



IBM Software Group

# RPG Overview of Enhancements

## 6.1 and 7.1 and Open Access

Barbara Morris IBM



# Agenda

## ➤ **What's new in 6.1**

- What's new in 7.1
- Rational Open Access: RPG Edition



# ILE RPG Enhancements for 6.1

- A new kind of RPG main procedure
- Defining files locally in subprocedures, and passing files as parameters
- Qualified record formats
- Data structure type definitions
- No more compile-time overrides
- Significantly higher limits for the size of variables and array elements
- Relaxation of some UCS-2 rules (available for V5R3/4 through PTFs)
- Store parameter information in the program
- New XML-INTO operations
- Run concurrently in multiple threads; RPG doesn't have to be a bottleneck



# A main procedure which does not use the RPG cycle

- MAIN keyword on the H specification designates one subprocedure as being the procedure that gets control when the program gets called.
- The main procedure is just like any other subprocedure. No RPG cycle.

```
H MAIN(ordEntry)
D ordEntry      PR          EXTPGM('W230E14X')
D   parms ...

* Here is the main procedure
P ordEntry      B
D ordEntry      PI
D   parms ...
...
P               E
```



## Files defined in subprocedures

- Local F specifications follow the Procedure-begin specification and precede the Definition specifications.

```

P writeRecs          b
Foutfile  o      e      disk
D data              ds          likerec(rec)
C              write  rec  data
P writeRecs          e
  
```

- No I and O specifications for local files.
- By default, the file is associated with the subprocedure call. The file is closed when the procedure ends.
- The STATIC keyword can be used to keep the file open when the procedure ends.

# File parameters

- Use the LIKEFILE keyword on a prototype to indicate that the parameter will be a file

```
Fmyfile  if     e      disk

D myProcedure      pr
D   fileParm      likefile(myfile)

// Pass file "myfile" to the procedure
myProcedure (myfile);
```



# File parameters

- Within the procedure, the file is used normally
- A result data structure must be used for I/O operations

```
P myProcedure      b

D myProcedure      pi
D  fileParm        likefile(myfile)
D ds               ds      likerec(myfile.custrec)
/free
```

```
read fileParm ds;
if not %eof(fileParm);
    ...
```



## Qualified record formats

- When a file is defined with the QUALIFIED keyword, the record formats must be qualified by the file name, MYFILE.MYFMT.
- Qualified files do not have I and O specifications generated by the compiler; I/O can only be done through data structures.
- When files are qualified, the names of the record formats can be the same as the formats of other files. For example, you can have FILE1.FMT1 and FILE2.FMT1.

```
Foutfile o e disk qualified
D data ds likerec(outfile.rec:*OUTPUT)
write outfile.rec data;
```

# TEMPLATE keyword for files and definitions

- **TEMPLATE** keyword
  - ▶ Can be coded for file and variable definitions
    - Indicates that the name will only be used with the LIKEFILE, LIKE, or LIKEDS keyword to define other files or variables.
- Template data structures can have the INZ keyword coded for the data structure and its subfields, which will ease the use of INZ(\*LIKEDS).

```

D info_type      ds      template qualified
D   name         25a     inz('**UNKNOWN**')
D   num          10i 0   inz(0)  inz(*sys)

D myInfo         ds      likeds(info_type)
D                inz(*liked)

```



# Avoid compile-time overrides

## EXTDESC keyword and EXTFILE(\*EXTDESC) for F specs:

- EXTDESC keyword Identifies the file to be used for an externally-described file at compile time.

```
FoutputF      o      e      disk      extdesc ( 'MYLIB/MYFILE' )
```

- The EXTFILE keyword is enhanced to allow the special value \*EXTDESC, indicating that the file specified by EXTDESC is also to be used at runtime.

```
FoutputF      o      e      disk      extdesc ( 'MYLIB/MYFILE' )
F                                                    extfile (*extdesc)
```

## EXTNAME enhancement for D specs to specify the library name:

```
D myDs                e ds                extname ( 'MYLIB/MYFILE' )
```



# Larger fields

## No RPG-imposed restrictions on the size of fields

- Data structures can have a size up to 16,773,104.
- Character definitions can have a length up to 16,773,104. (4 less for VARYING.)
- UCS-2 and Graphic definitions can have a length up to 8,386,552 double-byte characters. (2 less for VARYING)

# Larger limit for DIM and OCCURS

## No RPG-imposed restrictions on the number of elements in an array, table, or multiple-occurrence DS.

- The limit on the **total** size of an array or structure remains the same; it cannot be larger than 16,773,104 bytes.
- For example
  - ▶ If the elements of an array are 1 byte in size, the maximum DIM for the array is 16,773,104.
  - ▶ If the elements of an array are 10 bytes in size, the maximum DIM for the array is 1,677,310 (16773104/10).



# Relaxation of some UCS-2 rules

Historically, it was very difficult to use character and unicode fields together in an RPG program. It was necessary to use %CHAR or %UCS2 to convert one type to the other.

- The compiler will perform some implicit conversion between character, UCS-2 and graphic values, making it unnecessary to code %CHAR, %UCS2 or %GRAPH in many cases. This enhancement is also available through PTFs for V5R3 and V5R4.
- Implicit conversion is now supported for
  - ▶ Assignment using EVAL and EVALR
  - ▶ Comparison operations in expressions and using fixed form operations IFxx, DOUxx, DOWxx, WHxx, CASxx, CABxx, COMP.
  - ▶ Note that implicit conversion was already supported for the conversion operations MOVE and MOVEL.



# Eliminate unused variables from the compiled object

- New values \*UNREF and \*NOUNREF are added to the OPTION keyword
  - ▶ For the CRTBNDRPG and CRTRPGMOD CMDs, and for the OPTION keyword on the Control specification.
- The default remains \*UNREF.
- \*NOUNREF indicates that unreferenced variables should not be generated into the RPG module.
  - ▶ Reduces program size
  - ▶ No confusion when debugging
  - ▶ If imported variables are not referenced, it can reduce the time taken to bind a module to a program or service program.

# Store parameter information in the program

The ILE RPG compiler is enhanced to allow information about the parameters for the program or procedures to be stored in the program.

- The information is in the form of Program Call Markup Language (PCML). Starting in V5R2, the ILE RPG compiler was able to output PCML to a stream file.
- Starting in V6R1, the PCML can also be placed directly in the module. H spec PGMINFO(\*PCML:\*MODULE), or use the updated PGMINFO command parameter.
- The information can later be retrieved with the new QBNRPDI API.

## Two new XML-INTO operations introduced by a PTF

- The `datasubf` option gives the name for any subfields that are intended to receive text data associated with a data structure.

```
<emp type="reg" id="13573">John Smith</emp>
```

- The `countprefix` option enables RPG programmers to get more information from an XML document:
  - ▶ The number of RPG array elements that were filled by XML data
  - ▶ For non-array subfields, whether the subfield was filled by XML data or not
  - ▶ Using `countprefix` can eliminate the need for the `allowmissing` option.

# Ability to run concurrently in multiple threads

When `THREAD(*CONCURRENT)` is specified on the Control specification of a module

- ▶ Multiple threads can run in the module at the same time.
- ▶ By default, static variables will be defined so that **each thread** will have its own copy of the static variable. This makes them thread-safe.
- ▶ Individual variables can be defined to be shared by all threads using `STATIC(*ALLTHREAD)`. These variables are not thread-safe, by default.

```
D sharedFld      S      10A      STATIC(*ALLTHREAD)
```

- ▶ A procedure can be serialized so that only one thread can run it at one time, by specifying `SERIALIZE` on the Procedure-Begin specification.

```
P serialProc     B                          SERIALIZE
```



# Agenda

- What's new in 6.1

## ➤ **What's new in 7.1**

- Rational Open Access: RPG Edition



# ILE RPG enhancements for 7.1

- Enhancements for arrays
  - ▶ Sort and search a data structure array
  - ▶ Sort arrays either descending or ascending
- Enhancements for defining procedures
  - ▶ Optional prototypes
  - ▶ One string procedure to handle any string type
  - ▶ Fast return values
  - ▶ Soft-code parameter numbers
- Alias names in data structures
- Miscellaneous
  - ▶ Built-in function to scan and replace
  - ▶ Encrypted debug view
  - ▶ Teraspace storage model
- Open Access: RPG Edition



# RPG: Sort and search a data structure array

## Sort a data structure array using one subfield as a key

```
// sort by name
SORTA info(*).name;

// sort by due date
SORTA info(*).dueDate;
```

## Search a data structure array using one subfield as a key

```
// search for a name
pos = %LOOKUP('Jack' : info(*).name);

// search for today's date
pos = %LOOKUP(%date() : info(*).dueDate);
```



## RPG: Sort ascending or descending

Non-sequenced arrays can be sorted either ascending or descending.

```
D meetings          S          D DIM(100)
  /free
    // sort descending, with the
    // most recent date first
    sorta(d) meetings;
```

(D) extender indicates a descending sort.

(A) Extender indicates ascending (default).



# RPG: Optional prototypes

If a program or procedure is not called by another RPG module, it is optional to specify the prototype.

These are some of the programs and procedures that do not require an RPG prototype

- An exit program, or the command-processing program for a command
- A program or procedure that is never intended to be called from RPG
- A procedure that is not exported from the module



# RPG: Implicit CCSID conversion for parameters

Previous: implicit conversion between the different string types (alpha, unicode, dbcs) for assignment

New: implicit conversion on parameter passing

- Enables writing a single procedure that can handle any string type.
- The procedure is written to have unicode parameters and a unicode return value, and the RPG compiler handles any necessary conversions.

```
// makeTitle() upper-cases and centers the parameter  
alphaTitle = makeTitle(alphaValue : 50);  
ucs2Title = makeTitle(ucs2Value : 50);  
dbcsTitle = makeTitle(dbcsValue : 50);
```



# RPG: Performance returning large values

RTNPARM keyword greatly improves performance when a procedure returns a large value

- The speed of using a parameter with the convenience of using a return value
- Especially noticeable when the prototyped return value is a large varying length value

```

D center          pr          100000a    varying
D                                     rtnparm
D  text          50000a    const varying
D  len           10i  0    value
D title         s          100a    varying
/free
  title = center ('Chapter 1' : 60);

```



# RPG: Softcode the parameter number

The %PARMNUM built-in function returns a parameter's position in the parameter list.

```
D myProc          pi          10A    OPDESC
D  company                25A          Parameter 1
D  city                   25A          Parameter 2
```

Problem solved by %PARMNUM:

- ▶ Pass a parameter number to a Parameter-Information API

```
CEEDOD (2 : more parms);           // hard to understand
CEEDOD (%PARMNUM(city) : more parms); // better
```

- ▶ Check to see if the number of passed parameters is high enough for a particular parameter

```
if %parms > 1;                       // hard to understand
if %parms >= %PARNUM(company);        // better
```



# RPG: %PARMNUM is imperative with RTNPARM

When a procedure is defined with RTNPARM

- The return value is handled as an extra parameter under the covers
- The extra parameter is the first parameter
- %PARMS and the parameter APIs use the true number
- The apparent parameter number is off by one

```
D  myProc          pi          10A  RTNPARM
... RTNPARM hidden parameter          Parameter 1
D  company         25A          Parameter 2
D  city            25A          Parameter 3
```

- %PARMNUM must be used



# RPG: Support for ALIAS names

## Background

- Fields in externally described files can have a standard name up to 10 characters and an alternate (ALIAS) name up to 128 characters.
- RPG III only allowed 6 characters, so many customers have files with cryptic names like CUSNAM, CUSADR. The files often have alternate names such as CUSTOMER\_NAME and CUSTOMER\_ADDRESS, that can be used in SQL queries.
- RPG programmers would like to use the alternate names in their RPG programs.



# RPG: Support for ALIAS names in data structures

## New ALIAS keyword for RPG

- When ALIAS is specified, RPG will use the alternate name instead of the 10-character standard name.
- Supported on F specs for local files or qualified files. Used for LIKERECD data structures.
- Supported on D specs for any externally-described data structure.

The subfields of the LIKERECD or externally-described data structure will have the alternate names instead of the standard name.



# RPG: Support for ALIAS names

```
A    R CUSTREC
A      CUSTNM      25A      ALIAS (CUSTOMER_NAME)
A      CUSTAD      25A      ALIAS (CUSTOMER_ADDRESS)
A      ID          10P 0
```

```
D custDs          e ds      ALIAS
D                                     QUALIFIED EXTNAME(custFile)
```

```
/free
```

```
custDs.customer_name = 'John Smith';
```

```
custDs.customer_address = '123 Mockingbird Lane';
```

```
custDs.id = 12345;
```



# RPG: New built-in function %SCANRPL

The %SCANRPL built-in function replaces all occurrences a string with another string.

```
fileErr = 'File &1 not found. Please create &1.';  
msg = %scanrpl ('&1' : filename : fileErr);  
  
// msg = 'File MYFILE not found. Please create MYFILE.'
```

Problem solved by %SCANRPL:

Hand-written versions of scan-and-replace tend to be large, error prone, and difficult to maintain.



# Encrypt the debug listing view (all ILE compilers)

## The problem:

- You want to ship a debuggable version of your application to your customers, but you don't want them to be able to read your source code through the debug view

## The solution:

- Encrypt the debug view so that the debug view is only visible if the person knows the encryption key.

- ▶ `CRTBNDRPG MYPGM DBGENCKEY('my secret code')`

- Then either

- ▶ `STRDBG MYPGM DBGENCKEY('my secret code')`

OR

- ▶ `STRDBG MYPGM`

and wait to be prompted for the encryption key



# Teraspace storage model (also for COBOL)

## The problems:

- 16MB automatic storage limits with the single-level storage model, for a single procedure, and for all the procedures on the call stack
- RPG's %ALLOC and %REALLOC have a 16MB limit

## The solution: use the teraspace storage model

- Much higher limits for automatic storage.
- Can compile \*CALLER programs with STGMDDL(\*INHERIT) so they can be called from either single-level or teraspace programs
- RPG's %ALLOC and %REALLOC can allocate teraspace with a much higher limit
- Teraspace allocations are the default in the teraspace storage model
- Specify H-spec ALLOC(\*TERASPACE) to have teraspace allocations in any storage model



# Agenda

- What's new in 6.1
- What's new in 7.1

## ➤ **Rational Open Access: RPG Edition**



# What is Open Access: RPG Edition

Open Access provides a way for RPG programmers to use the simple and well-understood RPG I/O model to access resources and devices that are not directly supported by RPG.

Open Access opens up RPG's file I/O capabilities, allowing anyone to write innovative I/O handlers to access other devices and resources such as:

- o Browsers
- o Mobile devices
- o Cloud computing resources
- o Web services
- o External databases
- o XML files
- o Spreadsheets



## Three parts

An Open Access application has three parts:

1. An RPG program that uses normal RPG coding to define an Open Access file and use I/O operations against the file.



2. A handler procedure or program that is called by Open Access to handle the I/O operations for the file.

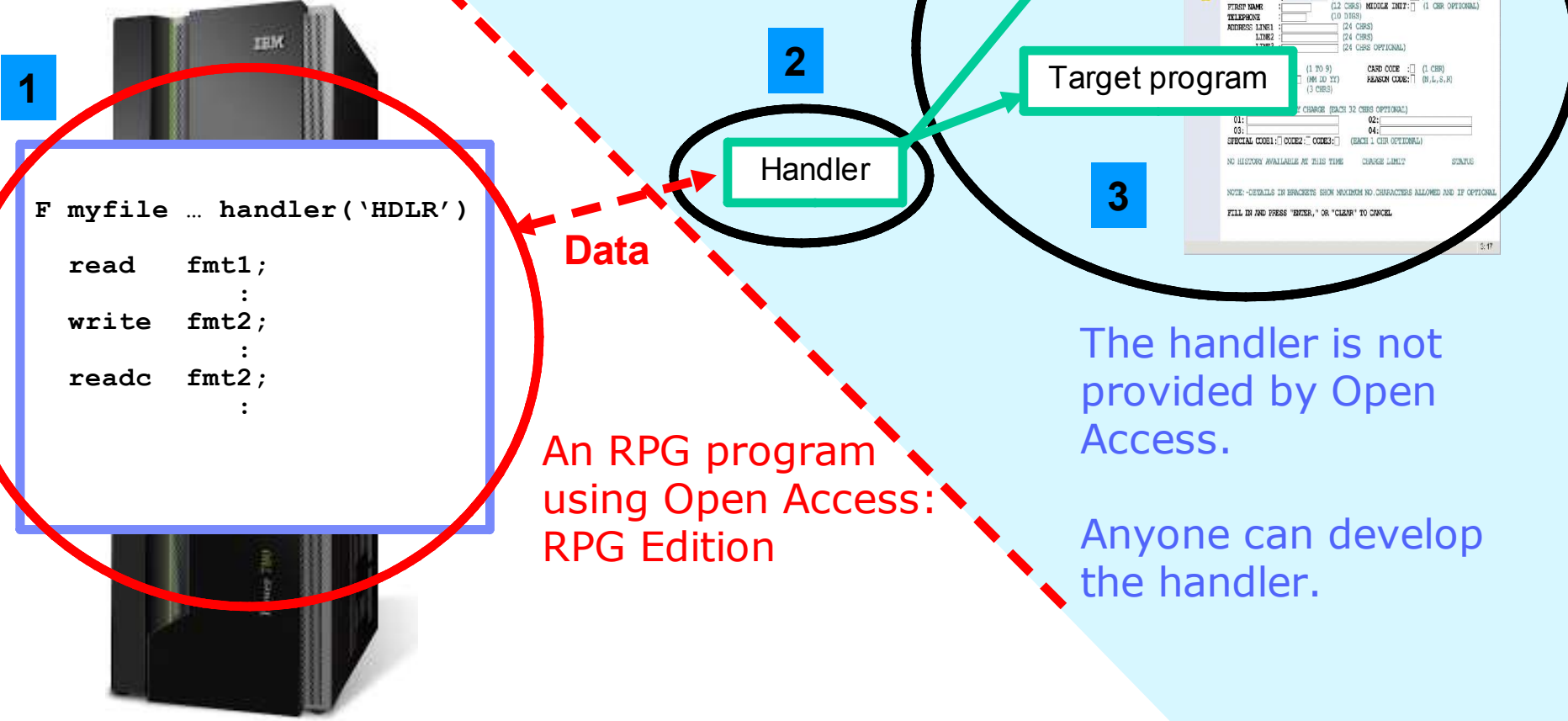


3. The resource or device that the handler is using or communicating with.

Open Access is the linkage between 1 and 2. Licensed program 5733-OAR is required at runtime to use Open Access. It is not required at compile time.

# Open Access

## RPG Applications



```

1
F myfile ... handler('HDLR')
  read  fmt1;
      :
  write fmt2;
      :
  readc fmt2;
      :
    
```

An RPG program using Open Access: RPG Edition

The handler is not provided by Open Access.

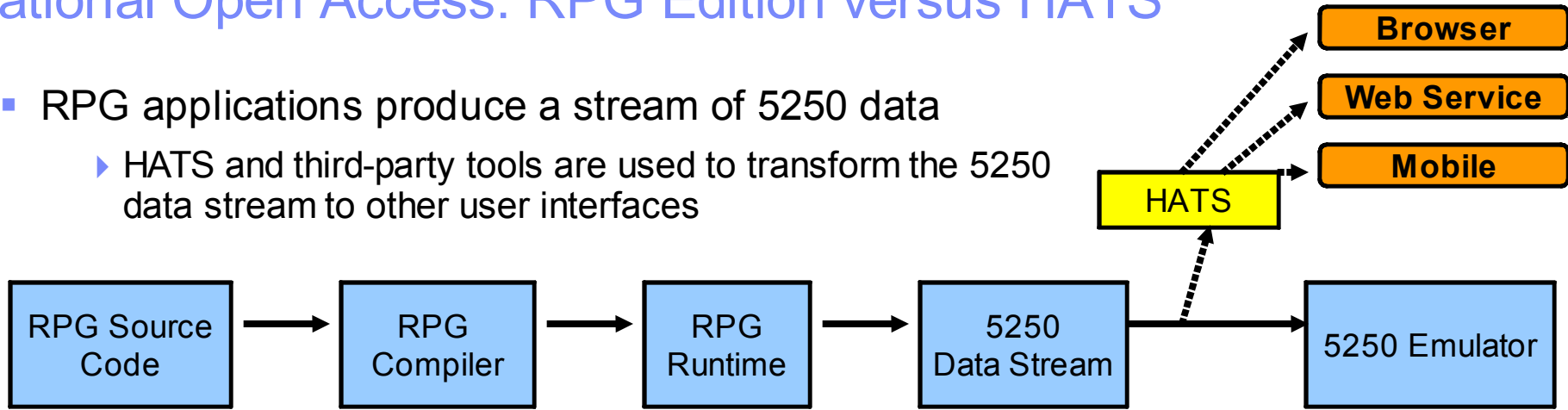
Anyone can develop the handler.

The developer continues to develop in RPG. The handler is called transparently.



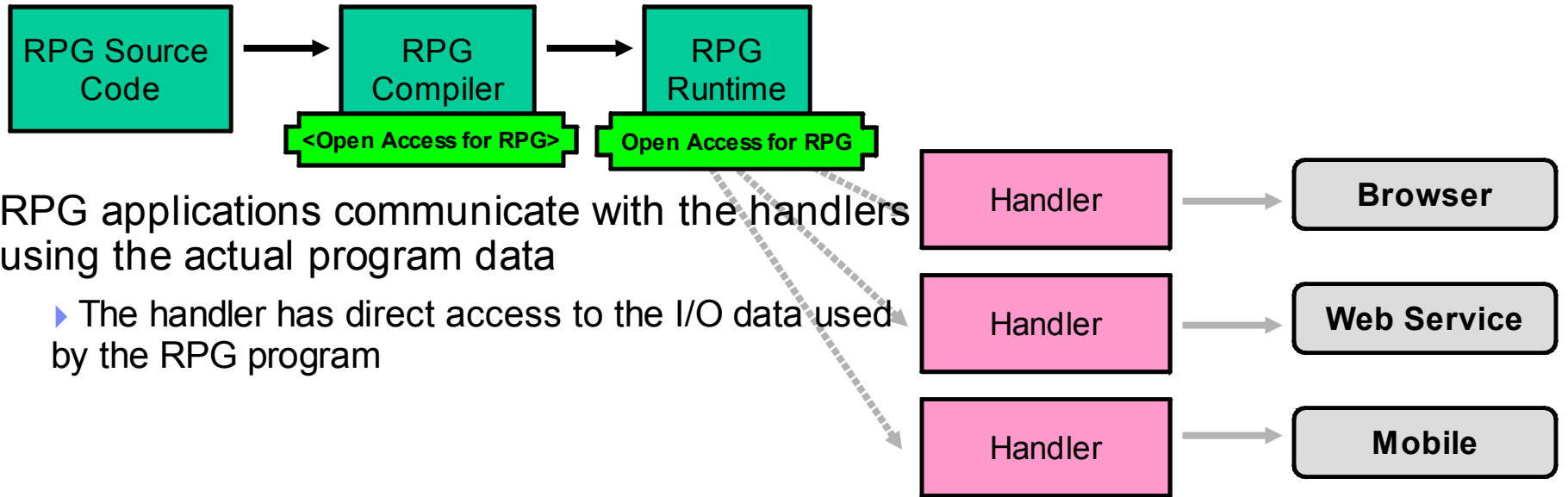
# Rational Open Access: RPG Edition versus HATS

- RPG applications produce a stream of 5250 data
  - HATS and third-party tools are used to transform the 5250 data stream to other user interfaces



*Traditional RPG applications*

*RPG Applications with Rational Open Access: RPG Edition*



- RPG applications communicate with the handlers using the actual program data
  - The handler has direct access to the I/O data used by the RPG program



# Who provides the handlers?

The real magic of Open Access is what the handler does.

Open Access does not provide the handlers.

Anyone can write the handlers that extend RPG IV's I/O capabilities to new resources and devices.

- Software tool vendors
- Business partners
- Services organizations
- You or someone in your shop who can write code to work with the target resource or device



## Any RPG device type

Any RPG device type can be defined as an Open Access file: DISK, PRINTER, or WORKSTN.

The provider of the handling procedure can choose the RPG device-type that best fits the function that the handler provides.

### Examples

- User interface: WORKSTN file
- Creating an Excel document: PRINTER file
- Accessing a Web service: keyed DISK file



## Two ways to use Open Access – 1. handler after

1. The handler is written after the application is written

**Example:** An existing application that uses 5250 display files modified to use Open Access for the WORKSTN files.

- The RPG program is modified by adding the HANDLER keyword to the WORKSTN files
- The handler must handle all the operations and requirements of the existing RPG program.
- This type of handler will normally be provided by an outside expert such as a software tool vendor or business partner.



## Two ways to use Open Access – 2: handler first

**Example:** The RPG programmer wants to use a Web service that returns the current weather for a particular city.

- The “request” is the city, and the data is the weather information.
- Conceptually like a keyed database file:
  - web service request = key
  - web service data = record matching the key
- The handler provider creates a keyed database file matching the web service. Not to contain data, but only for externally-describing.

continued on next slide ...

## Two ways to use Open Access – 2: handler first

... continued from previous slide

- The handler provider can tell the RPG programmer what I/O operations that the handler will support. For example, only OPEN, CHAIN, CLOSE.
- The RPG programmer codes that file as an externally-described keyed DISK file.
- The handler codes externally-described DS to get the correct buffer and key layouts.
- This type of handler may be written by the same RPG programmer who uses the Open Access file, or it may be provided by an outside expert.



# How Open Access works

When a RPG program performs an I/O operation for a “normal” file, a system data management function is called to handle the operation.

When the RPG program performs an I/O operation for an Open Access file, the Open Access handler is called.

The handler receives a data structure parameter with subfields that enable the handler to perform the correct I/O operation, and provide information back to RPG.



# RPG coding for Open Access

## RPG coding to use an Open Access file

Other than the HANDLER keyword, there is no new syntax related to using an Open Access file.

The RPG program can use all the operations that are allowed for the file as defined in the RPG program: EXFMT, WRITE, READE, CHAIN, SETLL etc.

- ▶ The handler may place its own limitations on the operations it will support.

The RPG program can use all the built-in functions that are relevant for the file: %EOF, %FOUND, %OPEN etc.



# The RPG coding to define an Open Access file

The HANDLER keyword identifies the location of the handler.  
The handler can be a program or a procedure.

```
Fmyfile    cf      e      workstn  extdesc('MYLIB/MYFILE')  
F                                     handler('MYLIB/MYSRVPGM(hdlMyfile)')  
F                                     usropt
```

## Other examples of the HANDLER keyword

- `handler('MYLIB/MYPGM')`
- `handler(charVariable)`  
where `charVariable = 'MYLIB/MYPGM' or 'MYSRVPGM(proc)'`
- `handler(rpgPrototype)`
- `handler(procptrVariable)`



# Optional second parameter for HANDLER keyword

The HANDLER keyword has an optional second parameter to pass information from the RPG programmer directly to the handler.

Example: The path to the IFS file

```
FmyIfsFile if e disk extdesc('MYLIB/READIFS') USROPN
F          handler(readIfs : ifsDs)

/copy MYLIB/QRPGLESRC,READIFS

D ifsDs          ds          likes(readIfs_t)

/free

ifsDs.path = '/home/mydir/myIfsFile.txt';
open myIfsFile;
```



# Coding the handler



# Documentation for handler writers

The documentation for Open Access is in the 7.1 version of the Info Center. It is currently available in English only.

## Navigate

IBM i 7.1 Information Center

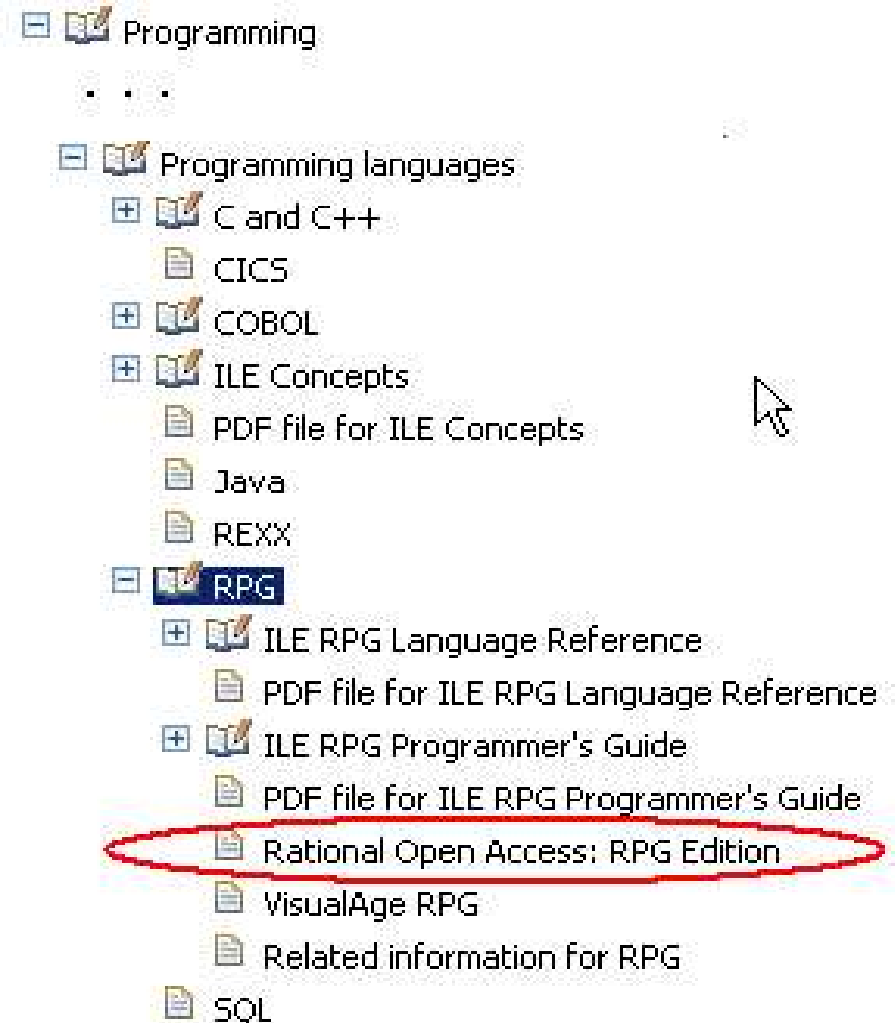
> Programming

> Programming languages

> RPG

> Rational Open Access: RPG Edition

The 7.1 documentation also applies to using Open Access in 6.1.



# Other information about Open Access in the RPG Café

Visit the RPG Café wiki.

<https://www.ibm.com/developerworks/rational/community/cafe/rpg.html>

You'll find

- Information about OA-related PTFs
- The same PDF that is in the Info Center
- PDFs of the copy files
- Some very simple examples that you can download as a savefile



# Parameter passed between RPG and the handler

The handler parameter is a data structure defined by the QrnOpenAccess\_T template in copy file provided as part of the Open Access product 5733-OAR.

```
D QrnOpenAccess_T...  
D          DS          TEMPLATE QUALIFIED ALIGN  
/* I  Length of this structure  */  
D  length...  
D          10U 0
```

# Parameter passed between RPG and the handler

The data structure has several kinds of subfields

1. Subfields set by RPG to be used by the handler
  - recordName
  - externalFile
  - compileFile
  - inputWithLock
  - rrn
  - etc
2. Subfields set by the handler and later used by RPG to provide feedback to the RPG program
  - rpgStatus
  - eof
  - found
  - printerOverflow
  - functionKey
  - rrn
  - feedback areas to set the INFDS
  - etc



# Internal interface between RPG and the handler

## 3. Subfields that are private to the handler and the RPG program

- state (private to the handler)
- userArea (private between the handler and the RPG program, unknown to Open Access, for example the "ifsDs" data structure in the earlier IFS example)

## 4. Subfields with I/O data (more info on following pages)

- input data that is provided by the handler for an input capable operation
- output data that is provided to the handler for an output capable operation
- key data that is provided to the handler for a keyed operation

# I/O data

The I/O data is communicated in two ways

1. Data structures matching the I/O buffers used by the external file
2. An array of name-value information about each field (externally-described files only)

During the OPEN operation, the handler can choose which way the I/O data will be communicated by setting the useNamesValues subfield to '0' (no) or '1' (yes).

```
/* I/O Whether the I/O operations */
/* will use the name-value */
/* info rather than the */
/* I/O buffers */
/* Set by handler during OPEN */
/* operation, used by handler */
/* and RPG for I/O operations */
D useNamesValues...
D 1N
```

# 1. I/O data – data structures

Data structures matching the I/O buffers used by the external file

```

/* O  Input buffer          */
/*  NULL if not used by opcode */
/*  - Length is given by      */
/*  inputBufferLen           */
D  inputBuffer...
D
*

/* I  Output buffer        */
/*  NULL if not used by opcode */
/*  - Length is given by    */
/*  outputBufferLen         */
D  outputBuffer...
D
*

```

- Subfield names and types are not directly available to the handler
- The handler must have some way to determine how to interpret and set the buffers

Either

- ▶ The handler is specifically written to handle that file

or

- ▶ The handler dynamically uses APIs to retrieve the structure layout and field names



## 2. I/O data – name-value information

An array of information about each field (externally-described files only)

```

D QrnNameValue_T...
D          DS          QUALIFIED TEMPLATE
/* I  Name from external file  */
D  externalName...
D          10A
/* I  Data type of field      */
/*      See QrnDatatype_*    */
D  dataType...
D          3U  0

```

- External name (the short 10-character form of the name)
- Data type
  - ▶ alpha, Unicode, DBCS, alpha varying, Unicode varying, DBCS varying
  - ▶ decimal, integer, float
  - ▶ date, time, timestamp
- Data CCSID, maximum length in bytes, decimals, date format etc
- Current length in bytes
- Pointer to the data
- The data is in human-readable form
  - ▶ For UCS-2 and DBCS, it is in the same CCSID as the field
  - ▶ For other types, it is the job CCSID, in the %CHAR form



## How the handler works – state information

A handler will usually need to maintain some state information about the file.

The “state” subfield of the handler parameter is a pointer. The handler sets this pointer during the OPEN operation, and the pointer is available for subsequent operations.

The particular state information that the handler needs depends on the nature of the resource or device being accessed by the handler, and it also depends on the individual nature of the handler.



## An example of simple state information

The handler is for a sequential DISK file to read lines from an IFS file.

The state information maintained by the handler might only be the "pointer" to the IFS file.

During the OPEN operation, the handler would call the open() API to set the pointer.

During the READ operation, the handler would use the pointer to call the read() API.

During the CLOSE operation, the handler would use the pointer to call the close() API.



# An example of simple state information in the handler

```
D state_t ds qualified template
D handle 10i 0
```

```
P myHandler b export
```

```
D pi
```

```
D parm likeds(QrnOpenAccess_t)
```

```
D state ds likeds(state_t) based(parm.state)
```

```
D ifsInfo ds likeds(ifsInfo_t) based(parm.userArea)
```

```
if parm.operation = QrnOperation_OPEN;
    parm.state = %ALLOC(%size(state_t)); // allocate the state information
    state.handle = open(ifsInfo.path); // set the state information
elseif parm.operation = QrnOperation_CLOSE;
    rc = close(state.handle); // use the state information
    dealloc(n) parm.state; // free the state information
endif;
```



## More complex state information

The handler is for a WORKSTN file with a subfile record. Some of the state subfields would be

- Array of subfile records
- Array of changed-record indicators
- Number of subfile records
- Next RRN for READC
- Current RRN of the record read by READC or CHAIN, that will be updated by an UPDATE operation

# Using more complex state information

## WRITE subfile control record

- If the SFLCLR indicator is on, set “number of subfile records” to zero

# Using more complex state information

## EXFMT subfile control record

- If the SFLDSP indicator is on
  - ▶ interact with the end user by showing all the subfile records and obtaining changes to the records
  - ▶ keep track of which records were changed in the “changed records” indicator array
  - ▶ set the “next record” to 1, for the first READC operation
  
- Set the input data for the control record itself



# Using more complex state information

## WRITE subfile record

- If rrn is too high, set handlerParm.eof to '1' and return
- Store output fields in an array of subfile records, indexed by the rrn provided to the handler
- Keep track of the number of subfile records in the array



## Using more complex state information

### READC (read next changed subfile record)

- Find next changed record starting at “next record”
- If no changed record. set handlerParm.eof to ‘1’ and return.
- Set “current record” to the changed-record rrn, in case of a later UPDATE operation
- Place output and input fields for current record in the I/O data
- Set “next record” to “current record” + 1

# State information in action

```
*in70 = *on; // sflclr  
write sflctl;  
*in70 = *off;
```

```
name = 'Jim';  
rrn = 1;  
write sfl;
```

```
Repeat for 2 'Mary'
```

```
Repeat for 3 'Sam'
```

```
*in71 = *on; // sfldsp  
exfmt sflctl;
```

```
readc sfl;
```

number of records

current rrn

next rrn

subfile data

changed



# Summary

- Using Open Access is both extremely simple (HANDLER keyword) and arbitrarily complex (the handler).
- The intention for Open Access is for RPG programmers to be able to use their existing expertise in using the RPG file I/O model, while using others' expertise in accessing new resources and devices.
- In many cases, the handler will be provided by an outside provider.

## PTFs necessary to use Open Access in 6.1 and 7.1

- PTFs are needed for the ILE RPG compiler if you want to compile a program that uses the HANDLER keyword
- PTFs are needed at runtime if you want to run a program that uses the HANDLER keyword
- PTFs may also be needed by handler developers who want the latest version of the copy files in library QOAR

For information on the latest PTFs for Open Access, see the “Rational Open Access: RPG Edition” page in the wiki section of the RPG Café:

<http://preview.tinyurl.com/rpgoa-info-on-rpg-cafe>



# Thank YOU

[→ Go to IBM](#)

© Copyright IBM Corporation 2010. All rights reserved.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

IBM, the IBM logo, the on-demand business logo, Rational, the Rational logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.

