**ILE RPG goes free and mobile!**

By Roger Pence

Back in my early days of the IBM midrange platforms RPG was mostly considered a report programmer generator (although the "generator" part was a little over optimistic!). This was back when writing interactive programs with RPG was a black art of shoe-horning the RPG cycle into something in which it clearly didn't fit.

But with enough effort, slamming your head into the wall, and a little time with a Shelly Cashman textbook or two, it was possible back in the day to write serviceable interactive RPG programs. That code was never going to win a Most Beautiful Code Contest, but it did the job. After spreading my wings and learning other languages, I realized the value of indented, expressive, and obvious code. Alas, even with the advent of the mid-90s era of ILE RPG, RPG persisted as a compile-only language very unfriendly to both eyeballs and quick comprehension. I’ll go out on a huge limb here and flatly state that RPG, I don’t care how good of a programmer you were, was simply impervious to being expressive and obvious.

I am late to the party but the other day I started noodling around with the 7.1 Technical Refresh of ILE RPR; the one that provides full "free-form" RPG capabilities. In very short order I was outrageously impressed. Nits remain to be picked with several syntactical aspects of the language, but for the first time ever, I am writing RPG that is expressive and comprehensible. If you write RPG for anything serious, you owe it to yourself to dig into the 7.1 Technical Refresh RPG. It will forever change how you write RPG. While the refresh doesn't add any great new functional changes, RPG programming power previously straight-jacketed by column reliance quickly comes shining through.

This article features an example RPG program I wrote with the TR 7.1 RPG free format syntax. I am a rusty RPG coder to be sure, so if you see anything glaringly stupid (for which I have a knack!) please let me know. I cut a few corners to keep the code short--and at 191 lines I'm probably pressing my luck. But, it's 2015 and articles like this are no longer constrained by the printed page, so bear with me. I won't cover every line of the RPG in detail in this article, but all of the project's source, including the RPG, is [available here for easy download](https://github.com/ASNA/Mobile-RPG-app-with-TR-7.1-RPG) or further inspection.

Making it mobile

I thought ILE RPG's spanking new syntax was worthy of spanking new user interface, so the program I wrote is an ASNA Mobile RPG (MR) app. MR apps are HTML5 apps primarily intended for smarthpones and tablets, but also run on desktop browsers. They feature performant and secure IBM i database connectivity. The app presented here is a simple little customer CRUD app with, for a little more sizzle, a Google map of the customer's address.

The intent of this article is two-fold:

1. To show TR 7.1 ILE RPG in action. A full RPG program shows more context and capabilities than just a few snippets of RPG here and there.
2. To show an RPG model that provides a (use your imagination here) a work-with panel-like UI for a mobile app. Mobile idioms are vastly different than we used to use in the green-screen, but as you'll see, MR abstracts away mobile uniqueness and empowers RPG coders to create IBM i mobile apps with nothing but RPG.

The ASNA Mobile RPG mobile display file

Before digging into the ILE RPG for this mobile app, let's take a quick tour of MR's mobile display file. This isn't really an article about ASNA Mobile RPG (MR) as much as it is about ILE RPG (I wrote an article a while back that goes into more Mobile RPG detail [that can be read here](http://www.mcpressonline.com/dev-tools/using-asna-mobile-rpg-to-create-a-mobile-app.html)). While there are a few things to understand about Mobile RPG, thanks to IBM's Open Access API, there is less than you think. Mobile RPG provides a Windows-based mobile UI designer. It lets you create mobile display files with record formats, just like old-school display files. MR enforces the traditional indicator-driven "contract" between an RPG program and its display file.

**The three steps to creating and running an ASNA Mobile RPG app**

[Figure 1 here]

Figure 1 above summarizes the three steps for creating and running an MR mobile app. A little detail for each step follows:

**Step 1 - Create the mobile UI.** MR provides a Windows-based designer for creating the mobile UI. It includes all of the user interface elements you'd expect for a mobile user interface (including text boxes, buttons, navigational bars, map, data charts, data list, signature capture, images, and many others). Once you've created the mobile display file, you export it through MR as a traditional display file object on the IBM i. This display file will *never* be seen by eyeballs; rather, it exists purely to compile an associated RPG program. During this export step you can optional save the exported display file source, which is mostly useful for learning purposes to see how all of MR's user interface elements map to RPG idioms.

**Step 2 - Write an RPG program.** This program provides the logic and file IO for your mobile app. It is compiled against the display file object created by the export process in Step 1.

**Step 3 - Run the mobile app.** The MR mobile app is an HTML5 browser-based mobile app that runs in mobile (and desktop) browsers. At runtime, IBM's Open Access API intercepts this RPG program's display file data and routes it to and from the MR mobile user interface. That the traditional display file is swapped out for the mobile display file at runtime is completely transparent to the RPG program.

Let's take a closer look at the example app's three display panels (which are actually surfaced as display file record formats).

**The mobile list panel**

[Figure 2A here]

Figure 2A. The mobile list display

The initial screen for this example app is this list panel and its format name is CUSTLIST. It is scrollable and a user can select a row one of two ways; the user can either tap on the customer's name or tap on the chevon on the right end of the row. This list effectively presents the work-with panel model for mobile devices.

Because this is an example of IBM i app, it assumes it might be using an input file with a great deal of records thus the "Next" button to get the next page of rows. For this example, six rows are shown and the display isn't scrollable. However, the number of rows displayed is controlled by a constant in the underlying RPG program. Unlike a teathered greens-screen, apps like this need to be written with performance in mind. The user doesn't want to wait for hundreds of rows to load (but it might also be reasonable to load more than I am loading here).

The Mobile RPG designer is used to map function keys to button (and other UI element) taps. In this panel, the F3 key is mapped to the "End" button and the F5 key is mapped to the "Next" button. The record format name for this list panel is CUSTINFO.

This customer list is populated in the ILE RPG as a very simple subfile. This subfile has the following configuration assignments:

Description Value

Subfile name CSTSBF

Subfile controller name CSTCTRL

Clear subfile indicator 99

Main text field name CSTXT

Main text field length 70

Secondary text field name CSTDTL

Secondary text field length 50

Selection field name CSTSEL

Value field CSTVAL

Value field length 30

The CSTSEL fields is a single-character field that is populated implicitly by MR with a '1' when a user taps the row. RPG's READC operation is then used to identify the row selected. The CSTVAL field is a "hidden" field used to stash hidden data for a row. In this case, the customer number is stored in this field (so that when a row is selected, via READC, the customer number is available).

**The update panel**

[Figure 2B here]

Figure 2B. The mobile update panel

The update panel, which is format CUSTINFO, is shown by tapping on a customer name from the list panel. The user can change any of the fields presented and tap OK or tap Cancel. Tapping either button returns the user to the list panel.

In this panel, the F2 key is mapped to the "Back" button, the F8 key is mapped to the "OK" button, and the F2 key is mapped to the "Cancel" button.

**The map panel**

[Figure 2C here]

Figure 2C. The map panel.

The map panel, which is format CUSTMAP, is displayed for customer when the chevron is tapped on the list panel. With an ILE RPG program providing of this mobile app's logic, Mobile RPG needed an idiomatic RPG way of providing the addresses for a map. In this case, a single address is mapped, but the Mobile RPG map control can map many addresses.

The map control is fed addresses through a simple RPG subfile. This subfile has the following configuration assignments:

Description Value

Subfile name MAPSBF

Subfile controller name MAPCTRL

Clear subfile indicator 99

Address field name Location

Main text field length 60

Clicking back returns the user to the list.

After exporting the display file these three record formats comprise, the residual DDS display file source member is shown in Figure 3 below. Take a look at it and notice that its fields and subfiles resolve to those explained above.

0001 A DSPSIZ(27 132 \*DS4)

0002 A INDARA

0003 \*--------------------------------------------------------

0004 A R CUSTLIST

0005 A OVERLAY

0006 A MOREROWS 10A O 1 2

0007 \*--------------------------------------------------------

0008 A R CUSTINFO

0009 A OVERLAY

0010 A CUSTKEY 50A O 1 2

0011 A CMNAME 40A B 2 1

0012 A CMADDR1 35A B 2 43

0013 A CMCITY 30A B 3 1

0014 A CMSTATE 2A B 3 33

0015 A CMPOSTCODE 10A B 3 37

0016 \*--------------------------------------------------------

0017 A R CUSTMAP

0018 A OVERLAY

0019 A CMNAME 40A O 1 2

0020 A CSZ 60A O 2 1

0021 \*--------------------------------------------------------

0022 A R CSTSBF SFL

0023 A CSTSEL 1A H

0024 A CSTTXT 70A O 1 5

0025 A CSTVAL 30A H

0026 A CSTDTL 50A O 3 1

0027 \*--------------------------------------------------------

0028 A R CSTCTRL SFLCTL(CSTSBF)

0029 A SFLSIZ(2)

0030 A SFLPAG(1)

0031 A N99 SFLDSP

0032 A N99 SFLDSPCTL

0033 A 99 SFLCLR

0034 A OVERLAY

0035 \*--------------------------------------------------------

0036 A R MAPSBF SFL

0037 A LOCATION 60A O 1 2

0038 \*--------------------------------------------------------

0039 A R MAPCTRL SFLCTL(MAPSBF)

0040 A SFLSIZ(2)

0041 A SFLPAG(1)

0042 A N99 SFLDSP

0043 A N99 SFLDSPCTL

0044 A 99 SFLCLR

0045 A OVERLAY

Figure 3. DDS display file source generated when MR mobile UI is exported as a display file object.

Think of MR's exported display file object as a proxy for the mobile UI. At compile time, the RPG program will reference this proxy display file; but at runtime, through Open Access, the MR mobile display file is used. Once again, the proxy IBM i display file is never seen by human eyeballs.

With the MR mobile display file and its IBM i proxy display file created, let's turn our attention to some of the interesting parts the ILE RPG source code. Because MR so well abstracts away the notion that we're really creating a mobile app, the RPG is written (and can be examined) without regard for it really being a mobile app. Just think of it as powering three simply traditional display file formats (which, by the way, is exactly what it's doing).

The ILE RPG source

The full RPG source is shown below in Figure 4a and the single /COPY member it includes is shown in Figure 4B. A brief code narrative follows these two listings.

0001 Ctl-Opt Option(\*srcstmt) Dftactgrp(\*No) ActGrp('rpmobile');

0002

0003 Dcl-F smplst WORKSTN Infds(infds)

0004 Handler('MOBILERPG')

0005 SFile(CSTSBF:CSTRRN)

0006 SFile(MAPSBF:CSTRRN);

0007

0008 Dcl-F CustomerL2 Disk(\*ext) Usage(\*Input) Keyed

0009 Rename(RCMMASTER:RCUSTL2);

0010

0011 Dcl-F CustomerL1 Disk(\*ext) Usage(\*Update) Keyed

0012 Rename(RCMMASTER:RCUSTL1);

0013

0014 Dcl-DS CustL2Key LikeRec(RCUSTL2:\*Key);

0015 Dcl-DS TopRow LikeRec(RCUSTL2:\*Key);

0016

0017 Dcl-S CstRRN Zoned(4:0);

0018 Dcl-C MAXROWS Const(6);

0019

0020 /copy RPMRDATA/QRPGLESRC,KEYMAPF

0021

0022 Dcl-DS Action Qualified;

0023 Exit Char(1) Inz(F03);

0024 Cancel Char(1) Inz(F02);

0025 Back Char(1) Inz(F02);

0026 ItemTapped Char(1) Inz(F01);

0027 ChevronTapped Char(1) Inz(F04);

0028 More Char(1) Inz(F05);

0029 Backward Char(1) Inz(F06);

0030 Find Char(1) Inz(F07);

0031 OK Char(1) Inz(F08);

0032 Display Char(1) Inz(F09);

0033 Remove Char(1) Inz(F10);

0034 End-DS;

0035

0036 SetLL \*LoVal CustomerL2;

0037 LoadCstLst();

0038 ExFmt CUSTLIST;

0039

0040 Dow ActionRequest <> Action.Exit;

0041 Select;

0042 When CurrentFormat = 'CUSTLIST';

0043 Select;

0044 When ActionRequest = Action.ItemTapped;

0045 ReadC CSTSBF;

0046 If ReadCustById(%INT(CSTVAL));

0047 // Customer not found.

0048 EndIf;

0049 ExFmt CUSTINFO;

0050

0051 When ActionRequest = Action.ChevronTapped;

0052 ReadC CSTSBF;

0053 If ReadCustById(%INT(CSTVAL));

0054 // Customer not found.

0055 EndIf;

0056 CSZ = %TRIM(CMADDR1) + ', ' +

0057 %TRIM(CMCITY) + ', ' +

0058 CMState;

0059 ShowMap(CSZ);

0060 ExFmt CUSTMAP;

0061

0062 When ActionRequest = Action.More;

0063 LoadCstLst();

0064 ExFmt CUSTLIST;

0065

0066 Other;

0067 ExFmt CUSTLIST;

0068 EndSl;

0069

0070 When CurrentFormat = 'CUSTINFO';

0071 Select;

0072 When ActionRequest = Action.OK;

0073 UpdateCust();

0074 RefreshCustList(CMName:

0075 CMCustNo);

0076 ExFmt CUSTLIST;

0077

0078 When ActionRequest = Action.Cancel;

0079 RefreshCustList(TopRow.CMName:

0080 TopRow.CMCustNo);

0081 ExFmt CUSTLIST;

0082

0083 When ActionRequest = Action.Back;

0084 SetLL \*LoVal CustomerL2;

0085 LoadCstLst();

0086 ExFmt CUSTLIST;

0087

0088 Other;

0089 SetLL \*LoVal CustomerL2;

0090 LoadCstLst();

0091 ExFmt CUSTLIST;

0092 EndSl;

0093

0094 When CurrentFormat = 'CUSTMAP';

0095 Select;

0096 When ActionRequest = Action.OK OR

0097 ActionRequest = Action.Back;

0098 RefreshCustList(CMName:

0099 CMCustNo);

0100 ExFmt CUSTLIST;

0101 EndSl;

0102

0103 EndSl;

0104 EndDo;

0105

0106 \*InLR = \*On;

0107 Return;

0108

0109 Dcl-Proc UpdateCust;

0110 Update RCustL1;

0111 End-Proc;

0112

0113 Dcl-Proc LoadCstLst;

0114 Dcl-S RowCount Packed(12:0);

0115

0116 \*IN99 = \*On;

0117 Write CSTCTRL;

0118 CstRRN = 0;

0119 RowCount = 0;

0120

0121 DoW (RowCount < MAXROWS);

0122 Read CustomerL2;

0123 If NOT %EOF;

0124 If RowCount = 0;

0125 TopRow.CMName = CMName;

0126 TopRow.CMCustNo = CMCustNo;

0127 EndIf;

0128 RowCount = RowCount + 1;

0129 CstRRN = CstRRN + 1;

0130 CSTSEL = '0';

0131 CSTTXT = CMName;

0132 CSTDTL = %Trim(CMCITY) + ', ' + CMSTATE;

0133 CSTVAL = %CHAR(CMCustNO);

0134 Write CSTSBF;

0135 Else;

0136 Leave;

0137 EndIf;

0138 EndDo;

0139

0140 \*In99 = \*Off;

0141 Write CSTCTRL;

0142 PeekForEOF();

0143 End-Proc;

0144

0145 Dcl-Proc PeekForEOF;

0146 Read CustomerL2;

0147 If NOT %EOF;

0148 Eval MOREROWS = 'More...';

0149 ReadP CustomerL2;

0150 Else;

0151 Eval MOREROWS = 'No More';

0152 SETLL \*LOVAL CustomerL2;

0153 EndIf;

0154 End-Proc;

0155

0156 Dcl-Proc ReadCustById;

0157 Dcl-Pi \*N Ind;

0158 CustNo Packed(9:0) CONST;

0159 End-Pi;

0160

0161 Chain CustNo CustomerL1;

0162 Return %EOF();

0163 End-Proc;

0164

0165 Dcl-Proc RefreshCustList;

0166 Dcl-Pi \*N;

0167 Name Like(CMName);

0168 CustNo Like(CMCustNo);

0169 End-Pi;

0170

0171 CustL2Key.CMName = Name;

0172 CustL2Key.CMCustNo = CustNo;

0173 SetLL %KDS(CustL2Key) CustomerL2;

0174 LoadCstLst();

0175 End-Proc;

0176

0177 Dcl-Proc ShowMap;

0178 Dcl-Pi \*N;

0179 Address Char(60);

0180 End-Pi;

0181

0182 \*IN99 = \*On;

0183 Write MAPCTRL;

0184

0185 CstRRN = 1;

0186 Location = Address;

0187 Write MAPSBF;

0188

0189 \*In99 = \*Off;

0190 Write MAPCTRL;

0191 End-Proc;

Figure 4A. The full ILE RPG source listing.

0001 Dcl-Ds Infds;

0002 ActionRequest Char(1) Pos(369);

0003 CurrentFormat Char(10) Pos(261);

0004 End-Ds;

0005

0006 Dcl-C F01 const(x'31');

0007 Dcl-C F02 const(x'32');

0008 Dcl-C F03 const(x'33');

...

0028 Dcl-C F23 const(x'bb');

0029 Dcl-C F24 const(x'bc');

0030

0031 // Other attention keys

0032 Dcl-C Clear\_Key const(x'bd');

0033 Dcl-C Enter const(x'f1');

0034 Dcl-C Help const(x'f3');

0035 Dcl-C PageUp const(x'f4');

0036 Dcl-C PageDown const(x'f5');

0037 Dcl-C Print\_Key const(x'f6');

0038 Dcl-C Auto\_Enter const(x'50');

Figure 4B. External source member to define the INFDS data structure and its constants.

Narrative for several chunks of the RPG follows:

Line Description

0001 Define compile time options for the program.

0002 – 0012 Declare the workstation file and two disk files (CustomerL2 is keyed by customer name and number and CustomerL1 is keyed by customer number only). The workstation file continuation on line 4 registers this RPG program with the Open Access API. This is the RPG program’s *only* nod to it not using a traditional display file.

0014 – 0015 A very cool RPG feature, these lines provide fully qualified key definition data structures for the two disk files. These one-line shortcuts serve as KLIST alternatives (albeit vastly more programmer friendly).

0017 – 0018 Two global variables for the program (local variables in procedures substantially reduces global variable dependence).

0020 Include the INFDS copy member (from Figure 4B) and its constants. This enables the very old-school trick of interrogating the hex value of INFDS’s position 369 to determine what function (or special) key was pressed. Positions 260-270 also define and interrogate the last-written display file format as “CurrentFormat”. This field is instrumental in guiding this program’s logic. (As an aside, I was amazed to realize that with Mobile RPG and IBM’s Open Access API this INFDS data structure feature works just fine! The INFDS data flows naturally out of, and back into, the RPG program through Open Access.)

0022 – 0034 Declare a fully qualified data structure named “Action” that maps the function key constants (declared in Figure 4B’s copy member) to semantic actions. This lets the programmer think not about function key presses, but user actions selected (such as a list’s chevron having been tapped). My longer term plan for this data structure is to map reusable and generic “actions” to function keys and then move that definition to the Figure 4b copy member. Note that a couple of these actions are defined multiple times. This is for those times when a “Back” action is different from a “Cancel” action (or example) semantically but both are implemented with an F2 key press. I was surprised to find what appeared to be keyword type conflicts with some field names in RPG data structures. For example, the ILE RPG does not at all tolerate an action being named “Next,” to my great disappointment. This is why there is a “More” action; I’d sure rather have a “Next” action.

0036 – 0038 Mainline code to bootstrap the program. These three lines populate the initial customer list. There aren’t any subroutines in this program—those have been dispatched and replaced with procedures. Thanks to TR 7.1’s free format procedure declaration, you no longer need to bleed to death to remember how to declare a procedure and its interface.

0040 -0104 The main loop of this program. This loop is wayyyy too long to be considered good code in anyone’s book. I sacrificed less code quality here for slightly shorter source listing and less eyeball hopping required to comprehend the code.

0042 – 0049 This is the point in the narrative where, were I explaining this to my father he’d throw up his hands and say, “enough about the labor pains, show me the baby!” This is where the fully qualified data structures, mapping function keys to semantic actions, and RPG automatically tracking the most recent record format really pay off. While it’s really a stretch, this RPG code almost presents itself as three controllers for three routes (the three record formats), each having at least one action. You can easily see that for the CUSTLIST format, the program allows three actions: ItemTapped, ChevronTapped, and More. These three actions (and the Exit action) are the actions available on the CUSTLIST panel.

0046 – 0046 You didn’t get to write RPG like line 46 when Saturday Night Fever was number 1 at the box office! This line calls a procedure that accepts a single integer argument and returns a Boolean value which indicates if a record was read. I’ve left the error handling as an exercise to the reader (That’s usually what writers says when they don’t know how to do something! In this case, I promise, it’s to minimize the code sample).

0074 – 0075 When the CUSTINFO panel is displayed (Figure 4B) and the OK button is tapped, the program needs to return to the customer list (Figure 4A), but reposition it to the customer just updated. RefreshCustList is a procedure that does just that. It is passed the customer name and number for which to position the list).

0113 – 0143 The LoadCstLst procedure loads up to the constant MAXROWS value of rows into the subfile displayed in Figure 4A. Note the variable RowCount is declared locally (so therefore it isn’t available to the rest of the program) and don’t forget that disk files can also be declared locally to procedures. Otherwise there isn’t much RPG here of any major interest, but it is interesting to note that this journeyman RPG is actually populating a list to be displayed on a smartphone.

0156 – 0163 The ReadCustById reads a customer record from the CustomerL1 file by customer number. Line 157 declares the procedure’s interface (where \*N is a much-needed shorthand for the procedure name). If an RPG data type is declared after the \*N, that is the type that the procedure returns. There must be Return opcode with a variable of that type in the procedure to return the value to the caller. Any variable declarations between the Dcl-Pi and the End-Pi define required arguments that must be passed to the procedure. For simple procedures (those that effectively take the place of subroutines) the Dcl-Pi/End-Pi declarations can be omitted.

There is more to explore in the RPG presented that what’s covered in the narrative, but it all follows the patterns and explanations provided in the code narrative.

This article intended to show TR 7.1 free-format ILE RPG in action and also that, with ASNA Mobile RPG, you can put that ILE RPG to work building a vastly non-traditional but very powerful new application type for your IBM i.

IBM deserves a big tip of the hat for finally coming ‘round and providing us with a powerful, expressive RPG syntax. The IBM i is much better for it and you’ll be a much better RPG programming using it. Please, please, investigate this new syntax. It is very much worth breaking out of your old school RPG comfort zone!

Resources for learning TR7.1 ILE RPG

[Four Reasons RPG Geezers Should Care About The New Free-Form RPG, Jon Paris in IT Jungle](http://www.itjungle.com/fhg/fhg021214-story01.html) (or anything else that Jon Paris or Susan Ganter have written about RPG—get your Google on! )

[Simon Hutchinson’s RPGPGM.COM blog](http://www.rpgpgm.com/)