RPG Free Form
In a Nutshell

Susan Gantner

Susan @ SystemiDeveloper.com

SystemiDeveloper.com



Notes

Most Summit sessions with RPG code samples will be coded using free format RPG - including free format data and procedure declarations. If you could use a refresher on those topics to make sure you'll be able to understand the syntax of the code samples you see, you should attend this session.

Perhaps you've heard about free form RPG - you may even have read an article or two, or even tried your hand at writing some free-form code. But many RPGers have stumbled over "how do I ... ?" issues and wonder if it is really worth the effort.

If this sounds like you then come to this session and Susan will do her best to clear up any misconceptions you may have and show you why and how to take this next step.

In this session, Susan will attempt to answer questions such as:

- Why is free format RPG important?
- What are the rules?
- Just how free can you go?
- Are there any likely gotchas?
- Are there tools available to convert fixed format code easily?

The session will include a brief refresher on free format logic, but will focus primarily on the new free-form declarations (i.e. the replacements for H, F, D and P specs) as well as the ability to use as many columns for your RPG logic as you need - starting in column 1 if you want!

The presenter, Susan Gantner, is co-founder of Partner400, a firm specializing in customized education and mentoring services for IBM i developers.

Susan is also a partner in System i Developer, a consortium of IBM i educators who host events such as the RPG & DB2 Summit. Susan and her partner, Jon Paris, author regular technical articles for a number of online publishers.

Find a comprehensive collection of Jon & Susan's published works - including ITJungle, IBM Systems Mag, Tech Channel and others - here:

authory.com/JonParisAndSusanGantner

This presentation may contain small code examples that are furnished as simple examples to provide an illustration. These examples have not been thoroughly tested under all conditions. We therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All code examples contained herein are provided to you "as is". THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED.

© Copyright Partner400, 2023.

Attending Summit Deep Dive Workshops?

SUMMIT Deep Dive

Workshops that contain RPG code examples will use free-form

- The goal of this session is to enable you to understand those examples
  - Even if your experience is currently primarily (or solely) with fixed-format RPG

Workshops featuring RPG code examples are highlighted in red below

- Getting to a Modern Database on IBM i
- **RDi Quick Start for PDM Fans**
- **Data Structures & Arrays for Today's RPG Apps**
- **Intro to Open Source on IBM i**
- Foundations in SQL on IBM i
- **RPG Procedures & SRVPGMs by Example**
- Python for RPGers
- **API Enable Your Application With Web Services**
- **A to Z of Embedded SQL in RPG**
- **Is Code for IBM i Right for You?**
- PHP for RPGers
- **Coding a Modern RPG Application**
- **SQL Procedures, Functions and Triggers**

For details on all Deep Dive workshops

- Stay tuned at the end of this session
- systemideveloper.com/pages/events/workshops/

© Copyright Partner400, 2023.

Notes

For information about any of the Summit Deep Dive workshops or to register, visit the web site:

http://systemideveloper.com/pages/events/workshops/

© Copyright Partner400, 2023.

Agenda

SUMMIT Deep Dive

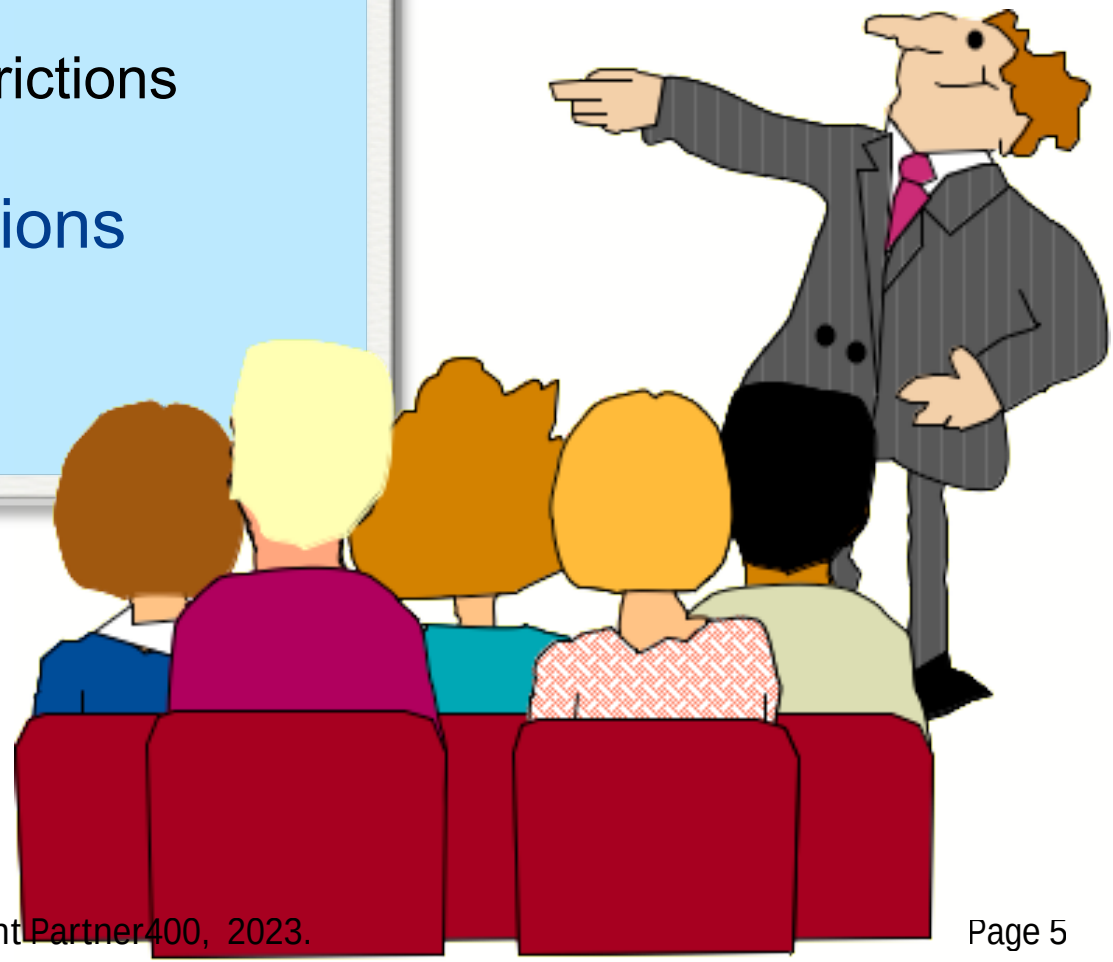
Why Free-form?

How Free do you want to go?

- Free-form declarations
- Free-form logic
- Freedom from all column restrictions

Rules & syntax for all 3 options

Tools to help you convert



© Copyright Partner400, 2023.

Notes

Free-form calculations in RPG IV first became available in V5R1 but it was not until V7.1 that data declarations (the former D-specs) and file declarations (F-specs) became available in free-form. Then in V7.2 (also supported in V7.1) we have the ability to do all the free-form coding without the 8-80 column restrictions.

© Copyright Partner400, 2023.

Why Free-form ?

SUMMIT Deep Dive

1. Much easier to read (and write!)

- Especially file declarations!
  - Keywords with intelligent default values
- Especially for developers experienced in other languages

2. Indented code is more understandable

- Match up nested If ... EndIf, DoW ... EndDo, etc.
- Nesting of levels in data structures

3. More efficient use of space in source code

- No more wasted space between C and Op Code column
- Complex expressions take fewer lines

4. More room for larger names

- More descriptive and/or qualified data names easier to code and use

5. Attraction of new RPG programmers

- Free-form looks and feels more like other modern languages
- Do you want your legacy to be discarded?
  - Free-form is more easily learned & maintained by RPG neophytes

© Copyright Partner400, 2023.

Notes

1) The V7 free format F and D spec replacements make data and file declarations far easier to read. No need to remember which characters go into which columns in the F spec. No more copying an F spec from another program because you can't be bothered to figure out which columns were which! An excellent set of default values means you rarely need to specify all the keywords.

2) When writing free format code, I can indent the statements to show the structure of the code. The code is more easily understood, especially in cases involving more complicated logic blocks, such as nested If statements, Select/When statements or Monitor blocks. The structure of data declarations - especially data structures - can also be made more obvious in free format. The easier the code is to understand, the faster and less error prone it will be to maintain.

3) I find that I seldom use Factor 1 in RPG IV. At the same time, because I am no longer limited to simple arithmetic operations, many expressions - especially those involving built-in functions - get quite lengthy, often requiring multiple lines to complete. Therefore with fixed format logic, the source edit screen typically has a huge amount of unused space on the left (i.e., where Factor 1 and the outdated left-hand conditioning and cycle control indicators would go) while the right side is complicated with many multi-line statements. Using free format logic, I have much more space for my expressions. My edit screens are "cleaner" and I can see more logic at a time because there are fewer multi-line statements.

4) There are no real length limitations for names when you can code declarations and logic in free format. I don't often set out to create names of 20+ characters. But it's nice not to have to think of an acceptable abbreviation for "ProcessCustomer" or "ErrorMsgDisplay", which are just slightly too long to fit into the old fixed format columns. Qualified data names allow powerful features such as data structure arrays and nested data structures, but could never have fitted into a 14 character Factor column.

5) Last, but not least, free format logic brings RPG more in line with other modern programming languages, all of which use free format logic. This is important for attracting new developers coming into the marketplace. Traditional fixed format RPG less attractive and gives RPG the undeserved appearance of an old-fashioned language not up to modern application tasks. RPG is a powerful and flexible language that many young developers come to prefer over other more popular language options for business applications. But they must first be attracted to learn the language.

© Copyright Partner400, 2023.

Old and New - Free Form Declarations

```
FCustMast  UF A E           K Disk
FReport    O    E             Printer OflInd(*IN96)
FScreen    CF   E             Workstn

D MyDS              DS
D  Address                       32A
D    Street                      15A   Overlay(Address)
D    City                        10A   Overlay(Address: *Next)
D    State                        2A   Overlay(Address: *Next)
D    Zip                          5A   Overlay(Address: *Next)
D  LstOrdDate                     D   DatFmt(*USA)
D  VaryingField                  25A   Varying
```

```
Dcl-F CustMast Usage(*Delete:*Output) Keyed;
Dcl-F Report   Printer OflInd(*IN96);
Dcl-F Screen   Workstn;

Dcl-DS MyDs;
 Address        Char(32);
    Street       Char(15) Overlay(Address);
    City         Char(10) Overlay(Address:*Next);
    State        Char(2)  Overlay(Address:*Next);
    Zip          Char(5)  Overlay(Address:*Next);
 LstOrdDate   Date(*USA);
 VaryingField Varchar(25);
End-DS;
```

© Copyright Partner400, 2023.

Notes

This small example of a comparison of fixed format RPG IV and Free-form format declarations illustrates some of the advantages. For example the omission of useless specifications. After all, when was the last time you defined an input capable printer!

© Copyright Partner400, 2023.

Old and New - Free Form Logic

```
C                   READ(E)   TransFile
C                   DOW       Not %EOF(TransFile)
C                   IF        %Error
C                   Eval      Msg = 'Read Failure'
C                   LEAVE
C                   ELSE
C     CustNo        CHAIN(N)  CustMast      CustData
C                   Eval      CustName = %Xlate(UpperCase:LowerCase
C                                        : CustName)
C                   EXSR      CalcDividend
C                   READ(E)   TransFile
C                   ENDIF
C                   ENDDO
```

```
READ(E) TransFile;
 DOW not %EOF(TransFile);
    IF %Error;
       Msg = 'Read Failure';
       LEAVE;
    ELSE;
       CHAIN(N) CustNo CustMast CustData;
       CustName = %Xlate(UpperCase : LowerCase : CustName);
       EXSR CalcDividend;
       READ(E) TransFile;
    ENDIF;
 ENDDO;
```

© Copyright Partner400, 2023.

Notes

This small example of a comparison of fixed format RPG IV and Free-form format logic illustrates some of the advantages such as the ability to nest the logic to match the conditioning. And of course no wasted space on the left-hand side.

© Copyright Partner400, 2023.

How Free is Free? A Quick History Lesson

SUMMIT Deep Dive

RPG IV was introduced in V3R1 (1994)

- Still mostly columnar except for extended Factor 2 in logic
  - Extended Factor 2 and built-in functions allowed for complex expressions
  - Extended columns allowed for somewhat longer names

Completely free format logic arrived in V5R1 (2001)

- Required /Free directive before all uses of free format
  - And /End-Free to insert fixed-format logic or declarations - e.g., P specs
- Allowed for indentation to illustrate nested logic
- Enabled some new features

Note: SEU editor does not support features added since V6!
Use a modern editor!

Free format declarations finally arrived in V7.1 (2012)

- Free format replacements for H, F, D and P specs
  - Indentation for structures and intelligent defaults ease coding
- But we were still limited to coding between columns 8 & 80

V7.2 (PTF'd to V7.1) did away with 80 column restrictions (2015)

- First statement must be **Free
- No fixed-format code allowed at all in the same source member
  - But can be added via /Copy

© Copyright Partner400, 2023.

RPG IV dipped its toe into free-format logic in the beginning with the semi-free-format expressions allowed with the extended factor. You may not know it, but fully free-form was part of the original design for RPG IV. But for a lot of reasons it didn't happen until much later in RPG IV's life. Jon Paris has written a series of blog posts on his perspective of this story and you can find links to all the articles in the series here:

https://authory.com/JonParisAndSusanGantner?collection=History-of-RPGIV

True free-format arrived in V5R1 - but only for logic statements. The /Free directive was required before any free format logic could be coded and the /End-free directive was required before anything that wasn't free-format was coded. This could include either fixed-format logic (perhaps because of using an op code that's not supported in free format) or for declaring procedures with the still fixed format P spec.

It was quite a long wait between free form logic and free form declarations, but they finally arrived in V7.1 (not at the beginning of V7.1 - the support came in at the same time as TR 4, in May 2012). This removed the requirement for /Free and /End-Free - you can still mix fixed and free format code, but no directive is required.

A few restrictions still existed with the 7.1 support - most notably that I and O specs are not replaced (but since they are rarely used in modern coding, it's rarely an issue) and the RPG code was still required to be between columns 8 and 80.

With the new support in V7.2 (and PTF'd back to V7.1) the 8 to 80 column restriction can also go away if you want. If the first statement in a source member is **Free, then the code can be in any column and can extend for as long as the source member or file allows. This support became available in November 2015.

One restriction with the **Free support should be noted that was not true for any of the earlier free-format releases - a source member that has **Free cannot contain any kind of non-free-format RPG code in it. If non-free-format RPG is required for the program, it can be /copied in from an external member at compile time. Effectively data and file declarations in each source member are required to be in a single format, either fixed or to begin with **Free.

RDi is my IDE/edior of choice for RPG coding, but it may not be right for everyone. Other options that support free form RPG include:

Liam Allan's plug-in to MS Visual Studio: Code for IBM i

https://www.seidengroup.com/2021/06/21/visual-studio-code-rpg-and-ibm-i/

MiWorkplace from Remain Software

https://remainsoftware.com/miworkplace-development-environment-ibm-i

IBM's newly announced Merlin tool also contains an editor that supports free-form RPG.

© Copyright Partner400, 2023.

We'll Cover the Features in Reverse

SUMMIT Deep Dive

1. ****Free format (no column restrictions) - V7.2 TR 3+ & V7.1 TR 11+**
   - Syntax rules
   - Overcoming the "no fixed format specs" restriction
2. **Free format declarations - V7.1 TR4+**
   - Ctl-Opt - the new H spec
   - Dcl-F - the new F spec
   - Dcl-xx - the new D spec
   - Dcl-Proc - the new P spec
3. **An overview of free format logic - pre-V7**
   - Syntax rules and op-codes not supported
     - We'll discuss replacing those unsupported op-codes later
   - New things you can do only with free-format logic
4. **Converting from fixed form to free form**
   - And tools to help

© Copyright Partner400, 2023.

Notes

We’ll be covering free format RPG syntax in reverse historical sequence. The newer features will be covered first and will be the primary focus of this session.

Since free form logic has been around for so long, we're hopeful that most people are at least somewhat familiar with that syntax. We'll cover those syntax rules as well but don't plan to spend as much time as the more recent free form enhancements.

During the syntax discussion, we’ll be assuming we’re talking mostly about writing new code in free format. Lastly we’ll talk about converting existing code from fixed format to free format, including some tools which you’ll want to look into using - because doing it manually is no fun at all!

© Copyright Partner400, 2023.

Want Completely Free Format RPG?

SUMMIT Deep Dive

If the first line of your source says **Free

- Beginning in line 1, column 1
  - Ironically, the only thing that MUST BE in a specific row & column
- Then all other source may be coded beginning in column 1 (or later)
  - And continue coding all the way to the end of the source line **
- Requires 7.2 TR3+ or 7.1 TR11+

If the first line of the source member does not begin **free

- The entire member is considered to be "column-limited"
  - i.e., only columns 8 to 80 may RPG contain code

** Source member line lengths are limited to 32,766 characters

- That's probably long enough!
- IFS files are unlimited in length

© Copyright Partner400, 2023.

Notes

Probably the most ironic thing about this support is the fact that the declaration of fully free-form code is the only thing that is strictly limited to a specific line & column!

Note that it's not just the first few columns on the left that can be used in these full free-form members. Code can span the entire length of the file where the source is - beyond column 80.

Source physical file members are limited to 32K length. But we seriously doubt any RPGer would be interested in source lines that long!

© Copyright Partner400, 2023.

**Free RPG

SUMMIT Deep Dive

**Free members may not contain ANY non-free-format statements

- If fixed-format code is needed, use /Copy to bring it in at compile time
  - Each /Copy member is assumed to be "column-limited" unless line 1 says **Free
  - /Include can be used in place of /Copy
- Note that Copy'd code must have **Free as well if fully free-form

/Free and /End-Free are not only not required - they are not allowed

SQL pre-compiler supports **Free

RDi 9.5 and later supports **Free

- Most other modern non-green-screen editors do, too
- But not SEU, of course

All my free-form code examples will work beginning in column 1 or 8, depending on whether **Free is coded

```
**free

Dcl-F InvMast;      // Free form file declare in Col 1 - No F specs allowed
Dcl-S array  Char(10) Dim(arrayMax);  // Also in col 1 - No D specs allowed

/copy OldRPGCode    // Older fixed-format code must be /copy'd in
```

© Copyright Partner400, 2023.

Notes

In fully free-form source, columns 6 and 7 have no special status. Therefore there can be no spec type specified in column 6 nor an * in column 7 to indicate a comment.

If it is necessary to have fixed-form statements (maybe Input or Output specs or - heaven forbid - TAG), these statements must be placed in a copy file.

The **Free status only applies to a single source file. Copy files are assumed to have column-limited source mode unless **Free appears in the first line of the file.

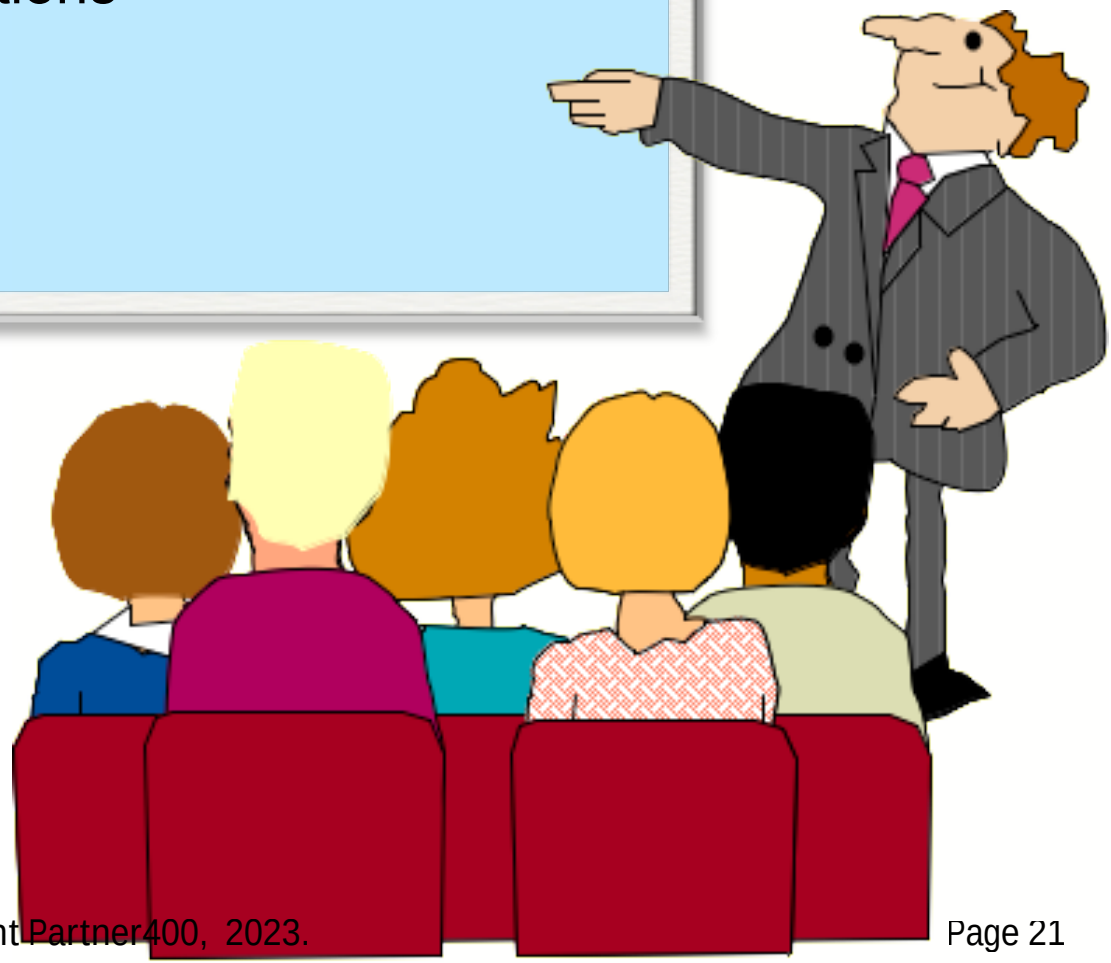
© Copyright Partner400, 2023.

V7+ Free-Form Declarations

SUMMIT Deep Dive

Now (almost) all RPG code can be free-format

- Control (aka H) specification
- File Declarations
- Data Declarations
- Procedure & Prototype declarations



© Copyright Partner400, 2023.

Notes

Beginning with V7.1, nearly all the RPG language can be coded in free format. The only exceptions are the I and O specs but since those are so rarely actually coded in RPG IV, that is not much of a limitation.

This section will describe the details of replacing control specs, file specs and data declarations specs with the new free format statements.

IMPORTANT NOTE: The SEU editor (aka the green screen editor accessed via PDM) does NOT support any of these new 7.1 free format versions of RPG. So you can code them using SEU but all the code will be indicated as in error because support for that syntax doesn't exist - and will not.

The editor in the RDi (Rational Developer for i) product and the new Merlin product recently announced are the only editors from IBM that support the V7+ RPG syntax. But other modern editors also support free form code - including the Code for IBM i plug-in to the Visual Studio Code IDE and the editor in the MiWorkplace IDE from Remain Software.

© Copyright Partner400, 2023.

V7 - Free Form Enhancements

SUMMIT Deep Dive

You can now specify an entire program in free form

- Only exception is that I & O specs are not available in free-form

No longer any need for /Free or /End-Free

- Just leave columns 6 and 7 blank

New free form options for:

- H-specs (CTL-OPT)
- F-specs (DCL-F)
- D-specs (DCL-xx)
  - Where xx = C, DS, PARM**, PI, PR, S, or SUBF**
    - ** These are rarely needed
- P-specs (DCL-PROC)

```
Ctl-opt option(*srcstmt) dftactgrp(*No);

Dcl-ds employeeDS;                          // Begin Data Structure
  firstName char(16)    Inz('James');       // Subfields
  lastname  char(30)    Inz('Joyce');
  salary    packed(7:2) Inz(12500);
End-ds;                                     // End Data Structure

Dcl-f qprint printer(80);  // Define PRTF and associated DS together
Dcl-ds prtDs len(80) end-ds;
```

© Copyright Partner400, 2023.

Notes

For many RPGers - especially those who use subprocedures - one of the best parts of this support is the ability to do away with /Free and /End-Free!

But the ability to actually code a file declaration without your secret decoder ring to tell you what character to put in what column is also very nice!

Another feature is illustrated in this example - the ability to intermix file declarations with "ordinary" (that is, D-spec-type) declarations. So files and standalone fields, data structures, etc may be in any order - but all must come before the logic.

© Copyright Partner400, 2023.

Format of the new Declarations

The declaration op-codes follow this basic format:

- First, the DCL-xx itself
- Next the name of the item
  - File, field, procedure, etc. (no name for Ctl-opt)
- Followed by keywords related to other fixed form elements
  - e.g. File usage, field type and length
- Then keywords from the old spec

We will look at a more complete code sample on the next chart

```
ctl-opt option(*srcstmt) dftactgrp(*No);

dcl-ds employeeDS;
  firstName char(16) Inz('James');
  lastname  char(30) Inz('Joyce');
  salary    packed(7:2) Inz(12500);
end-ds;

dcl-f qprint printer(80);

dcl-ds MyExtDs ExtName('MYFILE') end-ds;
```

From original D-spec keyword area

Fixed column entries

end-ds can be on the same line as dcl-ds in this case.

© Copyright Partner400, 2023.

Notes

When it comes to replacing D specs, DCL-xx includes:

DCL-S - Stand-alone field

DCL-DS - Data Structure (usually requires END-DS)

DCL-C - Named constant

DCL-PR - Prototype (usually requires END-PR)

DCL-PI - Procedure Interface (usually requires END-PROC)

Later, we'll see that we can also replace P specs with:

DCL-PROC - Define a procedure (requires END-PROC)

In all cases where an END-xx is allowed, the name of the DS/Prototype/Procedure may be included on the END-xx

© Copyright Partner400, 2023.

Simple Free Form Program

SUMMIT Deep Dive

File definitions can now be intermixed with data definitions

- Named Constants, Data Structures, etc.

Most new options have sensible defaults

- Disk files default to input, Printers to output, Decimals to zero, etc.

End-of-line comments are now useful in definitions!

```
ctl-opt option(*srcstmt) dftactgrp(*No);

dcl-ds employeeDS;  // Nice to be able to have comments here!
  firstName char(16) Inz('James');
  lastname  char(30) Inz('Joyce');
  salary    packed(7:2) Inz(12500);
end-ds;

// Define printer file and associated DS
dcl-f qprint printer(80);     // This PRTF is program described
dcl-ds prtDs len(80) end-ds;

dsply ('Hello to our new employee');
dsply ( %TrimR(firstName) + ' ' + lastName );

prtDs = 'The name of our new employee is ' +
         %TrimR(firstName) + ' ' + %TrimR(lastName) +
         ' his salary is $' + %Char(salary);
write qprint prtds;
```

Note that Printer file is defined along with the DS that is used for its output

© Copyright Partner400, 2023.

Notes

© Copyright Partner400, 2023.

CTL-OPT - The New H-Spec

SUMMIT Deep Dive

Defaults to DftActGrp(*No) if any ILE-specific options are used

- ACTGRP, BNDDIR, or STGMDL
  - This is the first of many "sensible" defaults that are added by this support

Other differences:

- Can start in any column from 8 onwards
  - Exception: In **Free members, can begin before column 8
- Must be terminated with a semi-colon

No other differences between this and the current H-spec

- Can occupy multiple lines
- Multiple CTL-OPT can appear in program
- Presence of CTL-OPT stops compiler from looking for H spec data area

```
H BndDir('UTILBNDDIR') Option(*SRCSTMT: *NODEBUGIO)

  Ctl-Opt BndDir('UTILBNDDIR');
  Ctl-Opt Option(*SRCSTMT: *NODEBUGIO);

   // Can also be coded as:

  Ctl-Opt BndDir('UTILBNDDIR') Option(*SRCSTMT: *NODEBUGIO);
```

© Copyright Partner400, 2023.

Notes

Not that much has changed about the H spec since it was almost free format before.

Ctl-Opt replaces the H and the semi-colon is used at the end.

Note that just as the H spec would prevent the compiler going to look for an H spec data area, so will the Ctl-Opt statement. If you just want to prevent the data area search but have no other options to specify, you could simply have an empty statement, such as:

```
Ctl-Opt;
```

With all the new free-format definitions and even logic (provided you're not in a **Free member with no column restrictions), you are allowed to intermix free-format with fixed format - although it usually produces an ugly mess, so it's best to avoid it.

Syntactically, the compiler would accept the following in a single member (but please don't do it!):

```
H option(*srcstmt : *nodebugio)
  ctl-opt datfmt(*YMD) nomain;
H timfmt(*usa)
```

© Copyright Partner400, 2023.

Declaring Files In Free-Form

SUMMIT Deep Dive

File Name listed first, followed by Device type keyword (if any)

- Device type defaults to DISK if omitted - i.e., a Database table

Externally described is the default

- File Keyword *EXT can optionally be specified as a parameter
- Program described files must specify the record length
  - e.g. PRINTER(132) for a program described printer file

No more copying F specs!

Sensible defaults for USAGE based on device type - more details coming

Add KEYED keyword for keyed database (disk) files

- Other F spec keywords may also be used

File name no longer limited to 10 characters

- EXTDESC used to specify actual name when needed

```
Dcl-F CustMaster Disk  Usage(*Update) Keyed;
Dcl-F Report    Printer(*EXT) Usage(*Output) OFLIND(*IN96);
Dcl-F qPrint    Printer(132) OflInd(PageFull); // Program described
Dcl-F MyDisplay WorkStn Usage(*Input:*Output);
Dcl-F InvMast;     // Defaults to Disk device, Input usage
Dcl-F InvoiceMaster ExtDesc('INVMAST') EXTFILE(*EXTDESC);  // Longer name
```

© Copyright Partner400, 2023.

Notes

In the example here, we have defined 3 files - one is a keyed database table (aka file), a report, and an interactive screen.

The example shows some ways of using Dcl-F to accomplish the same capability for the files that the fixed format F specs did. The second example of Dcl-F is a “minimal” example which takes advantage of all the defaults.

For example, **PRINTER(*EXT)** defining an externally described printer did not need the *EXT parameter since externally defined is always the default. Also the **USAGE(*Input:*Output)** on the Screen file is the default usage value for a display file and could have been omitted. Note that Usage(*Output) is implied by default for the Report file.

The OFLIND keyword on the Report file is “Overflow Indicator” which can be used by the program to determine when to go to a new page on the report.

The File name is no longer limited to 10 characters as it was on the F spec. However, since file object names (externally on the system) are limited to 10 characters, that means if a longer name is used, then keyword ExtDesc must be coded to give the real external name for the file. A longer, more meaningful name for the file may be useful for making the logic more readable.

There are other keywords that can be used when declaring files. These are the most commonly used features of file declarations.

© Copyright Partner400, 2023.

DCL-F : Very Helpful Defaults

SUMMIT Deep Dive

Device type implies default usage

- **Disk** device defaults to **Usage(*Input)**
- **Printer** device defaults to **Usage(*Output)**
- **WorkStn** device defaults to **Usage(*Input:*Output)**

Some Usage values imply others

- Usage values are: *Input, *Output, *Update, *Delete
  - **Usage(*Update)** implies ***Input**
    - Note: Unlike with with F spec, *Update does NOT imply *Delete
  - **Usage (*Delete)** implies both ***Input** and ***Update**

```
FCUSTMR0   UF A E           K DISK    USROPN
FREPORT    O    E             PRINTER OFLIND(*IN96)
FSCREEN    CF   E             WORKSTN

   Dcl-F Custmr0 DISK Usage(*Input:*Update:*Delete:*Output) Keyed UsrOpn;
   Dcl-F Custmr0 Usage(*Delete:*Output) Keyed UsrOpn;

   Dcl-F Report  Printer(*EXT) Usage(*Output) OFLIND(*IN96);
   Dcl-F Report  Printer OflInd(*IN96);

   Dcl-F Screen Workstn Usage(*Input:*Output);
   Dcl-F Screen Workstn;
```

© Copyright Partner400, 2023.

Notes

In the free-format version of the example, note that there are 2 versions of each file declaration. In each case, both versions have identical effect. Due to defaults and assumptions made by the compiler, the 2nd version of each is a "minimal" one, taking advantage of defaults and compiler implications.

DISK device type is the default if none is coded. Device type must be the first thing coded after the name of the file, with 1 exception. The exception is when the file is defined using the LIKEFILE keyword. In that case, the LIKEFILE must immediately follow the name.

The most commonly used devices are: DISK, PRINTER, and WORKSTN

There are less common device types: SPECIAL (where a user-created program handles the I/O operations for the file) and SEQ (where the file is sequentially organized - the actual device is specified in a CL command or in the file description).

It is important to note that in fixed form, Update implies that the files is also delete-capable. However, in free-form, *DELETE must be explicitly specified for the files to be delete-capable.

© Copyright Partner400, 2023.

DCL-xx - The New D-Spec

SUMMIT Deep Dive

xx = DS for Data structures

- In most cases there must also be a matching END-DS

xx = SUBF for DS subfields - Very Rarely Required

- Code only if field name is a valid free-form op-code
  - Yes, some strange people do use names like READ or SELECT as field names

xx = S for Stand-Alone fields

xx = C for Named Constants

```
D Address          DS                  Dim(20) Qualified
D  Street1                        30A
D  City                           30
D  State                           2
D  Zip                             5S 0
D  ZipPlus                         4S 0

  dcl-ds Address Dim(20) Qualified;
    Street1    char(30);
    City       char(30);
    State      char(2);
    Zip        zoned(5);  // Zero decimals assumed
    ZipPlus    zoned(4:0);
  end-ds Address;
```

DS Name is optional on end-ds but must match if present

© Copyright Partner400, 2023.

Notes

END-DS may be omitted if DS is externally described or has no named subfields. It can even be on the same line as the DCL-DS in such cases, such as the following DS which is externally described based on a table (aka file) named PRODUCT.

```
dcl-ds product  Ext end-ds;
```

Name of DS can be specified on end-ds and must match if present.

© Copyright Partner400, 2023.

DCL-xx - The New D-Spec - Data Types

SUMMIT Deep Dive

Data types are spelled out instead of using a one character code

- Chart below shows the most commonly used types
- Length or format of item follows in parentheses when required

Simpler definition for some data types

- VarChar avoids the need for the Varying keyword
- Date, Time, Timestamp - no need for separate DatFmt keyword
- 0 (zero) decimals assumed for all numeric data types if not specified.

Data Type	Free Keyword	Comment
A	Char(len)	
A + Varying	Varchar(len)	
I	Int(len)	Decimals not specified
U	Uns(len)	Decimals not specified
P	Packed(len:dec)	0 assumed if decimals not given
S	Zoned (len:dec)	0 assumed if decimals not given
N	Ind	
D + DatFmt	Date(format)	
B	BinDec (len:dec)	!! DO NOT USE !! (see notes)

See Notes for examples

© Copyright Partner400, 2023.

Notes

For an explanation on why we say "do not use" for the binary data type - whether in fixed format or free-form, check out this article The Difference between Binary and Integer (And Why You Should Care) https://techchannel.com/SMB/11/2017/rpg-binary-integer

The following is a partial list of RPG data types represented as D-specs:

```
d packedNum       s              7p 2
d zonedNum        s              7s 2
d integer         s             10i 0
d unsigned        s             10u 0
d float           s              8f
d character       s             20a
d varyingChar     s             20a   Varying
d dateMDY         s               d   DatFmt(*MDY)
d timeUSA         s               t   TimFmt(*USA)
d indicator       s               n
d nastybinary     s              9b 0
```

And here are their free form equivalents.

```
Dcl-S packedNum    Packed(7:2);
Dcl-S zonedNum     Zoned(7:2);
Dcl-S integer      Int(10);
Dcl-S unsigned     Uns(10);
Dcl-S float        Float(8);
Dcl-S character    Char(20);
Dcl-S varyingChar  Varchar(20);
Dcl-S dateMDY      Date(*MDY);
Dcl-S timeUSA      Time(*USA);
Dcl-S indicator    Ind;
Dcl-S nastybinary  Bindec(9);
```

© Copyright Partner400, 2023.

D-Spec DS Example

SUMMIT Deep Dive

This shows how a fixed form DS could be converted to free-form

- I have used my own personal preferences for alignment
  - At least they were my preferences at the time I coded this example!

```
D MyDS              DS
D  Address                          32A
D    Street                         15A   Overlay(Address)
D    City                           10A   Overlay(Address: *Next)
D    State                           2A   Overlay(Address: *Next)
D    Zip                             5A   Overlay(Address: *Next)
D  LstOrdDate                        D    DatFmt(*USA)
D  ASubfield                        25A   Varying

   Dcl-DS MyDs;
     Address      Char(32);
       Street       Char(15) Overlay(Address);
       City         Char(10) Overlay(Address:*Next);
       State        Char(2)  Overlay(Address:*Next);
       Zip          Char(5)  Overlay(Address:*Next);
     LstOrdDate   Date(*USA);
     ASubfield    Varchar(25);
   End-DS MyDs;
```

© Copyright Partner400, 2023.

Notes

I like the idea of aligning the field type definitions even though it is not required. I don't however insist on placing them in a specific column. Rather I start them 2 characters after the end of the longest field name in the block. For me it works but you'll devise your own style - just be consistent and remember that readability is critical.

© Copyright Partner400, 2023.

D-Spec - Unnamed Fields and POS

In fixed form no field name was required

- In free-form we must specifically inform the compiler
  - Done by coding *N instead of a field name

In free-form we are not permitted to OVERLAY the DS name

- So we must use POS(1) instead
  - POS(nn) can also be used for positioning any field within a DS
  - For example when specifying specific fields in the PSDS

```
Dcl-DS DayData;
  *n         Char(9) inz('Monday');    // *n = no name
  *n         Char(9) inz('Tuesday');
  *n         Char(9) inz('Wednesday');
  *n         Char(9) inz('Thursday');
  *n         Char(9) inz('Friday');
  *n         Char(9) inz('Saturday');
  *n         Char(9) inz('Sunday');

  DayArray   Char(9) Dim(7) Pos(1);

End-ds;
```

Fixed format version of this DS is shown in the notes

© Copyright Partner400, 2023.

Notes

The RPG compiler team decided that too many people got confused when the OVERLAY keyword was used against the DS name. To help avoid this confusion, OVERLAY is now limited to subfields within the DS and you must use the POS keyword to reference the DS starting position.

This is effectively the same as using a start position on the old D-specs.

This is what the DS in my example would have looked like in fixed-form. Another good use for POS is with Indicator Data Structures (INDDS).

```
D DayData         DS
D                                  9    Inz('Monday')
D                                  9    Inz('Tuesday')
D                                  9    Inz('Wednesday')
D                                  9    Inz('Thursday')
D                                  9    Inz('Friday')
D                                  9    Inz('Saturday')
D                                  9    Inz('Sunday')

D DayArray                         9    Overlay(DayData) Dim(7)
```

© Copyright Partner400, 2023.

2 Ways to Code a Nested DS

```
Dcl-Ds Divisions Qualified Dim(5);
   DivCode     Zoned(2);
   Departments LikeDS(DeptData) Dim(10);
End-Ds;

Dcl-Ds DeptData Qualified TEMPLATE;
   DeptCode    Zoned(3);
   Products    LikeDS(ProductData) Dim(99);
End-Ds;

Dcl-Ds ProductData Qualified TEMPLATE;
   ProdCode    Zoned(5);
   MonthsSales Packed(9:2) Dim(12);
End-Ds;
```

Option 1:
Use LikeDS

```
Dcl-Ds Divisions Qualified Dim(5);
   DivCode     Zoned(2);
   Dcl-DS Departments Dim(10);
      DeptCode    Zoned(3);
      Dcl-DS Products    Dim(99);
         ProdCode    Zoned(3);
         MonthsSales Packed(9:2) Dim(12);
      End-DS;
   End-DS;
End-Ds;
```

Option 2:
Inline Coding
Only in Free-form

Inline nested DS coding. Proof that the RFE process works!

© Copyright Partner400, 2023.

Notes

Nested Data Structures are not new for free form. However, there are 2 ways to code them in free-form.

The first way is just free-format version of what we did with the D specs using the LikeDS keyword. That's shown at the top of this chart.

The second way is a more direct form of nesting - where you actually code the nested DS inline and omit the LikeDS.

This second form was implemented as an RFE - Request for Enhancement - on IBM's developerWorks site. So when you come up with good ideas for enhancements to RPG or most any other IBM software product, submit it to the RFE site and get others to vote for it. It just may be implemented like this one was.

© Copyright Partner400, 2023.

More Fun with Data Definitions

SUMMIT Deep Dive

Constants can be used in many more places

- Including field length, decimal places, array dimensions - just about anywhere you would use a literal

Long names don't require ellipsis … or a continuation line

```
 D DateMDY          S                 D   DatFmt(*MDY-)
 D array            S               10a   dim(10)
 D CustomerInfo     DS
 D   CustomerName                   50A   Varying
 D   CustomerBalance...
 D                                   7P 2


dcl-s DateMDY  Date(*MDY-);
dcl-s array    Char(10) Dim(arrayMax);
dcl-c  Digits 7;
dcl-c  Decimals 2;
dcl-ds CustomerInfo;
  CustomerName     VarChar(50);
  CustomerBalance  Packed( Digits: Decimals );
end-ds;
```

© Copyright Partner400, 2023.

Notes

Note that in the fixed format example here, it made no sense to define Digits and Decimals as constants since they could not be used to define the length and precision in fixed format D specs. However, in the new free format D specs, they can be used - as illustrated in the 2nd example.

Note that not only are the ellipsis not required for longer variable names - they are not allowed in most cases. Unless the actual variable name is continued on the next line, ellipsis should not be used, even if the data type and other related keywords are on the next line. If ellipsis are found, the compiler assumes the next line will contain part of the variable name.

© Copyright Partner400, 2023.

Subprocedures, Prototypes and More

DCL-PROC declares a subprocedure

- Completed by an END-PROC {name}
  - Name is optional but if included must match DCL-PROC name

DCL-PI and DCL-PR define procedure interfaces and prototypes

- Placeholder *N can be used if you don't want to type the name again in PI
  - Or for unnamed parameters in a prototype
- DCL-PARM within PIs & PRs is like DCL-SUBF within DS
  - Only needed if name of parameter is the same as an RPG op-code
  - Such as a parameter named SELECT or CHAIN, as in
  - Dcl-Parm SELECT Char(2);

```
dcl-proc DayOfWeek Export;

dcl-pi *N Int(3);                  // To omit name, use *N placeholder
  InputDate Date(*USA) Const;
end-pi;

  dcl-s DayNumber int(3);

  // Do calcs populating value for DayNumber

  Return DayNumber;

end-proc DayOfWeek;
```

© Copyright Partner400, 2023.

Notes

Prototypes can be used when calling anything - a program, whether RPGLE or RPG or CLP or CLLE, etc. or a subprocedure or a C function, a system API - anything.

Prototypes were added to the RPG IV language in the V3R2 and V3R6 releases. Although their initial purpose was to support Subprocedures, the RPG developers realized that they could be used to provide additional support for program calls among other things. This helps to address one of the really annoying problems with the old CALL/PARM syntax. Namely that errors with parameter lists are not discovered until run-time. It would be much better if we could have the compiler validate the parameter lists for us, and that's what prototyping provides.

In addition to validating parameters, prototyped calls allow for automatic adjustment in the case of certain parameter mismatches. For example they can allow you to pass a 7 digit packed field when a 9 digit zoned is expected. The techniques involved are beyond the scope of this session but some references to more information about prototypes are included below.

A great many people mistakenly believe that prototypes and CALLP relate only to procedures. This is not true. CALLP can invoke either a program or a procedure. As we shall see shortly, which type of call is made depends on keywords on the PR line of the prototype.

If you need to know more about using prototypes, take a look at these articles:

RPG IV Prototypes: What are they and why should you care? - Susan Gantner

authory.com/JonParisAndSusanGantner/RPG-IV-Prototypes-What-are-they-and-why-should-you-care

Parameters and Prototypes - Scott Klement

http://www.scottklement.com/presentations/Parameters%20And%20Prototypes.pdf

The Geezer's Guide To Free-Form RPG, Part 4: Prototypes and Procedure Interfaces

https://www.itjungle.com/2014/07/09/fhg070914-story01/

© Copyright Partner400, 2023.

Prototypes (PR) and Procedure Interfaces (PI)

SUMMIT Deep Dive

Program name can be omitted from EXTPGM

- If the program name is the same as the prototype name
- The name, if specified, name must be upper case and in quotes

New option for EXTPROC - *DCLCASE

- Indicates the external name is exactly the same, including case
  - Avoids need to retype name when using mixed case procedure names
    - such as with mixed-case APIs or C functions

The PRs & PIs here are incomplete - no parms and missing End-xx

```
dcl-pr QCmdExc ExtPgm ; // Can be used to call 'QCMDEXC' program

dcl-pr QCmdExc ExtPgm('QCMDEXC') ; // Can also call 'QCMDEXC' program

dcl-pr MyProgram ExtPgm; // Can be used to Call 'MYPROGRAM'

dcl-pr DifferentName Extpgm('MYPROGRAM'); // Call 'MYPROGRAM' using the
                                          //   name DifferentName

dcl-pi cos float(8)  ExtProc(*DclCase);  // Calls C function 'cos'

dcl-pi cosine float(8)  ExtProc('cos');  // Also calls C function 'cos'
```

© Copyright Partner400, 2023.

Notes

Note that the *DCLCASE option is only available for EXTPROC, not for EXTPGM - for hopefully obvious reasons!

As the note indicates on the examples, these are all examples of just the first part of the declaration of PRs or PIs. Obviously most PRs and PIs have parameters and all of them require an END-PR or END-PI as well. We just didn't have enough room on the chart to show all the variations we wanted to illustrate if we had to include parms and END-xx as well.

If there are no parms - just a return value - then it is possible to include the END-PR or END-PI at the end of the DCL-PR or DCL-PI, such as the one below. But using a separate End-Pr; statement would also be valid.

```
Dcl-Pr  getCustName  Char(25)  End-Pr;
```

© Copyright Partner400, 2023.

"Gotchas" to Watch Out For

SUMMIT Deep Dive

OVERLAY keyword cannot be used against DS name

- Use POS(n) instead

Names in EXTNAME, EXTFLD, and DTAARA

- Must be in quotes and are case sensitive
- Without quotes, they are treated as variable or constant names

Ellipsis (...) for continuation only allowed when continuing a name

- But almost never needed anymore anyway

On F-Spec "U" enables both update and delete

- In free form *DELETE must be requested explicitly

End-DS, End-PR, End-PI are (almost) always required

- But may appear on same line as DCL-xx in some cases

RDi's "Convert all to Free-Form" means only convert "all logic"

- And will still generate /Free and /End-Free
  - With fixed form logic in between

I and O specs remain in fixed form

© Copyright Partner400, 2023.

Notes

Here are few things that have caught us and a few others using the new free form declarations by surprise.

If you know of or find additional gotchas, please let us know!

© Copyright Partner400, 2023.

High Points for Free-Form Declarations

SUMMIT Deep Dive

/Free and /End-Free no longer required

Intelligent defaults

- In file declarations
  - For device type (Defaults to DISK)
  - Usage (Default based on device type)
- In data declarations
  - Decimal places (Defaults to zero)

Check out content assist (Ctrl+Space) in RDi for help with remembering data type names, etc.

File and data declarations can be mixed

- Even in fixed form

Constants can be used as keyword values

- Including for lengths and decimal places

/Copy and other compiler directives no longer need to start in col 7

// style end of line comments can be used in data/file declarations

DFTACTGRP(*NO) is optional

- If any other ILE keyword such as ACTGRP or BNDDIR is used

© Copyright Partner400, 2023.

Notes

They had me at "no more /Free and /End-Free".

But then it's really nice to be able to actually read (and understand) a file declaration. And I love not having to copy an F spec from another program because I could never remember what things went in which columns.

And the intelligent defaults - such as no longer having to specify that I'm not planning on reading anything from my printer file!

© Copyright Partner400, 2023.

Free Form Logic - V5R1 +

SUMMIT Deep Dive

Completely Free-Form Logic

- Freeform specs originally had to be preceded by a /FREE directive
  - /END-FREE was used to return to fixed format
- As of V7.1 - simply leave columns 6 and 7 blank
- You can mix Free-form and Fixed .... but please don't do it

Syntax: Op-Code{(Extender)} { Factor1 } { Factor2 } { Result Field };

- Operands no longer limited to 14 characters
- Each statement must end with a semicolon (;)
- Not all Fixed form Op-codes are supported



Two Op Codes are (mostly) optional: Eval and CallP

```
/Free
   If CustomerNumber = *Blanks;
      Eval CustomerError = True;
    Else;
      CustomerError = False;
   EndIf;
/End-Free
```

/Free & /End-Free not needed since V7

© Copyright Partner400, 2023.

Notes

Although freeform can be mixed with traditional C specs, the result is code that is ugly and hard to maintain. Avoid it like the plague!

Wherever possible you should replace operations that do not have a direct freeform equivalent with options that will work in free form instead. The degree to which such replacement is "possible" depends on which release of the operating system you are writing for. We will be discussing the replacement options later in the session.

One of the really nice aspects of using Free-form form calcs is that you can indent your source statements to highlight the logic flow as shown in this short example.

Note the use of the semicolon (;) to specify the end of the statement. As you will soon discover, if you forget it, you will get some really interesting error messages!

As shown in the example, some op-codes can be omitted completely. In the example we have omitted the EVAL op-code In the assignment CustomerError = False. Since the EVAL is optional, either option will work. Later we will look at the exception to this rule.

By the way - the only "Convert to Free-form form" option supplied by IBM is provided within the RDi editor. BUT - that conversion is a very rudimentary one and will only convert logic - no conversion of H, F, D and P specs. Other software vendors, such as Linoma (Help Systems) and Arcad Software offer much more functional conversion options for a fee. Craig Rutledge offers an open source (free) tool which converts to free format, including declarations.

© Copyright Partner400, 2023.

Free-form Logic Rules - Pre V7

SUMMIT Deep Dive

Opcode & operands can appear anywhere in columns 8 - 80

- Positions 6 & 7 must be blank

Comments are specified by the traditional * in column 7

- Or by // in column 8 onwards
  - These // Style comments can also be placed at the end of any free form line

There is no way to define fields in Free-form form logic

- But then you wouldn't want to do such a naughty thing anyway!

Resulting indicators may not be specified

- BIFs such as %FOUND, %EOF, %ERROR are be used instead

Embedded SQL was enabled in Free-form in V5R4

```
// I can have a comment like this anywhere

EXEC SQL SELECT  NAME, POS
     INTO   :Name, :Job FROM  EMPL
     WHERE   NBR = :EmpNbr;   // Or at the end of a line
```

© Copyright Partner400, 2023.

Notes

If you are not in a position to move to V5's Free-form format yet, there are a number of steps you can take to prepare yourself for the change. These are all good programming style anyway so you can't lose!

Use only those opcodes supported by freeform

Defining all variables on D-Specs

Do NOT use conditioning indicators

Avoid resulting indicators

Note that beginning with V5R4, SQL can be embedded in Free-form RPG logic. Prior to V5R4, many SQL users placed their SQL statements in either subroutines or subprocedures in order to avoid the need for /End-Free to do SQL statements (which themselves were also free format!).

When embedding SQL statements in Free-form RPG logic, the End-Exec directive is not required. And the Exec SQL directive is no longer a "traditional" compiler directive, which would require a slash (/) in position 7 of the spec. Now, the only requirements are simply the words "Exec SQL" at the beginning of the statement and the semicolon (;) at the end of the SQL statement, just as all other Free-form logic statements end.

© Copyright Partner400, 2023.

Free-form Logic Example (V7.1+)

SUMMIT Deep Dive

Note that subroutines can be defined and invoked in Free-form

- And as of V7.1 /Free and /End-Free are no longer required

```
READ(E) TransFile;          // Read file to prime do loop

DOW not %EOF(TransFile);    // Continue until everything processed
  IF %Error;
    DSPLY 'The read failed';
    LEAVE;
  ELSE;
    CHAIN(N) CustNo CustMast CustData;
    CustName = %Xlate(Upper : Lower : CustName);
    EXSR CalcDividend;
    READ(E) TransFile;
  ENDIF;
ENDDO;

BEGSR CalcDividend;
  TotalSales = %XFoot(MthSales);
  EVAL(H) Dividend = TotalSales / 100 * DivPerc;
  Record_transaction();
ENDSR;
```

Notice the use of // for end of line comments

Note:
EVAL is required here because of the Extender

© Copyright Partner400, 2023.

Notes

Note again the benefit of indentation of the source. With the source coded this way, it is very clear which statements belong to the If block and the Else block. And it is clear that that the IF/ELSE block is all part of the DOW.

Notice that we must use the %EOF and %Error built-in functions because resulting indicators cannot be used in freeform calcs. That's OK, because even in fixed format source, the use of the BIFs makes the code far more readable and understandable.

Notice that in this code sample, the EVAL operation code has been omitted - except where it was necessary to include it because of the half adjust extender.

Now for a tough question: Assuming these calcs are syntactically correct (i.e. the program containing these calcs will compile), what could Record_transaction() be? Is it an array element? A subprocedure call? A program call?

It is either an subprocedure call or a program call (or, to be more specific, it is the name on the prototype given to a subprocedure or program to be called.) The CALLP (call with prototype) operation code, like the EVAL, can be omitted, as it is in this case.

You see another change in syntax for Version 5 in that same line, because prior to V5, if the procedure or program required no parameters to be passed, we could NOT have coded the empty parentheses. It is a good idea to use the empty parentheses notation so the name is not mistaken for a field name. In a free form spec, it is required to use the empty parentheses when no parameters are being passed.

© Copyright Partner400, 2023.

Obsolete Op-Codes & Replacements (1 of 2)

Op-Code	Free-form Substitute
ADD	Operator +
ADDDUR	Operator +
ALLOC	BIF %ALLOC
ANDxx	Logical operator AND
BITxx	(Bitwise operations in V5R2)
CABxx	(see GOTO)
CALL	Op-code CALLP
CALLB	Op-code CALLP
CASxx	Op-code IF with EXSR
CAT	Operator +
CHECK	BIF %CHECK
CHECKR	BIF %CHECKR
COMP	Operators =, <, >, etc.
DEFINE	LIKE or DTAARA on D-Spec
DIV	Operator / or BIF %DIV

Op-Code	Free-form Substitute
DOUxx	Opcode DOU
DOWxx	Opcode DOW
END	Opcodes ENDDO, ENDIF, etc.
EXTRCT	BIF %EXTRACT
GOTO	LEAVE, ITER, LEAVESR, etc.
IFxx	IF
KFLD	(See KLIST)
KLIST	(Various options in V5R2)
LOOKUP	%LOOKUPxx or %TLOOKUPxx
MxxZO	(Bitwise operations in V5R2)
MOVE	EVALR or %DATE, %TIME, etc.
MOVEA	%SubArr in V5R3 or …..
MOVEL	EVAL or BIFs for Date, Time etc.
MULT	Operator *
MVR	BIF %REM

© Copyright Partner400, 2023.

Obsolete Op-Codes & Replacements (2 of 2)

Op-Code	Free-form Substitute
OCCUR	BIF %OCCUR
ORxx	Operator OR
PARM	(see PLIST)
PLIST	D-Spec PI & PR definitions
REALLOC	BIF %REALLOC
SCAN	BIF %SCAN
SETON	EVAL *Inxx = *ON
SETOFF	EVAL *Inxx = *OFF
SHTDN	BIF %SHTDN
SQRT	BIF %SQRT
SUB	Operator -
SUBDUR	Operator - or BIF %DIFF

Op-Code	Free-form Substitute
SUBST	BIF %SUBST
TAG	(see GOTO)
TESTB	(Bitwise ops in V5R2)
TESTN	(Bitwise ops in V5R2)
TESTZ	(Bitwise ops in V5R2)
TIME	Time & Timestamp BIFs
WHENxx	Opcode WHEN
XFOOT	BIF %XFOOT
XLATE	BIF %XLATE
Z-ADD	Opcode EVAL
Z-SUB	Opcode EVAL

© Copyright Partner400, 2023.

Op-codes not supported in Free-form Logic

SUMMIT Deep Dive

These fall into 5 main categories:

1) "Old fashioned" Op-codes whose use is discouraged

- e.g. ANDxx, COMPxx, DEFINE, DOxyy, GOTO, and IFxx

2) Those for which new support has been provided

- e.g. ALLOC, CHECK, EXTRCT, LOOKUP, TIME, XLATE, KFLD, KLIST

3) Op-codes with existing (if not identical) expression support

- e.g. ADD, DIV, MULT and SUB

4) Those supported by a combination of enhanced and new support

- e.g. ADDDUR and SUBDUR

5) The ones the compiler writers don't like: MOVE, MOVEL, MOVEA

- We will talk more about Why they don't like it in a few minutes
- EVAL is the primary alternative but does not meet all needs

Many of these are good reasons to use automated conversion tools discussed later

© Copyright Partner400, 2023.

Notes

Over the next few charts we will study the substitution options for some of the op codes that are not supported in Free-form format.

Note that the Op-code substitutions are not necessarily one-to-one. The substitute might not work exactly as the original code, especially in the area of error handling. For example the default for numeric overflow on an ADD operation is to truncate the result and ignore the error. The default for an addition in an EVAL type operation is to blow up! You need to bear this in mind if you are converting. Personally we would rather know if numeric overflow occurs, which is why we have always preferred using EVAL to the older style operations.

© Copyright Partner400, 2023.

What's So Bad About MOVE ?

SUMMIT Deep Dive

It tells you (almost) nothing about what is happening

For example - what does this code do?

```
C                   MOVE      ARCode         PrARCode
```

Does it ... ?

- Simply copy the data - byte for byte?
- Change the data type?
  - And if so what other effect did that have?
- Truncate the data?
- Effectively concatenate two fields?
- Discard high-order digits without warning ?

And how about this ?

```
C     *MDY        MOVE      InvDate        WorkDate
```

- Which is the "real" date field (i.e. the D data type) ?
- Which one is in MDY format ?

© Copyright Partner400, 2023.

Notes

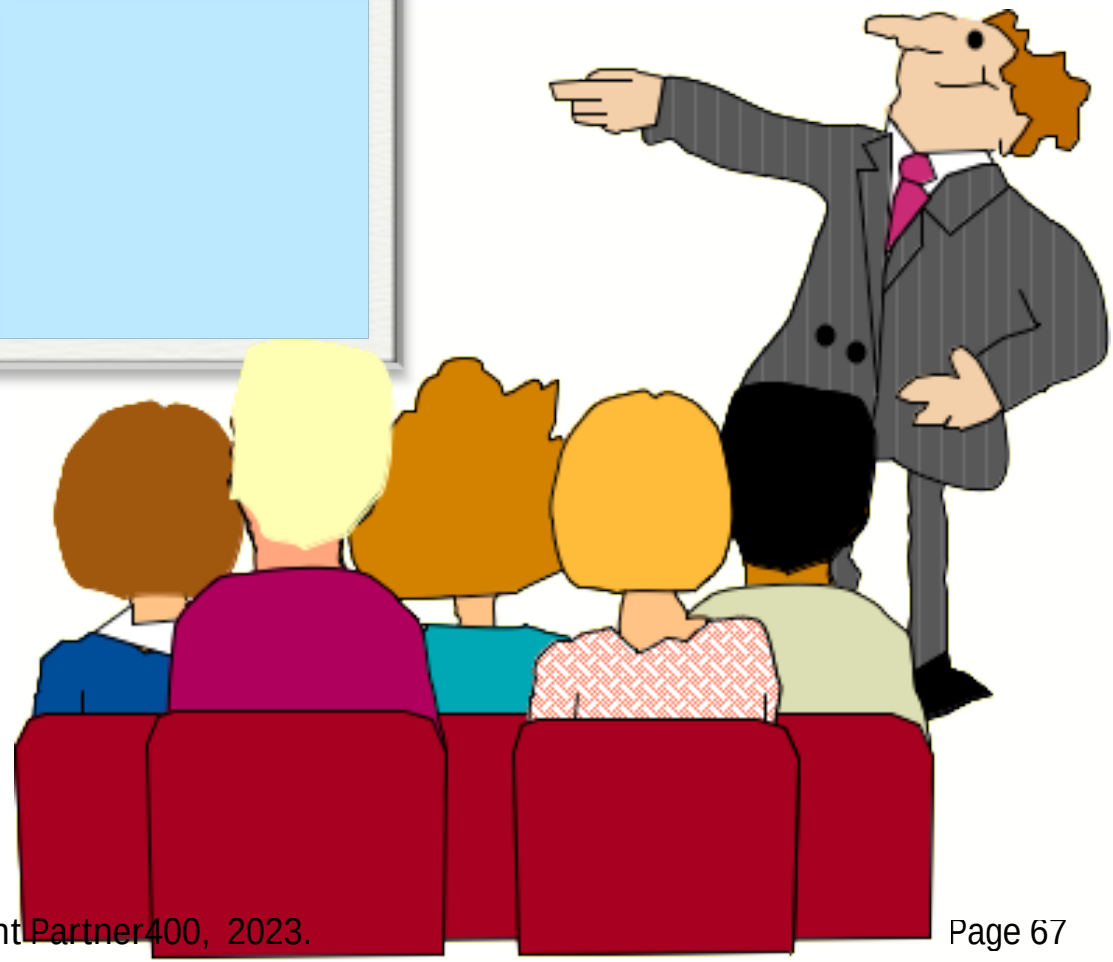
For many people the surprise omission from the list of op-codes supported in Free-form was MOVE. There were also a few people bent out of shape over the omission of GOTO - but that has been accepted as poor programming practice for so many years that they received little support from their fellow programmers.

MOVE on the other hand caused a lot more controversy. As a result, Internet flame wars raged on and off for several years on this topic with the pro and con MOVE camps fairly equally divided. However, IBM stuck by their guns and now some years later still show no sign of changing their minds. We tend to lean towards the "MOVE is evil" camp for reasons that we hope will be apparent from the chart above.

© Copyright Partner400, 2023.

Now Appearing in Free-form Logic Only

New options only available
in Free-form format logic



© Copyright Partner400, 2023.

Notes

© Copyright Partner400, 2023.

I/O Enhancements - New Key Options (1 of 2)

SUMMIT Deep Dive

%KDS is a free-form replacement for KLIST

- Used in Factor 1 position instead of a KLIST name

Syntax:

%KDS(keyDSName { : numberOfKeysToUse })

- Optional 2nd parameter specifies # of fields to be used in partial key

All keyed operations can use this new BIF

- Generate related DS by using LIKEREC(recname : *KEY)

```
// The ProductKeys DS will contain the three fields that make up the
//   key for the Product File (Division, PartNumber and Version)
 dcl-DS ProductKeys LikeRec(ProductRec : *Key);

  // Read specific record using full key
 Chain %KDS(ProductKeys) ProductRec;

  // Position file based on first 2 key fields
 SetLL %KDS(ProductKeys : 2 ) ProductRec;
```

© Copyright Partner400, 2023.

Notes

Until this feature was introduced, there were two ways to specify the key for a keyed operation. Specify the name of a single field or the name of a KLIST. KLISTs always annoyed me because you had to wander off elsewhere in the program to actually find the list. Only then did you know what keys were being used. This new support offers both an improved alternative to the KLIST approach and a new method of directly specifying the keys on the operation itself.

The new "KLIST" (actually a BIF called %KDS - Key Data Structure) references key definitions in the D specs where they belong. Remember the LIKEREC *KEY option we covered earlier? This is where it comes into play. You can use it to automatically generate a DS containing the file's key fields. This structure can then be referenced in the I/O operation by specifying the DS name to the new %KDS function.

So how do you specify that a partial key is to be used? Just use the second parameter of %KDS to tell the compiler how many of the key fields are to be used.

We will look at the second method on the next chart.

© Copyright Partner400, 2023.

I/O Enhancements - New Key Options (2 of 2)

SUMMIT Deep Dive

Keys can now also be specified as a list of values

- List is specified in Factor 1 position, enclosed in parentheses

Any type of field, literal or expression can be used

- As long as the base type matches AND the key is in parentheses
  - i.e. The result is numeric for a numeric key, Alpha for an alpha key, etc.
- The compiler will perform any required conversions
  - Using the same rules as for EVAL operations - much like CONST does for parameters
- Note that the absence of parentheses changes the behavior
  - See example below

```
// Read using specified keys
 Chain (Division : PartNumber : Version) ProductRec;

// Position file based on first 2 key fields
 SetLL (Division : PartNumber) ProductRec;

// Position file to specified Part number in Division 99
 SetLL ( '99' : PartNumber) ProductRec;

 Chain Field Record;   // requires an exact match of attributes for Field
 Chain (Field) Record; // only requires a matching base type
```

© Copyright Partner400, 2023.

Notes

The second method is an extension of the current ability to specify a single field as the key (the old Factor 1).

Instead of a single field, you can now supply a list of fields. The list should be specified within parentheses with colons (:) used to separate the individual key elements.

Note that the key elements do not have to be fields, they can be any character expression. The compiler will perform conversion if required.

Note that even if you have a single field key, it is often better to enclose it in parentheses because with the parentheses the field specified is not required to match exactly in type and size, but without parentheses they must match. This is much like the size & type accommodation made by the compiler with the use of the CONST keyword in prototypes for parameters.

© Copyright Partner400, 2023.

I/O Enhancements - %FIELDS

SUMMIT Deep Dive

Allows you to specify a list of fields to be affected by an UPDATE

- Only those fields specified will be updated

Syntax:

%FIELDS(name1 { : namen ... })

Used in the Result position of the op-code

```
Dcl-F Product Disk  Usage(*Update) Keyed;
Dcl-DS ProductData LikeRec(ProductRec : *All) end-DS;

// Update all fields in the record using values in ProductData DS
 Update Product ProductData;

// Update only UnitCost and UnitPrice fields
 Update Product %Fields( UnitCost : UnitPrice );
```

© Copyright Partner400, 2023.

Notes

In many ways this is one of the best of the new features, and we saved it for the end (almost)

This is the capability to limit which fields are modified by an UPDATE operation. We love this one! It provides a great way to protect your code from the worst efforts of (shall we say) less-gifted programmers. The list of fields is specified using the new BIF %Fields. Only those fields specified will be updated.

Why is this useful? Suppose that, during the operation of the program, only certain fields in the file should be subject to change. By specifying those fields to the UPDATE op code, you are assured that only those fields will be changed. If during subsequent maintenance tasks a mistake is made and the value of a field that should not be modified is accidentally changed in the code, it will have no effect on the database. Only if the %Fields list is also modified can this error result in database corruption.

© Copyright Partner400, 2023.

Short-form Expression Support

SUMMIT Deep Dive

We suspect you'll either love this or hate it

- It is kinda nice when using long or subscripted or qualified names
  - Particularly if you are a slow typist !!

It allows shortening expressions such as X = X + 1 to X += 1

- Instead of repeating the field name (X) you can use short-forms
  - += for addition
  - -= for subtraction
  - and you can guess what the others are ...

This is not limited to Free-form

```
// This calculation
     MonthTotal = MonthTotal + TotalSale;
// can be shortened to this
     MonthTotal += TotalSale;
// and this
     x = x + 1;
// can similarly be shortened to this
     x += 1;

// Be careful! The following is not the same
     x =+ 1;    // This is the same as x = 1;
```

Not strictly a Free-form feature but is more often used in free-form logic

© Copyright Partner400, 2023.

Notes

Numeric operations now support short-form notation for certain functions. Prior to this release, an addition of the type X = X + 1 required that you repeat the name of the target field. Some people considered this a step backwards since the old ADD op-code offered a short-form notation that only required the target field to be specified once, in the result field. e.g. ADD 1 X.

With this new feature, the expression can be written as X += 1. Similar shorthand can be used for subtraction, multiplication, division, and exponentiation.

Do not make the mistake of coding it like this: X =+ 1 This simply puts a positive value of 1 into X.

© Copyright Partner400, 2023.

Tools for Converting Old to New

SUMMIT Deep Dive

Built-in RDi Option (Source menu) - "Convert All to Free-Form"

- It doesn't "Convert All" - not even close
  - Only logic - no declarations - and only the "easy" conversions are done
  - There are much more complete options available

Help Systems RPG Toolbox (aka Linoma Software's RPG Toolbox)

- The latest evolution of the first commercial RPG III conversion utility
  - Includes conversion to Free-form logic, declarations, etc.
    - Also converts from RPG/400 all the way to free-form if you choose
  - Command interface + RDi plug-in - allows for partial "selection conversion" in RDi

Arcad Software's ARCAD Transformer RPG

- Converts source members to Free-form - including declarations
  - Command interface + RDi plug-in
  - Even converts some GoTo logic!

Craig Rutledge's open source tools

- JCRCmds free toolset contains some utilities related to Free-form
- RDi Plug-in available from a third party

RPGLE Free - Brian J Garland

- Extension to Visual Studio Code - works with Code for IBM i

Links for info on 3rd party tools on a later chart

Merlin from IBM

- Newly announced IDE that contains free-form conversion facilities

© Copyright Partner400, 2023.

Notes

Don't waste a lot of time with IBM's RDi free-form convertor. It was never intended to be a serious "contender" in this arena. It works on a line-by-line basis without any reference to the data types/lengths of the fields being used. As a result it reverts to fixed form for all MOVEs - even if the two fields involved are identical in size and type. This makes for really ugly code. If all of your code is in EVAL style - no MOVEs etc. - then it does an OK job. However, if you are doing a major modernization effort on RPG III style code don't bother with it - just go for one of the alternatives.

Speaking of the alternatives. Craig's tools are free - the only cost is your time to build and modify them. It can be more cumbersome to use. RPGLEFree is also free - but it must be plugged into a VS Code environment, which relatively few RPG shops are currently using (so far). Linoma comes next and many shops already have their RPG toolbox, but it is inexpensive even if you don't. Also, adding their low-cost RDi component is a really easy addition to justify. Arcad's tooling, while the most expensive, may be the most comprehensive. In fact if you give it a chance it can even rid your code of pesky GOTOs!

IBM's newly announced Merlin product also contains free format conversion capability. We can't speak to that tool at all since we haven't seen details or worked with it.

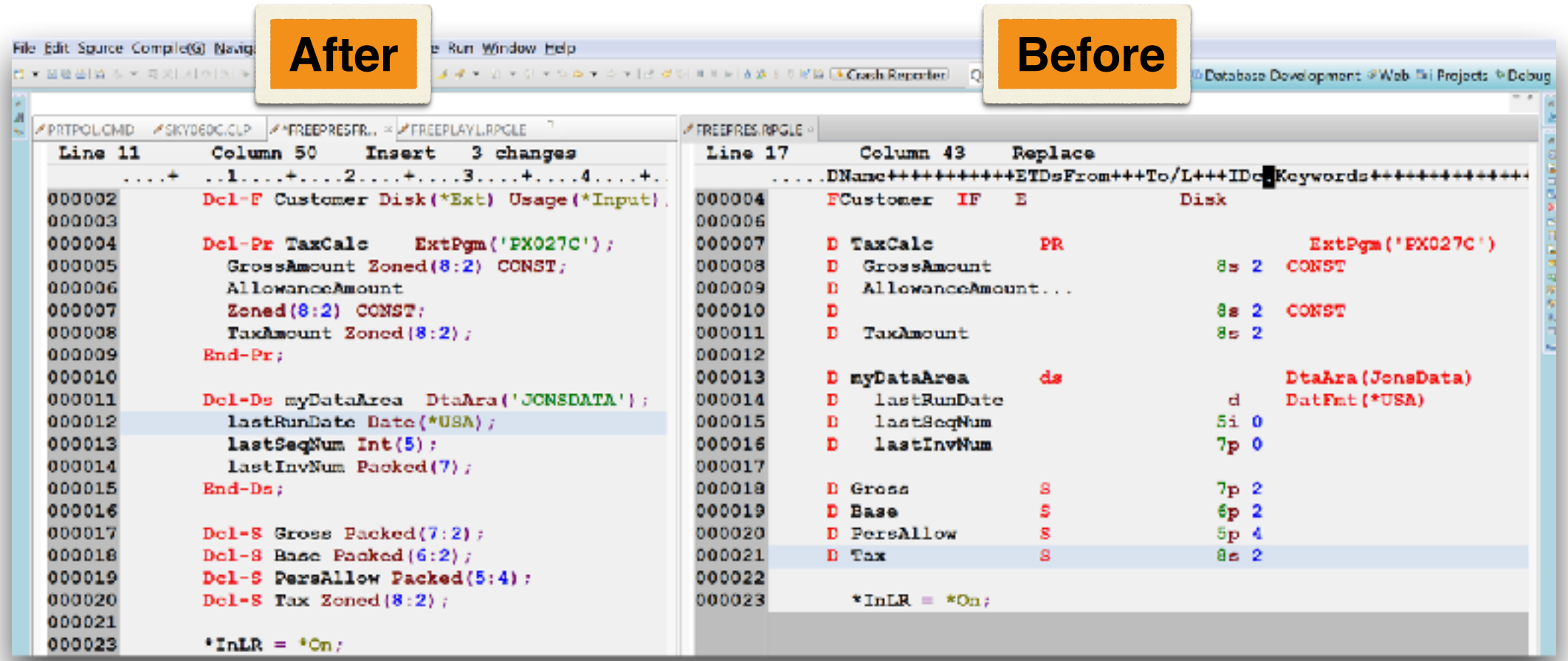
© Copyright Partner400, 2023.

RPG ToolBox - Side By Side Comparison

SUMMIT Deep Dive

Help Systems RPG Toolbox tool does a great job but ...

- It tends to play it safe and includes a lot of extra keywords
- I'm also not a fan of the way it aligns things - it doesn't!
  - But they are looking at changing this and providing more formatting options



© Copyright Partner400, 2023.

Notes

My biggest grouse with the Linoma tool was that it converts everything and that means it also adds a bunch of file declaration keywords that are not necessary. The first thing I do after running a conversion with the tool is to manually clean it up.

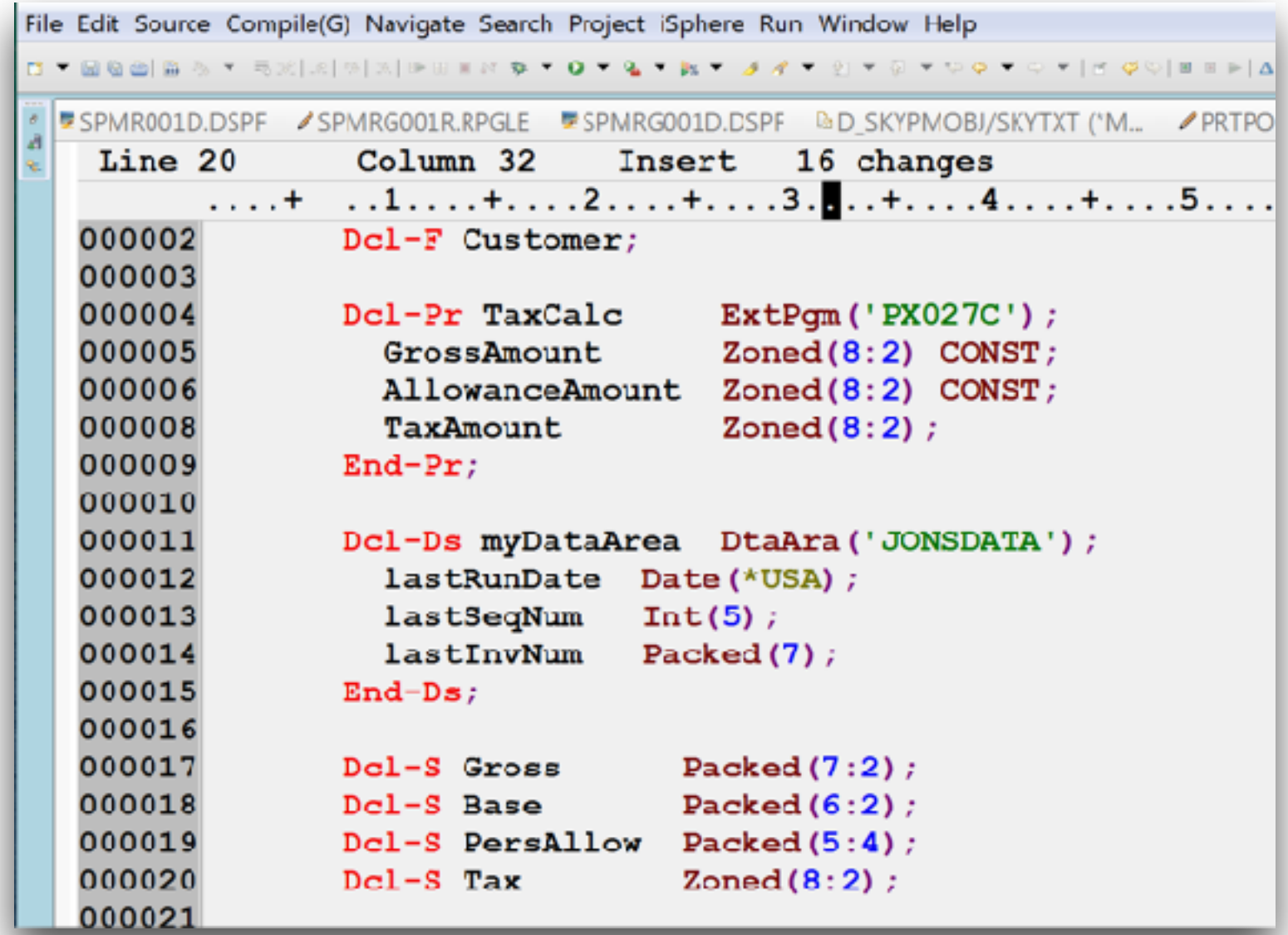
Their conversion of D-specs does not apply any alignment to the resulting code - just inserts a single blank between the components. For me the result looks messy so again I apply some manual clean up after conversion. An example on the next chart.

© Copyright Partner400, 2023.

Edited Version of Converted Code

SUMMIT Deep Dive

I removed the superfluous file entries and applied my own alignment style to the data declarations.



© Copyright Partner400, 2023.

Notes

Here you can see that I have removed the superfluous keywords from the file declaration. I have also added some alignment to the data declarations.

Right now my basic alignment rule is that in any given group (i.e. DS, PR, PI, series of standalone fields, etc.) I insert two spaces between the longest data name in the group and its data type/length definition and then align all other entries in the group to that. You can see the effect in the chart.

Note that I don't try and make all the entries in all the groups align - that is just too much work and I don't find it necessary - but visually I do like the entries in a specific group to align.

Choose your own style - but try to make sure that everyone in the shop uses the same (or very similar) style.

© Copyright Partner400, 2023.

Time to Update Your Editor

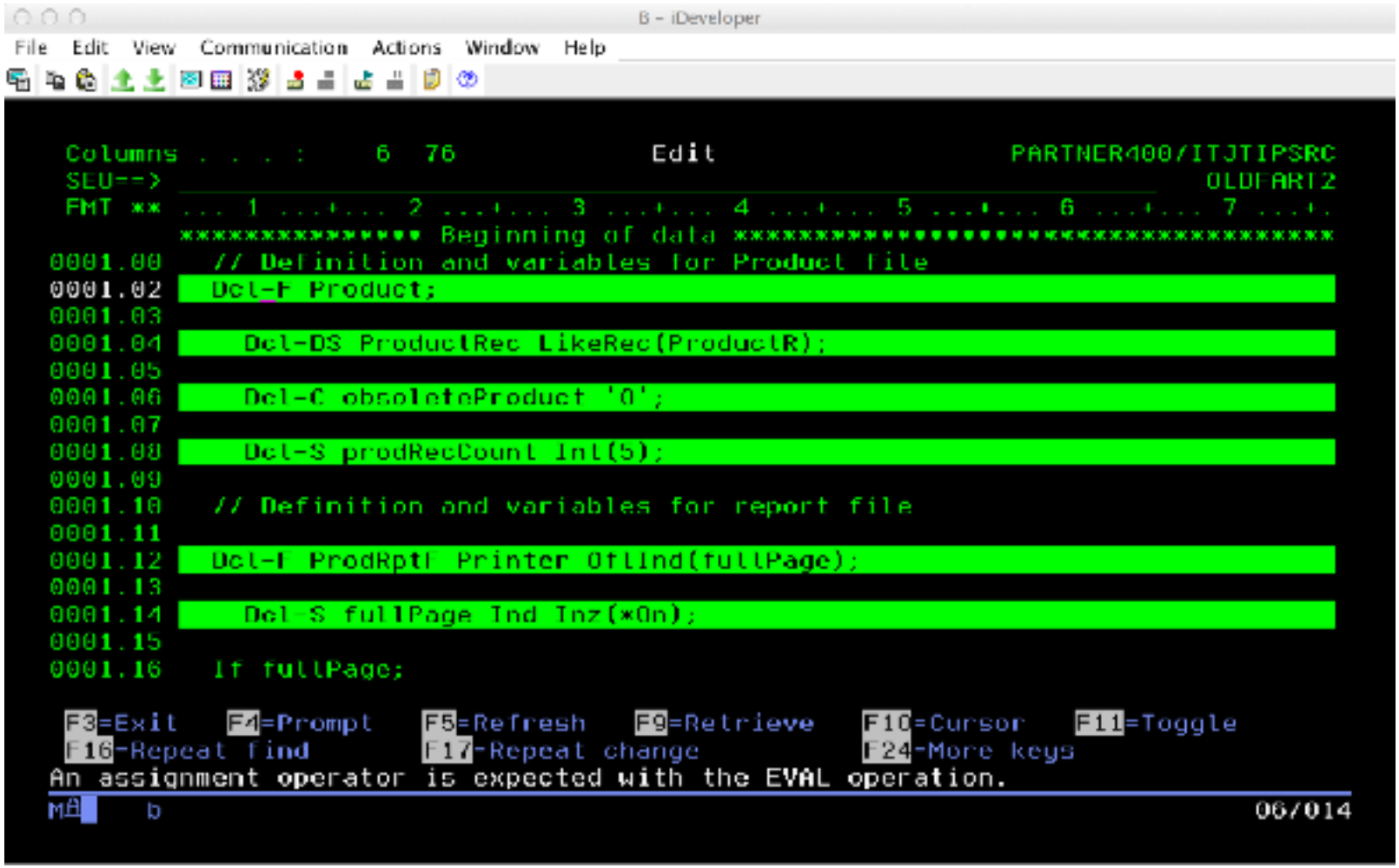
This is what free-form coding looks like in SEU

- IBM will NOT be enhancing SEU to handle these new definitions

Time you moved up to a modern editor - my preference is RDi

- or if not RDi, another modern editor, such as VS Code for IBM i, MiWorkplace or Merlin

See Notes page in PDF handout for more on this



© Copyright Partner400, 2023.

Notes

IBM has confirmed that SEU will not be updated to support this new functionality. In fact it was frozen as at V6 and supports none of the V7 functionality.

As a result a great many people have started doing what they should have done many years ago and moved their development to RDi. It fully supports the new free-form including excellent code-assist to help you remember the new formats.

RDi is my IDE of choice for RPG coding, but it may not be right for everyone. Other options that support free form RPG include:

Liam Allan's plug-in to MS Visual Studio: Code for IBM i

https://www.seidengroup.com/2021/06/21/visual-studio-code-rpg-and-ibm-i/

MiWorkplace from Remain Software

https://remainsoftware.com/miworkplace-development-environment-ibm-i

IBM's newly announced Merlin IDE also supports free-format RPG.

If you'd like to learn more about why RDi is my IDE of choice, here are a few of my ramblings on the subject:

Why RPG Developers are Adopting RDi

https://techchannel.com/SMB/02/2019/rpg-developers-adopt-rdi

A video on the same topic above: https://www.youtube.com/watch?v=-I-qrByv0zY

Using Modern IBM i Tools e-book - my chapter on 5 Reasons Susan Gantner loves RDi

https://techchannel.com/Admin/IBM/media/PDFs/TechChannel_02ebook_FINAL2.pdf

And > 100 articles on a wide variety of RDi topics can be found here:

https://authory.com/JonParisAndSusanGantner?collection=RDi&view=text

© Copyright Partner400, 2023.

Learn more ...

SUMMIT Deep Dive

Jon's 5-part Geezer's Guide to Free-Form RPG series in IT Jungle

- Covers free-form declarations in detail
  - authory.com/JonParisAndSusanGantner?collection=Geezers-Guide-to-Free-Form-RPG&view=text

How to Bring Your RPG Programs Into the 21st Century

- By Jim Martin - Get 3rd Edition or later for free format declarations
- Published by MCPress (www.MC-store.com)

Help Systems (formerly Linoma Software) RPG Toolbox

- Command interface + RDi plug-in
- www.helpsystems.com/products/rpgtoolbox-ibmi

Arcad Software's ARCAD Transformer RPG

- Command interface + RDi plug-in
- arcadsoftware.com/resource-items/arcad-transformer-rpg

Craig Rutledge's open source tools

- JCRCmds toolset includes free form conversion
- Green screen only but an RDi plug-in is available
  - RDi plug-in is only for free-form declarations (not logic)
- www.jcrcmds.com/ - see section "Free H F D"

RPGLE Free - Brian J Garland

- Extension to Visual Studio Code - works with Code for IBM i
- Part of IBM i Development Pack
- marketplace.visualstudio.com/items?itemName=HalcyonTechLtd.ibm-i-development-pack

© Copyright Partner400, 2023.

Thanks for Joining Me

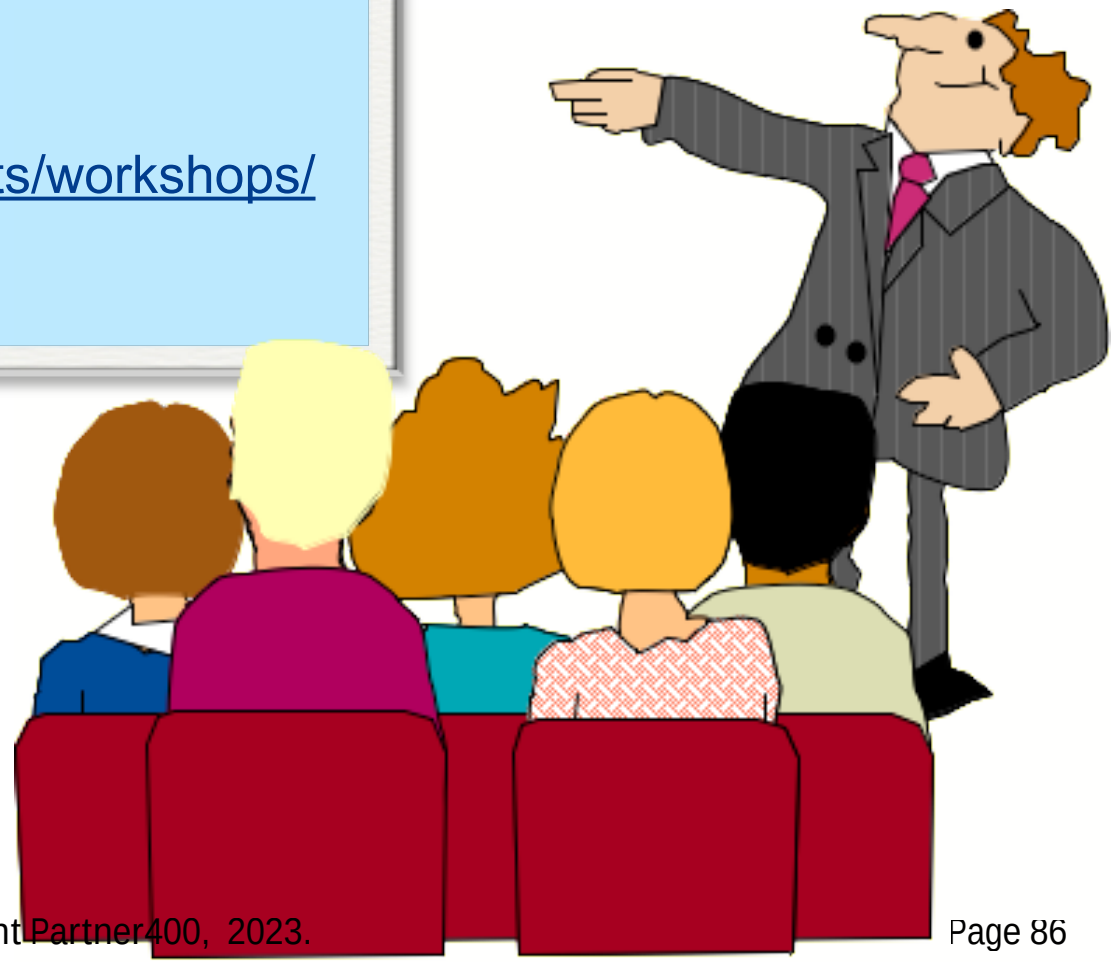
Hope to see you in a Summit Deep Dive workshop!

systemideveloper.com/pages/events/workshops/

Stay tuned for info on other skills needed for today's applications & how our workshops can help

Extra handout materials:

Details on replacing some obsolete logic op-codes



© Copyright Partner400, 2023.

Essential Skills for Today's RPG Developers

Jon Paris

Jon @ SystemiDeveloper.com

Susan Gantner

Susan @ SystemiDeveloper.com



© Copyright Partner400, 2023.

Free-form RPG is Just the Beginning

SUMMIT Deep Dive

Today's applications require many more skills

- Modern RPG practices
- Database & SQL
- Powerful IDEs
- Open Source

Summit Deep Dive Workshops can help

© Copyright Partner400, 2023.

Modern RPG Practices

SUMMIT Deep Dive

Free-form is the foundation

Move towards an API-based architecture

- Procedures & Service Programs are essential components

Upgrade your RPG data handling skills

- Defining and using Dynamic arrays, DS arrays, Templates, ...
  - You'll need this for handling JSON, XML, etc.

Utilize the power of SQL

- Replacing SETLL/READ/CHAIN loops with embedded SQL

© Copyright Partner400, 2023.

Modern RPG Practices

SUMMIT Deep Dive

Free-form is the foundation

Move towards an API-based architecture

- Procedures & Service Programs are essential components

Upgrade your RPG data handling skills

- Defining and using Dynamic arrays, DS arrays, Templates, ...
  - You'll need this for handling JSON, XML, etc.

Utilize the power of SQL

- Replacing SETLL/READ/CHAIN loops with embedded SQL

Summit Deep Dive Workshops

- Data Structures & Arrays for Today's RPG Apps (Jon Paris)
- RPG Procedures & SRVPGMs by Example (Susan Gantner)
- API Enable Your Application With Web Services (Jon Paris & Paul Tuohy)
- A to Z of Embedded SQL in RPG (Paul Tuohy)
- Coding a Modern RPG Application (Paul Tuohy)

© Copyright Partner400, 2023.

Database & SQL Skills

SUMMIT Deep Dive

Essential SQL Skills

- The SQL Language
- Utilizing SQL within RPG programs

A Modern Database

- Exploits Tables, Constraints, Indexes, Views, and more
- Migrates definitions from DDS to DDL
- Includes logic callable from SQL

© Copyright Partner400, 2023.

Database & SQL Skills

SUMMIT Deep Dive

Essential SQL Skills

- The SQL Language
- Utilizing SQL within RPG programs

A Modern Database

- Exploits Tables, Constraints, Indexes, Views, and more
- Migrates definitions from DDS to DDL
- Includes logic callable from SQL

Summit Deep Dive Workshops (all led by Paul Tuohy)

- Getting to a Modern Database on IBM i
- Foundations in SQL on IBM i
- A to Z of Embedded SQL in RPG
- SQL Procedures, Functions and Triggers

© Copyright Partner400, 2023.

Powerful IDEs

SUMMIT Deep Dive

Green screens are inadequate for editing modern code

- And SEU has not been updated in 15 years!
- New developer recruits will expect better

Luckily we have excellent alternatives customized for IBM i

- RDi & Code for IBM i

© Copyright Partner400, 2023.

Powerful IDEs

SUMMIT Deep Dive

Green screens are inadequate for editing modern code

- And SEU has not been updated in 15 years!
- New developer recruits will expect better

Luckily we have excellent alternatives customized for IBM i

- RDi & Code for IBM i

Summit Deep Dive Workshops

- RDi Quick Start for PDM Fans (Susan Gantner)
- Is Code for IBM i Right for You? (Jon Paris & Susan Gantner)

© Copyright Partner400, 2023.

Open Source Skills

SUMMIT Deep Dive

Learn a New Language

- Even if there is no obvious immediate need
- It will teach you to write better RPG code!

Use it to help modernize your IBM i apps

- New programmers can help address your skill shortages

Use Open Source packages to fill gaps in your software inventory

© Copyright Partner400, 2023.

Open Source Skills

SUMMIT Deep Dive

Learn a New Language

- Even if there is no obvious immediate need
- It will teach you to write better RPG code!

Use it to help modernize your IBM i apps

- New programmers can help address your skill shortages

Use Open Source packages to fill gaps in your software inventory

Relevant Workshops - All led by Mike Pavlak

- Intro to Open Source on IBM i
- Python for RPGers
- PHP for RPGers

© Copyright Partner400, 2023.

Join Us

Enhance your skills quickly

- 1/2 day of in-depth online learning
- Interactive, live instruction
  - Recording will also be available to registrants
- Real-time Q&A
- Easy to schedule around daily work
- Lots of code examples
- Quantity/Group discounts
  - Apply to the total number of workshops attended by an individual and/or a group



SystemiDeveloper.com/pages/events/workshops/

© Copyright Partner400, 2023.

Bonus Free-Form Reference Materials: Replacing Obsolete Op-Codes

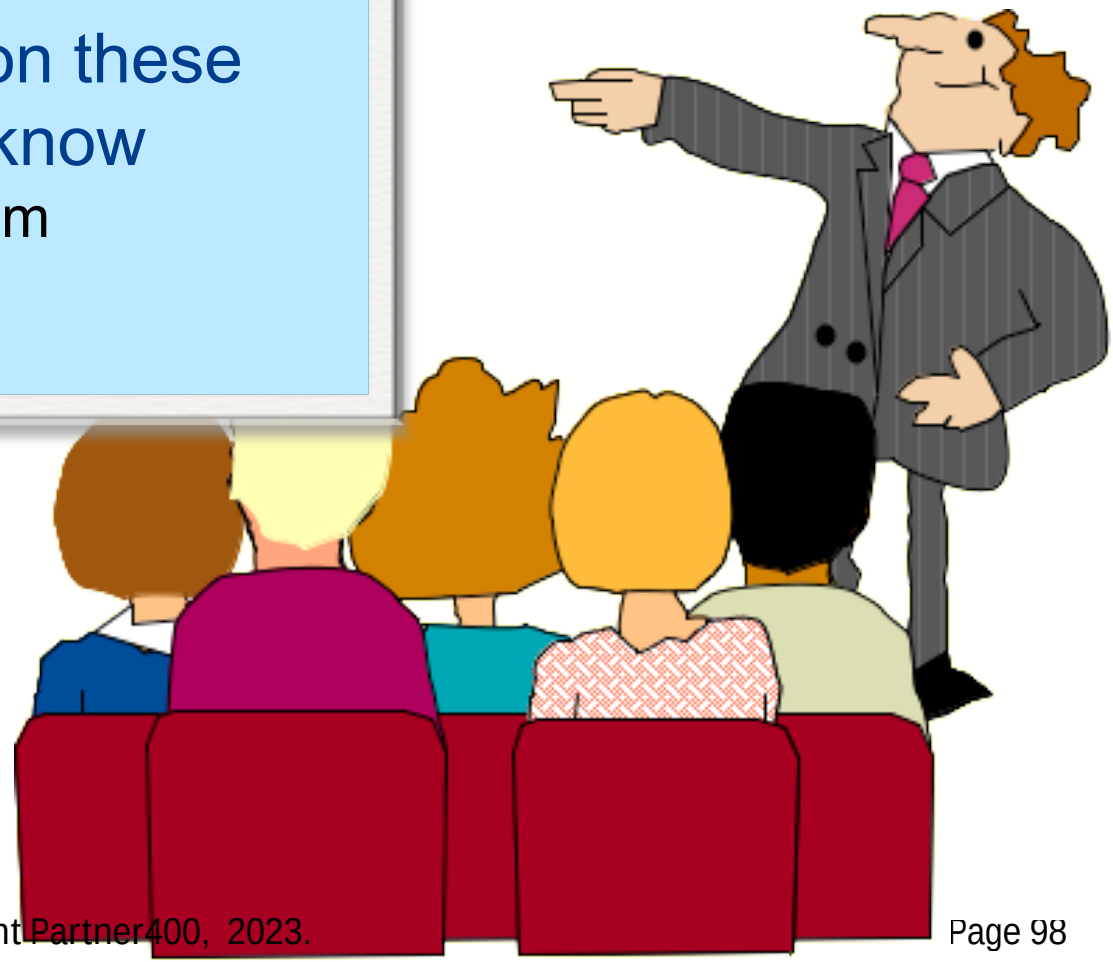
SUMMIT Deep Dive

I will not take time to cover this now

I hope the following charts will be useful to any of you who need it as reference material

If you have any questions on these extra charts please let me know

Susan @ SystemiDeveloper.com



© Copyright Partner400, 2023.

Notes

© Copyright Partner400, 2023.

Replacing Call - CallP and Prototypes

SUMMIT Deep Dive

The modern way to call anything

- e.g. Programs, procedures or subprocedures

Parameter mismatches can become a thing of the past!

- The compiler will validate your program calls
  - It checks the number of parameters passed
  - Plus their data type and size
- Also allows the compiler to "Help" you when things don't quite match
  - More on this aspect later

The op-code CALLP is used with Prototypes

- It means CALL using Prototype (not Call Procedure)

```
Dcl-Pr TaxCalc  ExtPgm('PX027C');
   GrossAmount        Zoned(8:2);
   AllowanceAmount    Zoned(8:2);
   TaxAmount          Zoned(8:2);
End-Pr;

CALLP TaxCalc( GrossPay : Allowances : TaxDue );
```

Note:
Parameter names on prototypes are used purely for documentation.

Prototypes do NOT define data

© Copyright Partner400, 2023.

Notes

Prototypes can be used when calling anything - a program, whether RPGLE or RPG or CLP or CLLE, etc. or a subprocedure or a C function, a system API - anything.

Prototypes were added to the RPG IV language in the V3R2 and V3R6 releases. Although their initial purpose was to support Subprocedures, the RPG developers realized that they could be used to provide additional support for program calls among other things. This helps to address one of the really annoying problems with the old CALL/PARM syntax. Namely that errors with parameter lists are not discovered until run-time. It would be much better if we could have the compiler validate the parameter lists for us, and that's what prototyping provides.

A great many people mistakenly believe that prototypes and CALLP relate only to procedures. This is not true. CALLP can invoke either a program or a procedure. As we shall see later, which type of call is made depends on keywords on the PR line of the prototype.

If you need to know more about using prototypes, take a look at these articles:

RPG IV Prototypes: What are they and why should you care? - Susan Gantner

authory.com/JonParisAndSusanGantner/RPG-IV-Prototypes-What-are-they-and-why-should-you-care

Parameters and Prototypes - Scott Klement

www.scottklement.com/presentations/Parameters%20And%20Prototypes.pdf

The Geezer's Guide To Free-Form RPG, Part 4: Prototypes and Procedure Interfaces

www.itjungle.com/2014/07/09/fhg070914-story01/

© Copyright Partner400, 2023.

Replacing *ENTRY PLIST

SUMMIT Deep Dive

A Procedure Interface (PI) replaces the *ENTRY PLIST

- It is almost identical to the prototype

Unlike the prototype a PI DOES define the items referenced

- In other words GrossAmount, AllowanceAmount and TaxAmount are all defined by this PI.

Procedure Interfaces are also used when coding subprocedures

- More on this later

```
Dcl-PI TaxCalc  ExtPgm('PX027C');
   GrossAmount         Zoned(8:2);
   AllowanceAmount     Zoned(8:2);
   TaxAmount           Zoned(8:2);
End-PI;
```

© Copyright Partner400, 2023.

Notes

Most RPGers seem to pick up on prototyping and procedure interfaces fairy quickly. Perhaps because they are essential for writing subprocedures.

For some reason though, the idea of replacing *ENTRY PLISTS with procedure interfaces seems to cause more of a mental hiccup. But it really shouldn't.

As you will see when you gain more experience with prototypes and procedure interfaces, they are so much more versatile that the old CALL/PARM and *ENTRY PLISTs could ever be.

© Copyright Partner400, 2023.

Replacing MOVE - Like to Like

SUMMIT Deep Dive

If fields are the same length

- Replacement is EVAL

If both fields are character but of different lengths

- Replacement is (if blank padding is OK):
  - EVAL for MOVEL
  - EVALR for MOVE
- However, if padding is not desired
  - %Replace and/or %Subst will be needed
  - Or use a Data Structure

If both fields are numeric but of different length / decimals

- Replacement is EVAL but ...
  - An error will occur in the event of overflow
  - MOVE would have ignored it

© Copyright Partner400, 2023.

Notes

Handling for character to character moves depends on the length of both the from and to fields.

Very often you will see code that moves blanks to a field and then moves another value in. Sometimes this is done even when the two fields are both character fields and the same length! At one point in time they might have been different lengths - but nobody removed the blank fill when the field length changed.

In fact the "P" extender (i.e. MOVE(P) or MOVEL(P)) to cause target fields to be blank filled has been around for some time and most blank fills could have been removed years ago!

It might not have occurred to you to use %SUBST to replace MOVE / MOVEL because most people don't realize that it can be used as part of the result field definition. So in a situation where MOVE is being used to replace the first three characters of a ten character filed, switching to EVAL and coding the result field as %SUBST(targetname: 1: 3) would have the same effect. It will also be much more obvious what is going on when yourself or others look at the program in future!

© Copyright Partner400, 2023.

Replace MOVE - Character to Numeric

SUMMIT Deep Dive

Use %DEC (expression { : precision : decimals })

- Expression can be character or numeric
- This capability also applies to %FLOAT, %INT, and %UNS
- As well as the Half adjust variants (%DECH, %INTH, and %UNSH)

Exception (Status 105) is signalled if character field not valid

- Use MONITOR to catch this error
  - See Rules and comments on "clean up" on Notes page

```
Dcl-S  CharField  Char(12) Inz(' -12345.678');
Dcl-S  NumField   Packed(9:2)          ;
Dcl-C  Length     %LEN(NumField);
Dcl-C  Decimals   %DECPOS(NumField);


NumField = %DecH( CharField : 9 : 2 );
// This version "self adjusts" if NumField's definition changes
NumField = %DecH( CharField: Length: Decimals);
```

© Copyright Partner400, 2023.

Notes

Rules for format of character string:

- Sign is optional. It can be '+' or '-', leading or trailing
- The decimal point is optional
- Blanks are allowed anywhere in the data.
- For %DEC(H) both the Precision and Decimal Place parameters are compulsory
- Floating point data (e.g. '2.5E4') is not allowed

In the example below, the assignment at <A> will work correctly. <B> though will cause an exception (Status code 105) because of the dollar ($) sign. IBM suggests a nice technique to handle this situation, as demonstrated at <C>. Simply put, the %XLATE BIF is applied to the character expression, substituting blanks for all unwanted characters (our example only replaces $ and , (comma). The resulting blanks will be ignored while processing the expression.

This deals with most common situations, but will not remove all sources of error. (e.g. there might be two decimal points in the field). One way to validate the character string is to use the subprocedure ChkNbr, which is part of the CGIDEV2 library available free from IBM. Go to www-922.ibm.com

If you do not wish to validate every character field before converting it, "Wrap" the operation in a MONITOR group (another wonderful V5R1 innovation) and catch the errors if and when they happen.

```
          Dcl-S CharField1      Char(14) Inz('      1525.95-');
          Dcl-S CharField2      Char(14) Inz('    $1,525.95-');
          Dcl-S NumField        Packed(9:2);
          Dcl-C NumberEdits '$,';

<A>       NumField = %Dec(CharField1 : 9 : 2);
<B>       NumField = %Dec(CharField2 : 9 : 2);
<C>       NumField = %Dec( %Xlate('$,' : '  ' : CharField2) : 9 : 2);
```

© Copyright Partner400, 2023.

Replace MOVE - Numeric to Character

SUMMIT Deep Dive

Basic option - no decimal places

- First choice is to use the BIF %CHAR
  - But only if it's OK to suppress any leading zeros
- Alternative is to use %EDITC with the 'X' edit code
  - Retains leading zeros after conversion

If the numeric field contains decimal places

- The best bet is to use %EDITC with the 'X' edit code
  - And take the time to study exactly what you are trying to achieve

The BIFs will produce a character string

- Once the BIF has been selected and you know the length of the result apply the same rules as for Character to Character

```
Dcl-s  NumField   Packed(7)  Inz(12345);
Dcl-s  CharField  Char(7);

charField = %CHAR(numField);  // charField = '12345  '

charField = %EDITC(numField: 'X');  // charField = '0012345'
```

© Copyright Partner400, 2023.

Notes

It is worthwhile when replacing MOVEs to take the time determine exactly what was the intent of the original operation.

Note that %Char converts numeric fields to character, but it suppresses leading zeroes. If you want to retain leading zeroes in the character field, use %EDITC instead.

© Copyright Partner400, 2023.

MOVE - Date to Numeric

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

- Length of returned value is the number of digits in the Date
  - Or Time or Timestamp
- e.g. %DEC(date : *MDY) = length of 6
- Length of %DEC(Timestamp) is always 20

If format is not specified - the format of the date field is used

- In the example below it would have been *ISO and used 8 digits

```
Dcl-s  mmddyy   zoned(6);

// DateFld uses the default (*ISO) format
Dcl-s  DateFld  Date  Inz(D'2003-06-27');


// Convert date to numeric
 mmddyy = %Dec(DateFld: *MDY);

// Result:  mmddyy = 062703
```

© Copyright Partner400, 2023.

Notes

%DEC can be used to convert character strings and dates to numeric.

© Copyright Partner400, 2023.

MOVE - Character/Numeric to Date

%DATE ({ expression { : date format } })

- Converts both character and numeric fields to date
- The second parameter specifies the format of the source field
  - Just as it did with MOVE
- If no parameters are passed then the system date is returned

Companion BIFs are %TIME and %TIMESTAMP

```
   Dcl-S CharDate1  Char(6) Inz('011549');
   Dcl-S CharDate2  Char(6) Inz('031954');
   Dcl-S ISODate    Date(*ISO);
   Dcl-S USADate    Date(*USA);

 * Loading a date field from a character field fixed form version
C      *MDY0          MOVE      CharDate1      ISODate
C      *MDY0          MOVE      CharDate2      USADate

  // Equivalent code in Free-form
   ISODate = %Date(CharDate1: *MDY0);
   USADate = %Date(CharDate2: *MDY0);
  // Expressions are also supported - e.g. Join 3 separate character fields
   USADate = %Date( Day + Month + Year): *DMY0);
```

© Copyright Partner400, 2023.

Notes

The %DATE, %TIME and %TIMESTAMP BIFs are used to remove the requirement for a MOVE or MOVEL operation to convert data from character or numeric fields to date/time/timestamp fields.

Much like %CHAR will format date data into a character field, %DATE will do the reverse for character fields. The second (format) parameter specifies the format of the data being converted (i.e. the data represented by the first parameter). If the format is not specified, the default format for the program is used. If you recall, the default format for the program is the format specified with DATFMT (or TIMFMT for time fields) on the H spec or, if nothing is specified on the H spec, it will be *ISO.

Note that the format of the date (or time) returned will always be *ISO.

If you specify either *DATE or UDATE (to retrieve the job date) as the first parameter you should omit the format parameter altogether.

Note the difference between the job date and system date. *DATE or UDATE return the job date, %DATE() returns the system date (i.e. the current date). Also note that initializing a date field to *SYS using the D spec keyword only sets the INITIAL value for the field. So if a program runs over midnight, the initialized value in a date field defined with INZ(*SYS) will be different from that returned by %DATE().

© Copyright Partner400, 2023.

Dates - Replacing SUBDUR

Calculating Durations

%DIFF(Date1 : Date2 : DurationType)

- Calculates durations between date, time or timestamp values
- Other duration calculations are performed by simple + and - operators
  - In conjunction with new duration Built-Ins

Duration types are as per ADDDUR / SUBDUR

- That is *MS, *S, *MN, *H, *D, *M, *Y
  - And the long-winded versions *MSECONDS, *SECONDS, *MINUTES, etc.

```
 * Is the loan due within the next 6 months ?
C     DueDate        SUBDUR    Today          MonthsToRun:*M
C                    IF        MonthsToRun < 6


   MonthsToRun = %Diff(DueDate : Today : *M );
   If MonthsToRun < 6;

   If %Diff(DueDate : Today : *M ) < 6; // Alternate coding
```

© Copyright Partner400, 2023.

Notes

Hurray! Courtesy of the Free-form support, we finally have date duration support in expressions! This is a feature that RPG IV programmers have been wanting since V3R1. Most RPG IV programmers quickly became addicted to the extended factor 2 operation codes (e.g., EVAL, IF, etc.) and were dismayed to discover that in order to use the powerful date duration support built in to RPG IV from the beginning, they were forced to revert to the Factor 1, Factor 2, etc. format to use the powerful date duration operations.

The code examples here illustrate the ability to replace the ADDDUR and SUBDUR operation codes with expressions using either %DIFF or a simple arithmetic add (+) operation in combination with the duration BIF (%Years, in this example).

It also demonstrates that, because the new support is through BIFs, it can be used directly in expressions without the need for creating intermediate results.

On the next chart, we will show how to also replace the requirement for a MOVE operation code to get numeric data into a date or time field.

Now, all date data type operations can be done with free form operation codes.

© Copyright Partner400, 2023.

Dates - Replacing ADDDUR/SUBDUR

Adding durations to a date, time or timestamp

- This function can now be performed by simple addition
- The durations are supplied via the appropriate BIFs
  - e.g. %MINUTES, %HOURS, %DAYS, %MONTHS, %YEARS, etc.
- Multiple durations can be handled in a single statement

Subtracting a duration works the same way

- i.e. It is achieved through the use of simple subtraction with BIFs

```
 * These original duration calculations
C     ContrDate      AddDur    Cyears:*Y      ExpDate
C     ExpDate        AddDur    CMonths:*M     ExpDate
C     ExpDate        AddDur    1:*D           expedite

C     ExpDate        SubDur    90:*D          WarnDate

 // Can be replaced by these Free-form calculations
  ExpDate = ContrDate + %Years(CYears) + %Months(CMonths) + %Days(1);

  WarnDate = ExpDate - %Days(90);
```

© Copyright Partner400, 2023.

Notes

As on the previous chart, we are demonstrating here that since the new functionality is supplied by BIFs, they can be combined in a single expression - no need for the multiple and potentially error prone intermediate steps.

© Copyright Partner400, 2023.

Combining The New Date Functions

Full power of the new BIFs appears when combining them

- In the example below it has significantly reduced the amount of code
  - The difference is actually greater than shown since in the new version there is no need to define the work fields Temp, Today and WorkDate
- %Date is used twice in the calculation
  - The first time to obtain the system date
  - The second to convert the numeric field DueDate to a "real" date

```
 * Determine if payment is more than 90 days overdue - prior to V5R1
C      *MDY0          MOVE      DueDate        WorkDate
C                     TIME                     Today
C      Today          SubDur    WorkDate       Temp:*D
C                     If        Temp > 90
C                     : ......  : ..............................
C                     EndIf
  // And the same calculation in Free-form
   If %Diff( %Date(): %Date(DueDate: *MDY0): *Days) > 90;
    ...............................
   EndIf;
```

Returns the system date

© Copyright Partner400, 2023.

Notes

In this example, you can see how much more powerful it is to be able to use date operations in expressions. In this small example alone, we did away with 3 temporary work fields. We think the new operation is also more obvious as to its purpose.

Note the use of %Date for 2 functions - first to retrieve the current date for the calculation and second to convert the DueDate field to a date form so it can be used in the duration calculation (via %DIFF).

© Copyright Partner400, 2023.