RPG IV - V5R2 and beyond . . .

George Farr, iSeries RPG and eclipse Development Manager
farr@ca.ibm.com
IBM Toronto Laboratory
SP13

WDS/400
New World
New
New Tools

ITSO iSeries Technical Forum
SP13

Disclaimer

Acknowledgement:

- This presentation is a collaborative effort of the IBM Toronto AS/400 Application Development presentation team, including work done by:
  - Phil Coulthard, George Farr, Claus Weiss, Don Yantzi, John Steinbacher, Barbara Morris

Disclaimer:

- The information contained in this document has not been submitted to any formal IBM test and is distributed on an as is basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customers' ability to evaluate and integrate them into the customers' operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environment do so at their own risk.

Reproduction:

- The base presentation is the property of IBM Corporation. Permission must be obtained PRIOR to making copies of this material for any reason.

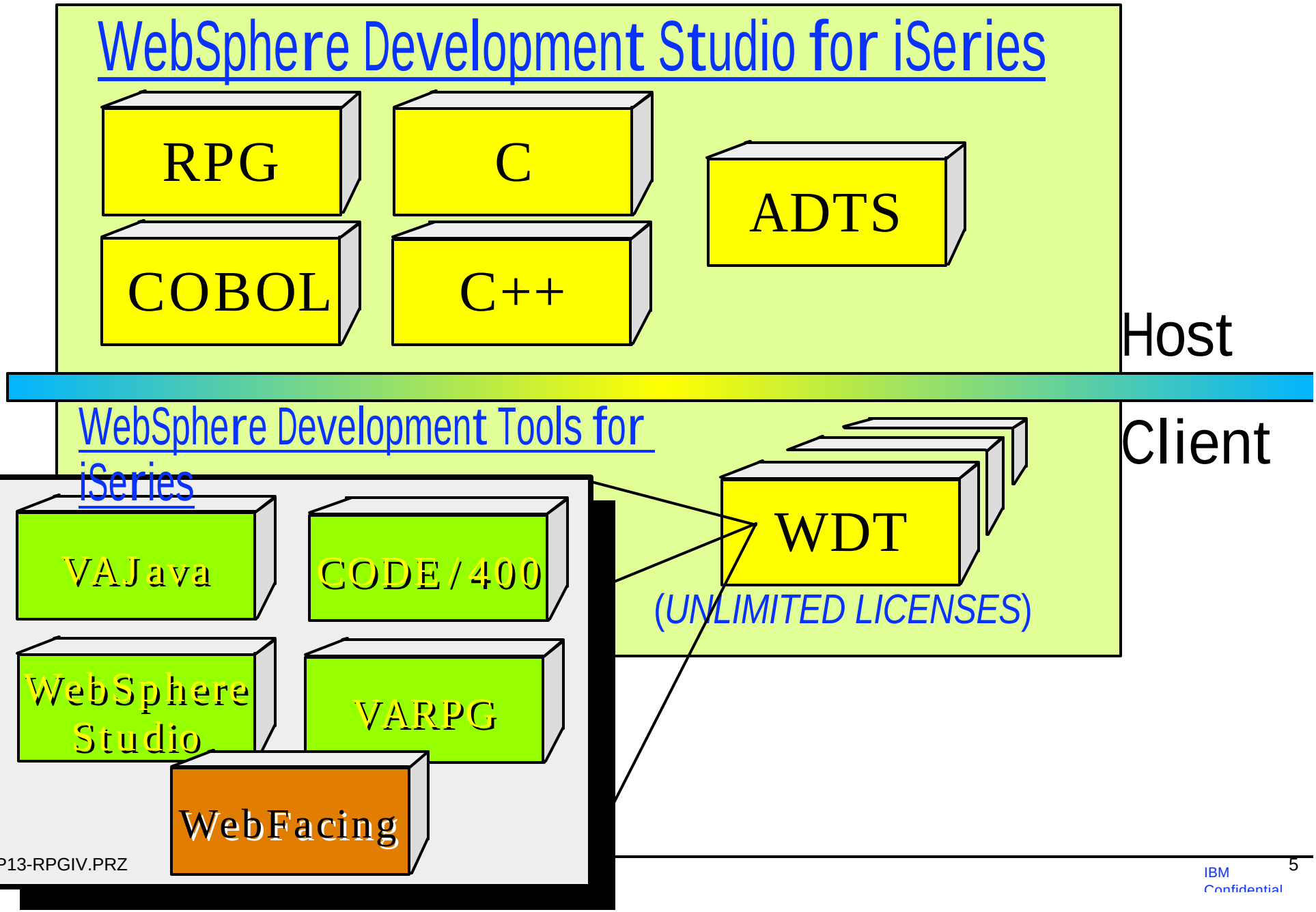
Another BIG release for RPGIV

- 31 digit support
- Allow Char. Parms for built-ins
- IFS Source File Support
- Lib qualified access to data areas
- PCML generation
- Short form operators
- I/O enhancements
- What's Next for RPGIV?

AGENDA

RPG IV
Packaging

WDS Packaging V5R1



WDSc 4.0!

5722-WDS
Refresh code 2655

WebSphere Development Studio for iSeries

RPG C COBOL C++ ADTS

Host

~~WDT~~
WDSc

WSSDa + iSeries Tools

Client

Team

Projects

UI F/W

Plugin Tools

Java Tools

WAS Unit Test Environment

Web Tools

AppSrvr Tools

WebSvc Tools

XML Tools

DB Tools

iSeries Projects

iSeries Extns

iSeries Extns

iSeries Extns

Remote Systems Explorer

WebFacing

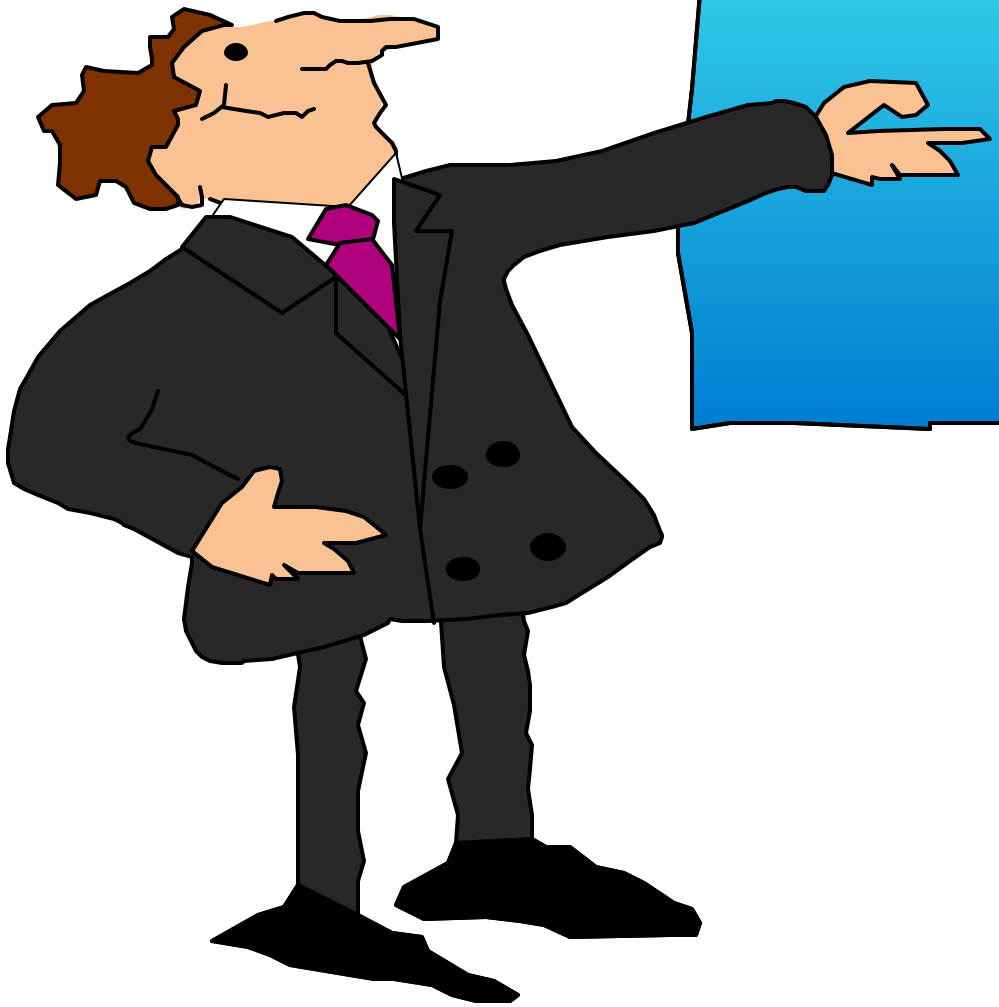
CODE

VARPG

VAJava

Studio

AGENDA



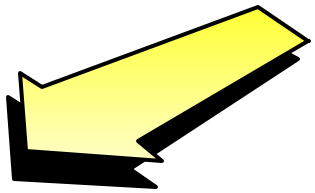
Fast V5R1 Review!

V5R1

- V5R1 enhancements . . .
  - *New built-in functions:*
  - *%CHECKR, %XLATE,%SCANR, %LOOKUP*
  - *Monitor operation group*
  - *Increasing Java-related support*
  - *CLASS keyword on the D spec*
  - *EXTPROC maps to Object (CLASS) methods*
  - *STATIC keyword for static methods*
  - *Free Form Calcs!*
  - *Runtime control of file to be opened*
  - *Date/time/timestamp operations in expressions*
  - *ELSEIF*
  - *Predefined compiler directives*
  - *Qualified names in data structures*

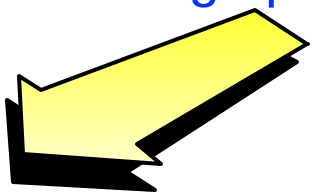
Built-ins

For date/time/timestamp operations in expressions:



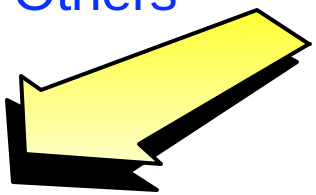
- %DIFF
- %MSECONDS
- %SECONDS
- %MINUTES
- %HOURS
- %DAYS
- %MONTHS
- %YEARS
- %DATE
- %TIME
- %TIMESTAMP
- %SUBDT

From existing opcodes:



- %ALLOC
- %REALLOC
- %CHECK
- %CHECKR
- %LOOKUPxx
- %TLOOKUPxx
- %OCCUR
- %SHTDN
- %SQRT
- %XLATE

Others



- %DATE
- %TIME
- %TIMESTAMP
- %CHAR

MONITOR GROUP

The MONITOR block consists of the READ statement and the IF group.

'1211' is issued if file is NOT opened

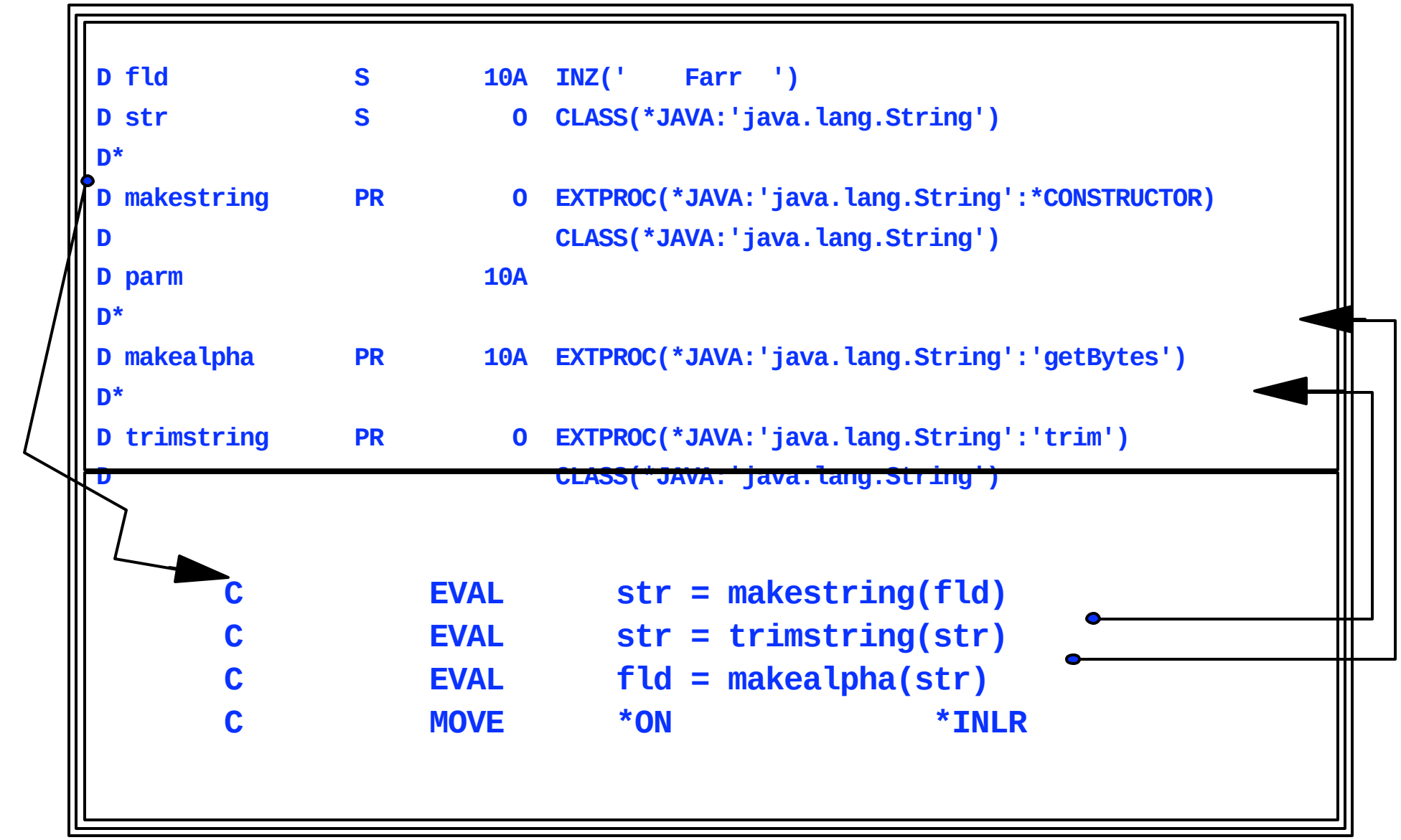
```
C           MONITOR
C           READ      FILE1
C           IF        NOT %EOF
C           EVAL      Line = %SUBST(Line(i) :
C                           %SCAN('***':Line(i)) + 1)
C           ENDIF
C*
C           On-Error        1211
C            ... handle file-not-open
C           On-Error        *FILE
C            ... handle other file errors
C           On-Error        00100 : 00121
C            ... handle string error and array-index error
C           On-Error
C            ... handle all other errors
C           ENDMON
```

'100' and '121'are string operation status codes

We could have *ALL

If no errors occur, control passes to ENDMON

Java Enablement



```
D fld             S        10A  INZ('    Farr  ')
D str             S          O  CLASS(*JAVA:'java.lang.String')
D*
D makestring      PR         O  EXTPROC(*JAVA:'java.lang.String':*CONSTRUCTOR)
D                               CLASS(*JAVA:'java.lang.String')
D parm                     10A
D*
D makealpha       PR       10A  EXTPROC(*JAVA:'java.lang.String':'getBytes')
D*
D trimstring      PR         O  EXTPROC(*JAVA:'java.lang.String':'trim')
D                               CLASS(*JAVA:'java.lang.String')

     C           EVAL       str = makestring(fld)
     C           EVAL       str = trimstring(str)
     C           EVAL       fld = makealpha(str)
     C           MOVE       *ON                *INLR
```

Free Form C-Specification

```
/FREE
      read  file;                 // Get next record
      dow not %eof(file);         // Keep looping while we have
                                  // a record
          if %error;
              dsply &csq.The read failed&csq.;
              leave;
          else;
              chain(n) name database data;
              time = hours * num_employees
                        + overtime_saved;
              pos = %scan (&csq.,&csq.&colon. name);
              name = %xlate(upper&colon.lower&colon.name);
              exsr handle_record;
              read file;
          endif;
      enddo;

    begsr handle_record;
      eval(h) time = time + total_hours_array (empno);
      temp_hours = total_hours - excess_hours;
      record_transaction();
    endsr;
 /end-free
```

OVRDBF

CL

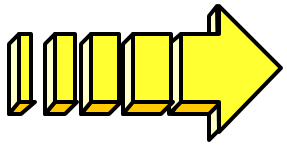
```
OVRDBF OVRNAME OVRLIB/OVRFILE
CALL RPG
```

RPG

```
FFILE1   IF   E           EXTFILE(OpenThisFile)

 * This OPEN operation will open SOMELIB/SOMEFILE
C                   EVAL      OpenThisFile = 'SOMELIB/SOMEFILE'
C                   OPEN      FILE1

 * This OPEN operation will open OVRLIB/OVRFILE because there
 * is an override for filename 'OVRNAME' in effect.
C                   EVAL      OpenThisFile = 'OVRNAME'
C                   OPEN      FILE1
```

ELSEIF

```
H*
C*                :
C                 IF          *IN01
C*                :
C                 ELSE
C                 IF          *IN03
C*                :
C                 ELSE
C                 IF          *IN99
C*                :
C                 ELSE
C*                :
C                 END
C                 END
C                 END
```



```
H*
C*                :
C                 IF          *IN01
C*                :
C                 ELSEIF      *IN03
C*                :
C                 ELSEIF      *IN99
C*                :
C                 ELSE
C*                :
C                 END
```

Qualified Subfield Name

```
D cust          DS                QUALIFIED   <---- specify subfields ds.subf
D   name                    50a               <---- has subfield called "name"
D   id_num                  10i 0
D part          DS                QUALIFIED
D   name                    25a               <----also  has subfield called "name"
D   id_num                  10i 0

D part1         DS                LIKEDS(part) <---- define with same subfields
D part2         DS                LIKEDS(part) <---- new DS is also QUALIFIED

C     eval       cust.name = 'ABC Electronics'
C     eval       part1.name = 'Radio' <---- ds name is required
C     eval       part1.id_num = 100035
C     eval       part2.name = 'Telephone'
C     eval       part2.id_num = 100036
```

AGENDA

AGENDA

31 Digit Support

- 31 Digit support:
  - **Field definitions**
    - Packed and Zoned now support 31-digit total length
    - Explicit (D-spec) and implicit (C-spec) declarations
    - Internal (explicit, implicit) and external (record formats)
  - **Expressions**
    - Decimal numeric max precision now 31 digits (vs 30)

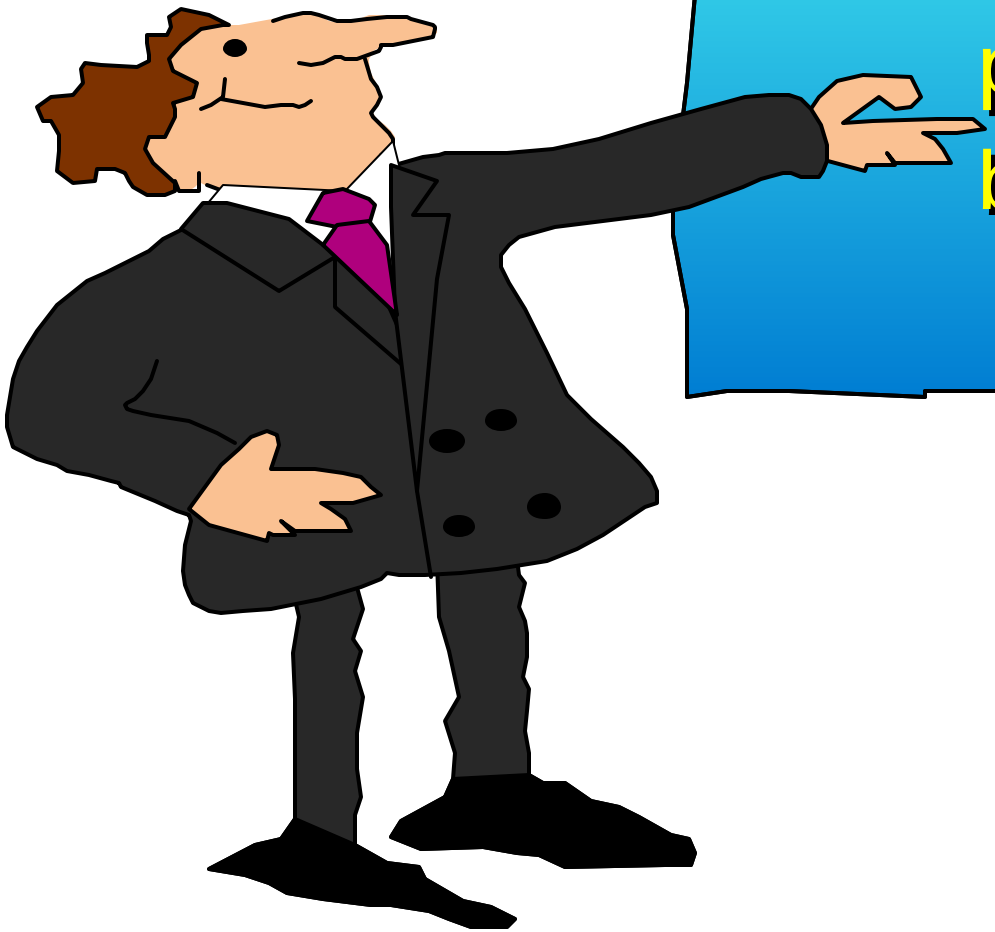
- Why increase it from 30?

31 Digit support

DECPREC keyword:

- DECPREC(30|31)
- specifies the precision of decimal values within expressions
- specific to %EDITC and %EDITW operations
- default value is 30
- If expression contains a decimal variable declared as 31, keyword does not apply and 31 digits will be used

AGENDA



Character parameters for built-ins!

Char. for built-ins

- Character parms for built-ins
  - **Currently %INT, %INTH, %UNS, %UNSH, %DEC, %DECH, and %FLOAT allow only numeric operands**
    - Enhanced to allow character data for V5R2
    - It represents valid numeric value, built-in will return that value
      - ✓ %DEC('-15.57' : 7 : 2)
      - ✓ will return a packed(7,2) value of -00015.57
  - **Rules for %INT, %INTH, %UNS, %UNSH, %DEC, %DECH:**
    - ✓ The sign (+/-) can precede or follow the value
    - ✓ The decimal point can be either period (.) or comma (,)
    - ✓ For %UNS and %UNSH, only positive signs (+) are allowed
    - ✓ The second and third parameters are required for %DEC(character)
      - ✓ This is similar to the existing rule for a float parameter.
    - ✓ Floating point data, for example '1.5E3', is not allowed.
      - ✓ For other builtins, this value will result in an "Invalid Numeric Data" exception.
    - ✓ Blanks are allowed anywhere. For example ' 1 2 3 . 4 5 - ' is a valid input string

Char. built-ins

- Rules for %FLOAT:
  - The sign must precede the value for both mantissa and exponent
  - The decimal point can be either period (.) or comma (,).
  - Blanks are allowed anywhere. For example ' 1 2 3 . 4 5 E - 5' is a valid input string
  - The 'E' for the exponent may be in upper or lower case

Example

- Character parms for built-ins
  - **Examples:**

```
BIF                                  result          notes
-------------------------------------------------------------------
%INT('123')                           123
%UNS('123.6')                         123            decimal truncation
%INTH('123.6')                        124            rounding
%INTH('123.6-')                      -124            trailing sign
%UNSH(' 1 2 3 . 6 ')                  124
%DEC('-123.6' : 5 : 2)               -123.60
%DEC('123.6 -' : 5 : 2)              -123.60          trailing sign
%DEC('   5 123 456,18 ' : 15 : 2) 5123456.18        comma decimal point
%DEC('   5 123 456.18-' : 15 : 1)-5123456.1        decimal truncation
%DECH('  5 123 456.188' : 15 : 1) 5123456.2        rounding

%FLOAT('123E9')                       123E9
%FLOAT('-12345')                     -1.2345E4
%FLOAT('  1,2 E 9  ')                 1.2E9          comma decimal point
```

Examples - Exceptions!

BIF	status	problem
%DECH(' 5 123 456.18' : 5 : 1)	00103	overflow
%UNS('-1')	00105	neg. sign not allowed for %UNS
%INT('-1+')	00105	too many signs
%INT('1234A')	00105	invalid character (A)
%DECH('1.5E7' : 5 : 1)	00105	Only floating point allowed
%DECH('1.5.3' : 5 : 1)	00105	too many decimal points
%DECH(' ' : 5 : 1)	00105	no numeric data
%FLOAT('123-')	00105	invalid character (sign must precede number for float)

Example...

```
 *----------------------------------------------------------------
 * If the character data is known to contain non-numeric characters
 * such as thousands separators (like 1,234,567) or leading
 * asterisks and currency symbols (like $***1,234,567.89), some
 * preprocessing is necessary to remove these characters from the
 * data.
 *----------------------------------------------------------------
D data            s              20a   inz('$1,234,567.89')
D num             s              21p 9
 /free

    // Use the %XLATE builtin function to replace any currency
    // symbol, asterisks or thousands separators with blanks

    num = %dech(%xlate('$*,' : '   ' : data)
              : 21 : 9);

    // If the currency symbol or thousands separator might
    // vary at runtime, use variables to hold these values.

    num = %dech(%xlate(cursym + '*' + thousandsSep : '   ' : data)
              : 21 : 9);
```

AGENDA

Data Structure

Enhancements

Data Structure enhancements

- Data structure enhancements
  - **Keyword DIM is allowed on a data structure definition to improve usability of multiple-occurrence data structures**
  - **Keyword LIKEDS is allowed on a subfield definition**
  - **Arbitrary levels of indexing and qualification are allowed:**

```
ds(x).subf1.s2.s3(y+1).s4
```

Data Structure enhancements

```
D CustomerInfo     DS                        QUALIFIED BASED(@)
D    Name                              20A
D    Address                           50A

D ProductInfo      DS                        QUALIFIED BASED(@)
D    Number                             5A
D    Description                       20A
D    Cost                               9P 2

D SalesTransaction...
D                  DS                        QUALIFIED
D    Buyer                                   LIKEDS(CustomerInfo)
D    Seller                                  LIKEDS(CustomerInfo)
D    NumProducts                       10I 0
D    Product                                 LIKEDS(ProductInfo)
D                                            DIM(10)

 /free
     TotalCost = 0;
     for i = 1 to SalesTransation.Numproducts;
         TotalCost = TotalCost + SalesTransaction.Products (i).Cost;
     endfor;
     dsply SalesTransaction.Products (i).Cost;
     dsply ('Total cost is ' + %char(TotalCost));
 /end-free
```

Array Data Structure

Description:

- Keyword DIM allowed on data structure
  - similar to multiple occurence data structure
    elements referenced by array index
- Multiple Occurences may be referenced in one expression
- LIKEDS allowed on subfield definition
  - when specified subfield is defined to be a data structure
    has its own set of subfields

Example:

- Data Structure (DS) has subfield S1
- S1 defined as a data structure with subfield S2
  coded as:

```
DS.S1.S2
```

Array Data Structure

Definitions:

- Simply Qualified Name
  - form "A.B"
  - allowed as
    - argument on keywords in F and D specs
    - in Field-name entries on I and O specs
    - Factor 1, Factor 2 and Result-Fields on fixed form C specs
  - no white space allowed between names and the dot
- Fully Qualified Name
  - name with qualifications and indexing to arbitrary number of levels
    - e.g. "`A(X).B.C(Z+17)`"
  - Allowed in
    - free-form C specs
    - and extended Factor-2 entries

Array Data Structure

Example:

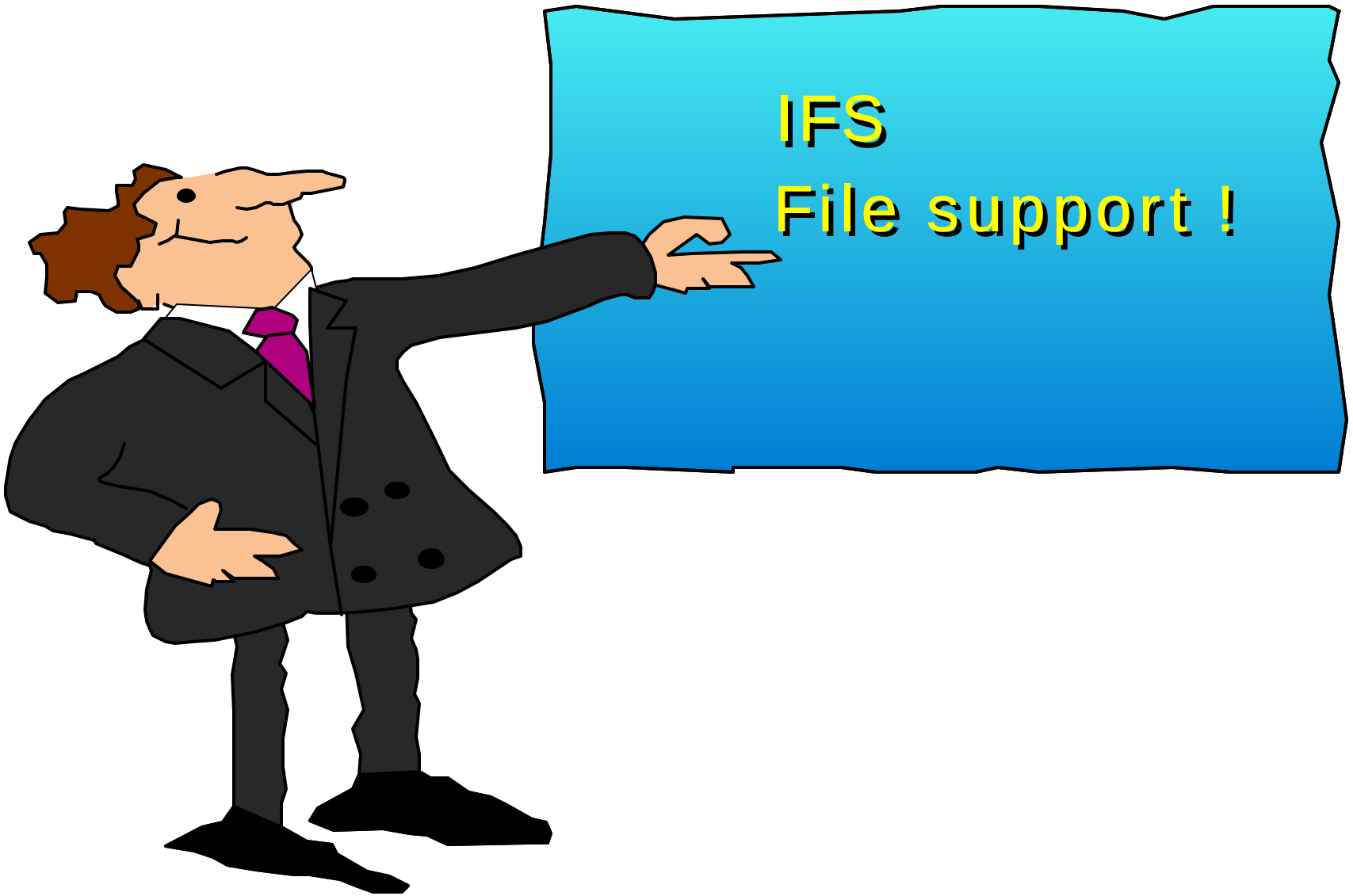
```
D CustomerInfo          DS                  QUALIFIED   BASED(@)
D   Name                            20A
D   Address                             50A

D ProductInfo           DS                  QUALIFIED   BASED(@)
D   Number                           5A
D   Description                     20A
D   Cost                             9P  2

D SalesTransaction...
                        DS                  QUALIFIED
D   Buyer                                   LIKEDS(CustomerInfo)
D   Seller                                  LIKEDS(CustomerInfo)
D   NumProducts                      10I  0
D   Product                               LIKEDS(ProductInfo)
D                                         DIM(10)


/free
     TotalCost = 0;
     for i = 1 to SalesTransaction.Numproducts;
         TotalCost = Total Cost + SalesTransaction.Product(i).Cost;
     endfor
     dsply SalesTransaction.Product(i).Cost;
     dsply ('Total cost is ' + %char(TotalCost));
/end-free
```

Nested Data Structures - Initializing

INZ(*LIKEDS)

- allowed on LIKEDS subfield
- initialized exactly the same as corresponding data structure
- if no INZ specified, subfield is initialized to x'40's
  - even if INZ is coded at the main data structure definition that contains the LIKEDS subfield
- INZ may be coded for a LIKEDS subfield
  - initialized to default values
- Includes all levels of nested subfields
  - NOT like nested LIKEDS subfields with INZ(*LIKEDS)
- If INZ is specified at main data structure definition, INZ is implied for all LIKEDS subfields in the data structure
  - except for the LIKEDS subfields with INZ(*LIKEDS)

AGENDA



IFS Support

- IFS Support!
  - **Existing support**
    - possible to access data on IFS
    - APIs are available
  - **New V5R2 support for compiling out of IFS**
    - source files and copy files can now be in IFS file system
  - **Parameters added to the create commands (CRTBNDRPG, CRTRPGMOD)**
    - **SRCSTMF** is used instead of SRCFILE and SRCMBR to indicate a stream file is the main source file
    - **INCDIR** is used to list the copy-file directories
    - new directive allows inclusion of files from IFS source files
      - similar to /COPY
- However...
  - CVTRPGSRC will not be changed to support IFS
  - Compiled objects cannot live in IFS

AGENDA

Short Form Assignment Operators

- Short-form Assignment Operators
  - **Compressed syntax when modifying field based on its old variable:**
    - <eval> target **op=** expression
  - **... same as...**
    - <eval> target **=** target **op** expression

Operation	Short Form Operator	Example	Full
Add	`+=`	`a += b`	`a = a + b`
Subtract	`-=`	`a -= b`	`a = a - b`
Multiply	`*=`	`a *= b`	`a = a * b`
Divide	`/=`	`a /= b`	`a = a / b`
Exponential	`**=`	`a **= b`	`a = a**b`

AGENDA

Bitwise operations

- Bitwise built-in functions
  - **%BITAND(expr:expr<:expr...>)**
    - Logically AND the bits
  - **%BITOR(expr:expr<:expr...>)**
    - Logically OR the bits
  - **%BITXOR(expr:expr)**
    - Logically EXCLUSIVE-OR the bits
  - **%BITNOT(expr)**
    - Logically NEGATE the bits

Bitwise operations

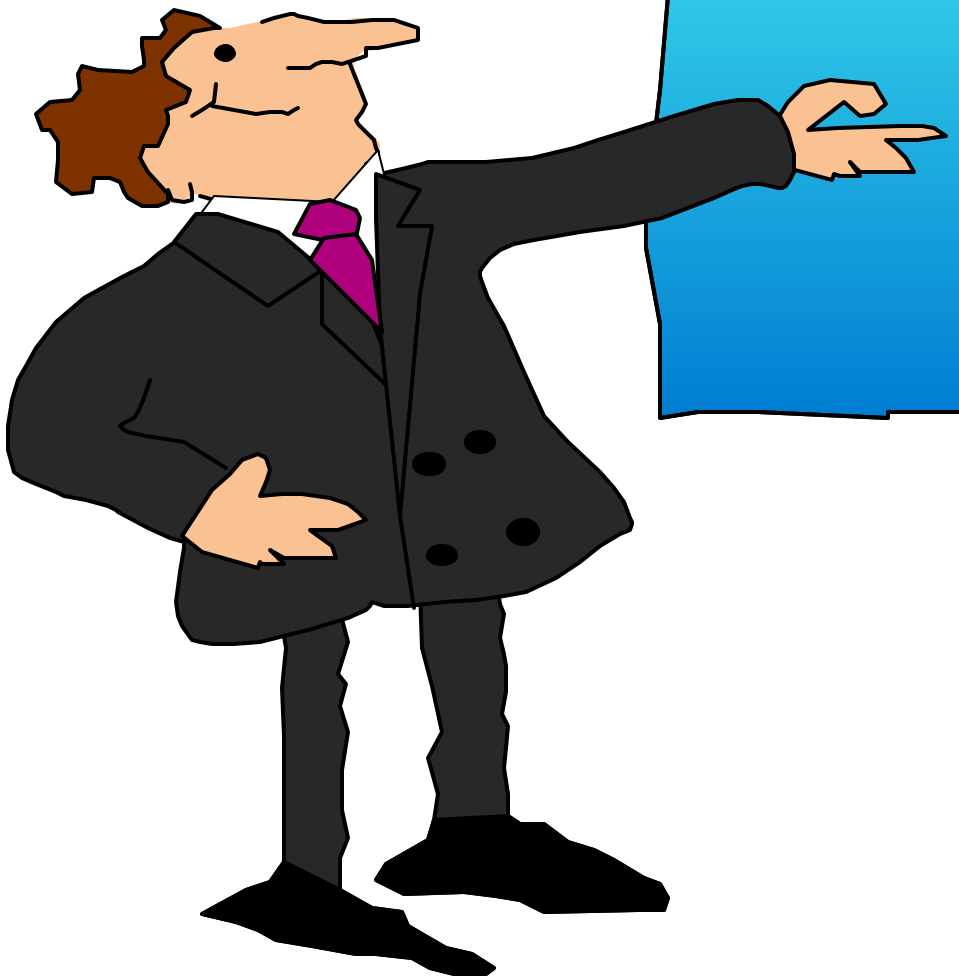
```
D const          c                  x'0007'
D ch1            s            4a    inz(%BITNOT(const))
 * ch1 is initialized to x'FFF84040'
D num1           s            5i 0 inz(%BITXOR(const:x'000F'))
 * num is initialized to x'0008', or 8
D char2a         s            2a
D char2b         s            2a
D uA             s            5u 0
D uB             s            3u 0
D uC             s            5u 0
D uD             s            5u 0
```

```
C                eval       char2a = x'FE51'
C                eval       char2b = %BITAND(char10a : x'0F0F')
 * char2b = x'0E01'
C                eval       uA = x'0123'
C                eval       uB = x'AB'
C                eval       uc = x'8816'
C                eval       uD = %BITOR(uA : uB : uC)
```

```
* operand1   = b'1111 1110 0101 0001'
* operand2   = b'0000 1111 0000 1111'
* bitwise AND:   0000 1110 0000 0001
```

```
* operand1   = b'0000 0001 0010 0011'
* operand2   = b'0000 0000 1010 1011'
* operand3   = b'1000 1000 0001 0110'
* bitwise OR:    1000 1001 1011 1111
```

AGENDA



Qualified data areas

RPG V5R2 Part 6

Qualified Data Areas

- **Name of data area specified in character constant or variable must be in following form:**
  - 'NAME'
  - 'LIBRARY/NAME'
  - '*LIBL/NAME'
- **valid for single parameter form of DTAARA or using new DTAARA(*VAR) form**
  - ** DTAARA keyword had un-quoted name of data area
- **Naming must follow same rules as called programs and EXTFILE F-spec keyword**
- **External name of the DTAARA used when RPG handles the IN op code**
  - except when DTAARA already locked by a previous *LOCK IN operation ... with no interim OUT or *UNLOCK operation

Qualified Data area

- Qualified Data Areas Example:

```
D dtal             S       10A   DTAARA(*VAR : pgmvar)
D pgmvar           S       21A

C                EVAL            pgmvar = 'LIB1/DTAARA1'
C           IN         dta1
* The data area LIB1/LIB1/DTAARA1 is read into variable dta1

C           eval       pgmvar = 'LIB1/DTAARA2'
C  *LOCK    IN         dta1

* Data area LIBL1/DTAARA2 is locked, and its contents are read
* into  "pgmvar".  Until data area is unlocked, the data area
* being used by dta1 is fixed.

C           eval       pgmvar = 'LIB1/DTAARA3'
C           IN         dta1
* The data area LIB1/DTAARA2 is read again.  The value of the
* DTAARA variable is ignored, since the data area is already
* locked and in use by the RPG program

C           OUT        dta1
* The data area LIB1/DTAARA2 is updated with the contents of dta1
* and unlocked. Since the data area has been unlocked, the
* external data area can now be changed.
```

Example continue...

- Qualified Data Areas Example Ct'd:

```
C             EVAL       pgmvar = 'LIB1/DTAARA3'
C             IN         dta1
* The data area LIB1/DTAARA3 is read.  Since the data area is not
* locked, the external data area can be changed for the next use
* of dta1.

C             eval       pgmvar = 'LIB1/DTAARA4'
C   *LOCK     IN         dta1
* The data area LIB1/DTAARA4 is locked and read into dta1
C   *LOCK     OUT        dta1
* The data area LIB1/DTAARA4 is updated with the contents of dta1
* but it is not unlocked.
C             eval       var = 'LIBL/DTAARA5'
C             IN         dta1
* The data area LIB1/DTAARA4 is read into dta1 again.  Since it
* was not unlocked by the OUT command, it is still locked , and
* the existing locked data area DTAARA4 is used.
```

AGENDA

Data Structures

Say we change the RPG code to do this . . .

```
FCUSTOML3  IF   E           K DISK
DCUSTINFO        DS
D Number                    1      7A
D Name                      8     47A
C      *ENTRY    PLIST
C                PARM                    CUSTINFO
C      Number    SETLL     CUSTOM01
C      Number    READE     CUSTOM01                  9091
C                EVAL      Name = CUSTNA
C                MOVE      *ON             *INLR
****************** End of data **************************
```

Pass in Customer ID and receive back customer name.

PCML for Data Structures

```
<pcml version="1.0">

 <!-- Create a Data Structure -->
 <struct name="custinfo">
   <data name="Number" type="char" length="7"
         usage="inputoutput" init="0014400"> </data>
   <data name="Name"   type="char" length="40"
         usage="inputoutput" init=" "> </data>
 </struct>

 <!-- Program getcust -->
 <program name="getcust"
          path="/QSYS.lib/FARR.lib/GETCUST.pgm">
   <data name="gotback" type="struct"
         usage="inputoutput" struct="custinfo"> </data>
 </program>

</pcml>
```

Name and describe Data Structure

Reference named DS

Pass DS as input and output

Receiving DS in Java from RPG

File: GetCust.java
Class: GetCust

```
public static void main(String[] argv)
{
    AS400 as400System = new AS400();
    ProgramCallDocument pcml = null;
    String msgId, msgText;
    Object value = null;

    try {
      System.out.println(
         "Creating ProgramCallDocument for GetCust pgm.");
      pcml = new ProgramCallDocument(as400System, "GETCUST");
      boolean ok = pcml.callProgram("getcust");
      System.out.println(" rc is---> " + rc);
      if (!ok)
        { /* Retrieve list of AS/400 messages & display them */ }
      else
        {
          value = pcml.getValue("getcust.gotback.Name");
          System.out.println("Customer name: " + value);
        }
    } catch (PcmlException exc) {
      System.out.println("*** Call to getcust failed. ***");
      System.exit(0);
    }
    System.exit(0);
} // end main method
```

Name of PCML file

Retrieve Name

Results . . .

```
Command Prompt
f:\toolbox\examples>javac GetCust.java
f:\toolbox\examples>java GetCust
Constructing ProgramCallDocument for GetCust pgm...
 rc is---> true
Customer name: Great Neck Industries

f:\toolbox\examples>
```

What about arrays?

```
<data type="{ char | int | packed | zoned | float | byte | struct }"
        [ ccsid="{ number | data-name }" ]
        [ count="{ number | data-name }" ]
        [ init="string" ]
        [ length="{ number | data-name }" ]
        [ maxvrm="version-string" ]
        [ minvrm="version-string" ]
        [ name="name" ]
        [ offset="{ number | data-name }" ]
        [ offsetfrom="{ number | data-name | struct-name }" ]
        [ outputsize="{ number | data-name | struct-name }" ]
        [ precision="number" ]
        [ struct="struct-name" ]
        [ usage="{ inherit | input | output | inputoutput }" ]>
</data>
```

Specifies number of elements!

PCML

- PCML Support!
  - **PCML will be generated from RPG compiler**
  - **PCML will generally be complete...**
    - However, when requested from the CRTRPGMOD and CRTCBLMOD commands, some manual fixup is needed since the compilers don't know how the module will be used. In particular, for each program tag, the path and entrypoint attributes will have to be manually fixed up
- What is PCML?
  - **Program Call Markup Language**
    - XML language Toolbox for Java uses to define entry points into programs and service programs
    - Eases effort to call RPG from Java
- Why PCML?
  - **To enable tools: ie Program Call wizard in WDSc**

AGENDA

I/O operations

- Enhanced I/O operations
  1. **Extract specific fields for externally described DS**
     - EXTNAME(filename{:extrecname}{:***ALL|*INPUT|*OUTPUT|*KEY**})
  2. **Keyword LIKEREC to define a DS with same subfields as an externally described record format**
     - LIKEREC(intrecname{:***ALL|*INPUT|*OUTPUT|*KEY**})
  3. **List of keys on keyed I/O operations**

```
/free
     chain (a: b+c: %subst(d:e:1)) record;
/end-free
```

  4. **%KDS on keyed I/O operation**
  5. **Data structure name on keyed I/O operations to externally described files**
  6. **List of fields to update on UPDATE operation**

Extract Specific Fields

```
EXTNAME(filename{:extrecname}{:*ALL|*INPUT|*OUTPUT|*KEY})
```

- D-Spec keyword EXTNAME can take optional 2nd or 3rd parameter
  - indicates the types of fields to extract for the externally described data structure
- If no subfields meet requirements, data structure has no subfields

*ALL	All fields in external record are extracted
*INPUT	All input capable fields are extracted (default??)
*OUTPUT	All output capable fields are extracted
*KEY	Only key fields are extracted In order specified on K spec of DDS

* Fixes problem of only extracting *Both fields from *DSPF

LIKEREC Keyword

```
LIKEREC(intrecname{:*ALL|*INPUT|*OUTPUT|*KEY})
```

- Extract fields from an internal record format
- D-Spec keyword; optional 2nd parameter indicating types of fields to extract

*ALL	All fields in internal record are extracted
*INPUT	All input capable fields are extracted (default??)
*OUTPUT	All output capable fields are extracted
*KEY	Only key fields are extracted In order specified on K spec of DDS

- First parm must be name of internal record format
- Second optional parameter must match the definition of the associated record or file
- Similar to LIKEDS but not for DS

List of Keys on Keyed I/O

Current Keyed I/O format:

```
CHAIN(EHMNR) fieldname file-or-record-name {ds-name};
DELETE(EHMR) klistname file-or-record-name;
```

Future Keyed I/O format:

```
CHAIN(EHMNR) (expression{:expression ..}) file-or-record-name {ds-name};
DELETE(EHMR) (expression{:expression ..}) file-or-record-name;
READE(EHMNR) (expression{:expression ..}) file-or-record-name {ds-name};
READPE(EHMNR)(expression{:expression ..}) file-or-record-name {ds-name};
SETLL(EHMNR) (expression{:expression ..}) file-or-record-name;
SETGT(EHMNR) (expression{:expression ..}) file-or-record-name;
```

```
/free
    chain (a:b+c: %subst(d:e:1)) record;
```

- List of expressions is allowed as the search argument for any keyed I/O operation in a free-form group
- Search argument is the compound key formed from all expressions in the list

List of Keys on Keyed I/O

```
%KDS(data-structure-name{:num-keys})
```

- %KDS is allowed as a search argument for any keyed I/O operations coded in a free-form group
  - CHAIN, DELETE, READE, READPE, SETGT, SETLL
- may be an externally described data structure
- Rules
  - first argument must be a ds name
    - includes subfields defined with LIKEDS
  - second argument provides number of subfields to use as argument
  - subfields used to form compound key may not be arrays
  - more to follow

```
D ds            e  ds                        extname(extfile:*key)
/free
            chain %kds(ds) record;
/end-free
```

List of Keys on Keyed I/O

Future Keyed I/O format:

1.
```
CHAIN(EHMNR) fieldname file-or-record-name {ds-name};
```

2.
```
DELETE(EHMR) klistname file-or-record-name;
```

3.
```
CHAIN(EHMNR) (expression{:expression ..}) file-or-record-name {ds-name};
DELETE(EHMR) (expression{:expression ..}) file-or-record-name;

        /free
            chain (a:b+c: %subst(d:e:1)) record;
        /end-free
```

4.
```
D ds          e  ds                          extname(extfile:*key)
/free
              chain %kds(ds) record;
/end-free
```

Data Structure name on I/O to Externally Described Files

- Today:
  - can use data structure name on I/O for program described files
- Future:
  - may specify data structure name on I/O to externally descibred files also
    - CHAIN, READ, READC, READE, READP, READPE, UPDATE, WRITE
  - name may be fully qualified
    - may be a LIKEREC subfield
  - types of fields included must match the type of I/O operation being executed
    - e.g. for input or update operations - base externally described data structure must be extracted with *INPUT
- Example:

```
Read        X           Dsname
```

BIF - List of Fields to Update

```
%FIELDS (name{:name ...})
```

- List of fields can be specified as final argument to I/O operation UPDATE in a free-form group
- Built-in function %FIELDS allowed for externally described files
  - on UPDATE operation
- Each name must be name of a fields in the input buffer for the record

```
/free
           chain empno record;
           salary = salary + 2000;
           status = STATEXEMPT;
           update records %fields(salary:status);
/end-free
```

AGENDA

V5R3

What's in?

- SQL Preprocessor
- 64 Digit support (Packed and Zoned)
- ... More

Possible items fo V5R3 . . .

- ***The selected items are:***
  - Allow dynamic resizing of arrays and multi-occurrence DS.
  - Conversion of date/time/tstamp to numeric using %INT, %DEC.
  - Second parameter on %TRIMx - char to trim.
  - Option to pass trimmed string as parameter.

- ***If time permits, high priority:***
  - Allow EXTPGM to be coded without a parameter.
  - LEAVE-WHEN & ITER-WHEN: single-statement conditional leave or iter.
  - DEBUG(*RETVAL) to allow debugging of procedure return values

- ***If time permits, lower priority:***
  - Procedure name overloading based on parm types.
  - EVALC move corresponding from one data structure to another.

Dynamic resizing of arrays

SUBARR(array:start:{elements})

- Built-in function %SUBARR() returns a contiguous subset of the specified array starting at the specified starting index
- Can be used in any place an array can be used except as a procedure parameter.
- Array parameter can be any array including procedure return values.
- This can be used to implement dynamically sized arrays with varying number of elements.

Dynamic resizing of arrays

EXAMPLE

```
a(1)=9;
a(2)=5;
a(3)=16;
a(4)=13;
a(5)=3;
```

```
// Copy part of an array to another array:
array = %subarr(a:4:n);
%subarr(b:3:n) = %subarr(a:m:n);

// Sort subset of array:
sorta %subarr(a:1:4);  // Now, A=(5,9,13,16,3);

// Dynamically allocated arrays:
// (parr is array based on pointer p)
n = 17;
p = %alloc(%size(parr) * n);
%subarr(parr:1:n) = *blank;
b = %subarr(parr:1:n);
```

%DEC (date | time | timestamp { : format })

- Allow date, time and timestamp parameters for %DEC, with an optional format parameter, to allow conversion from these types to numeric, in expressions.
- Currently it is simple to do this conversion using the MOVE operation, but MOVE is not supported in /free calculations.
- By enhancing %DEC to take date, time and timestamp parameters and an optional format, conversion in /free calculations will be straightforward.
- The length of the result is the number of digits in the date, time or timestamp value. For example:
  --->%DEC(date : *MDY) = length of 6; Length of %DEC(timestamp) is 20.
- If the format is not specified, the default format will be used. (*ISO).
  For date and time conversions, the default may be overridden by the DATFMT and TIMFMT keywords on the Control Specification.

Example . . .

```
 * Using %DEC to convert dates, times and timestamps to numeric
D   mmddyy        S              6S 0
D   yyyymmdd      S              8P 0
D   hhmmss        S              6P 0
D   ts            S             20S 0
D   date          S               D     inz(D'2003-06-27')
D   time          S               T     inz(T'09.25.59')
D   timestamp     S               Z     inz(D'2003-06-27.09.25.59.')
```

Free form

```
/free
// equivalent /free coding

   mmddyy = %dec(date : *mdy);
   yyyymmdd = %dec(date : *iso);
   hhmmss = %dec(time : *hms);
   ts = %dec(timestamp);

   // Results:
   //   mmddyy = 062703
   //   yyyymmdd = 20030627
   //   hhmmss = 092559
```

Fixed form

```
 * Move the date, time and timestamp to the numeric
 * fields using the MOVE operation
C     *mdy          movel     date          mmddyy
C     *iso          movel     date          yyyymmdd
C     *hms          movel     time          hhmmss
C                   movel     timestamp     ts
```

%TRIMx 2nd parameter . . .

%TRIMx(string{:trim-chars})

- An optional second parameter is allowed on the %TRIMx built-in functions to indicate the set of characters to trim off the end(s) of the specified string.
- The second parameter must be the same type as the first parameter, which include:
  - character
  - graphic
  - ucs2
- If 2nd parameter is not specified, the trim character defaults to blank.

%TRIMx 2nd parameter . . .

Examples

```
%trim('xxxABCxxx':'x');
```

Returns 'ABC'

```
%trimr('>>>>ABC<<<<<':'<>');
```

Returns '>>>>ABC'

```
tc = 'xyz';
%triml('xxyyzzyyzzABCxyzyxzxy':tc);
```

Returns 'ABCxyzyxzxy'

OPTION (*TRIM)

OPTION(*TRIM)

Add a new prototyped-parameter option OPTIONS(*TRIM), to request RPG to trim a string before passing

it to the called procedure.

OPTIONS(*TRIM) is valid with

CONST or VALUE varying-length character, UCS-2 or graphic parameters

CONST or VALUE fixed-length character, UCS-2 or graphic parameters when OPDESC is coded on the prototype

Pointer parameters when OPTIONS(*STRING) is also coded

The benefit of this enhancement is that it allows the writer of a prototype to tell the compiler

the parameter must be trimmed, rather than telling the callers of the procedure through documentation.

This ensures that a trimmed parameter is always passed, without relying on programmers to read the documentation.

It also simplifies the calls to the procedure, by eliminating the need to code %TRIM.

OPTION (*TRIM)

```
 * Using OPTIONS(*TRIM) to pass a trimmed OPTIONS(*STRING) parameter

 * The function stat() does not allow trailing blanks in the
 * IFS path.  By coding OPTIONS(*TRIM) on the filename
 * parameter, we assure that trailing blanks are never passed.

D   stat          PR                  LIKEDS(statResult)
D                                     EXTPROC('stat')
D      path                     *     VALUE
D                                     OPTIONS(*STRING : *TRIM)


 /free

       filename = '/home/george/myfile.txt';

       // Since filename is a fixed length field, its
       // value is now '/home/george/myfile.txtbbbbbbbbbb...bbb'
       // (where b represents blanks)

       // However, the programmer does not need to worry about
       // trimming the filename on the call, because OPTIONS(*TRIM)
       // will handle the trimming.

        result = stat (filename);
```

Leave / Iter

LEAVE WHEN expression;

ITER WHEN expression;

A "when clause" can be coded on opcodes LEAVE and ITER. When the specified expression is true, the operation is performed.

Example

```
// Loop through all records of file
dow '1';
   read record data;
   leave when %eof(file);// Are we done?
   iter when data.code <> 'B3E';// Process only B3E records
   process(data);
enddo;
```

AGENDA

Summary

Summary

- **V5R1 &2 are Monster Releases**
  - **Major enhancements for RPG IV**
  - **New Tools**
  - **Major repackaging**
    - Now you get it all
- **Why You Should Feel Good:**
  - IBM is committed to iSeries
  - IBM is committed to Java for *eBusiness*
  - IBM is committed to RPG IV for *your business*

AGENDA

COBOL V5R2 At a Glance

- Another big release for ILE COBOL
  1. **More intrinsic functions**
  2. **Recursive support**
  3. **Local storage support**
  4. **IFS source file support**
  5. **PCML generation**

COBOL V5R2 Part 1

- More ANSI Intrinsic Functions!
  - **MAX**
  - **MEDIAN**
  - **MIDRANGE**
  - **MIN**
  - **ORD-MAX**
  - **ORD-MIN**
  - **PRESENT-VALUE**
  - **RANGE**
  - **STANDARD-DEVIATION**
  - **SUM**
  - **VARIANCE**
- Plus UTF8STRING
  - **Converts strings to UTF-8 format**
    - For better Java interoperability

COBOL V5R2 Part 2

- Recursion Support
  - **Optional RECURSIVE clause**
    - COBOL programs can call themselves!
    - You can now write games in COBOL :-)

- Local Storage Support
  - **Local variables!!**
    - A new data section that defines storage allocated and freed on a per-invocation basis.
    - You can specify the Local-Storage Section in recursive programs, and in non-recursive programs

COBOL V5R2 Part 3

- IFS Support!
  - **Existing support**
    - possible to access data on IFS
    - APIs are available
  - **New V5R2 support for compiling out of IFS**
    - source files and copy files can now be in IFS file system
  - **Parameters added to the create commands (CRTBNDCBL, CRTCBLMOD)**
    - **SRCSTMF** is used instead of SRCFILE and SRCMBR to indicate a stream file is the main source file
    - **INCDIR** is used to list the copy-file directories
    - new directive allows inclusion of files from IFS source files
- However...
  - Compiled objects cannot live in IFS

COBOL V5R2 Part 4

- PCML Support!
  - **PCML will be generated from COBOL compiler**
  - **PCML will generally be complete...**
    - However, when requested from the CRTCBLMOD and CRTRPGMOD commands, some manual fixup is needed since the compilers don't know how the module will be used. In particular, for each program tag, the path and entrypoint attributes will have to be manually fixed up
- What is PCML?
  - **Program Call Markup Language**
    - XML language Toolbox for Java uses to define entry points into programs and service programs
    - Eases effort to call COBOL from Java
- Why PCML?
  - **To enable tools: ie Program Call wizard in WDSc**

Trademarks & Disclaimers

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.
The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

AS/400	IBM(logo)
AS/400e	iSeries
e (logo) business	OS/400
IBM	

Lotus, Freelance Graphics, and Word Pro are registered trademarks of Lotus Development Corporation and/or IBM Corporation.
Domino is a trademark of Lotus Development Corporation and/or IBM Corporation.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.
UNIX is a registered trademark of The Open Group in the United States and other countries.
SET and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.
Other company, product and service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information in this presentation concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of the specific Statement of Direction.

Some information in this presentation addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in IBM product announcements. The information is presented here to communicate IBM's current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Photographs shown are of engineering prototypes. Changes may be incorporated in production models.