CodePort: Preserving Existing Programs

Revised 3 January 2011

Disclaimer

The procedures described in this document have been designed and implemented by AddonSoftware and have received limited testing by the AddonSoftware staff. The CodePort utility is constantly being upgraded to address new issues as they are encountered, so some elements of this document may be obsolete. Please forward any issues and/or suggestions to AddonSoftware.

Introduction to CodePort

The CodePort utility is intended to provide a tool for migrating existing AddonSoftware Version 6.x or 7.x to the standards used in the Barista version. CodePort expects to use a compiled PRO/5 program as input and further expects that the program generally complies with the AddonSoftware standards and conventions for program code.

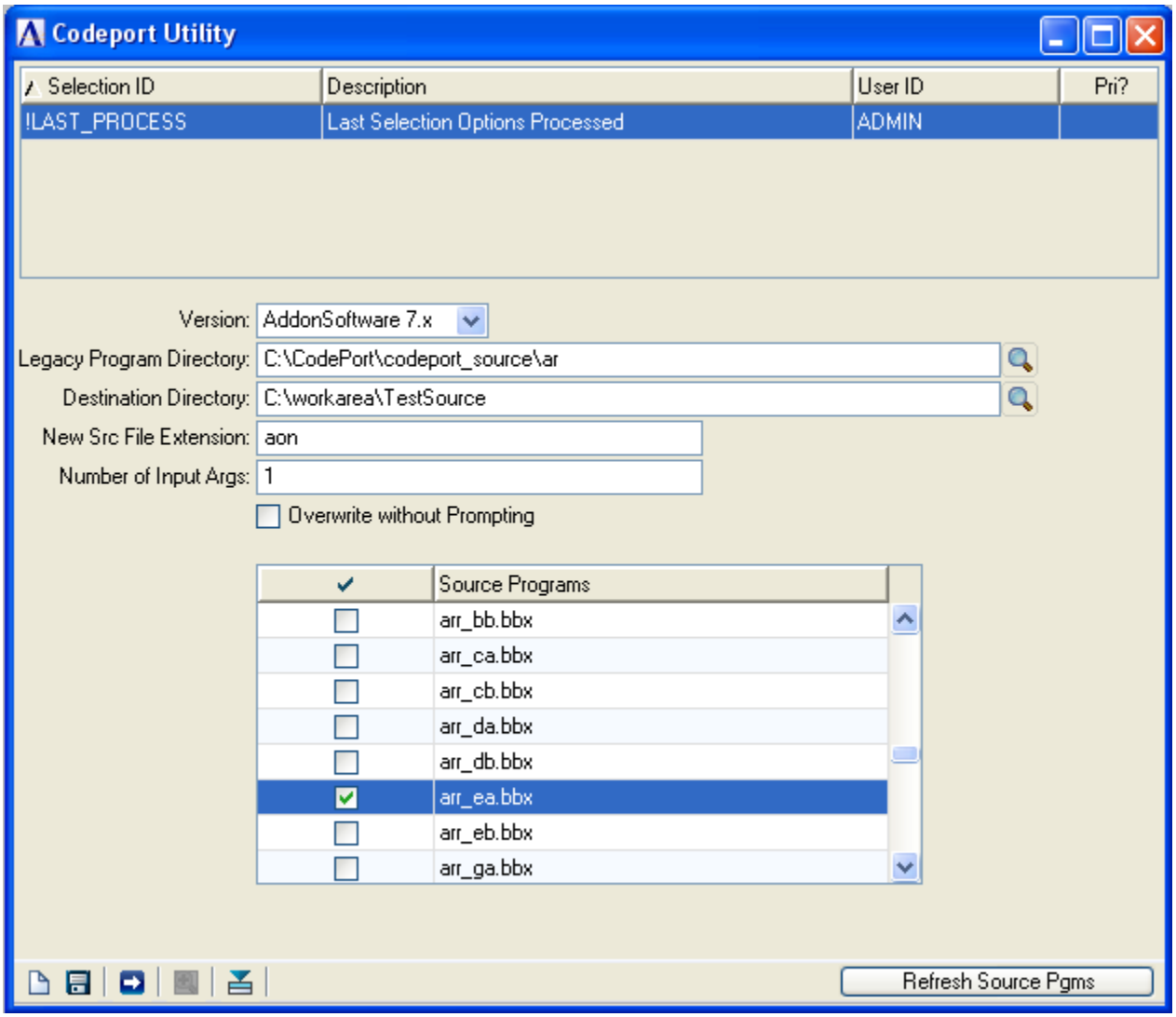
Running the CodePort Utility

Run the CodePort Utility from the AddonSoftware Administration -> Utilities menu.

You'll need to know where your PRO/5 tokenized Addon Version 6 or 7 code resides. CodePort uses the compiled code as input, and creates an ASCII program source file containing the converted code.

CodePort performs several tasks to modernize the code and convert it to Barista standards. You can learn more about the conversion process and program flow in the CodePort Utility Program Flow document. In this document we'll convert a program and illustrate typical post-conversion edits.

The CodePort Option Entry form appears in Figure 1 below. Because CodePort uses a Barista Option Entry form, you can double-click the LAST_PROCESS or other saved-selection options to re-use previous selection criteria.



- Version: Select AddonSoftware 6.x or 7.x; this selection controls what sorts of text replacements take place in the selected program file(s).
- Legacy Program Directory: browse to the directory containing the tokenized program file(s) to convert.
- Destination Directory: browse to the directory where new program source files should be saved. You can create a new directory from the directory chooser, if the destination directory doesn't already exist.
- New Src File Extension: this defaults to "aon." Use a different three-character extension if desired.
- Number of Input Args: this entry determines the number of placeholders that will be created in the new source file for gathering any user input. Use this if you are converting something such as a report -- you will typically not convert the "pick screen" program, but only the overlay. The legacy user selections will be replaced by a Barista Option Entry form, which will run the program overlay. Once the Option Entry form is created, you can come back to thse placeholder statements and fill in the actual field names from the form.

- Overwrite without Prompting: As a courtesy, CodePort will always warn you before it begins if the Destination Directory is not empty. This checkbox provides a further override so you replace files that already exist without interactive prompting. If not selected, CodePort will ask if you want to overwrite each time it encounters a file that already exists in the Destination Directory.
- Source Programs: select one or more files for conversion from the Source Programs grid. Use the Refresh Source Pgms button to reload the Source Programs directory if you want to uncheck all selected programs, if you have changed the Version and/or Legacy Program Directory, or if you have automatically set run-time options using the Saved Selections grid.

Press the Run Process button (or F5) to begin the conversion process. Each program typically completes in a few seconds. As it runs, the log file (see the <install>/aon/util/codeport/logs/ directory) is updated, and the progress meter also displays conversion status. When the conversion is finished, the progress meter will display “process completed” and wait for you to press the OK button to dismiss CodePort.

Post-Conversion Changes

The CodePort utility can automate several of the required steps toward producing code that will run in Barista. Other post-conversion modifications are discussed below. Make these modifications to the code using a text edit. Note that CodePort will help by listing any errors or exceptions encountered during the conversion as comments at the top of the program. It will also show any IOLIST's removed as part of the conversion.

```
rem --- Customer Name & Address Listing
rem --- Program converted from arr_db, 03Jan2011
rem --- Created by Codeport Utility (01/03/2011 02:14:46)

rem --- AddonSoftware
rem --- Copyright (c) 1981-2011 AddonSoftware
rem --- All Rights Reserved

rem -------------- Errors, Warnings and Conversion Information ----------------------

rem --- The following error(s) were encountered during the conversion:

rem --- v7.x administrator reference ["syc_ig.bbx"] (Line 0277)
rem --- BBx reference ["syc_ig.bbx"] (Line 0277)
rem --- v7.x administrator reference ["syc_es.bbx"] (Line 0284)
rem --- BBx reference ["syc_es.bbx"] (Line 0284)

rem --- The following IOLIST's were removed from this program:

rem --- arm01a: iolist a0$(1),a1$(1)

rem --- The following channel references have been identified:

rem --- ars01a: (Generated by CodePort)

rem ------------------------------------------------------------------------------------
```

In our example, CodePort reports two errors stemming from a #include of the standard input routine. Since our input will be coming from a separate Selection Form, the standard input routine code can be removed from this version. CodePort also tells us that it has replaced an IOLIST used for the Customer Master file. In addition we see a note identifying a channel reference to the AR Parameter file.

Always remember to check the initial lines of code to be sure that any BEGIN (or similar) statements are removed. CodePort also replaces all program code from line 8700 through the "end" statement of the program and replaces those sections (functions, error routine, program end).

Any custom code after the "end" verb (i.e., in statements 10000 and beyond) is saved in a separate vector during processing and placed back at the end of the new source file. Note that CodePort does not attempt any modifications or replacements on custom code except removing line number references (via the "_label" utility) and converting to lower case.

CodePort will automatically insert code to open files and retrieve the record templates for any IOLIST's encountered during the conversion.

CodePort has already supplied the code to open the Customer Master and AR Parameter files, and to dimension the associated string templates.

```
rem --- Open/Lock files

    files=2,begfile=1,endfile=files
    dim files$[files],options$[files],ids$[files],templates$[files],channels[files]
    files$[1]="arm-01",ids$[1]="ARM_CUSTMAST"
    files$[2]="ars_params",ids$[2]="ARS_PARAMS"
    call pgmdir$+"adc_fileopen.aon",action,begfile,endfile,files$[all],options$[all],
:                                   ids$[all],templates$[all],channels[all],batch,status
    if status goto std_exit
    arm01_dev=channels[1]
    ars01a_dev=channels[2]

rem --- Dimension string templates

    dim arm01a$:templates$[1],ars01a$:templates$[2]
```

The next piece of code has been place by CodePort in anticipation of fields coming from the Selection Form or other initialization program. Leave this code as is for now:

```
rem --- Assign form input values to local variables

    value_01$=Option!.getOptionData("form_value_01")
```

The rest of the post-conversion work will be the most labor-intensive, and involves five main classes of modification. Much of this will quickly become routine, but there will no doubt be times when you'll need to refer back to the original Version 6 or 7 code and file layouts.

- Identify variables that correspond to data supplied by the Selection Form or other initialization program, and note them as REM's in the section of code depicted above.
- Resolve IOLIST elements to their templated names. While CodePort will define any used templates and replace the related I/O statements, it does not replace IOLIST variable references with their templated names. This must be performed manually and is generally the most time-consuming part of the conversion. While not strictly required (you could for example move the templated record into the old IOLIST variables) it is strongly recommended. Once performed, the need to modify programs when new fields are added or field lengths change is either eliminated or greatly reduced.
- Eliminate GOTO statements, wherever possible, by using while/wend or other structured syntax.
- Remove/replace the statement number labels placed in the code by CodePort. While purely optional, it's recommended that you replace the arbitrary and non-intuitive labels generated by the utility with more meaningful names. Taking the time (and it's not much time) to perform this simple search and replace will make things easier for you in the future.
- Remove other unneeded code (such as the standard input routine in this example).

Incoming variables:

First, we'll look at the initialization section of the program to see what variables we're setting, and which ones should be coming from the Select Form:

```
rem --- Initializations

    dim a0$(8),a1$(325),headings$[3]
    h=0
    h1$=n1$
    h2$=n3$
    l9=59
    l=l9+1
    a0$(1)=firm_id$
    headings=3
    width=132
    clock$=""
    m0=len(m0$)
    m8$="(###)-###-####"
    m7$="00000"
    headings$[0]=h1$
    headings$[1]=h2$
    if p[2]=9 m7$="00000-0000"

rem --- Open printer, position file

    call pgmdir$+"adc_printer.aon",printer_dev,1,"","",status
    if status goto std_exit


    if p9$="A" p8$="A"+p8$; if p7$<>"" p7$="A"+p7$
    read (file_dev,key=firm_id$+p8$,dom=l1000)
    k0$=firm_id$+p8$
    goto l1100
```

We can fairly quickly ascertain that n1$, n3$, mo$, p[2], p7$, p8$ and p9$ are already known when this program executes, as well as the device channel file_dev. Create a REM in the "assign form input values to local variables" section of code noting these variables.

IOLISTS

Now let's look at the main section of the code, with an eye toward replacing IOLIST variables with their string template counterparts:

```
l1000: rem --- Report loop

    k0$=key(file_dev,end=done)
    if pos(firm_id$=k0$)<>1 goto done
    if p7$<>"" if k0$(3,len(p7$))>p7$ goto done
    if p9$<>"A" goto l1100
    if k0$(3,1)<>"A" goto done
    call pgmdir$+"adc_progress.aon","S","","","",k0$(4,10),0,0,1,meter_num,status
    k0$=k0$(1,2)+k0$(14,6)
    read (file_dev)

l1100: rem --- Read customer master

    read record (arm01_dev,key=k0$,dom=l1000) arm01a$
    if p9$<>"A" call pgmdir$+"adc_progress.aon","S","","","",
:       fnmask$(a0$(3,p[0]),m0$),0,0,1,meter_num,status
    if l+1>l9 gosub report_heading
    address$=a1$(31,72)+a1$(179,48)+a1$(103,9)+a1$(265,24)
    call pgmdir$+"adc_address.aon",address$,24,5,9,24
    print (printer_dev)fnmask$(a0$(3,p[0]),m0$)," ",fnmask$(a1$(112,10),m8$)," ",
:       a1$(1,28)," ",address$(1,24)," ",address$(25,24)," ",address$(49,24)
    l=l+1

rem --- End of report loop

    goto l1000
```

We read the Customer Master file (arm-01), not using the old IOLIST, but the string template variable we dimensioned earlier. By consulting both the legacy and Barista table layouts, we can change the code to use string template (<table>.<field>) notation:

```
l1100: rem --- Read customer master

    read record (arm01_dev,key=k0$,dom=l1000) arm01a$
    if p9$<>"A" call pgmdir$+"adc_progress.aon","S","","","",
:       fnmask$(arm01a.customer_id$,m0$),0,0,1,meter_num,status
    if l+1>l9 gosub report_heading
    address$=arm01a.addr_line_1$+arm01a.addr_line_2$+arm01a.addr_line_3$+
:       arm01a.addr_line_4$+arm01a.city$+arm01a.state_code$+arm01a.zip_code$
    call pgmdir$+"adc_address.aon",address$,24,5,9,24
    print (printer_dev)fnmask$(arm01a.customer_id$,m0$)," ",fnmask$(arm01a.phone_no$,m8$),
:       " ",arm01a.customer_name$," ",address$(1,24)," ",address$(25,24)," ",address$(49,24)
    l=l+1
```

GOTO's/Labels

Now we'll tackle the GOTO's and CodePort's line labels. We'll replace the main loop with a while/wend structure that uses symbolic labels (break), eliminating all but one goto. For that label, we'll replace the non-intuitive l1100 with next_arm01a:

```
rem --- Report loop

    while 1
        k0$=key(file_dev,end=*break)
        if pos(firm_id$=k0$)<>1 then break
        if p7$<>"" if k0$(3,len(p7$))>p7$ break
        if p9$<>"A" then break
        if k0$(3,1)<>"A" then goto next_arm01a
        call pgmdir$+"adc_progress.aon","S","","","",k0$(4,10),0,0,1,meter_num,status
        k0$=k0$(1,2)+k0$(14,6)
        read (file_dev)

rem --- Read customer master
    next_arm01a:

    read record (arm01_dev,key=k0$,dom=*break) arm01a$
    if p9$<>"A" call pgmdir$+"adc_progress.aon","S","","","",
:       fnmask$(arm01a.customer_id$,m0$),0,0,1,meter_num,status
    if l+1>l9 gosub report_heading
    address$=arm01a.addr_line_1$+arm01a.addr_line_2$+arm01a.addr_line_3$+
:       arm01a.addr_line_4$+arm01a.city$+arm01a.state_code$+arm01a.zip_code$
    call pgmdir$+"adc_address.aon",address$,24,5,9,24
    print (printer_dev)fnmask$(arm01a.customer_id$,m0$)," ",fnmask$(arm01a.phone_no$,m8$),
:       " ",arm01a.customer_name$," ",address$(1,24)," ",address$(25,24)," ",address$(49,24)
    l=l+1

rem --- End of report loop

    wend
```

Finally, we'll scan through the code and remove anything we know is no longer needed, in this case the standard input routine.

Sort Files

In the Barista version of Addon, most sort files have been replaced by secondary indices on the source file. If the code that you're porting uses an existing sort file record, check the Barista file layouts (Barista Development -> Table Layout Inquiry) to see if the sort record has been replaced.

Improved Program Structure

Moving to BBj introduces many new programming structures that aren't available in PRO/5. Many of them can be incredibly useful. Structures such as if/endif, while/wend and switch/case can improve the readability and flow of code, decreasing or eliminating the need for goto statements. In addition, symbolic labels such as next, break, and continue are highly recommended wherever possible. See the BASIS help docs for more on these structures.

Tidying up…

In addition to the newer coding structures, using indentation, white-space, comments, and breaking long lines with the colon (":") continuation character will all help to create source that is easy for other developers or support reps to read and understand. If you aren't familiar with using the colon for long lines, see the BASIS help docs.

Last but not least, don't forget to remove any error or exception comments at the top of the program. You might also consider removing any unreferenced functions.