONTRACTS.PRCONT)
  From LABOE021P As CONTRACTS, LABOE023P As SHIPTOS,
       LABOE020P As PROMOS
  Where (CONTRACTS.PRRQTY > 0 Or PROMOS.PRPPAID = 'N')
  And CONTRACTS.PRBTKY = :BILLTOCUST
  And (SHIPTOS.PRSTKY = :DSJOB.CUSTNUMBER Or SHIPTOS.PRSTKY = '*ALL')
  And CONTRACTS.PRSDAT <= :CURRDATEYMD
  And CONTRACTS.PREDAT >= :CURRDATEYMD
  And CONTRACTS.PRCONT = SHIPTOS.PRCONT
  And PROMOS.PRCODE = CONTRACTS.PRCODE
  Group By CONTRACTS.PRCODE ;

Exec Sql Open C1;

Dow SQLStt = sqlOK;
  Exec Sql Fetch C1 Into :PROMOCODE, :PROMOCONTRACT ;
  If SQLStt = sqlOK;
```

I have sqlOK, sqlNoRow, sqlDupRow, etc. defined in ConstSQL /copy member.

Tip: Do the SQL's over physical files, not logicals whenever possible.

You want to use SQE not CQE, and you don't want logicals with select/omits forcing you into CQE. To change globally, add a record to QUSRSYS/QAQQINI to set IGNORE_DERIVED_INDEX to *YES.

Service Program procedure using embedded SQL

```
//-----  
// Procedure:   getCat  
// Description: Get product category of part  
// Programmer:  Anders Roede  
// Date:       06/17/2008  
// Narrative:   This procedure accepts a part number  
//             and returns the product category.  
//-----  
P getCat      B          Export  
  
d getCat      pi          4A  
d partNumber  15A      Const  
  
/-- Variables  
d category    s          4A  Inz  
  
/FREE  
  
Exec Sql  
Select Substr(PMUSR1, 1, 4)  
Into :CATEGORY  
From PMP  
Where EMPART = :PARTNUMBER ;  
  
If SQLStt <> sqlOK;  
category = '****';  
EndIf;  
  
return category;  
  
/END-FREE  
P getCat      E
```

Not all SQL statements need to be intimidating... this one is very simple but useful in a service program procedure.

Prior to i6.1, you can't have local files in your procedures...

No F-specs at top of this module, because all the procedures use SQL. Plus I don't need to worry about opening/closing files, etc.

By the way, write the procedure first, then just copy the prototype from the procedure into the prototype copy member and change the pi to a pr.

Modernized User Interface

Many options exist:


- CGIDEV2
- PHP
- BCD Websmart and Websmart PHP
- .Net
- RPGOA
- EGL
- CNX Valence (this is what we're using)
- Etc., etc...

CNX Valence

- HTML/Javascript front-end
- RPG back-end
- Both of these are edited from within RDp!
- No proprietary IDE or language
- Valence service program procedures handle all the communication between front-end and back-end

Javascript framework included with Valence: ExtJS


Includes a portal with security, has grids, trees, forms, windows, toolbars, menus, charts, uploading/downloading from IFS, PDFs etc.



Valence: Reload Your RPG for Web 2.0

No WebSphere® or external servers required.

▶ DOWNLOAD NOW




Give Your Business an Easy to Use, Powerful Web 2.0 Application

Create System i apps with highly intuitive [Web 2.0 features](#), including drop-downs, grids with sortable columns, trees and charts. The new [Valence 2.1](#) also allows you to create PDFs and emails using RPG, route SQL results straight to the browser, and much more!



Keep Your Back-End RPG Unencumbered and Fast

Don't settle for a performance-robbing screen scraper. Get Valence's user friendly Web 2.0 front-end served at top speed through standard ILE RPG, all running natively on your [AS/400, iSeries or System i](#).



Retrofit Existing Programs & Create New Ones Faster than Ever

Valence uses JavaScript, a simple, non-proprietary language to interact with RPG and give existing and new apps fresh Web 2.0 interfaces. Valence's [new AutoCode](#) feature speeds up development by helping write front and back end code simultaneously.

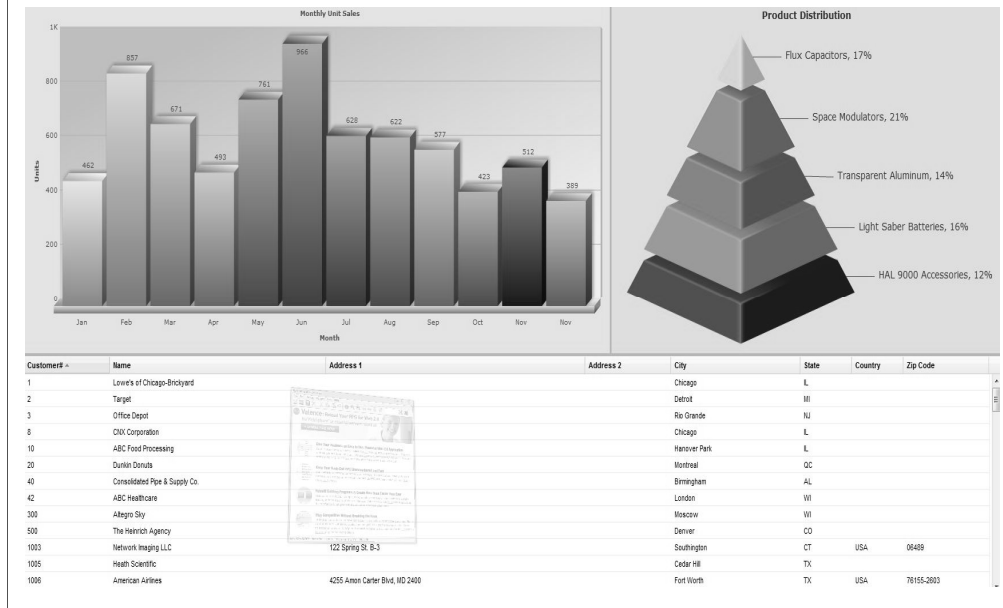


Stay Competitive Without Breaking the Bank

Valence is licensed on a per developer basis — no need to worry about the size of your IBM i or number of users. Run all your applications using ILE RPG and the free Apache Web Server — no additional hardware, WebSphere® Application Server or Java required. See the [Licensing & Services](#) page for full pricing details.

www.CnxCorp.com/Valence

Browser-based User Interface



Flexibility and dynamic nature (and wow-factor) of a Javascript front-end, with the power and stability of RPG on the back-end.

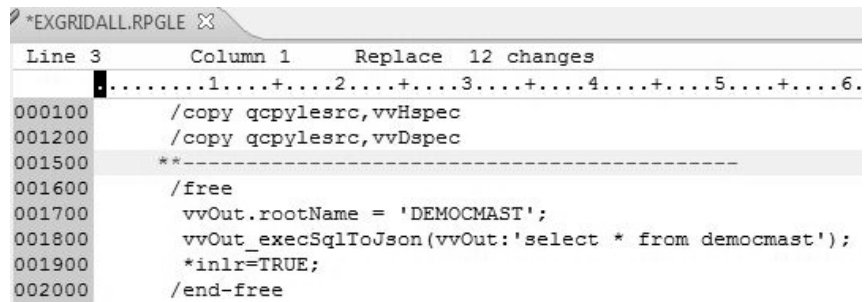
This is not a marketing slide... this is a live demo app running on my system, and the demo customers listed are being pulled from a file on my system.

(Flux capacitors are doing pretty well, which we'd expect...) 😊

HTML with Javascript front-end (partial):

```
// data store
var dsCustomers = new Ext.data.JsonStore({
  autoLoad: true,
  url: 'vvcall.pgm',
  baseParams: {
    pgm: 'exgridall',
    action: 'getCustRecs'
  },
  root: 'DEMOCMAST',
  sortInfo: {
    field: 'CUSNO',
    dir: 'ASC'
  },
  fields: ['CUSNO', 'CNAME', 'CCITY', 'CSTATE']
});
```

RPG back-end program:



```
*EXGRIDALL.RPGLE
Line 3      Column 1      Replace 12 changes
.....1.....+.....2.....+.....3.....+.....4.....+.....5.....+.....6.
000100      /copy qcpylesrc,vvHspec
001200      /copy qcpylesrc,vvDspec
001500      **-----
001600      /free
001700          vvOut.rootName = 'DEMOCMAST';
001800          vvOut_execSqlToJson(vvOut:'select * from democmast');
001900          *inlr=TRUE;
002000      /end-free
```

Both of these are edited from within RDp! Again, no proprietary IDE. They do have an autocode feature to help speed up development.

Possible challenges

1. Takes time to learn

It will likely take some of your own time... invest in yourself!

2. Less efficient initially

Work overtime to compensate until you're up to speed and reaping the benefits, and read at home... invest in yourself!

3. Cost

Programming in /free is free. WDSC is free*. Using procedures and service programs is free. Embedded SQL is free. Resources on the web are free. Even Valence is free for the community developer license!

Possible challenges to implementing the use of more modern techniques

*sort of... it should be on your install disk from IBM

Possible challenges (cont)

4. Unreceptive management

This is a tough one... could be rooted in fear of the unknown and/or not understanding the point. Don't go against your manager and standards, but work to show the benefits, maybe doing it both ways to show them.

5. Establishing standards that work with your environment

This takes some research and thinking about your environment, but you can learn a lot from others online, etc.

6. Takes time to get things set up

Not much once you have #1-5 done.

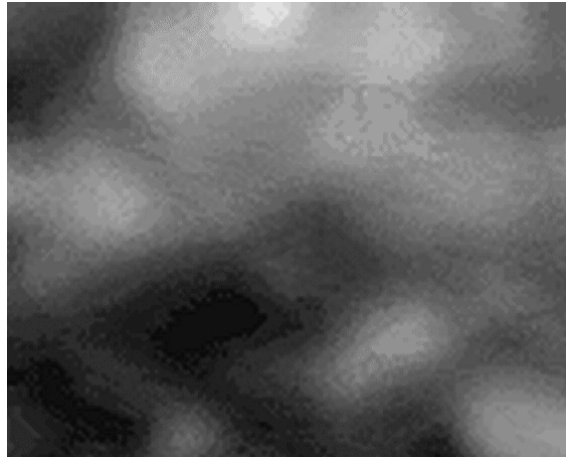
All these are real challenges. But if you don't make a change, someday an executive is going to come in and say "this isn't modern, get rid of it!". But the machine is modern, it's just us that may not be modern and taking advantage of what it can do!

And a big benefit to all this... It makes programming fun again!

Thank you!

Anders Roede

aroede@signetarmorlite.com



Some reference material:

- www.SystemiNetwork.com
- www.RPGWorld.com
- Free-Format RPG IV by Jim Martin
- Programming in RPG IV by Bryan Meyers and Jim Buck
- Functions in Free-Format RPG IV by Jim Martin
- RPG TnT by Bob Cozzi
- SQL for DB2 by James Cooper and Paul Conte
- Javascript for the business developer by Mike Faust
- www.CNXCorp.com/Valence
- Lots of online resources