System i Developer

Simplify your RPG Applications with RPG OA

Jon Paris

Jon.Paris @ Partner400.com
www.Partner400.com
www.SystemiDeveloper.com

Paris Gantner400

Notes

Paris Gantner400

Jon Paris is co-founder of Partner400, a firm specializing in customized education and mentoring services for IBM i (AS/400, System i, iSeries, etc.) developers.

Jon's career in IT spans 40+ years including a 12 year period with IBM's Toronto Laboratory.

Together with his partner Susan Gantner, Jon devotes his time to educating developers on techniques and technologies to extend and modernize their applications and development environments. Together Jon and Susan author regular technical articles for the IBM publication, IBM Systems Magazine, IBM i edition, and the companion electronic newsletter, IBM i EXTRA. You may view articles in current and past issues and/or subscribe to the free newsletter at: www.IBMSystemsMag.com.

Jon and Susan also write a weekly blog on Things "i" - and indeed anything else that takes their fancy. You can find the blog here: ibmsystemsmag.blogs.com/idevelop/

Feel free to contact the author at:

Jon.Paris @ Partner400.com

Agenda

Partner400

What is Rational Open Access: RPG Edition ?

- Most people know it as Open Access for RPG or OA
- Why did IBM develop it
- What can you do with it

The basic mechanics of OA

- The buffer interface (Record level)
- The name/value pairs interface (Field level)
- How to choose which version to use

Three OA Examples

- A simple web service handler that uses the buffer interface
- A name/value handler that creates CSV files
- And one that "Webulates" Printer output

Notes

Partner400

What Needs does Open Access Help Meet?

Are you one of many who have RPG programs that write to the IFS

- You probably clone code every time you need a new one
- But it requires that everyone in the shop who writes or maintains these programs be aware of at least the basics of using C-style I/O functions

Perhaps you write to a database and then use CPYTOIMPF

- What a waste of time, effort, and disk storage

Wouldn't it be better if regular RPG could write directly into the IFS ?

- With an OA handler you could do just that
- And only the person who writes the handler has to understand C functions
  - Others in the shop only need know how to code conventional I/O operations

This is the type of requirement that drove IBM to develop OA

Notes

This to me sums up one of the major benefits of OA - anyone with basic RPG skills can write programs that can do anything the writer of the handler can imagine - or at least code.

When we write a program that writes to a database we don't have to worry about the mechanics of how it is achieved. The compiler writers and folks who write the operating system and DB2 code take care of all that for us. Why should writing to the IFS be any different? Or calling a web service? Or ...

OA allows business programmers to focus on what they do best - and leaves the "twiddly bits" to those of us poor soles who take pleasure in such things.

So Why Did IBM Develop Open Access ?

Partner400

They can't add new features fast enough to meet user's needs

- And even if they could, we take too long to move to new releases
  - From IBM's decision to implement something to the average shop having it available takes between 2 and 3 years

Most modern languages have an API integration mechanism

- This allows third parties to add features to the language
  - PHP, Java, Python, Ruby, etc. all have this

That's what OA brings to RPG

- A mechanism to extend RPG's I/O model to any "device" you like
- Without having to use the sometimes clumsy API approach

The mechanism that utilizes this interface is known as a "Handler"

- This is a program/procedure that performs the underlying processing
  - In other words for this particular file it replaces the operating system

The handler can do anything it wants

- Call Java programs, C functions, stored procedures on other systems, ...
- All while RPG controls the gathering/distribution of the fields involved

Notes

Partner400

When we say 2 -3 years we are actually being quite generous. It took a lot of people many, many, years to get off V5R4 - some are sadly still there even though it is now out of service. V7.2 has been available for a while now - add to that the fact that it probably takes IBM a minimum of 18 months to design and implement a new function and you can see the problem.

Today's world moves too fast for that kind of delay. Would anyone have guessed the impact that mobile devices, cloud computing and social media would have 10 years ago? Very few did.

Other languages have been able to deal with this by having a clearly defined mechanism whereby new functionality can be "plugged in" while still taking advantage of the language's native functionality. Many people have tried to do that for RPG by supplying Subprocedures in Service Programs - for example Scott Klement's HTTPAPI. But the problem that all of these face is that calling APIs can be messy by comparison with what RPG has taught us to expect.

Think of a CHAIN operation for example. With one single instruction we cause the compiler to gather up the key fields, request the matching record, and if it is found distribute the data to the requisite fields. If it is not found then a "not found" status is set and we can interrogate that in subsequent operations. To do the same with an API interface requires that we provide the key fields (probably as parameters) and supply a data structure to hold the retuned information. If the data is located then we are responsible for distributing it to the required fields. Many RPGers find that difficult to get their heads around. If they are accessing a web service they would rather it be more like a CHAIN operation.

If you find yourself tempted, as many have been, to say that APIs aren't that tough to use, then you are probably not the target audience. Or are you? Perhaps you should be writing OA Handlers to allow others in your shop to more easily access the functionality. Not everyone needs to be an API expert!

So What Might You Do With It ?

Use it to let RPG directly interface with ANY non-standard "device"

- While still using standard RPG I/O operations
  - READ, WRITE, CHAIN, EXFMT, etc. etc.

What kind of devices?

- Browsers
- Mobile devices
- Cloud-based storage or other resources
- Web Services
- Other databases (SQL Server, Oracle, MySQL, ...)
- XML files
- Excel Spreadsheets

Who can take advantage of it?

- Software vendors
- Open Source developers
- YOU !

Notes

Imagine if ...

You could use simple READ. WRITE, etc. operations to process files in the IFS - Or read through IFS directory structures - Or produce web pages - Or create Excel spreadsheets - Or ...

IBM hasn't actually added these capabilities to RPG, but third party vendors are doing so - and you can too. IBM's Open Access for RPG gives us the tools to do the job.

Rational Open Access: RPG Edition

Originally available with V7.1

Originally released as a Tier Priced product

- Luckily Rational saw the error of their ways
- It is now included with the compiler
  - Some V7.1 users require PTFs to instal - see the notes page for details

Many experienced RPG programmers involved in the design

- Many vendors have developed products
  - Asna, BCD, looksoftware, Profound Logic, RJS, and Seagull to name a few
- All except RJS have focussed on web/mobile interfaces
  - They are after all the Silver Bullet
- Look for more vendors and Open Source projects down the road
  - Now that there is no charge the barrier to OS developers has disappeared

Notes

These are the required PTF numbers to install OA on releases 6.1 and 7.1 as at May 1st, 2012. For the latest information on PTF levels etc. go to the RPG Café (http://tinyurl.com/rpg-oar-ptfs) - that will also give you access to other related ares of the RPG Cafe.

Release	Run Time	Compiler	Copy Sources
6.1	SI45906	SI45904	SI45910 and SI45908
7.1 (Current)	SI45905	SI45903	SI45909 and SI45907
7.1 (Previous)	N/A	SI45902	N/A

How Does Open Access Work?

The Handler programmatically determines the amount of data it wants

- It makes the selection at the time of the open operation

It can elect to simply be passed the buffers

- In other words, pretty much the same as the old SPECIAL file interface
  - But with the addition of full file and record names, key data, etc.

Or choose to be passed a full description of each field

- Known as a Name/Value handler
- Along with the data and keys
- The description includes information such as:
  - Field name, Length, Data type, Decimal places, etc.

The Handler can pass back full status data to the user program

- RRN, Status codes, etc.
  - In other words all of the information that an existing program being converted might currently be using

Notes

The full IBM documentation for IBM OA can be found in the RPG Cafe at: http://tinyurl.com/29jyk8j

Sorry about the tiny url but IBM's direct page links are about 70 characters long!

It does not include much in the way of examples. IBM are in the process of developing those and will be shipping them in source form.

Writing a handler to perform the kinds of functions discussed earlier is not that hard if it is specific to a particular task/file. Writing a generic handler that can be flexible enough to handle (say) conversion of an arbitrarily chosen display file is a very different matter and requires an in depth understanding the workings of display files. It is hardly surprising that vendors from the 5250 web enablement arena were the first to offer commercial handlers.

What type of Handler do you need?

Use a "Buffer" handler if:

- Processing requirements are specific for each type of "file"
  - For example UPS web services are different to FedEx web services
  - SOAP web services require different processing from REST web services
- Any variations are not readily parameterized

Or ... When the handler does not need to know the content of the data

- e.g. It simply passes it on as a "lump" such as when using a data queue

Use a "Name/Value" handler when:

- You need to adapt the output of an existing program
- The behaviour is consistent
  - For example writing to or reading from an IFS file
  - Reading and writing Excel spreadsheets

The more generic a handler is the more complex it is to write

- Generally you won't want to write your own 5250 handler
  - we will talk more about why later
- But you can easily write handlers for IFS files, spreadsheets, etc.

Creating a "Buffer" Style Handler

Determine the type of RPG file and Op-code(s) that are "Best fit"

- Perhaps DISK file and CHAIN for a web service
- WORKSTN and EXFMT for browser/smart phone
  - Or just for situations where multiple input and output formats are required

Determine the data required

- Key fields (if any)
- Input layout
- Output layout

Create appropriate externally-described file (PF, PRTF, DSPF)

- This will be used to externally describe the buffer layout in the handler
  - In some cases you may require multiple record formats

Determine if additional data is required

- For example the name of an IFS file

Write the handler to process the new file type

- The HANDLER keyword will identify the specific handler to use

What Data is Passed to/from the Handler?

Copious information about the file such as:

- Library, file and member names
- Whether file is keyed, externally described, blocked, a subfile ...
- The file's Device type, format level indicator, RRN, number of keys
- The RPG operation code requested

Record data (input and/or output as appropriate)

- In either raw buffer or Name/Value forms

The Handler must supply feedback data to the RPG run-time such as:

- I/O Status
- Open or I/O Feedback information unique to how this handler works
- Whether the handler wants raw buffer data or Name/Value pairs

Notes

The full IBM documentation for IBM OA can be found in the RPG Cafe at:

http://tinyurl.com/29jyk8j

This chart summarizes the information passed in the OA parameter to the hander - yes, there is only one parameter, but it's a somewhat complex nested Data Structure!

Due to the length and complexity of the DS used for this parameter, we won't go into the full details of the RPG description of it in this session. /Copy members with the descriptions are supplied by IBM for those writing handlers. You can find copies of them in RPG Cafe.

One of the more critical pieces for the Handler developer to do is to specify how they want to receive the record buffer information. That is, whether they want to receive the raw buffer, much like the Special File programs do (plus a bit more information, such as the file/member names, keys, if any, etc.) or whether they prefer to receive the data formatted into what the OA documentation refers to as "Name/Value pairs". Actually, as we'll see on the next few charts, there's a lot more information there than just names and values for each field.

The handler needs to be able to supply to the RPG run-time all relevant status information related to the type of file it represents. For example, if the handler is designed to be used as a DISK file it would include things such as the EOF status, RRN, etc. It is important that the programmer using the handler be able to obtain all of the information he might reasonably expect for this device.

What Data is Passed to/from the Handler? (cont.)

Optional parameter passed to the Handler from the User Program

- Can be in any format defined by the handler
- Would normally be supplied as a /Copy source
- Only a single parameter is passed so you may need to use nested DS
  - Or use pointers for structures that can vary in length

Handler-specific information to handle the “state” of the request

- Needed so that the same handler can be used for multiple files
- And to allow it to "remember" its current status

Notes

A well written handler also needs to be able to handle multiple instances of the file so “state” information must be maintained. The RPG run-time will pass back the link you supplied to the state information on each call.

If the Handler required extra parameter data, the pointer to that data is supplied in the main OA parameter structure.

Note that there is only one extra parameter permitted but of course it could very well be a Data Structure if more than one piece of information is needed.

How Does Your Handler Get This Information ?

It is passed a single parameter

- This contains data about the file and the operation requested
- It also includes fields that the handler can use to indicate errors etc.
- much of the data is represented by pointers
  - This is for the more complex structures such as those for field values
    - Don't go getting "pointer phobic" they are very very simple to deal with
- A pointer is also used to supply the link to the optional extra parameter
  - Since its size and layout is not known it is really the only way it could be done

To work with this parameter there are 2 things that you MUST know

- Or at least be familiar with their basic concepts and usage

BASED fields and structures

- So you can map the data referenced by the pointers

Qualified Data Names, Nested Data Structures and DS Arrays

- Because that is how the RPG run-time passes data to the handler

Charts are included in the handout but we won't have time to study them

Notes

As noted on the chart, you have to have a basic understanding of pointers and the relatively new V5R1/R2 features for Nested Data Structures and DS Arrays along with the qualified data names that go with them.

The pointers are really very easy to handle - you don't need to do anything other than copy and/or reference them. Similarly the DS are not complex - just different to what you may be used to.

Refresher on Nested DS and DS Arrays

These features were added in the V5R1 and V5R2 timeframes

- They allow for one DS to be nested within another
- And for DS to be defined as arrays - no more Multi-Occurrence DS!
- The TEMPLATE keyword used in the OA definitions was added in V6.

The LIKEDS keyword is the magic that allows them to be nested

- These are QUALIFIED data structures and must use the . notation

```
 // Example of a Template DS - used to nest one DS in another
D myData          DS                  TEMPLATE
D   field1                         5i 0
D   field2                       256a   Varying

D myComplexDS     DS                  Qualified
D   aField                        10a
D   anArray                        7p 2 Dim(12)
D   aDSArray                           LikeDS(myData) Dim(99)

D x               S                5i 0

 /Free
  // This is how the fields are referenced
  For x = 1 to %Elem(myComplexDS.aDSArray);

    myComplexDS.aDSArray(x).field1 = x;

  EndFor;
```

Notes

We really don't have time to go into all of the details of Qualified DS, nested DS and DS arrays but we have written several articles on them that you can reference.

But the example here is sufficient for everything you will encounter in OA. And remember - IBM supplies all the definitions - you just have to be able to read them!

Refresher on BASED Storage

When a variable or DS is defined as BASED ...

- The compiler will not allocate memory to it as it normally would
- It is your responsibility to set the Basing Pointer with storage

In the case of OA the pointers are passed to you as parameters

- The example below shows the process in a significantly simplified form

Many handlers will also require to use dynamic storage (%ALLOC)

- We will look at this when we look at a handler that requires this

Note - what I am showing here is actually a "visible" version of how all parameters are passed - i.e. By Reference

```
D MyProcedure     PI
D   p_myData                      *

  // Simple Example of a BASED DS
D myData          DS                  BASED(p_myData)
D   field1                       5i 0
D   field2                     256a   Varying
```

Pointer is passed in as a parameter

And used here as the basing pointer for the DS

Notes

As you can see from the example - which shows how you might allocate space for the state information storage - there is really nothing very complex here. You just define the field or DS as BASED and name the pointer to be used.

You can then either allocate storage as we have done here or - in the case of the extra parameter - simply reference the pointer passed by the RPG run-time.

Our First OA Handler

Let's start with a simple buffer handler

- It will use CHAIN to access currency data from a web service

I created a physical file with three fields

- From currency (FROMCNTRY)
- To currency (TOCNTRY)
- Exchange rate (RATE)

The file is keyed

- On FROMCNTRY and TOCNTRY

Remember that we will never put any data in this file

- We are using it simply as a model for the data we want

Before we look at the Handler, let's see how it is used

Notes

The physical file we are using is intended only to provide a template for the data we will pass to and from the web service. So we will not normally ever store data in the file. However, if we were at all concerned about the web service failing we could always store the latest value received in the actual file ... just a thought.

Using The Handler

Looks pretty normal doesn't it!

- Nothing that you wouldn't see in a regular RPG program
- With the exception of the HANDLER keyword of course

```
      // Tests an RPG Open Access SOAP web service handler
     H DFTACTGRP(*NO)
     FCurrRate  IF   E           K DISK    Handler('CURRRTHND')
     D forever         s               n   Inz(*Off)
      /free
         Dou ( forever );
           Dsply ('Enter "From" currency: ') ' ' FROMCNTRY;
           If ( FROMCNTRY = 'end' );
             Leave;
           EndIf;
           Dsply ('Enter "To" currency: ')   ' ' TOCNTRY;
           Chain (FROMCNTRY: TOCNTRY) CurrRate;
           If %Found(CurrRate);
             Dsply ('Conversion rate from ' + FROMCNTRY +
                    ' to ' + TOCNTRY + ' is ' + %Char(RATE));
           Else;
             Dsply ('No record found for ' + FROMCNTRY +
                    ' to ' + TOCNTRY );
           EndIf;
         EndDo;
```

Using The Handler

Full source of the code on the chart

```
      // Tests an RPG Open Access SOAP web service handler
     H DFTACTGRP(*NO)
     FCurrRate  IF   E           K DISK    Handler('CURRRTHND')
     D forever         s               n   Inz(*Off)
      /free
         Dou ( forever );
           Dsply ('Enter "From" currency: ') ' ' FROMCNTRY;
           If ( FROMCNTRY = 'end' );
             Leave;
           EndIf;
           Dsply ('Enter "To" currency: ')   ' ' TOCNTRY;
           Chain (FROMCNTRY: TOCNTRY) CurrRate;
           If %Found(CurrRate);
             Dsply ('Conversion rate from ' + FROMCNTRY +
                    ' to ' + TOCNTRY + ' is ' + %Char(RATE));
           Else;
             Dsply ('No record found for ' + FROMCNTRY +
                    ' to ' + TOCNTRY );
           EndIf;
         EndDo;
```

This is the ONLY difference from dealing with a regular disk file

The Handler Itself - Data Definitions

The record layout we need is the Input one

- Because the user program is "reading" from the handler

```
 // Set up Rates file as template to facilitate LikeRec definitions
FCurrRate  IF   E           K DISK     Template

D CurrRateHndlr   Pr                  ExtPgm('CURRRTHND')
D   info                              LikeDS(QrnOpenAccess_T)

  // DS to map record layout to return data
D CurrRateD       DS                  LikeRec(CurrRateR: *Input)
D                                     Based(info.inputBuffer)

  // DS to map supplied keys
D CurrRateK       DS                  LikeRec(CurrRateR: *Key)
D                                     Based(info.key)
```

The Handler Itself - Data Definitions

This is the portion of the source that the extracts on the chart were taken from

```
H DFTACTGRP(*NO) BNDDIR('HTTPAPI':'QC2LE')
  // Set up Rates file as template to facilitate LikeRec definitions
FCurrRate  IF   E           K DISK     Template

D CurrRateHndlr   Pr                  ExtPgm('CURRRTHND')
D   info                              LikeDS(QrnOpenAccess_T)

D CurrRateHndlr   PI
D   info                              LikeDS(QrnOpenAccess_T)

  // Standard IBM supplied Open Access definitions
 /copy simplify,qrnopenacc

  // DS to map record layout to return data
D CurrRateD       DS                  LikeRec(CurrRateR: *Input)
D                                     Based(info.inputBuffer)

  // DS to map supplied keys
D CurrRateK       DS                  LikeRec(CurrRateR: *Key)
D                                     Based(info.key)
```

The Handler Itself - Main Handler Logic

Partner400

Handler must issue an error if told to do anything it cannot do - e.g. SETLL

- A "real" handler would issue a descriptive message for such errors
  - Without it you don't know the real cause of the problem - you just have a generic error

```
If info.rpgOperation = QrnOperation_CHAIN;
    doChain();
// Open/Close are basically no-ops but need handling
//   since they are valid ops. Close just sets on LR
elseIf (info.rpgOperation = QrnOperation_OPEN);
    // Nothing to do here ...
elseIf (info.rpgOperation = QrnOperation_CLOSE);
    *InLR = *On;
```

```
// Meanwhile in the doChain() subprocedure
.......
If (charRate <> NOT_FOUND) ;
   currRateD.rate = %dech(charRate: 15: 5);
   info.found = *on;
else;
   info.found = *off;
endif;
```

This places the rate value in the buffer

This logic is effectively setting the %Found BIF

The Handler Itself - Main Handler Logic

Partner400

This is the portion of the source that the extracts on the chart were taken from

```
If info.rpgOperation = QrnOperation_CHAIN;
    doChain();
// Open/Close are basically no-ops but need handling
//   since they are valid ops. Close just sets on LR
elseIf (info.rpgOperation = QrnOperation_OPEN);
    // Nothing to do here ...
elseIf (info.rpgOperation = QrnOperation_CLOSE);
    *InLR = *On;
else;
   // Any other operation is unsupported so notify RPG
   info.rpgStatus = 1299;  // general error status
endif;
Return;

.....................................

// Meanwhile in the doChain() subprocedure
If (charRate <> NOT_FOUND) ;
   currRateD.rate = %dech(charRate: 15: 5);
   info.found = *on;
else;
   info.found = *off;
endif;
```

Name/Value Style Handlers

In most cases these will be written to be generic

- If it is not, you may as well use the buffer approach

They typically adapt an existing process to a new purpose

- A "print" file is output as a browser page
- Writing to a "disk" file actually produces a CSV file in the IFS
- Reading a "disk" file reads data directly from a spreadsheet

They use the information supplied to process the data

- The more truly generic you try to make the handler the more complex
- Or you compromise and limit the functionality to programs that "fit"

Name/Value is requested via an indicator set during the “Open”

- On every subsequent I/O operation the Handler receives the data
- It is in the form of a DS array containing information about each field
  - You'll see some of the information passed on the next chart
- Note that the actual data is always in human readable form

Notes

The decision about which way to receive the information is supplied by the Handler by turning on (or off) an indicator in the OA parameter structure. The indicator is turned on to request the “name/value pair” style of data. Once set it should not be changed.

If the indicator for Name/Value pairs is turned on, then the pointers to the buffers will be null on subsequent calls to the Handler. Conversely, if the indicator is off, then pointers to the buffer(s) will contain values and the pointer to the name/value pair data will be null.

There’s a lot more information there than just names and values for each field. The DS that contains the Name/Value information is defined as a DS than contains the number of fields and a DS array that contains all the details about each field. The list of field information on this chart is a summary and not 100% complete. However on a later chart, we will show what the RPG definition of this DS might look like.

Name/Value Info - QrnNamesValues_T (extract)

The highlighted fields are the ones we will use in our first example

Subfield	RPG Type	Set by	Used by
externalName	10 A	RPG	Handler
dataType	3U 0	RPG	Handler
numericDefinedLen	3U 0	RPG	Handler
Decimals	3U 0	RPG	Handler
dtzFormat	3U 0	RPG	Handler
dtSeparator	1A	RPG	Handler
Input	N	RPG	Handler
Output	N	RPG	Handler
isNullCapable	N	RPG	Handler
hasNullValue	N	RPG and Handler	RPG and Handler
valueLenBytes	10U 0	RPG and Handler	RPG and Handler
valueMaxLenBytes	10U 0	RPG	Handler
valueCcsid	10U 0	RPG	Handler
value	*	RPG	Handler

Notes

Note that the data structure defined on this chart represent only a portion of the parameter structure that is passed to the OA Handler program.

The field in the chart **numericDefinedLen** is only used for numeric fields and contains the declared length of the numeric fields. This information is in digits for decimal fields and in bytes for Integers, unsigned integers and floating point fields.

Template D specs for this structure as well as all the other structures used in the OA parameter structure are provided with the OA product. The name of the supplied structure template for the structure shown in this chart is QrnNameValue_T.

This structure is nested inside another structure that looks like the one below. Note that this DS contains an integer (num) that holds the number of fields in the record (and therefore the number of active entries in the DS array (field) that is also in the structure. We'll see how this information gets processed in a moment.

```
D QrnNamesValues_T...
D                 DS                  QUALIFIED TEMPLATE ALIGN
D   num                           10I 0
D   field                             LIKEDS(QrnNameValue_T)
D                                     DIM(32767)
```

The Need for State Information

What do we mean by "state" ?

- All the things that are related to a specific file at a given point in time
  - e.g. Current RRN, status information such as open/closed, EOF, etc.

If you ever coded MRTs for S/34, S/36, or CICS you know this stuff

- If you have ever coded a web application you also had to deal with it

Most handlers will normally need to cater for this

- They may be handling multiple files in one program
  - We'll see an example of this soon
- And if they don't handle it there may be a nasty mess!

Luckily IBM have provided a very simple and elegant solution

- You just tell it the address of where you stored the data and it passes it back to you on all subsequent calls
- You just have to know how to declare a Basing pointer
  - More on this in a minute if you don't know

Notes

State information is critical to the correct functioning of almost any handler you will ever write. Luckily for us IBM have given us a very simple and elegant solution. It does all the heavy lifting - we just have to know how to use a basing pointer on the DS or variable in which we are keeping the state information and RPG guarantees that it will give us the correct pointer each time it calls the handler.

Using Dynamic Storage for State Data

An OA Handler should be able to process any number of files

- So we must isolate any file specific information

This "file specific" data is what we mean by state information

- Normally RPG (or rather the operating system) worries about these things
- But when writing an OA Handler you must deal with it yourself
- The programmer who uses your handler knows nothing about this
  - Just as with a regular RPG file

```
  // Example of a BASED DS such as might be used to store state information
D myData          DS                  BASED(p_myData)
D   field1                         5i 0
D   field2                       256a   Varying

 /Free
  // Allocate sufficient storage to hold the DS
  p_myData = %Alloc( %Size(myData) );

  // When you are done with it - set it free!
  DeAlloc p_myData;
```

Using the Handler

```
  // IFS output file - file definition specifies fields to output
FIFS_OUT1  o    e             Disk  Handler('HND_IFS_J2' : ifs_info1)
F                                   UsrOpn

FIFS_OUT2  o    e             Disk  Handler('HND_IFS_J2' : ifs_info2)
F                                     UsrOpn

FTEST1     if   e             Disk

  // Define fields for IFS file names
D ifs_info1       s           5000a  varying
D ifs_info2       s           5000a  varying

 /free
  ifs_info1 = '/Partner400/IFS_J2_1.csv';
  ifs_info2 = '/Partner400/IFS_J2_2.csv';

  open IFS_OUT1;
  open IFS_OUT2;

  read TEST1;

  dow not %eof(TEST1);
     Write IFS_OUT1R;
     Write IFS_OUT2R;
    // read next record
     Read TEST1;
  enddo;

  *inlr = *On;
```

Files IFS_OUT1 and IFS_OUT2 contain a subset of the fields in file TEST1. The format of those records determine the action of the handler.

Notes

Notice that because the handler maintains state information, the exact same handler can be used to for multiple files in the same program as shown here. The file, and field information passed enables the routine to customize its output based on the file definition used.

These charts are based on the article "Getting a Handle on RPG's Open Access" which you can find here:
www.ibmsystemsmag.com/ibmi/OAR_handlers/33511p1.aspx

In the article you will also find a link to download the code if you are interested in "playing" with these programs.

The HANDLER Keyword

Identifies the program or Service Program procedure to be used

- Reference can be any of the following:
  - A literal (as in the example below)
    - Subprocedure name is enclosed in parentheses if required
  - Name of a prototype for a bound procedure.
  - A procedure pointer (or procedure pointer literal - %PADDR)

Optional parameter can be specified

- Used to pass data that is not available directly from the I/O operation
  - For example the name of the IFS file/directory to be used
  - Similar to the extra parameters used by SPECIAL files
- Only a single parameter can be used
  - But it could be a nested DS or DS array or ...

This is the ONLY additional syntax added to support Open Access

```
FIFS_OUT1  o    e             Disk    Handler('HND_IFS_J2' : ifs_info1)
F                                     UsrOpn

FIFS_OUT2  o    e             Disk    Handler('HND_IFS_J2' : ifs_info2)
F                                     UsrOpn
```

Notes

With RPG OA the file's Device type remains as it was – either printer, disk, workstation, etc. but the use of the Handler keyword indicates that this file is "special" in that the program or procedure specified is called whenever any operation is done to this file.

Just like the Special File program can accept extra parameters, the OA handler can also get extra parameters from the program using the handler. So, to pass in the name of the IFS file for writing, we saw how we could do this using the PList when using the Special File.

With RPG OA, there is only one additional parameter that can be passed (SPECIAL files could have any number), so if there is more than 1 piece of information needed to be passed in, that parameter will need to be a data structure containing the pieces of data needed, as shown in the D specs in this example.

Simple IFS Handler - Basic Data Definitions

```
  // Interface uses QrnOpenAccess_T to define the parameter
D HND_IFS_J2      PI                  ExtPgm('HND_IFS_J2')
D   info                              likeds(QrnOpenAccess_T)

  // Field Names/Values structures
D nvInput         ds                  likeds(QrnNamesValues_T)
D                                     based(info.namesValues)

  // Structure to map the "additional info" parameter passed
  //   by the RPG program. I use it to pass the IFS file name.

D ifs_info        ds                  Qualified
D                                     based(info.userArea)
D   path                       5000a  varying

  // Used by the IFS routines - determines which file is to be used
  //   Maps to dynamic storage allocated when opening the file.
  //   Stored in the rpgStatus field in the info structure

D fileHandle      s              10i 0 based(info.stateInfo)
```

Notes

As you can see the majority of data definition is done by simply referencing the IBM supplied templates.

Only the additional parameter used to pass the IFS name and the definition of the state information have to be "hand coded" as they will be different for each and every handler you write.

Main Logic From IFS Sample Handler

```
If info.rpgOperation = QrnOperation_WRITE;

   If ( writeFile(fileHandle) = fileError );
      info.rpgStatus = errIO;
   EndIf;

elseIf info.rpgOperation = QrnOperation_OPEN;

   info.useNamesValues = *On; // Set up to use Name/Value info

   info.stateInfo = %Alloc(%Size(fileHandle));

   clear fileHandle;

   fileHandle = openFile (ifs_info.path); // Open file

   if fileHandle = fileNotOpen;
      info.rpgStatus = errImpOpenClose; // Open failed
   EndIf;
```

Allocate memory for file handle and place in State Information area for later calls. See notes.

Notes

This simple Handler program simply writes to a CSV file to the IFS. We're taking advantage of RPG OA's ability to get the data and the format of the data in the handler so that we can make this program generic – it can write any file in any format to the IFS.

In addition, this Handler takes advantage of the ability to keep track of state information between calls to the Handler program. All well written handlers should take advantage of this facility. It is essential if, for example, one program wants to write to multiple IFS files. The RPG run time will keep track of which piece of state information belongs to which file. In this example, the state information is simply the file handle that was returned by the IFS open() API for a particular IFS file. If more than 1 file that used this handler were used in a program, usage of the state information ensures that it will receive the appropriate file handle on each subsequent call.

The D spec code for the **fileHandle** is below. Note that is based on a pointer. That pointer is received in the parameter block. In a similar fashion, the userArea pointer is passed in the parameter block as well. It's value is used as the basing pointer for a DS called ifs_Info in our program. That DS will contain the extra parameter which has the IFS path/file name for the file to be written.

```
D fileHandle      s             10i 0 based(info.stateInfo)
```

This chart shows the main logic of the Handler program. Note that the logic simply detects what RPG operation was requested and takes the appropriate action in each case. We have only coded logic for the Open, Write and Close operations. Any other op-code would result in an error.

Note that the DS named **info** which is used to qualify many of the fields such as rpgOperation, stateInfo and useNamesValues, is the name we have given to the structure received as the parameter to the handler. It is defined as being likeds(QrnOpenAccess_T) .

Extract From Simple IFS Handler (contd.)

```
elseif info.rpgOperation = QrnOperation_CLOSE;

   closeFile (fileHandle);

   // free the state information storage and null out the pointer
   dealloc(n) info.stateInfo;

   info.stateInfo = *null;

else;
   // Any other operation is unsupported so notify RPG
   info.rpgStatus = 1299;  // general error status

endif;

Return;
```

Notes

It is essential that you deallocate any storage that you allocated to hold state information.

It is also a good idea to set the related pointer to null so that the data can never be referenced by mistake.

It would be a good idea to send an info message to the program's message queue - not just place the error status in the RPG control area. The program will work without it - but the message in the log will not indicate the cause of the failure in the way that you are used to with normal RPG errors.

Simple IFS Handler's Write Routine

```
For i = 1 to nvInput.num; // Process all fields in record
   pvalue = nvInput.field(i).value; // set up to access data

   If ( nvInput.field(i).dataType = QrnDatatype_Alpha )
     Or ( nvInput.field(i).dataType = QrnDatatype_AlphaVarying);
       buffer += quote
     + %trimR( %subst( value: 1: nvInput.field(i).valueLenBytes ))
     + quote;
   ElseIf ( nvInput.field(i).dataType = QrnDatatype_Decimal );
      buffer += %subst(value: 1: nvInput.field(i).valueLenBytes);
   ElseIf ( nvInput.field(i).dataType = QrnDatatype_Date );
      buffer += %subst(value: 1: nvInput.field(i).valueLenBytes);
   EndIf;

   If i <> nvInput.num; // Add comma after all except last field
      buffer += comma;
   EndIf;
EndFor;

buffer += CRLF; // Add record termination
// reply contains length of data written or -1 in case of error
reply = write ( handle: %Addr(buffer:*Data): %Len(buffer) );

Return reply;
```

D specs for this procedure are on the notes page

Notes

The Open and Close routines for the OA handler are very simple so we won't look at them in detail.

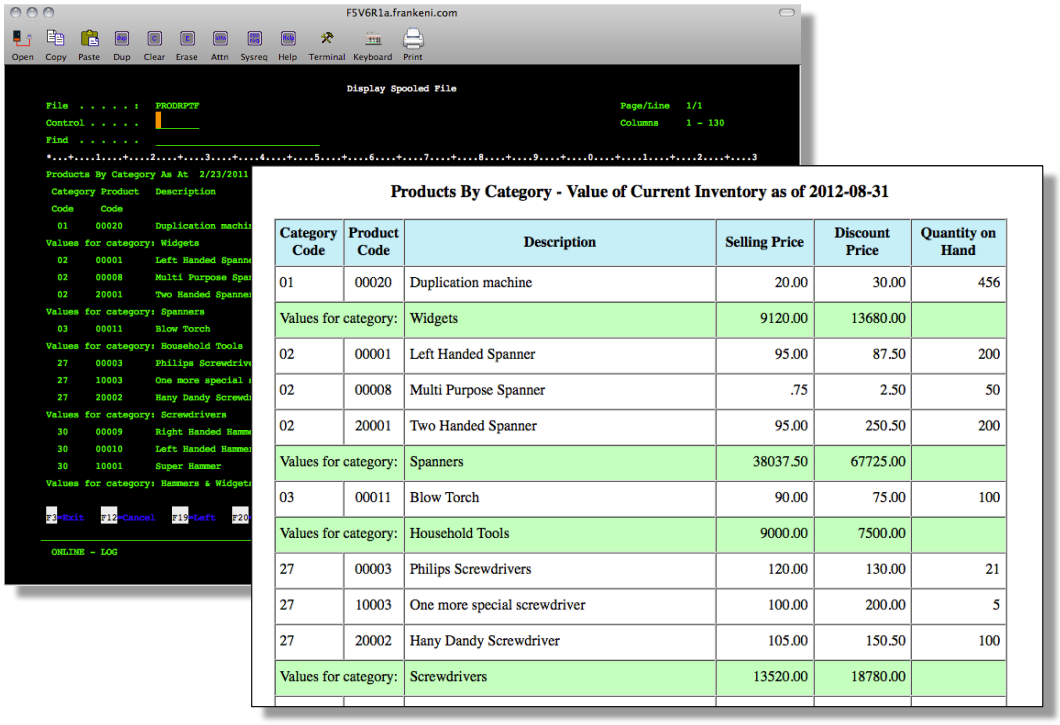
However, the Write routine is worth looking at as it uses the Name/Value pairs to put the data into the IFS file's buffer prior to the write. Note that the DS named nvInput contains the information for each field as shown in the table in the earlier chart, using the field names shown in that table.

Note that because we have the field names, attributes and values, compliments of RPG OA, this handler can write out data in the format of any externally described file. This is in contrast to a Special File program which would have to be hard-coded to a specific file.

There was no room on the chart for the D specs of this procedure, so we're including them here.

```
D                 pi                    like(filehandle)
D   handle                              like(fileHandle) value

D buffer          s          32740a     Varying Inz
D value           s          32470a     Based(pvalue)
D i               s             5i 0
D reply           s            10i 0
D comma           c                     ','
D quote           c                     '"'
D CRLF            c                     x'0d25'
```

Another Example - From Print to Web



Products By Category - Value of Current Inventory as of 2012-08-31

Category Code	Product Code	Description	Selling Price	Discount Price	Quantity on Hand
01	00020	Duplication machine	20.00	30.00	456
Values for category:		Widgets	9120.00	13680.00	
02	00001	Left Handed Spanner	95.00	87.50	200
02	00008	Multi Purpose Spanner	.75	2.50	50
02	20001	Two Handed Spanner	95.00	250.50	200
Values for category:		Spanners	38037.50	67725.00	
03	00011	Blow Torch	90.00	75.00	100
Values for category:		Household Tools	9000.00	7500.00	
27	00003	Philips Screwdrivers	120.00	130.00	21
27	10003	One more special screwdriver	100.00	200.00	5
27	20002	Hany Dandy Screwdriver	105.00	150.50	100
Values for category:		Screwdrivers	13520.00	18780.00	

Notes

This transformation required only a one line change in the program as you will see in a minute. Of course we could have done a little more work and enabled the program to output either to the printer or the web - it isn't hard to do. At the simplest level just duplicate the F spec, add USROPN, and add the logic to determine which file to open etc. etc.

The programs used here are described in the article **"Webulating" Printer Output Revisited** which you can find on the IBM Systems Magazine site at:

www.ibmsystemsmag.com/ibmi/developer/rpg/-Webulating--Printer-Output-Revisited

You can find the full sources (and download them in a zip file if you wish) from our web site - you will find them here:

www.Partner400.com/Examples/WebulatingPrinter.html

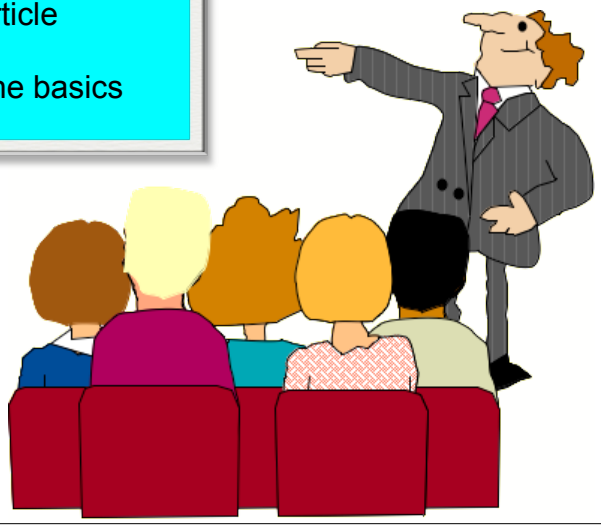
You can also run the program in your own browser by going to: www.partner400.com/CGI/PRODRPT2OA

Just in case ...

In case we don't have time to go through these charts ...

But if we don't you can find all the details in the referenced magazine article

The following pages explain the basics



The Modified Report Program

```
FXProducts IF   E           K DISK
FXCategors IF   E           K DISK    Prefix('XC')
FProdRptF  O    E             PRINTER OFLIND(*IN99)
F                                     Handler( 'WEBPRINTER' )

D WriteTotals     Pr

D totalSell       s              11p 2
D totalDisc       s              11p 2
D currentCategory...
D                 s                    Like(CatCode)
D descr           s              30a

D endOfPage       c                    99

 /Free

  Write Heading;

  // Read product file to prime Do loop and set category control
  Read XProducts;
  currentCategory = catCode;

  // Continue reading until EOF
  Dow Not %EOF(XProducts);
    If catCode <> currentCategory;
      WriteTotals();
      currentCategory = catCode;
    EndIf;
    ...
```

This is the modified line. Nothing else was changed

Notes

This really is the only change that was made to the report program.

We might, of course, want to add the capability to direct the generated HTML to a file rather than to a browser. This would probably require that an additional parameter be added to notify the handler. Later we will see the line of code that could be added to the handler to cause this output to occur.

Other possible changes might include the ability to specify the HTML skeleton to be used for output. Right now, as you will see, the name used is derived from the name of the file being used.

Basic Architecture Behind the Handler

Partner400

Best approach seemed to be an architecture based on existing names

- I used CGIDEV2 - but any HTML template-driven approach would work

HTML Template

- Store template in known location
  - Use the printer file name as the variable part of the file name
- Have one template section for each record format
  - Use the record format name as the section name
- Within the section use field names for the substitution variable names
- Always have a footer section to wrap up the HTML on file close
  - Use the same name name for the footer in each template

Logic

- Build template name from file name and load template
- Use field names in name/value data to update HTML variable content
- Once fields are processed use record format name to write HTML section
- When file close request received write footer section

Notes

Partner400

This architecture was the result of many false starts - and when I finally came to writing the code I was quite disappointed as to how simple it was. I kept thinking that I must have forgotten something.

Please understand that while this handler is generic it cannot simply be used with _any_ existing printer file. It cannot for example handle overlapping print fields. Nor can it handle situations where multiple records are used to build a single line. The second situation can be handled without too much difficulty (a simple matter of programming!) but the first is harder because there is a shortage of APIs to examine the internals of print files. As a result processing the DDS is really the only way to go that we can determine.

But with a little ingenuity you can certainly use the technique to allow non-web RPGers to build simple print reports. In the future we want to try using CNX's Valence and other similar tools that use JQuery and other Javascript libraries to produce richer reports that can, for example, sort their columns etc.

We should also point out that using this approach for a 60 page report is probably not a good idea! Let common sense guide your usage!

HTML Skeleton

```
<!-- Heading -->
...
<table width=800 border=1
       cellspacing=1 cellpadding=5>
  <tr>
    <th width=75>Category Code</th>
    <th width=75>Product Code</th>
    <th width=300>Description</th>
    <th width=100>Selling Price</th>
...
  </tr>
<!-- Detail -->
  <tr>
    <td width=75 height=39 align="center">/%CatCode%/</td>
    <td width=75 align="center">/%ProdCode%/</td>
    <td width=300 align="left">/%Descr%/</td>
    <td width=100 align="right">/%SellPrice%/</td>
...
  </tr>
<!-- CatTotals -->
  <tr>
    <td colspan=2 height=39>Values for category:</td>
    <td width=300 align="left">/%XCCatName%/</td>
    <td width=100 align="right">/%TotalSell%/</td>
...
  </tr>
<!-- Footer -->
</table>
</CENTER></BODY></HTML>
```

```
A            R HEADING
  ...
A            R DETAIL
A              CATCODE    R              3
A              PRODCODE   R             10
  ...
A            R CATTOTALS
A              XCCATNAME      30A        22
A              TOTALSELL      11  2      52
A              TOTALDISC      11  2      67
```

As extract above shows, Section and Substitution variable names match the record and field names used in the original print file

The Handler's Main Driver Routine

```
    If info.rpgOperation = QrnOperation_WRITE;

          writeFile();

       elseIf info.rpgOperation = QrnOperation_OPEN;

          // Specify that we want to use Name/Value intformation
          info.useNamesValues = *On;

          openFile();

       elseIf info.rpgOperation = QrnOperation_CLOSE;

          closeFile();

       else;
          // Any other operation is unsupported so notify RPG
          info.rpgStatus = 1299;  // general error status

       endif;

     Return;

    /end-free
```

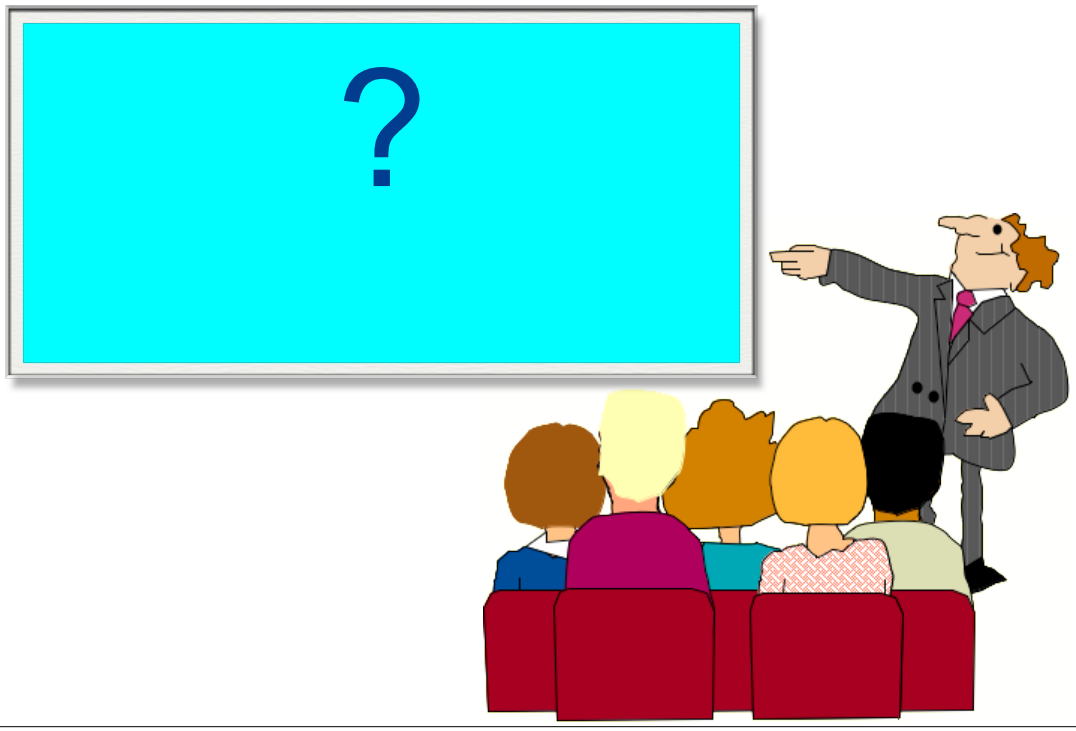
The open() and close() Functions

```
P openFile        b
D openFile        pi

D skeletonName    s             256a   Varying Inz('/Partner400/')

 /free

    // Build HTML template name using File name then retrieve template
    skeletonName += ( %TrimR(info.externalFile.name)
                    + '_Template.html' );

    GetHTMLIFS( skeletonName: '<!-- ': ' -->');

    UpdHTMLVar( 'Date': %Char(%Date()): '0');

 /end-free

P openFile        e

P closeFile       b
D                 pi

 /free

  WrtSection( 'Footer *Fini' );

  return;

 /end-free
```

The write() Function

```
P writeFile       b
D                 pi

D value           s          32470a   Based(pvalue)
D i               s             5i 0

 /free
  // Process all fields in record
  For i = 1 to nvInput.num;
    pvalue = nvInput.field(i).value; // set up to access data

  // Variable names are extracted from name/value information and used as the
  //   substitution variable name
    UpdHTMLVar( nvInput.field(i).externalName:
                %subst( value: 1: nvInput.field(i).valueLenBytes ));

  EndFor;

  // Once all fields are processed, write out current record format

  WrtSection( info.recordName );

  Return;

 /end-free

P writeFile       e
```

That's All Folks - Any Questions ?



Notes

Please feel free to talk to me after the session if you have any questions - or if you think of something later just email me - the email address is on the front of this handout.