IBM Software Group

What's new in RPG for V6.1

Barbara Morris
IBM

ILE RPG enhancements - overview

Major enhancements for ILE RPG include

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

ILE RPG – TEMPLATE definitions

RPG programmers often code variable definitions that are intended as “type definitions”, to be used by later LIKE or LIKEDS definitions. To prevent the compiler from defining storage for these definitions, they code the BASED keyword, hoping that no maintenance programmer will try to use the definitions as program variables.

The RPG compiler supports INZ(*LIKEDS) to initialize a LIKEDS data with the same initialization value as the parent data structure, but initialization values are not supported with the BASED keyword.

TEMPLATE solves both these problems:

- The compiler doesn’t allow the definition to be used as program variable.
- The INZ keyword is supported with TEMPLATE, so default initializations can be coded for a template data structure

TEMPLATE example – defining the template

```
D custId_t         S              10A   TEMPLATE
D CUST_ID_NOT_SET...
D                  C                    -1
D custInfo_t       DS                   TEMPLATE
D                                       QUALIFIED
D   name                          35A   VARYING
D   city                          25A   VARYING
D                                       INZ('Toronto')
D   province                      25A   VARYING
D                                       INZ('Ontario')
D   postal                         6A   INZ(*BLANKS)
D   id                                  LIKE(custId_t)
D                                       INZ(CUST_ID_NOT_SET)
```

TEMPLATE example – using the template

```
D cust1          DS                  LIKEDS(custInfo_t)INZ
D cust2          DS                  LIKEDS(custInfo_t)
D                                    INZ(*LIKEDS)
```

Both CUST1 and CUST2 are defined like the template data structure, so they have subfields NAME, CITY etc.

CUST1 has default initializations, so CUST1.CITY is initialized with an empty name.

CUST2 is initialized with *LIKEDS, so CUST2.CITY is initialized with 'Toronto'.

ILE RPG – avoid compile-time overrides

A common complaint by RPG programmers is that they have to do file overrides at compile time, for externally-described files and data structures.

RPG added the EXTFILE keyword in V5R1, which removed most of the necessity to do overrides at runtime, but the EXTFILE keyword was not used at compile time.

A new F spec keyword EXTDESC handles this problem for files, and an enhancement to the EXTNAME keyword handles this problem for data structures.

ILE RPG – avoid compile-time overrides

For F specs, new keyword EXTDESC and enhanced keyword EXTFILE(*EXTDESC)

- EXTDESC('LIBNAME/FILENAME') or EXTDESC('FILENAME') locates the file at compile time
- EXTFILE(*EXTDESC) indicates that the file specified by EXTDESC is also to be used at runtime.

For D specs, the EXTNAME keyword is enhanced

- EXTNAME('LIBNAME/FILENAME') or EXTNAME('FILENAME') locates the file for the data structure at compile time.

ILE RPG – local files

V6.1 introduces the ability for RPG programmers to define files locally in subprocedures. This has two main advantages:

1. **Maintainability** : Prior to V6.1, all files had to be defined globally. There was no way for an RPG programmer to limit the file's access to only one subprocedure, which could cause some maintenance difficulties. By coding the file definition in a subprocedure, the RPG programmer can explicitly limit the file's access to only that subprocedure.
2. **Reduction of static storage**: Defining the file locally reduces of the static storage required by the module, if the file is defined to use automatic storage for the internal storage required to manage the file.

ILE RPG – local files – some rules

- Local F specifications follow the Procedure-begin specification and precede the Definition specifications.
- I/O to local files can only be done with data structures. There are no I and O specifications for local files.
- By default, the storage associated with local files is automatic; the file is closed when the subprocedure returns normally or abnormally.
- The STATIC keyword can be used to indicate that the storage associated with the file is static, so that all invocations of the procedure will use the same file. If a static file is open when the procedure returns, it will remain open for the next call to the procedure.

ILE RPG – local files – example 1

```
P fixPostalCode    B
Fcust      UF      E            DISK
D custDs          E DS                    EXTNAME(cust:*INPUT)
 /free
      read custrec custDs;
      dow not %eof;
         if postCode = ' ';
            postCode = getPostalCode(addr:city:prv);
            update custRec custDs;
         endif;
         read custrec custDs;
      enddo;
```

The file is not defined with STATIC, so it will be closed when the procedure ends.

ILE RPG – local files – example 2

```
P getCust          B
Fcust      IF      E        K     DISK    STATIC
D getCust          PI                N
D   id                                    LIKE(custrec.id)
D                                         CONST
D   custDs                                LIKEREC(custrec)
 /free
      chain id custrec custDs;
      return %FOUND;
```

The file is defined with STATIC, so it will stay open when the procedure ends.

ILE RPG – qualified record formats

Another file-related enhancement is Qualified record formats. Using qualified record formats makes it easier to read and maintain RPG code.

Consider this code which does not use any qualified names:

```
read custRec custDs;
if not %eof(inFile);
   if amtOwing > 1000;
```

Compare to this version which uses qualified names for the file and the data structure:

```
read inFile.custRec custDs;
if not %eof(inFile);
   if custDs.amtOwing > 1000;
```

ILE RPG – qualified record formats

Some of the rules for using qualified 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.

A review of how RPG I and O specs work

When the compiler does generate I specs, you get program fields defined from the fields in the file.

```
1 Fcustfile  if   e   disk
2=ICUSTREC
3=I                                P    1    6 0ID
4=I                                A    7   56  NAME
5=I                                A   57  106  CITY
6 C                   if           name = *blanks


Global Field References:
        Field      Attributes      References
        NAME       A(50)           4D      6
```

What if there are no I and O specs?

When the compiler does not generate I specs, you don't get any program fields defined from the fields in the file.

```
5 P myProc      b
6 Fcustfile  if   e   disk
7 C                   if        name = *blanks


Field References for subprocedure MYPROC:
        Field      Attributes      References
RNF7030 NAME       **UNDEF**       7
```


Instead of I and O specs

Instead, you define a data structure to hold the fields from the file.

```
1 Fcustfile  if   e   disk   qualified
2 D custDs      ds            likerec(custfile.custrec)
3  /free
4       read custfile.custrec custDs;
5       if custDs.name = *blanks;


Global Field References:
        Field      Attributes      References
        CUSTDS     DS(100)         2D       4M       5
          ID       P(11,0)         2D
          NAME     A(50)           2D       5
          CITY     A(50)           2D
```

ILE RPG – LIKEFILE – files defined like other files

- Using the LIKEFILE keyword, a file can be defined to use the same settings as another File specification.
- QUALIFIED keyword is implied for externally-described. I/O to the file can only be done through data structures.
- The LIKEFILE keyword is mainly used with file parameters.

```
Fcustfile  if   e   disk    qualified
Fcust2                      likefile(custfile)
F                           extfile(cust2name)
```

The like-file CUST2 doesn't specify any of the entries like "Input" or "DISK". It inherits those from the parent file CUSTFILE.

ILE RPG – file parameters

Another major file-related enhancement for RPG is the ability to pass files as parameters between procedures and programs.

Passing files as parameters allows you to control which programs and procedures can "share" a file.

ILE RPG – file parameters

Scenario 1: A service program handles all the I/O for a file.

- Two different programs call the procedures in the service program.
- PGM1 calls procedures to open the file and read record 1.
- It calls PGM2, which calls a procedure to chain to record 15.
- When PGM1 calls the read procedure again, it doesn't get record 2, it gets record 16.
- **The programs are inadvertently sharing the file.**

ILE RPG – file parameters

By using file parameters, the application can avoid these problems.

The service program would not have its own file defined. Instead, it would use the file parameter passed by PGM1 or PGM2. PGM1 and PGM2 would be independent of each other.

ILE RPG – file parameters

Scenario 2: A file is shared using OVRPRTF so PGM1 and PGM2 can use it together

- PGM1 opens the printer file, writes some headers, then calls PGM2
- PGM2 opens the printer file, and writes the detail records
- PGM2 calls PGM3 to get some information about a particular record
- PGM3 opens the printer file, and writes some logging records
- **The shared override is inadvertently used by more programs than was intended.**

ILE RPG – file parameters

By using file parameters, the application can avoid these problems.

PGM1 would open the file and pass it as a parameter to PGM2. No override would be necessary for PGM1 and PGM2 to "share" the file. When PGM3 opens the file, there would be no shared override, so its use of the file would be independent from PGM1 and PGM2.

ILE RPG – file parameters

Scenario 3: There are several procedures in a module, and there is a file that only two of the procedures should be able to use.

The file could be defined in global F specs, but then any procedure in the module could use it.

By defining the file in one procedure and passing it as a parameter to the other procedure, the programmer can be sure that no other procedure can access the file.

ILE RPG – file parameters

- A prototyped parameter can be defined as a File parameter using the LIKEFILE keyword.
- Any file related by LIKEFILE keywords to the same original File specification may be passed as a parameter to the procedure.
- Within the called procedure or program, all supported operations can be done on the file parameter. However, I/O to the file parameter can only be done through data structures.
- RPG file parameters are not compatible with file parameters in other languages such as C or COBOL.

ILE RPG – file parameter example

Here is an ordinary file and an ordinary data structure.

```
FcustF     IF E   K   DISK
D custDs         DS            LIKEREC(custrec)
```

Here is a prototype for a procedure with a file parameter.

```
D getCust        PR          N
D   custfile                     LIKEFILE(custF)
D   id                           LIKE(custDs.id)
D   info                         LIKEDS(custDs)
```

Here is a call to the procedure, passing the file as a parameter.

```
   ok = getCust (custF : 12345 : custDs);
```


ILE RPG – file parameter example part 2

Here is the procedure that uses the file parameter.

```
P getCust        B
D getCust        PI          N
D   custfile                     LIKEFILE(custF)
D   id                           LIKE(custDs.id)
D   info                         LIKEDS(custDs)
 /free
        chain id custFile.rec info;
        return %found(custFile);
```

The procedure is working directly on the file from the calling procedure. If the file in the calling procedure had been at end-of-file, it will now be positioned at the record for ID.

ILE RPG – file templates

A file can be a template too. Just add the TEMPLATE keyword, and the file will only be used at compile time for any LIKEFILE definitions that you need.

Use EXTDESC if the RPG file name is different from the actual name.

```
FcustF_t    IF E    K    DISK    TEMPLATE
F                                EXTDESC('CUSTF')
```

To define a "real" file that can be used at runtime, and that can also be passed as a parameter, define the file using LIKEFILE.

```
FcustF                           LIKEFILE(custF_t)
F                                EXTFILE(*EXTDESC)
```


ILE RPG – /COPY file with F and D specs

This is the /copy file for the getCust procedure. You should use conditional compile to make sure your other RPG modules can pick up either the F specs or the D specs.

```
 /if defined(getFSpecs)
FcustF_t    IF E    K    DISK    TEMPLATE
 /endif

 /if defined(getDSpecs)
D getCust         PR         N
D   custfile                    LIKEFILE(custF_t)
D   id                          LIKE(custDs.id)
D   info                        LIKEDS(custDs)
 /endif
```

ILE RPG – using the /COPY file

The calling module has to do the /COPY twice, so it can define its own file in the F specs.

```
 /define getFSpecs
 /copy getCust
 /undefine getFSpecs
Fcustf                  likefile(custF_t)
F                       extfile(*extdesc)

 /define getDSpecs
 /copy getCust
 /undefine getDSpecs
```


ILE RPG – using the /COPY file

The called module can do one /COPY to pick up both at once, since it doesn't need any of its own F specs.

```
 /define getFSpecs
 /define getDSpecs
 /copy getCust

P getCust          B
 ...
```

ILE RPG – what files can be parameters?

```
Fprtf1     O  F  80   PRINTER
Fprtf2     O  F  80   PRINTER
Fprtf3                        LIKEFILE(prtf1)
Fprtf4                        LIKEFILE(prtf3)

D prtLine       PR
D   fileparm                  LIKEFILE(prtf1)
```

The four files PRTF1, PRTF3, PRTF4, FILEPARM are in the same "LIKEFILE family", because they are all related to the "parent file" PRTF1.

Any file in the same LIKEFILE family as the prototyped file parameter can be passed as a parameter to the procedure, so PRTF1, PRTF3 or PRTF4 could be passed to the procedure.

- What about PRTF2? It's identical to PRTF1 … but no.

Using RPG data structure I/O – a review

"I/O can only be done through data structures"

How many times has that appeared in this presentation so far?

The rule applies to qualified files, local files and file parameters.

Using RPG data structure I/O – continued

The steps to using a data structure for I/O are …

1. Define the data structure using LIKEREC or EXTNAME

```
FcustFile  IF   E     DISK
D custDs1      DS           LIKEREC(custRec:*INPUT)
D custDs2    E DS           EXTNAME(custfile:*INPUT)
```

2. Code the data structure as the result field of the I/O operation

```
read custFile custDs1;
```

3. Use the fields of the data structure instead of the standalone fields that you would use if there were I specs

```
if custDs1.active = 'Y';
```


ILE RPG – result DS for EXFMT

EXFMT allows a result data structure. Prior to V6.1, EXFMT was the only I/O operation that did not allow a result data structure, due to the fact that the I/O buffers are different for the input and output parts of EXFMT.

The data structure used with EXFMT is defined with usage type *ALL

```
EXTNAME(file : fmt : *ALL)
   or
LIKEREC(fmt : *ALL)
```

ILE RPG – result DS for EXFMT

```
FaskQuest  CF   E      WORKSTN QUALIFIED
D QA       DS          LIKEREC(askQuest.ask : *ALL)
  /free
       // Set the output-capable subfields
       QA.question = 'Do you want to continue?';
       QA.answer = 'Y';  // set the default

       exfmt askQuest.ask QA;

       // Check the input-capable subfields
       if QA.answer <> 'Y';
          return;
       endif;
```


ILE RPG – larger fields

- RPG programmers have been struggling with the size limits for string variables and data structures for several releases.
- Historically, RPG programmers had little need for very large variables, since all their data came from DB2/400 files which have a maximum record length of 32K.
- More recently, RPG programmers have been working with data from other systems where the data can be much larger. It is possible to deal with large amounts of data using RPG, but the techniques are poorly understood, and the implementations are error-prone.

ILE RPG – larger fields

No more artificial limits on the way variables are defined.

The system's limit of 16,773,104 for a single variable still applies, but data structures, and A, C and G variables can now have a size up to 16,773,104 bytes.

```
D bigDS           DS          500000
D  ...

D bigField        S         1000000A  VARYING
```


ILE RPG – larger fields

The LEN keyword can be used instead of the Length entry. It is necessary if the length has more than 7 digits.

```
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords
D bigField        S         9999999A

D biggerField     S                 A   LEN(10000000)
D biggestField    S                 A   LEN(16773104)
```

ILE RPG – larger fields

The VARYING keyword allows a parameter of either 2 or 4 indicating the number of bytes used to hold the length prefix. VARYING(4) is assumed if the defined length of the definition is over 65535.

```
D smallVarying    S             10A  VARYING
D bigVarying      S          65536A  VARYING
D smallVarying4   S             10A  VARYING(4)
```

- SMALLVARYING has 2 bytes for the length prefix.
- BIGVARYING has 4 bytes for the length prefix.
- SMALLVARYING4 has 4 bytes for the length prefix.

ILE RPG – larger fields

%ADDR(varying_field :*DATA) can be used to get a pointer to the data portion of a varying length field. Prior to V6.1, the data was always 2 bytes after the start of the field; now it can be 2 or 4 bytes.

```
D firstName    S             10A  VARYING
D lastName     S             10A  VARYING(4)
D pNameData    S               *
 /free
       pNameData = %addr(firstName : *DATA);
       pNameData = %addr(lastName : *DATA);
```

Equivalent to this more error-prone code

```
       pNameData = %addr(firstName) + 2;
       pNameData = %addr(lastName) + 4;
```

ILE RPG – more array elements

Larger limit for DIM and OCCURS

- There is no arbitrary limit on the number of elements in an array or occurrences in a multiple-occurrence data structure.
- 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).

ILE RPG – longer literals

The RPG compiler also increased the size limits for string literals. This is especially important for companies that generate RPG source with literals for prepared SQL statements, or for HTML data.

- Character literals can now have a length up to 16380 characters.
- UCS-2 literals can now have a length up to 8190 UCS-2 characters.
- Graphic literals can now have a length up to 16379 DBCS characters.

ILE RPG - main without cycle

Most RPG modules being written today do not take advantage of the RPG cycle; many RPG programmers are unaware of the presence of the RPG cycle in their modules. V6.1 introduces an RPG module with a main procedure that does not rely on the RPG cycle.

- MAIN keyword on the H specification designates one subprocedure as being the main procedure, that is, the procedure that gets control when the program gets called.
- Other than being the program-entry procedure, the main subprocedure is like any other subprocedure. It does not use the RPG cycle.
- The prototype for the main subprocedure must have the EXTPGM keyword; the main subprocedure can only be called by a program call.

ILE RPG - main without cycle, example

```
H MAIN(ordEntry)
D ordEntry        PR            EXTPGM('ORDENTRY')
D   custname                10A  CONST

P ordEntry        B
D ordEntry PI
D   custname                10A  CONST
 ... code the main procedure logic here
 ... when it reaches the end, it just returns
 ... *INLR has no meaning
P ordEntry        E
```

ILE RPG – relaxed UCS-2 rules

Programmers are being asked to enable their applications to use data from different character sets. This requirement comes from web-enabling applications, and from companies operating in more than one country.

One of the steps in supporting multiple character sets is to use Unicode fields in the database, instead of character or DBCS fields that only support a single CCSID.

ILE RPG supports the UCS-2 data type, which includes support for UTF-16, but the nature of the support means that it is extremely difficult to change the data type of a database character or graphic field to be UCS-2.

RPG considers Character, UCS-2 and Graphic to be three separate data types. To use these data types together in the same statement required specific conversion using RPG built-in functions %CHAR, %UCS2 or %GRAPH.

ILE RPG – relaxed UCS-2 rules

When a database field is changed from character to UCS-2, the RPG programs using the field may be using the field with other character data. The RPG compiler gives diagnostic error messages saying that the UCS-2 field cannot be used with the character data.

To make it easier to change the datatype of database fields to be UCS-2, the compiler has changed to allow any of the string types to be used in assignment and comparison operations without explicit conversion. The compiler performs any needed conversions implicitly.

UCS-2 variables can now be initialized with character or graphic literals without using the %UCS2 built-in function.

This enhancement is available in V5R3 and V5R4 with PTFs:

- `V5R3M0 TGTRLS(*CURRENT): SI24532`
- `V5R4M0 TGTRLS(*CURRENT): SI26312`
- `V5R4M0 TGTRLS(*PRV) : SI25232`

ILE RPG – reduce module size

To reduce the amount of static storage required by an RPG module, the ILE RPG compiler now has an option to eliminate unused variables from the compiled object:

- New values *UNREF and *NOUNREF are added to the OPTION keyword for the CRTBNDRPG and CRTRPGMOD commands, and for the OPTION keyword on the Control specification.
- The default remains *UNREF, meaning that unused variables are still generated into the module.
- *NOUNREF indicates that unreferenced variables should not be generated into the RPG module. This can reduce program size, and if imported variables are not referenced, it can reduce the time taken to bind a module to a program or service program.

ILE RPG - threads

We are all familiar with two jobs running simultaneously. The jobs could be running different programs or the same program.

It is possible for two jobs to access the same resources and interfere with each other. For example, two jobs could each get a pointer to the same user-space and if they both tried to change the user-space data at the same time, the results would not be correct.

But this type of interference is rare, and it's easy to prevent it.

ILE RPG - threads

When an application is running multi-threaded, the job has more than one “thread of execution”. Each thread has its own program stack. Two or more things can be happening in the job at exactly the same time.

The “thread safety” problem: If two parts of the application try to use the same resource at the same time, the result might not be correct.

With multiple threads, this interference is much more common, and much more difficult to predict or prevent, than with multiple jobs.

ILE RPG - threads

Example: The shared resource is program variable “i”, an integer.

A job is running with two threads, and both happen to be running in the same procedure.

These two statements happen at exactly the same time. What will be the value of “i”? It’s impossible to guess.

```
ThreadA: i = 3          ThreadB: i = 5
```

This application is not thread safe.

To be thread safe, this application cannot allow two threads to use this variable at the same time.

ILE RPG – threads prior to V6.1

Prior to V6.1, RPG could run safely in multiple threads, using the THREAD(*SERIALIZE) support, but each RPG module could be accessed by only one thread at a time.

- prevents multiple threads accessing the variables in the module
- impacts the overall performance and scalability of the application
- each RPG module has the potential of being a bottleneck

ILE RPG – threads in V6.1

In V6.1, RPG programmers have the additional option of having an RPG module run concurrently in multiple threads.

ILE RPG - 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.
- Individual variables can be defined to be shared by all threads using STATIC(*ALLTHREAD).
- Individual procedures can be serialized so that only one thread can run them at one time, by specifying SERIALIZE on the Procedure-Begin specification.

ILE RPG - threads

Using THREAD(*CONCURRENT) increases the total amount of static storage used by the application.

Since each thread has its own copy of the static storage, the total static storage used by the application is the size required by the module times the number of threads using the module.

Some of the other enhancements in V6.1 are aimed at reducing the amount of static storage required by an RPG module.

- OPTION(*NOUNREF)
- Local files and file parameters

ILE RPG - threads

If the application will be multi-threaded, the THREAD keyword must be coded in every ILE RPG module.

THREAD(*SERIALIZE) and THREAD(*CONCURRENT) modules can be mixed in an application.

The “Multithreaded Applications” section in the ILE RPG Programmer’s Guide has some guidance about how to choose.

http://publib.boulder.ibm.com/infocenter/systems/scope/i5os/topic/rzasc/sc09250758.htm#mthreadap

Threads

There is much more to thread-safety than protecting the static storage in the modules.

Search for “multithreaded” in the Info Center.

http://publib.boulder.ibm.com/infocenter/systems/scope/i5os/advanced/browse.jsp?searchQuery=multithreaded

Store parameter information in the program

In V5R2, the ILE RPG and ILE COBOL compilers were enhanced to produce a file containing PCML (Program Call Markup Language). This file was intended to be used by WDSC tooling to help programmers use the Program Call wizard.

It was not really convenient to have the PCML separate from the program, but it was much more convenient and less error-prone for programmers to use the generated PCML than to manually enter the parameter information using the wizard.

Integrated Web Services also uses PCML, but it requires it to be part of the program or service program.

http://www-03.ibm.com/systems/i/software/iws/index.html

Store parameter information in the program

Both the ILE RPG and ILE COBOL compilers are enhanced to allow information about the parameters for the program or procedures to be stored in the program.

The PCML can be placed in a stream file as before, or directly in the module, or both.

The information can later be retrieved from a program or service program containing the module, using the new QBNRPII API.

Parameter info in program - commands

The PGMINFO command parameter for the CRTRPGMOD, CRTCBLMOD, CRTBNDRPG and CRTBNDCBL commands is enhanced to specify where the PCML is to go.

- The default location is the stream file specified by the INFOSTMF parameter
- PGMINFO(*PCML:*MODULE) says to place the PCML information directly in the module. The PCML information becomes part of the program or service program containing the module.
- PGMINFO(*PCML:*ALL) says to place the PCML information both in the module and in the INFOSTMF stream file.

Parameter info in program – source files

H spec keyword for RPG or PROCESS option for COBOL

- The PGMINFO command parameter can be augmented or overridden by an H spec keyword (ILE RPG) or PROCESS option (ILE COBOL).
  - RPG: `PGMINFO(*PCML:*MODULE) or PGMINFO(*NO)`
  - COBOL: `PGMINFO(PCML MODULE) or PGMINFO(NOPGMINFO)`
- If the source keyword specifies “module”, then it augments the PGMINFO command parameter. For example, if the command requested *STMF, and the source keyword specifies *MODULE, then the PCML will be generated both to the stream file and into the module.
- If the keyword specifies “no”, then it overrides the PGMINFO command parameter. No matter what was specified by the command parameter, no PCML information will be generated by the compiler.

Parameter info in program – V5R4 PTFs

PTF support for V5R4

- **Part of this support is available in V5R4 with PTFs.**
  - The H specification keyword for ILE RPG
  - The PROCESS option for ILE COBOL
  - The QBNRPII API
- **The following V5R4M0 PTFs will provide the various parts of this function**
  - `5722SS1 SI23544 (QBNRPII API)`
  - `5722SS1 SI27064 (Support for compilers)`
  - `5722WDS SI27061 (ILE RPG compiler PTF 1)`
  - `5722WDS SI27065 (ILE RPG compiler PTF 2)`
  - `5722WDS SI27154 (ILE COBOL compiler)`

PTF: New options for XML-INTO

Two new options for XML-INTO available via a PTF for V6.1

- datasubf

  Handles XML elements in this form

  ```
  <employee type="manager">Jack Spratt</employee>
  ```

- countprefix

  Reduces the need for the allowmissing=yes option by allowing you to add extra "count" subfields.

See this RPG Café announcement:

http://www-949.ibm.com/software/rational/cafe/docs/DOC-2975

SQL Precompiler

- Supports all the new features in ILE RPG V6R1
- Except that it does not support VARYING(4), or any fields larger than 32767. LOBs must still be used to handle large data in SQL
- Host variables are scoped to the procedure instead of having to be unique for the whole module. Also available via PTFs for V5R4. See this RPG Café announcement:

  http://www-949.ibm.com/software/rational/cafe/docs/DOC-2947

Alignment of floating point data on PowerPC

On PowerPc architecture, binary floating point operations are much faster if the floating point variables are aligned.

- RPG aligns floating point standalone fields
- By default, RPG does not align floating point subfields in data structures.
- Code the ALIGN keyword on a data structure to have floating point subfields aligned

For more information:

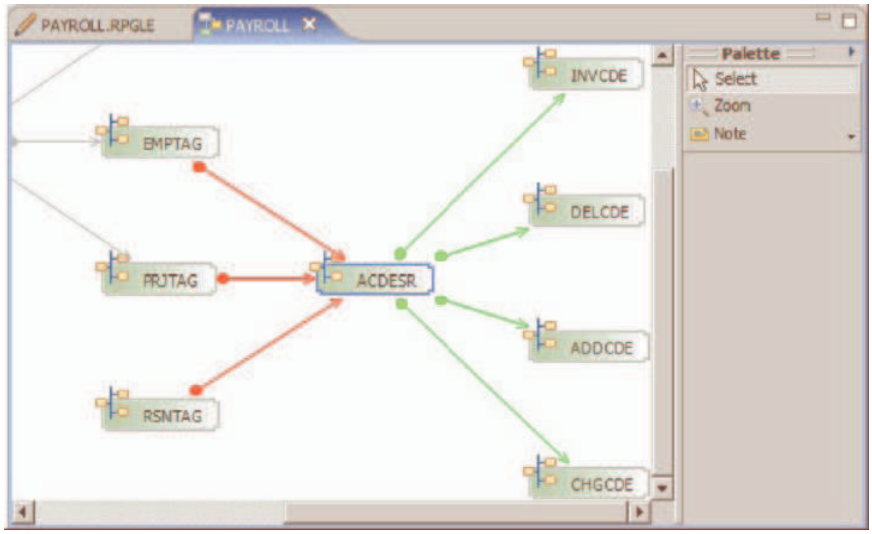
http://www-03.ibm.com/systems/resources/pcrm_oct2008.pdf

Tooling for RPG programmers - RDi

RDi (Rational Developer for i)

- Remote Systems Explorer (RSE)
- Integrated debugger
- Application diagram
- Screen designer
- More …

Application diagram

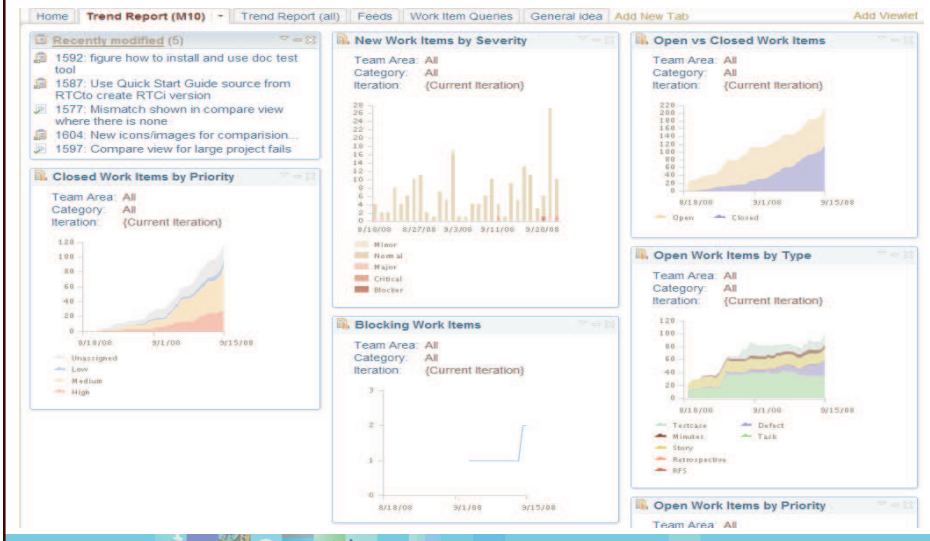


Tooling for RPG programmers - RTCi

RTCi (Rational Team Concert for i)

- Collaboration tool to help manage projects
- Development environment for a whole team (RDi is for each developer)
- Automate and enforce workflow practices
- Source control
- Change management
- Track enhancements, tasks, defects

RTCi dashboard



More

- **RDi SOA**
  - Develop RPG and COBOL applications
  - Create and consume web services
  - Create Web 2.0 applications using RPG and COBOL via EGL
  - more
- **Rational Business Developer (RBD)**
  - EGL
  - Web 2.0
  - Web services
  - More
- **Rational Host Access Transformation Services (HATS)**
  - Extend 5250 applications to new users on web, mobile etc
  - Fast return on investment with short learning curve
  - More

Sandbox: Try out Rational Tools for i

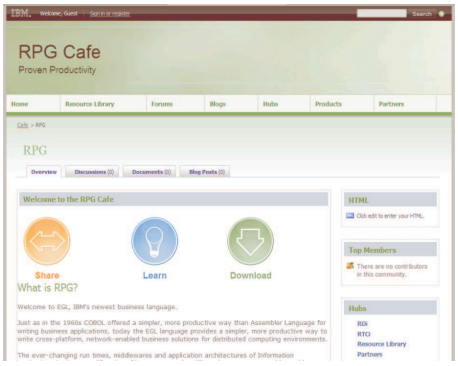
- Rational Developer for i (RDi)
- RDi SOA (with EGL)
- Rational Host Access Transformation Services (HATS)
- X-Analysis from Databorough

http://www.ibm.com/developerworks/downloads/emsandbox/

RPG Cafe

Visit the RPG Cafe for articles, tutorials, podcasts and forums about RPG, RDi and RTCi.

www.ibm.com/rational/cafe



Go to IBM