BCI433 Lab 3A (updated Fall 2021)

**Writing an interactive RPGLE screen program**

**Lab objectives:**

* **Create a display file (screen)**
* **Code an interactive RPGLE program using the display file**

**Lab Requirements:**

* **Show the compile listing for AUTOINSRPG.RPGLE (as a pdf in ACS)**
* **Successfully run AUTOINSRPG similar to instructor program AUTOINS3A.**

Start an RDi session

Start a ‘Green Screen’ (emulator) Session.

**Using** **Rational Developer for i (RDi):**

Lab 3AB Overview

Lab 3 has been separated into two related labs. Lab 3A involves screen design with RDi and the related RPGLE code to display screens. Validation with a display file and with an RPGLE program is also covered.

The final version of your program for our imaginary auto insurance company calculates the annual insurance cost to cover the replacement value of your vehicle. They look at the replacement price you value your vehicle, how much you drive in a year and your driving record. An interactive program is used to gather information and then supply a quote. The age of your vehicle is calculated but not used in the quote. We are only coding a limited version of this for Lab 3A. After you successfully complete Lab 3A more information will be supplied for Lab 3B on how to calculate an accurate quote.

Some business rules are introduced in Lab3A. Most of the computational information and additional rules are supplied in Lab3B.

**Part A**

**Objectives:**

* **Use RDi Screen Designer to Create a Display File**

Display File screens in IBM i are very similar to forms you see on Web pages. Entering and compiling DDS code can produce these interactive screens. One option that may be used for entering DDS code is to use a GUI tool that allows you to visually select and enter what appears on the screen. Your selections are converted to DDS program code that is compiled to produce interactive screen records in a display file object. Those interactive screens are available to various program languages.

We will create a display file object called AUTOINSDSP that uses two overlapping screen records. RPGLE, CLLE and COBOL programs can interact with the screens in the AUTOINSDSP object. We will code an RPGLE program called AUTOINSRPG to view and interact with the screen records. The following shows a run of the RPGLE program.

==>CALL AUTOINSRPG

first screen called AUTOINFO is displayed and blank fields are filled in

Text

Description automatically generated with medium confidence

Data is entered for all the fields and the ENTER key is pressed

A second screen called AUTOQUOTE overlays the first screen and the input fields are protected. You will not be doing any calculations for LAB3A and the bottom screen will be different with LAB3B. It will correctly calculate the vehicle age and annual cost based on the input. In Lab 3A the screen always shows 99 as the vehicle age and 9,999.00 as the annual cost.

Graphical user interface, text, application, email

Description automatically generated

## Using Screen Designer to Produce a Display File

Source code for a Display File may be stored in a source file called QDDSSRC. You have already created this file in Lab 2 to store DDS code in the STUDENTS member that produced a physical file called STUDENTS.

DDS Stands for Data Description Specifications. The native IBM i language used to describe \*FILE objects.

Create a new member inside of QDDSSRC. The member name should be specified as AUTOINSDSP and the member type should be DSPF. If you are not immediately put in the GUI and are just in the editor after creating the member, close the AUTOINSDSP tab.   
  
Right click on your AUTOINSDSP member, click on Open With and select SCREEN DESIGNER.

You should still see a Remote Systems view on your upper left. The following should appear:

An outline view on the right.

A tab in the centre that has your member name (AUTOINSDSP) with two boxes below the tab. One box has a title of Screens and the other box includes a Screen and Records tab

Below this a work area should appear with a ruler that counts from column 1 to column 100

A palette is available to the right of this work area and this will be used to select and place fields and constants on our screen record.

Immediately below this work area will be tabs that allow us to work with the GUI or to work with actual DDS code

Further down is a Source Prompter tab. We have already used a source prompter in lab 2 to enter RPGLE code and to enter DDS code for a physical file program.

A properties box on the lower left which can be used to enter useful information about records and fields.

|  |  |
| --- | --- |
|  | .  Left click on Standard Record in the Palette area and drag this over to the left work area.  Record1 should show up in several spots. In a right top window and in a Properties box on the lower left. This name can be changed to something else here. We will change it to AUTOINFO |

Create a date constant ( MM/DD/YY ) by clicking on Date Constant which is found in the palette underneath the Constants folder. Move your cursor over to the top left of your black grid screen and click once to place the date constant. As you move your cursor with the mouse, you should see the row and column numbers changing.

After you have placed this date, go back to the constants folder in the palette and use the arrows to scroll through other constant selections. We are also using the user constant. Refer to the finished first screen on a previous page to determine the placement of these constants and then add them to your work area.

Click on Text Constant in the palette and then click to place this centred at the top of your screen.

You can change the default text of “Text constant” to “A U T O I N S U R A N C E Q U O T E ” in one of two ways. Type directly in the box provided. If this is awkward click on the word “Text Constant” in the properties box in the lower left corner.

A text constant shows below the user constant on the demo screen above as “Instructor Program”. **Put your actual name here to identify this screen as your screen for the professor marking the interactive program.**

There are several more text constants to include on the first screen record to be named AUTOINFO: “Your Replacement Value”, “Annual Kilometres Driven” and “Year of Vehicle” appear. There are more that show on the screenshot. Include the Driving Record fields. (check the screen shot for these text constants) Do not include the Vehicle Age and Annual Cost text constants or the text constant referring to the F3 key.

We need to put in fields beside the some of the text constant prompts.

Click on Date Field below the Fields folder in the palette. Place the Date Field beside the “Year of Vehicle:” constant.

This shows up as yyyy-mm-dd on your work screen. Double click on the Properties box to specify information about this field. Change the provided field name to VYear and set the usage as Both. (This means it is an Input/Output field)

The next field beside the “Your Replacement Value:” prompt will require selection of Named Field from the same location where you found the date field.

The default properties for this field are set as 9 digit Input/Output field. The usage reflects this in Properties with a setting of “Both”.

Make this a 7 digit numeric field by typing over the “7” that appears under Length on the right side of the properties view and indicate this field has 0 decimal positions. The field name should be changed to REPLACEVAL from the default name supplied by the system.

The usage for the field should remain as Both. This means it is input capable (the user can enter something into the field when it is presented on the screen record) and it is output capable, the program can change what the user entered or leave the value and redisplay the field contents the next time the screen is displayed.

The display attributes for this field will use an indicator. When this indicator is on, we want the cursor positioned to this field and we want this field to show in reverse image.

First select Reverse image and Position cursor and click on the Add button.

Graphical user interface, application

Description automatically generated

Then click on the Indicators button and use indicator 97

Table

Description automatically generated

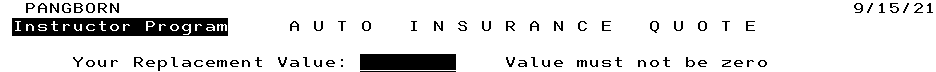
This means indicator has to be on in order for reverse image and position cursor to occur.

Graphical user interface, application

Description automatically generated

Indicators are either on or off, true or false, “1” or “0”. You have 99 of them available to use and their default setting when your program starts is off.

The program will check this field for a zero and turn on indicator 97 to position the cursor and to get reverse image for the REPLACEVAL field when a zero has been entered into the field. When we redisplay the screen, our attention will be on this field. We can also use indicator 97 to condition a text message informing the user that the field should not be set at zero.



Graphical user interface, text, application, email

Description automatically generated

The same technique can be used to insure the annual kilometers are not zero and the year of the vehicle is not before cars were first made (January 29, 1886)

We are going to include some data validation with the display file rather than have the program handle it. For the at fault accidents and number of traffic tickets we are going to allow numbers from 0 to 10 to be entered. For the Demerit points the numbers from 0 to 15 will be allowed. Use the Keywords tab and select the RANGE keyword.

This will prevent the user from entering numbers outside the allowed range of values.

The completed AUTOINFO screen record shows below. (You will have your name in place of Instructor Program)

A screenshot of a computer

Description automatically generated

In the screen shot showing above at the bottom of the screen record being designed with the GUI, there is an option to look at the actual code that is being generated. We are currently on the Design tab. You can click on the Source tab and view the actual code. You can make changes to the code as well, but this should be done carefully when you are a beginner. You can’t introduce syntax errors when working in design mode, but you could inadvertently touch a key and modify the code and not be able to recover.

One thing you can safely do when viewing the source code is to use a compile option that appears near the top of your screen: File, Edit, Source, **Compile**, Navigate …

When you compile the code a new object is placed in your library. In this case AUTOINSDSP This display file object can be used by CLLE, RPGLE, and COBOL programs.

Object Type Attribute

QDDSSRC \*FILE PF-SRC

AUTOINSDSP \*FILE DSPF

QDDSSRC holds the member that contains the display file code for AUTOINSDSP and AUTOINSDSP is the object available to be used by programs.

Click on the Source tab

Graphical user interface, text

Description automatically generated.

Compile what you have so far. Remember to click on the Design tab to get back to the GUI interface.

Here is source code.

Note a DSPATR(PR) is included in the code, but was not discussed in this lab handout. It will have been discussed in class.

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

You are ready to provide the second screen record that will overlay the first screen record. (It already shows in the code above)

Click on Standard Record in the palette and drop it onto your work area. You should get a RECORD1 with a blank work area. You can toggle between AUTOINFO and RECORD1 by clicking on the appropriate name at the top box in the screen. Try it. Change RECORD1 to the record name AUTOQUOTE

Make sure all your entries are below the area for AUTOINFO (in order for OVERLAY to work properly)

Graphical user interface

Description automatically generated

The fieldnames used for the above screen are Age, and Cost. Age is 3 digits with zero decimal positions, and Cost is 7 digits with 2 decimal positions. These are to be output only fields (the user cannot type anything into the field and the program loads these fields with appropriate values.)

**Specifying Function Keys**

We specified a constant which told the user to use F3 to exit the program. Constants are only text; they do not do anything! We need to make the F3 function key available to the user. Making sure your focus is not on a constant or a field in the AUTOQUOTE work area, double click on the properties view tab at the lower left of your screen and then select the Function Keys tab. Graphical user interface, application

Description automatically generated

Click on F3 and on the right-hand side select the Command Function radio button.

Under Parameters click on Indicator and beside that change the number to show as a three.

Click on Description and type in **Exit Program**.

Let’s review what you have done:

By clicking on F3 you have enabled the F3 key for AUTOQUOTE. That means when the user is looking at AUTOQUOTE they can press F3. They cannot press F4 or F5 because those keys have not been enabled.

The Enter key is by default enabled for the screen records. Now, the user can press F3 or Enter. Our program will exit when F3 is pressed and reshow AUTOINFO if Enter is pressed.

Command Attention means that no data is passed back to the program. If you entered data into an input capable field, that data would not be available to the program. Command Function allows the data entered on the screen to be passed back to the program. Our application could have used either the CA or CF setting because when F3 is pressed our program does not do anything with the input data entered on that final screen.

The program needs to know if F3 or Enter was pressed. You have enabled a response indicator by checking of Indicator 3 under Parameters. There are 99 indicators available to you. They are like switches with an “On” or an “Off” setting. This logical field either has a ‘1’ or a ‘0’ in it.

The EXIT Description is a comment that is placed in your code.

What was the actual DDS code for all of this? \_\_F3 Exit-Key\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(You already know how to look this up)

Does this function key setting apply to AUTOINFO and AUTOQUOTE or just one of those records? AUTOQUOTE

Add the OVERLAY keyword for AUTOQUOTE by using the KEYWORDS tab when setting AUTOQUOTE properties.

This will allow AUTOQUOTE to display without wiping out AUTOINFO. If any of AUTOQUOTE’s fields or constants are on the same line as a AUTOINFO field or constant, AUTOINFO will still be wiped out.

Check your source code for the second screen record with the DDS code shown on a previous page.

Close your AUTOINSDSP tab and save your work.

Right click on your closed Display file member (in the Remote Systems View) and compile it.

What is the command used to compile a display file DDS member? \_\_\_CRTDSPF\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

You should see feedback that the “Events file contains no messages” under the Error List Tab.

Click on the Commands Log Tab. You should see dialogue similar to the following:

SBMJOB CMD(CRTDSPF SRCFILE(DB433A40/LAB3) SRCMBR(AUTOINSDSP) REPLACE(\*YES) OPTION(\*EVENTF) FILE(DJ433A40/AUTOINSDSP)) JOBD(\*USRPRF)

Job 262777/DB433A40/QDFTJOBD submitted to job queue QBATCH in library QGPL.

CRTDSPF SRCFILE(DB433A40/LAB3) SRCMBR(AUTOINSDSP) REPLACE(\*YES) OPTION(\*EVENTF) FILE(DB433A40/AUTOINSDSP)

**File AUTOINSDSP created in library DB433A40.**

If you don’t get this there may be an error introduced into the code when you were working with the Source tab.

There also may be trouble with your connection. If the **File AUTOINSDSP created in library** dialogue does not appear and no syntax errors show, then close up RDi, reopen it and then retry the compile.

You can also compile this code using the green screen. Sign on to a Client Access session.

Type WRKMBRPDM QDDSSRC at the command line.

(This won’t work properly if you didn’t call your source physical file QDDSSRC)

Use option 14 beside the AUTOINSDSP member name.

Instead of typing WRKSPLF at the command line to view your listing, type in SP where you had typed in “14”.

Use option 5 and scroll up and down your listing or just type in “B” at the top to get to the bottom of your listing and look for the **File AUTOINSDSP created in library** message.

We are ready to code our program. Here is some help with entering the code.

Although a lot of help is provided, this help assumes that you made it to class and saw your instructor enter the code. If you missed that class, you should be able to get help from a lab aid or an instructor if you are stuck on any step. If you missed both the class and the lab, you should see a BCI433 tutor.

Add a new member called AUTOINSRPG with an RPGLE member type to your QRPGLESRC source physical file. It will contain your program code for an RPGLE program that uses the display file records you have just created.

We are going to code a free format RPGLE program. In the past, students have coded fixed format file and definition specs and then coded free format statements to indicate the processing logic and order. This was acceptable to the editor we worked with in lab 1 (PDM - SEU – Program Development Manager – Source Entry Utility). It was possible to work with the code in that environment and only code errors would be flagged.

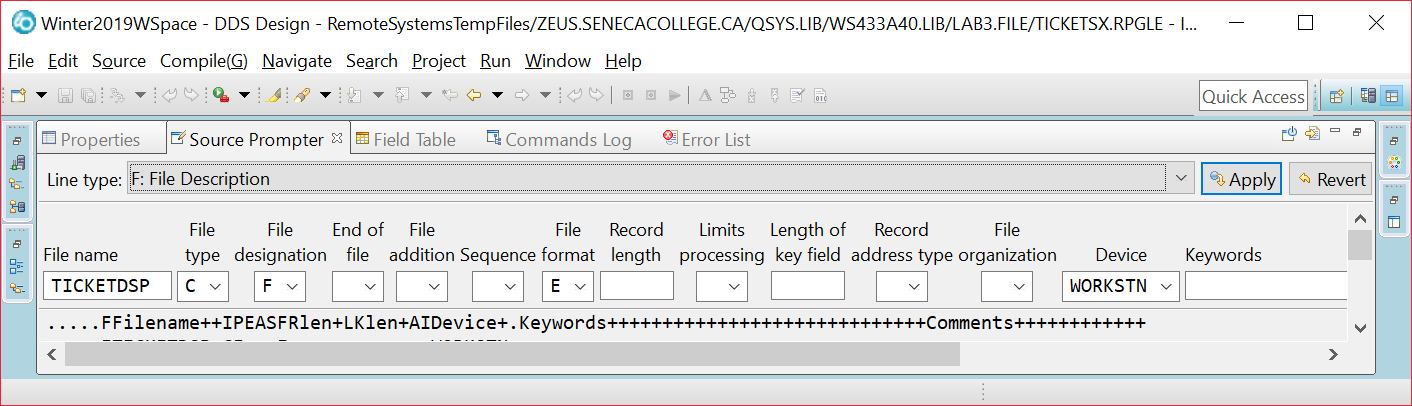
This semester we are going to code the file and definition information using free format code. If you look at this code using the source entry utility, syntactically correct free format file lines will be flagged as errors.

They are not flagged as errors when using RDi. So lets be aware of that problem and use RDi.

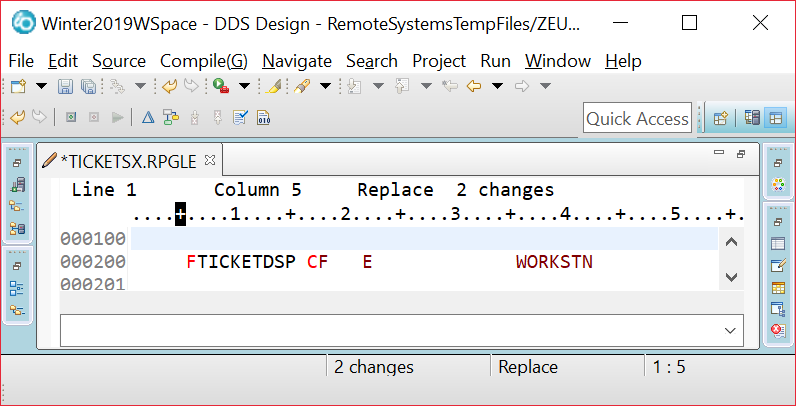
What file name are we processing? \_\_AUTOINSRPG.rpgle\_\_\_\_\_\_\_\_\_ . We are using a workstation device file.

Make sure you start your code anywhere after column 7.

**dcl-f filename workstn;**

This is the fixed format way to do the same thing that shows above with a workstation file called TICKETDSP. With a prompter 

and how it appears in code.



**Enter your file declaration using the easier free format technique.**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

If we had started with a fixed format file spec would have had to type /FREE on the next line to indicate we were going to free format RPG. The slash would have gone in column 7.

As you move your cursor, feedback should be provided about which column number you are in.

The code you need to enter for the main routine will be supplied for you. This version will not be perfect and may require a few adjustments in a later lab.

It is important to remember that each program statement needs to end with a semicolon. If you don’t do this, a syntax error may pop up when you go to the next line. The compiler will also reject an RPGLE statement that does not end with a ” ; ”

If a syntax error pops up it may not always appear immediately below the line that is missing the semicolon. Figure out where you need to make the adjustment and then press CTRL + F5 to get rid of the message that shows up in the LPEX editor.

Common PDM and LPEX editor commands:

A line can be deleted by typing a D in the first position of the line number and pressing enter.

A line can also be copied by typing a C at the line and a B or A at the line it is to be copied before or after

Multiple lines can be deleted by typing DD at the start of the block and DD at the end of the block

CC and MM also work with a block of lines. There are more modern ways of editing available with the LPEX editor.

Here is the code you need to enter.

**DCL-F AutoInsDsp Workstn; ;**

**EXSR CLEAR; ;**

**EXFMT AutoInfo; ;**

**DOW NOT(\*IN03);**

**If ReplaceVal = 0;**

**\*IN97 = \*ON ;**

**EXFMT AutoInfo;**

**\*in97 = \*OFF;**

**ITER;**

**?**

**ElseIf ?;**

**?**

**?**

**ELSE;**

**EXSR DetermineCost; ;**

**ENDIF;**

**// PROTECT FIRST SCREEN RECORD FIELDS**

**// REDISPLAY FIRST SCREEN RECORD AND THEN OVERLAY SECOND RECORD**

**\_\_\_Attribute => protected => indicators => 75\_\_\_\_\_\_;**

**\_\_\_AUTOQUOTE => Keywords => OVERLAY\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;**

**IF \*IN03=\*OFF;**

**EXSR CLEAR;**

**EXFMT AUTOINFO;**

**ENDIF;**

**ENDDO;**

**\*INLR = \*ON;**

**RETURN;**

EXFMT is a Read/Write operation

A write of the screen record to the display station and a pause while the user reads it. The user may fill in some fields if there are input capable fields on the screen. At minimum the user will press the return key. A read back to the program occurs and the next program statement is executed.

DOW / ENDDO is for looping. The test is done at the start of the loop and if the test is not passed, none of the loop statements are executed.

NOT(\*IN03) is a test of the response indicator 03 which was specified when developing AUTOINSDSP.

Where was that specified? \_ AUTOQUOTE properties keyboard\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

You could have also said DOW (\*IN03 = ‘0’) or DOW (\*IN03 = \*OFF)

EXSR DetermineCost is sending control down to a subroutine called DetermineCost. This subroutine needs to appear after the RETURN statement and will determine the contents of AUTOQUOTE output only fields.

Here is some sample code to use for this subroutine:

BEGSR DetermineCost;

Cost = 9999; // this is a stub coded amount. We will worry about getting the right amount in lab 3B

Age = 99 // this is not an actual age calculation based on the year of the vehicle

ENDSR;

Notice that all those fields have not been determined properly in the code above. You can demonstrate your program with this stub code for this lab and the next lab will include the information on how to determine the amounts. In Lab 3B we will look at solving for these fields.

When the overlayed second screen shows, the user will not be allowed to change the entries made above. They can press return to redisplay the first screen with the year of the vehicle field set to **D’0001-01-01’** (default) and the rest of the fields set as zeroes or press F3 to exit the program.

Compile AUTOINSRPG.

Fix any errors

Switch to your Access Client session and run AUTOINSRPG. This is done by typing

\_\_\_\_\_CALL AUTOINSEPG\_\_\_\_\_\_\_\_\_\_\_\_\_ at the command line.

Enter values for all the input fields and press enter.

When your results screen shows, can you still type data into the top screen fields? \_\_\_YES\_\_\_

Make sure you can exit your program by pressing the F3 key. A previous lab showed what do to if you can’t stop your program.

What do you need to do to stop this program if it won’t end?

\_\_\_press the F3 key\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Instructor sample programs are available to check yours against. They have reverse image text to self-identify as an instructor program. There is AUTOINS3A allowing you to check how your AUTOINSRPG program should behave after doing LAB3A (showing screens and doing validation). Next week you will compare your AUTOINSRPG RPGLE program to the instructor program AUTOINSB which correctly calculates the insurance cost and vehicle age.

To run the first sample program, change your current library to BCI433LIB and then invoke the program:

CHGCURLIB BCI433LIB

CALL AUTOINSRA

Notice the instructor program is flagging problems with zeroes for replacement value or annual Kilometers driven.



Text

Description automatically generated with low confidence

**Here is a sample screen when the date of the car is too low (before cars were invented)**

Text

Description automatically generated

**This shows a display file validation (before the program gets data from the screen record)**

Graphical user interface, text, application, email

Description automatically generated

Your program should run the same way as the instructor version runs.

After you have finished, you should reset your current library back to your own library.

What happens if you do not change the current library when running the instructor’s version of this program?

It will depend on the existence of AUTOINSDSP in your library.

Try running this program with your library as the current library after you have successfully created your version of AUTOINSDSP.

What happened?

\_Error occurs like this: Job 109209/DB433D24/QPADEV000M started on 09/25/22 at16:30:14 in subsystem

Error message CPF4131 appeared during OPEN for file MARKSDSP (C S D F)\_\_

Lab 3 Summary

A display file is coded with DDS code in a member that has a DSPF type.

Once this DDS code has been successfully compiled a Display File object is produced that can be used by different programming languages that want to take advantage of the interactive screens allowing data entry and data display.

The easiest way to define how the display file screens or records will look is to use the Screen Designer GUI in RDi.

The GUI allows you to define field names, types, usage, editing, validations and optional indicators that allow viewing of the field.

You would use the GUI to provide this information and the appropriate line is entered as DDS code.

Here is a sample line with explanations

This is how it actually appears



Some separation is shown here to better highlight concepts – the line would not be entered this way.

Field Field Data Decimal Line Column

Indicator Name Length Type Positions Use Number Position Functions

A 90 TEST1 3 S 0 B 5 32 RANGE(0 100)

Indicator 90 has to be on in order to view this field called TEST1 on the screen. If indicator 90 is off, the TEST1 field will not be visible when the screen record is displayed. For our application it would not be a good idea to use an indicator with the TEST1 field.

L shows a field length of three. So at this point XXX could be entered

T or Type shows a field type of zoned decimal and D or Decimal shows 0 decimal positions. So now we cannot put X’s in the field. 999 would work.

U or Usage indicates if this field is for input only (I) , output only (O) or both input and output (B).

P or position on the screen shows that this field would reside on line 5 at column 32.

The RANGE function allows us to validate the field before it is passed to a program. The user can enter 0 to 100. They cannot enter a negative number or 101.

Here is some more partial DDS code with explanations

A R RECORD2

A CA03(03 'EXIT')

A OVERLAY

A 11 18'Tests:'

A 13 18'Final Mark:'

A 15 18'Final Grade:'

A 22 3'F3=Exit'

A TESTOVRALL 3Y 0O 11 32EDTCDE(1)

A NUMGRADE 3Y 0O 13 32EDTCDE(1)

A LETGRADE 2A O 15 33

This screen record is called RECORD2 and has some text and fields that are displayed.

At the record level we see CA03 – that enables the pressing of function key F3 when the screen record is displayed. CA is command attention and no data from the screen gets passed back to the program. Using CF will allow data entered on the screen record to be passed back to the program.

(03 ‘Exit’) indicates when F3 is pressed indicator 03 will be turned on and the comment is ‘Exit’

OVERLAY is used to allow RECORD2 to overlay a screen record that is already being displayed. This only works of the first screen record does not use lines 11 – 25. You can see RECORD2 is using lines 11 to 22.

EDTCDE(1) can be used to make a numeric field show with commas, a decimal point and suppressed leading zeros

So 090 would show as 90. If the field was larger 0293334^33 would show as 293,334.33. The decimal point is not stored in the field but can be shown with proper editing. We used the “^” symbol to show where the placeholder for a decimal point is.

RPGLE

A Display file is declared in an RPGLE program with a DCL-f statement

Dcl-F HWYTOLLDSP Workstn;

EXFMT – write a screen record from a display file, pause and when the user presses enter read back what has been inputted into the screen record fields. Sometimes there are no fields and the read back just acknowledges that the screen has been viewed and the user is letting the rest of the program proceed.

DOW - a loop where the test on entering the loop or repeating the loop is done at the start of the loop.

ENDDO

DOU – the loop code is executed at least once and the test on repeating the loop is done at the bottom.

WRITE – can be used to have a screen record display. There is no pause and the program continues. This is useful when showing a screen record and then overlaying a second record and then pausing to let the user look at both screen records.

EXSR GETGRADE – control goes from here to a subroutine at the bottom of the program. The named subroutine has a BEGSR and ENDSR to indicate what code is executed and then control goes back up and executes the next line after the EXSR line.

\*IN01 - \*IN99 – in RPGLE indicators are referred to by an asterisk and a number. There are 99 indicators available for use and their default setting is \*OFF or ‘0’ . They can be turned on during the program run and then checked to see if they are on or off to determine a course of action.

\*INLR = \*ON – these two statements are how you should end all your RPGLE programs.

RETURN The last record indicator is turned \*ON and RETURN is used to return control to the operating system.