
```
exec sql
declare rs cursor for
        select CUSTNO, NAME, CONTACT, STR
               POSTAL, TITLE, BALANCE
         from CUSTFILE
        where CUSTNO=:CUSTNO;

exec sql open rs;
exec sql fetch next from rs into :CUSTFILE;

dow sqlcode = 0;

  output = 'CUSTNO  = ' + %char(CUSTNO);
  snd-msg output;
  output = 'NAME    = ' + NAME;
  snd-msg output;
  output = 'CONTACT = ' + CONTACT;
  snd-msg output;
  output = 'STREET  = ' + STREET;
  snd-msg output;
  output = 'CITY    = ' + CITY;
  snd-msg output;
  output = 'STATE   = ' + STATE;
  snd-msg output;
  output = 'POSTAL  = ' + POSTAL;
```

Agenda

- *There isn't enough time to teach a whole programming language in this session!*
- But we can look at:
  - Basic examples of the syntax
  - How it makes your life easier
  - What makes RPG a great business language
  - A few powerful features of RPG

What is 

- A business-oriented programming language
- Makes it very easy to work with databases, write business rules, work with screens, and print reports.
- It does *not* stand for Report Program Generator.
  - *It used to -- however... In 1994 IBM changed it to officially not stand for anything!*
- Even amongst season professionals, there is a lot of confusion about the name, the "versions", and languages released.
- Versions are backward compatible, allowing code to move forward to new versions when they are available.
- Very much under development, new enhancements released every spring and fall. More than 30 major enhancements in the past 5 years.

 Languages

Language	Introduced		Stopped Enhancing/Supporting
FARGO (Fourteen-O-One Automatic Report Generation Operation)	1959	IBM 1401	1960 / 1971
RPG ("RPG I") (Report Program Generator)	1960	IBM 1401	1971
RPG II	1968	IBM System/3, 34, 36, Mainframes, Others, AS/400 with System/36 Environment	1988 / still supported
RPG III	1978	IBM System/38, AS/400, Windows, Others	1993 / still supported
RPG IV	1994	Windows, IBM i, Others	still enhanced / supported

NOTE: The numbers I, II, III, IV are not properly known as "versions" – but different languages. Each of the above has many different versions available.

Compilers

Compiler	Language	Nicknames
System/36 Compatible RPG II	RPG II	RPG/36
System/38 Compatible RPG III	RPG III	RPG/38
RPG/400	RPG III	
Visual RPG (discontinued)	RPG III	VRPG
VisualAge for RPG (discontinued)	RPG IV	VARPG
ILE RPG for IBM i (formerly ILE RPG/400)	RPG IV	RPGLE, RPGILE, RPG/FREE

NOTE: Versions of the ILE RPG compiler that are still supported are 7.2, 7.3, 7.4 and 7.5. (Sometimes called V7R2, V7R3, etc)

Fixed vs Free-Format

```
     C     *INLR         DOUEQ     *ON

     C                   WRITE     INVINQ2F
     C                   EXFMT     INVINQ2C
     C                   MOVEL     *BLANKS       SCMSG

     C     *IN03         IFEQ      *ON
     C     *IN12         OREQ      *ON
     C                   LEAVESR
     C                   ENDIF

     C     *IN25         IFEQ      *ON
     C                   MOVE      *OFF          *IN25
     C                   EXSR      OPENURL
     C                   ENDIF

     C     *IN08         IFEQ      *ON
     C                   MOVEL     'B'           MODE
     C                   Z-ADD     2             STEP
     C                   LEAVESR
     C                   ENDIF

     C                   ENDDO
```

```
dou *inlr = *on;

  write INVINQ2F DSP2F;
  exfmt INVINQ2C DSP2C;
  scMsg = *blanks;

  if *in03 = *on or *in12 = *on;
    leavesr;
  endif;


  if *in25 = *on;
    *in25 = *off;
    exsr openURL;
  endif;

  if *IN08 = *ON;
    mode = 'B';
    step = 2;
    leavesr;
  endif;

enddo;
```

It surprises a lot of people who are new to RPG that these aren't different languages, or even different versions of RPG. They are both compiled with the ILE RPG compiler, both compatible with version 7.5, and both considered RPG IV!

Recommendation: Don't Bother Learning Fixed!

- Don't bother learning fixed format (until you need it)
- There are free tools to convert from fixed format to free
- Also, once you understand free format, it's easy enough to read fixed format.
- Code the new stuff in free.

```
dou *inlr = *on;

  write INVINQ2F DSP2F;
  exfmt INVINQ2C DSP2C;
  scMsg = *blanks;

  if *in03 = *on or *in12 = *on;
    leavesr;
  endif;


  if *in25 = *on;
    *in25 = *off;
    exsr openURL;
  endif;

  if *IN08 = *ON;
    mode = 'B';
    step = 2;
    leavesr;
  endif;

enddo;
```

Hello World

```
      snd-msg 'Hello World';
      *inlr = *on;
```

snd-msg just writes an informational message.

*inlr = *on ends the program.

```
**free

dcl-s name varchar(30);

name = 'Scott Klement';

if %subst(name: 1: 5) = 'Scott';
   snd-msg 'Hello, Scott!'
endif;

*inlr = *on;
```

**FREE allows us to start in column 1, and make lines as long as we want.

DCL-S declares a variable.

IF <condition>;

...stuff to do...

Endif;

Notice that there is a semicolon after the if condition.

Declaring Variables

Variables in RPG are declared with a DCL-xxx keyword.

Most of the time, this means using DCL-S

- "declare stand alone variable"

Sometimes you declare data structures, prototypes, procedure interfaces, and other things.

- DCL-S = declare standalone
- DCL-C = declare constant
- DCL-DS = declare data structure
- DCL-F = declare file (database table, screen, printer, tape)
- DCL-PR = prototype (for calling other routines)
- DCL-PI = procedure interface (parameter interface to a routine)
- DCL-PROC = procedure/function

DCL-S <variable name> <data type> (length) keywords.

Declaring Variables

DCL-S <variable name> <data type> (length) keywords;

```
dcl-s Var1 char(10);                    // fixed-length 10 characters
dcl-s Var2 varchar(1234);               // variable-length 1234 chars
dcl-s Var3 varchar(1234) ccsid(*utf8);  // variable-length, but in utf-8
dcl-s Var4 ucs2(4321)    ccsid(*utf16); // fixed-length but in utf-16

dcl-s Var5 packed(9: 2);                // xxxxxxx.xx
                                        // packed decimal 9 digits, 2 decimals
                                        //  (digits is the total incl the decimals)
                                        // xxxxxxx.xx
dcl-s Var6 packed(18: 0);               // xxxxxxxxxxxxxxxx
dcl-s Var7 zoned(11: 3);                // xxxxxxxx.xxx

dcl-s Var8 int(10);                     // 32-bit (10 decimal digits) integer
dcl-s Var9 float(8);                    // 64-bit (8 byte) floating point

dcl-s Var10 date;                       // date variable (default YYYY-MM-DD format)
dcl-s Var11 date(*usa);                 // date variable in MM/DD/YYYY format
dcl-s var12 time;                       // time variable
dcl-s var13 timestamp;                  // date+time variable
```

DCL Keywords

- CCSID = character set of the variable (special values *utf8 and *utf16 for Unicode)
- INZ = initialize (set the initial variable value)
- LIKE = variable is the same data type/length as another variable
- LIKEDS = variable is like a data structure
- Many others exist – I won't try to name them all here!

```
dcl-s Var3 varchar(1234) ccsid(*utf8);
dcl-s Var4 ucs2(4321)    ccsid(*utf16);

dcl-s Var4 ucs2(4321)    ccsid(*utf16);

dcl-s Var5  packed(9: 2);
dcl-s price like(Var5) inz(10.45);


dcl-ds Name_t;
  FirstName varchar(15) inz('Scott');
  LastName  varchar(15) inz('Klement');
end-ds;

dcl-ds OtherName likeds(Name_t) inz(*likeds);
```

There are many DCL keywords and an endless list of things you can do with them – they are a super powerful feature of RPG!

Why is RPG Good For Business?

- Strictly-typed variables allow you to catch more potential errors at compile-time.
- This prevents bad data from getting into your system.
- DCL keywords make it easy and powerful

- True decimal arithmetic.
- Database is better integrated into the language.
- Easy to build full-screen applications
- Easy to work with date, time and timestamp (date+time) variables.
- Easy to work with REST APIs, XML, JSON

Floating Point vs. True Decimal Math

RPG supports true decimal arithmetic, whereas most programming languages using floating point numbers.

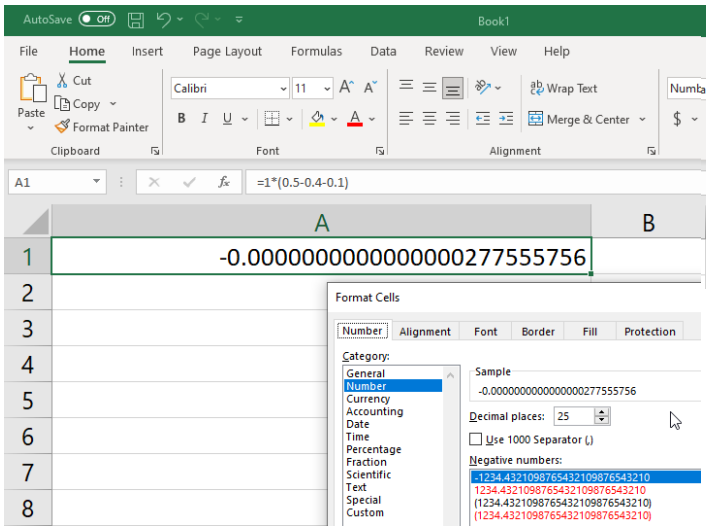
Why does this matter?

=1*(.5-.4-.1)

Consider this Excel formula. What do you expect the result to be?

Excel uses floating point – like most programming languages – and is prone to the same problems.

Floating Point vs. True Decimal Math



It should've been 0.

But all numbering systems (decimal, binary, hex, etc) represent certain fractions with repeating numbers.

In decimal, 1/3 is 0.33333333333 (repeats forever).

But the computer can't store an infinite number of decimal places – so it has to round it off at some point.

The numbers that cause this in binary are different than the ones in decimal – but the same problem exists.

As a business that serves humans – it's better to round off in decimal, since that's what people expect.

RPG works in decimal numbers, not binary.

Floating Point vs. True Decimal Math

```c
#include <stdio.h>

int main(int argc, char **argv) {

   double result;

   result = 1*(.5-.4-.1);
   printf("%26.25f\n", result);

   return 0;
}
```

```java
public class numbers2 {
  public static void main(String[] args) {

    double result;

    result = 1*(.5-.4-.1);
    System.out.println(String.format("%26.25f", result));

  }

}
```

To prove my point, here are examples in both C and Java. (Other languages are similar – even RPG would do the same if you forced it to use floating point math)

In all cases, they print the following (same as Excel):

```
-0.0000000000000000277555756
```

How Does That Affect Business?

Is it okay for your business to have values that are slightly off?

- Payroll
- Revenue
- Inventory
- Quantity shipped to a customer

RPG Packed vs. Zoned

```
dcl-s result1 packed(9: 2);
dcl-s result2 zoned(9: 2);

result1 = 1*(.5-.4-.1);
result2 = 1*(.5-.4-.1);

snd-msg %char(result1);
snd-msg %char(result2);

*inlr = *on;
```

```
This will print:
0.00
0.00
```

Packed and Zoned are both numeric data types.

- Difference is how they're stored in memory.
- Packed is a form of "binary coded decimal", typically uses ½ the memory of zoned.
- But zoned is easier to read if you see the raw unformatted value in memory.

In both cases, you specify the size as a number of digits and decimal places.

But they are "true decimal" types, not subject to binary rounding

Loops

```
**Free

dcl-s X int(10);

dow X > 0;
  // do something
enddo;
```

DOW = Do While

Pretty much the same as a while loop in any other language.

```
dou X = 0;
  // do something
enddo;
```

DOU = Dou Until (condition is checked at the 'enddo', so loop is always done once)

For Loops

```
for x = 1 to 10;
  // do something
endfor;
```

Loop through a range of numbers

```
for element in array;
  // do something
endfor;
```

Loop through the items in an array

```
for item in %list('Item1': 'Item2': 'Item3');
  // do something
endfor;
```

Loop through a fixed set of arbitrary valuesv

I'm just scratching the surface of what the different types of loops can do – just to give you a feel for it.

There are many, many, many more options available!

Sub-Procedures ("functions")

```
name = MyProcedure(last: first);
snd-msg name;

. . . other stuff could be here . . .

dcl-proc MyProcedure;

  dcl-pi *n;
    last varchar(15) const;
    first varchar(15) const;
  end-pi;

  dcl-s fullname varchar(30);

  fullname = first + ' ' + last;
  return fullname;

end-proc;
```

You can write your own functions and make them available within the current program.

Or export them to make them available to other programs as well.

RPG calls them "sub-procedures".

Notice that RPG's built-in functions always begin with a % character. The functions you write cannot begin with that character.

This makes it easy to distinguish the origin of a function, and also makes it easy to avoid naming clashes.

Integrated Database

```
dcl-f CUSTFILE disk keyed;

CUSTNO = 1500;
chain CUSTNO CUSTFILE;


// the CUSTFILE database table contains columns named
// CUSTNO, NAME, CONTACT, STREET, CITY, STATE, POSTAL
// and BALANCE -- all are ready to use!
```

DCL-F (declare file)

- Declares a database table
- keyed = Allows keyed (indexed) access.
- Automatically declares variables for all of the columns ("fields") in the table ("file").
- CHAIN = loads a record by it's key.

Integrated Database – Java Comparison 1/3

In most other languages (Java, in this example) you have to:

- Import a database driver
- Use it to connect to the database
- Build an SQL statement in a character string
- Run the statement
- Read the results into columns
- Clean up after the statement
- Close the connection

```
import java.sql.*;
import com.ibm.as400.access.AS400JDBCDriver;

public class JavaTest {

  public static void main(String[] args) {

    try {
      // Load driver

      Class.forName("com.ibm.as400.access.AS400JDBCDriver");

      // connect to database

      String jdbcUrlFmt = "jdbc:as400://localhost;user=%s;password=%s;naming=system;";
      String jdbcUrl = String.format(jdbcUrlFmt, Credentials.user, Credentials.password);

      Connection conn = DriverManager.getConnection(jdbcUrl);
```

Integrated Database – Java Comparison 2/3

```
      PreparedStatement stmt = conn.prepareStatement("select CUSTNO, NAME, CONTACT, "
                                                          + "STREET, CITY, STATE, "
                                                          + "POSTAL, TITLE, BALANCE "
                                                   + "from CUSTFILE "
                                                   + "where CUSTNO=?");
      stmt.setInt(1, 1500);

      ResultSet rs = stmt.executeQuery();
      while (rs.next()) {

        int CUSTNO     = rs.getInt   ("CUSTNO");
        String NAME    = rs.getString("NAME");
        String CONTACT = rs.getString("CONTACT");
        String STREET  = rs.getString("STREET");
        String CITY    = rs.getString("CITY");
        String STATE   = rs.getString("STATE");
        String POSTAL  = rs.getString("POSTAL");
        double BALANCE = rs.getDouble("BALANCE");


        // Do something with the column values

      }
```

Integrated Database – Java Comparison 3/3

```
        rs.close();
        conn.close();
      }
      catch (Exception e) {
        System.out.println(e.getMessage());
      }

    }

}
```

Or, in RPG:

```
dcl-f CUSTFILE disk keyed;

CUSTNO = 1500;
chain CUSTNO CUSTFILE;

if %found;
  // do something with the column values
endif;
```

The bulk of work in a business application is calculations and database:

- True decimal arithmetic
- Easier database operations

NOT FAIR! Not an Apples-To-Apples Comparison

```
dcl-ds CUSTFILE ext end-ds;
dcl-s output varchar(500);

CUSTNO = 1500;

exec sql
declare rs cursor for
       select CUSTNO, NAME, CONTACT, STREET, CITY, STATE,
              POSTAL, TITLE, BALANCE
         from CUSTFILE
        where CUSTNO = :CUSTNO;

exec sql open rs;
exec sql fetch next from rs into :CUSTFILE;

dow sqlcode = 0;

  // Do something with the column values

  exec sql fetch next from rs into :CUSTFILE;
enddo;

exec SQL close rs;
```

RPG's proprietary native I/O isn't totally equivalent to SQL. SQL is more modern and can do many more things.

So let's compare the same example with using SQL in RPG.

Notice the use of :CUSTNO instead of ?. This makes the code much more readable.

Also, notice the SQL is not built in a string. This lets the compiler & IDE detect errors in your syntax.

It also makes the coding a lot easier, no need to concatenate values (unless you want to), just type what you want.

Easy Dates/Times

```
**Free

dcl-s DeliveryDate date(*usa);
dcl-s output varchar(80);
dcl-s LeadTime int(10);

LeadTime = 14;
DeliveryDate = %date() + %days(LeadTime);

output = 'Delivery Date = ' + %char(DeliveryDate);
snd-msg output;

*inlr = *on;
```

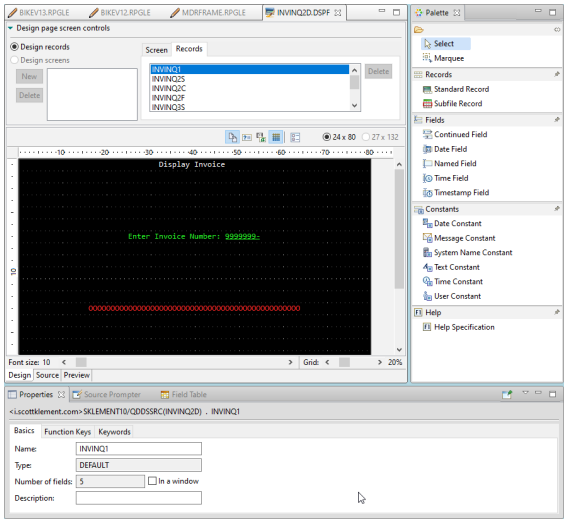
Just a quick example.

You can easily add or subtract days, months, years, etc.

Or hours, minutes, seconds, etc from a time.

This type of logic is very common and necessary in business!

Screens Are Easy



For basic text screens

- IBM provides an easy-to-use WYSIWYG screen editor, where you can build your screens using drag/drop, etc.
- Then you refer to these screens as "files" in your program.

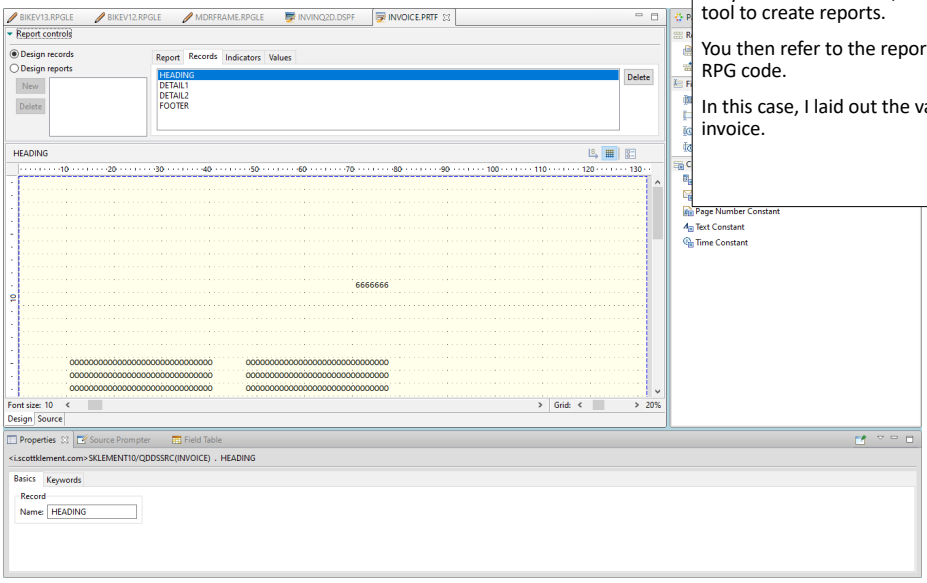
Screens Are Easy

```
dcl-f INVINQ2D workstn INDDS(dspIndMap);

... more code here ...

  dou DSP1.MSG = *BLANKS;

    exfmt INVINQ1 DSP1;       // Displays the screen
    DSP1.MSG = *BLANKS;

    if EXIT or CANCEL;
      return QUIT;
    endif;

    if DSP1.INVNO <= 0;
      DSP1.MSG = 'Please enter an invoice number!';
      iter;
    endif;

  enddo;
```

For basic text screens

- EXFMT = "execute screen format".
- User can use the screen until they press ENTER or a function key.
- Then control comes back to the program.

Reports Are Easy



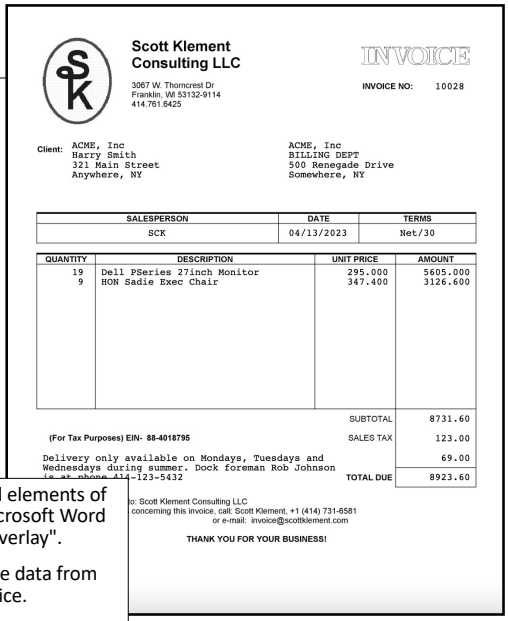
Very much like screens, IBM provides a GUI tool to create reports.

You then refer to the report as a file in your RPG code.

In this case, I laid out the variable data for an invoice.

Reports Are Easy

```
dcl-f INVOICE PRINTER oflind(overflow) usage(*output)
                      usropn;


setll (myInvNo) INVDET;
reade (myInvNo) INVDETF;

write HEADING;

dow not %eof(INVDET);

  lineamt = %dec(qty * price: 9: 3);

  if overflow;
    write FOOTER;
    write HEADING;
    first = *on;
  endif;
  write DETAIL;

  reade (myInvNo) INVDETF;
enddo;

write FOOTER;
```



Scott Klement
Consulting LLC

3067 W. Thorncrest Dr
Franklin, WI 53132-9114
414.761.6425

INVOICE

INVOICE NO: 10028

Client: ACME, Inc
Harry Smith
321 Main Street
Anywhere, NY

ACME, Inc
BILLING DEPT
500 Renegade Drive
Somewhere, NY

SALESPERSON	DATE	TERMS
SCK	04/13/2023	Net/30

QUANTITY	DESCRIPTION	UNIT PRICE	AMOUNT
19	Dell PSeries 27inch Monitor	295.000	5605.000
9	HON Sadie Exec Chair	347.400	3126.600

SUBTOTAL	8731.60
SALES TAX	123.00
	69.00
TOTAL DUE	8923.60

(For Tax Purposes) EIN- 88-4018795

Delivery only available on Mondays, Tuesdays and Wednesdays during summer. Dock foreman Rob Johnson is at phone 414-123-5432

o: Scott Klement Consulting LLC
concerning this invoice, call: Scott Klement, +1 (414) 731-6581
or e-mail: invoice@scottklement.com

THANK YOU FOR YOUR BUSINESS!

The fixed headings and graphical elements of the invoice were designed in Microsoft Word – I then told it to generate an "overlay".

The overlay gets merged with the data from the printer file to make this invoice.

Easy REST APIs

```
Ctl-Opt DFTACTGRP(*NO) ACTGRP('WEBAPI') PGMINFO(*PCML:*MODULE);

Dcl-F CUSTFILE Usage(*Input) Keyed PREFIX('CUST.');

Dcl-DS CUST ext extname('CUSTFILE') qualified End-DS;

Dcl-PI *N;
   CustNo                   like(Cust.Custno);
   Name                     like(Cust.Name);
   Street                   like(Cust.Street);
   City                     like(Cust.City);
   State                    like(Cust.State);
   Postal                   like(Cust.Postal);
End-PI;

Dcl-PR QMHSNDPM  ExtPgm('QMHSNDPM');
   MessageID      Char(7)    Const;
   QualMsgF       Char(20)   Const;
   MsgData        Char(32767) Const options(*varsize);
   MsgDtaLen      Int(10)    Const;
   MsgType        Char(10)   Const;
   CallStkEnt     Char(10)   Const;
   CallStkCnt     Int(10)    Const;
   MessageKey     Char(4);
   ErrorCode      Char(8192) options(*varsize);
End-PR;
```

To write a quick & dirty REST API, all you need to do is write a program that gets it's input/output through parameters.

The operating system provides a tool called "Integrated Web Services" that can be configured to call this program.

It will handle all of the communications work, converting data between JSON or XML and the parameters, etc.

Easy REST APIs

```
Dcl-DS err  qualified;
   bytesProv      Int(10)    inz(0);
   bytesAvail     Int(10)    inz(0);
End-DS;

Dcl-S MsgDta       Varchar(1000);
Dcl-S MsgKey       Char(4);
Dcl-S x            Int(10);

chain CustNo CUSTFILE;
if not %found;
   msgdta = 'Customer not found.';
   QMHSNDPM( 'CPF9897': 'QCPFMSG   *LIBL': msgdta: %len(msgdta): '*ESCAPE'
           : '*PGMBDY': 1: MsgKey: err );
else;
   Custno = Cust.Custno;
   Name   = Cust.name;
   Street = Cust.Street;
   City   = Cust.City;
   State  = Cust.State;
   Postal = Cust.Postal;
endif;

*inlr = *on;
```

This API simply returns a customer's address given a customer number.

The QMHSNDPM routine sends an error when the customer wasn't found. (This is an older example, a newer one might use SND-MSG to do the same thing.)

Handling XML or JSON Yourself

```
Ctl-Opt OPTION(*SRCSTMT: *NODEBUGIO) DFTACTGRP(*NO);

Dcl-F CUSTFILE Usage(*Input) Keyed prefix('CUST.');
dcl-ds CUST ext extname('CUSTFILE') qualified end-ds;

Dcl-PR getenv Pointer extproc('getenv');
   var  Pointer    value options(*string);
End-PR;

dcl-s custno like(CUST.custno);
Dcl-S pos    int(10);
Dcl-S uri    varchar(1000);
Dcl-S json   varchar(1000);
Dcl-C ID1    '/cust/';
Dcl-C ID2    '/custinfo/';

dcl-ds failure qualified;
  error varchar(100);
end-ds;
```

If you need to process XML or JSON from a file, parameter, or other means (aside from the Integrated Web Services) or if you want to write the API yourself without a special tool, it's relatively easy to do.

RPG provides a built-in way to map XML or JSON to an RPG variable called DATA-INTO. (or the older XML-INTO.)

Likewise, it can generate XML or JSON using another tool called DATA-GEN.

Easy REST APIs

```
uri = %str(getenv('REQUEST_URI'));

monitor;
  pos = %scan(ID1: uri) + %len(ID1);
  custno = %int(%subst(uri:pos));
on-error;
  failure.error = 'Invalid URI';
  DATA-GEN failure %DATA(json) %GEN( 'YAJLDTAGEN'
   : '{ "http status": 500, "write to stdout": true }');
  return;
endmon;

chain custno CUSTFILE;
if not %found;
  failure.error = 'Unknown customer number';
  DATA-GEN failure %DATA(json) %GEN( 'YAJLDTAGEN'
    :'{ "http status": 500, "write to stdout": true }');
  return;
endif;

DATA-GEN cust %DATA(json)  %GEN( 'YAJLDTAGEN'
  :'{ "http status": 200, "write to stdout": true }');
return;
```

In this example, the CUSTFILE database table is automatically loaded into the CUST data structure.

DATA-GEN is being used to convert the data structure to JSON format and write it out.

As with most everything in this presentation, I am just scratching the surface! This functionality is full of loads of different features and options to make it extremely versatile!

Rich Community of Developers and Tools

Although this session focuses on what's in RPG itself, it's worth mentioning the huge community of developers and tools out there:

- People in this community like helping each other!
- We are very close knit!
- Rich world of open source tools to help you (including things like Git and Jenkins)

- Vendors provide tools that make a lot of this stuff even easier... I can't list them all, but some of my own are:
  - MDCMS = Change Management / devops / agile
  - MDREST4i = Easier APIs

  - PROFOUND UI (previous job) make GUI web-based screens just as easily as text based ones.
- Many other vendors offer these types of things – lots of good choices available!

Recommended Resources:

- IBM Docs – the official source of documentation:
  (online books, official IBM manuals)
  Programming / ILE Languages / RPG

- Programming in ILE RPG (5th Edition) by Jim Buck & Bryan Meyers
  (in-depth book with exercises, 664 pages)
  https://www.amazon.com/Programming-ILE-RPG-Jim-Buck/dp/1583473793

- COMMON Bootcamp: Programming in ILE RPG
  (getting started video -- no charge to COMMON members!)
  http://www.common.org/education-events/boot-camp-training/programming-ile-rpg

- imPOWER Technologies
  (instructor-led online courses)
  https://impowertechnologies.com

Questions?

For this presentation visit my web site:

http://www.scottklement.com/presentations/