- DocOut -

Interfacing Reports with the Barista Document Management System

By Chris Hawkins

Introduction

'AddonSoftware by Barista' takes advantage of Barista®'s versatile Document Management System knows as DocOut. With DocOut, users can preview reports on the screen, save them in any of several formats – Portable Document File (.pdf), tab-delimited text (.txt), or comma-separated text (.csv), etc. – and also print or send as an e-mail or fascimile.

Developers can easily convert any report that currently prints in a well-defined tabular format to use DocOut. This tutorial describes how to alter a traditional report program, that is, a report that prints directly to a print channel, into a program that interfaces with DocOut.

Overview

DocOut report programs use a series of vectors and string arrays to store all of the output data for a report rather than printing lines directly to a printer or file. The report program, therefore, does not actually print, but simply builds the necessary vectors and string arrays.

When the report program finishes, it runs the Barista public `bas_process_end.bbj`. Aside from other housekeeping functions, `bas_process_end.bbj` checks to see if the main output vector contains data. If so, it calls the DocOut utility, which processes and then presents the formatted output in the DocOut display window. When the user closes the DocOut window, `bas_process_end.bbj` either runs the next overlay or terminates in the usual manner. The next overlay is optional, and typically is used in register/update pair of programs.

From the DocOut display window, users can adjust print settings such as column or font size, as well as generate a print version or any of several file versions of the report. With appropriate setup on the users' system, they can send the saved files as an e-mail or fax.

DocOut Basics

String Arrays

DocOut reports must use the `headings$[]` and `columns$[]` string arrays for report and column headings, respectively. Dim the `headings$[]` array to number of banner lines you want such as the report title, from/to customer name, etc. (*most Addon reports use this array already*). If you are using forced page breaks and subheadings (*more on that later*), dim `headings$[]` with an additional item and set it to

```
headings$[n]= "[HEADER]"
```

so DocOut can replace the `"[HEADER]"` token with the actual subheading as it processes data in the various vectors.

Dim the `columns$[]` array to have 12 slots for each report column. For each column, initialize the column heading text, column type (C=character, N=numeric), size (number of characters), formatting mask (if numeric), and whether or not to underline the column as part of a total row. Although not required by DocOut, use a numeric variable such as "columns" to specify the number of columns in the report. This makes it easier to add blank lines and use the BrkVect! and TotVect! vectors, all described below.

> ***Important***: Name these arrays `headings$[]` and `columns$[]`. In addition, the second coordinate in the `columns$[]` array is fixed: `[n,0]` is the column heading text, `[n,1]` is the column type (C/N), `[n,2]` is the column size in characters, `[n,3]` is the mask, if applicable, and `[n,10]="T"` if this column is underlined in a total row.

The following parameters are available for the columns$[] array:

[col,0]	= Header text
[col,1]	= Data type
[col,2]	= Column width original - Characters
[col,3]	= Output mask
[col,4]	= Control type on form
[col,5]	= Justification (L, R or C); overrides default based on [col,1] type
[col,6]	= Column width defined by Inquiry/Query export (reserved by Barista)
[col,7]	= Truncated data length
[col,8]	= Column width adjusted - Pixels
[col,9]	= Column width adjusted - Characters

[col,10] = Column flags. One or more of the following:

"T" = Underlined total column

"1-9" Repeated data on new page. (By row: 1=first row, 2=second row, etc.)

[col,11] = Number of columns to span header text

[col,12] = Not used

The following code shows how we might create `headings$[]` and `columns$[]` for a simple customer report:

```
rem --- Retrieve sysinfo data

    sysinfo_template$=stbl("+SYSINFO_TPL",err=*next)
    dim sysinfo$:sysinfo_template$
    sysinfo$=stbl("+SYSINFO",err=*next)

rem --- Document headings

    dim headings$[3]
    headings$[0]=sysinfo.firm_name$
    headings$[1]=sysinfo.task_desc$
    headings$[2]="Customer Report"
    headings$[3]="[HEADER]"

rem --- Document columns

    columns=3
    dim columns$[columns,10]
    columns$[0,0]="Customer",columns$[0,1]="C",columns$[0,2]="10"
    columns$[1,0]="Name",columns$[1,1]="C",columns$[1,2]="30"
    columns$[2,0]="Phone",columns$[2,1]="C",columns$[2,2]="15"
    columns$[3,0]="Balance",columns$[3,1]="N",columns$[3,2]="15",
:       columns$[3,3]=m$,columns$[3,10]="T"
```

You can create multiline headings by using the `^` character in the `[n,0]` element of the `columns$[]` array as follows:

```
columns$[0,0]="Customer^Number"
```

formats the column headings like:

```
Customer
```

```
  Number         Name                          Phone          Balance
```

Enter the alignment for the heading within the column: C=center, R=right, L=left. The default is R, right justified. For multi-line headings, use ^ (Shift-6) to separate alignments for each line. For example, for a 3-line heading, entering C^L^R will center the first line, left justify the second line, and right justify the third line.

The above example uses simple literals to create the column headings. Addon reports use a different syntax that resolves a column heading *key* to the appropriate *translation* for the language in use by the client. The key/translation pairs are stored in translation bundles - a series of flat files containing translations for the desired languages. See the BBjabber section of the Barista Language Utilities document to learn how you can localize DocOut reports.

Vectors

A BBj vector is an array, the elements of which can contain any allowed data type or object (see BBjVector for more information). All of the vectors used in DocOut contain string data only.

> ***Important***: The formatting and presentation of DocOut reports are in grids (*think of a spreadsheet*). Every report row must contain the same number of fields/cells to avoid a wrapping or stair-stepping effect. Therefore, when adding data to the main report vector, you always need to add enough items – even if some are blank – to constitute a complete row.

Place the following piece of code in the initialization section of a report program to create the vectors and set up some other standard strings for DocOut:

```
rem --- Document initializations

    OutVect!=bbjAPI().getSysGui().makeVector()
    HdrVect!=bbjAPI().getSysGui().makeVector()
    BrkVect!=bbjAPI().getSysGui().makeVector()
    TotVect!=bbjAPI().getSysGui().makeVector()
    rep_date$=date(0:"%Mz/%Dz/%Yd")
    rep_date_stamp$=date(0:"%Yd%Mz%Dz")
    rep_time$=date(0:"%hz:%mz %p")
    rep_time_stamp$=date(0:"%Hz%mz%sz")
    rep_prog$=pgm(-2)
```

<u>OutVect!</u>

OutVect! is the main vector, containing the majority of the report data. Replace lines that print data to a printer channel with code that adds the data to OutVect!:

from

```
print (prt_chan) @(10),cust_rec.cust_id$,
:       @(20),cust_rec.name$,
:       @(50),cust_rec.phone$,
:       @(65),str(cust_rec.balance:m$)
```

to

```
OutVect!.addItem(cust_rec.cust_id$)
OutVect!.addItem(cust_rec.name $)
OutVect!.addItem(cust_rec.phone$)
OutVect!.addItem(str(cust_rec.balance))
```

Remember, since you need to add a complete row's worth of items to the vector for each row, you may have some blank items. Add blanks to the vector one at a time like this:

```
OutVect!.addItem("")
```

or by creating a function like this:

```
def fnblank(q0)
    for q1=1 to q0
        OutVect!.addItem("")
    next q1
    return q1
fnend
```

and using instead of `OutVect!.addItem("")` in this manner:

```
x=fnblank(3)
```

<u>BrkVect! and HdrVect!</u>

Many reports are built to break and start a new page when certain data changes. DocOut uses BrkVect! and HdrVect! to determine where in OutVect! a break should occur and what heading information should appear at the top of the new page. For example, suppose we have a report that breaks when the customer changes. The traditional code may look like this:

```
customer_break:
       while old_cust_id$
             print (prt_dev)@(80),"Total for: "+old_cust_id$,@(100),cust_tot
             break
       wend
       read record (cust_chan,key=new_cust_id$) cust_rec$
       gosub report_headings
       print (prt_dev)@(5),"Customer: "+cust_rec.cust_id$+" "+cust_rec.name$
return
```

In a DocOut version of the report, we would alter this code slightly:

```
customer_break:
       while old_cust_id$
             OutVect!.addItem("")
             OutVect!.addItem("")
             OutVect!.addItem("Total for: "+old_cust_id$)
             OutVect!.addItem(str(cust_tot))
             break
       wend
       read record (cust_chan,key=new_cust_id$) cust_rec$
         BrkVect!.addItem(str(OutVect!.size()/(columns+1)))
       HdrVect!.addItem("Customer: "+cust_rec.cust_id$+" "+cust_rec.name$)
return
```

The calculation for BrkVect! tells DocOut at what point to force a page break. Here, we are using the zero-based variable "columns" to store our row size so "columns" of 3 indicates that each row holds 4 cells/fields. If the size of OutVect! is 60, then setting BrkVect! to 60/4 tells DocOut to begin a new page after row 15. The "[header]" token loaded in the `headings$[]` array is replaced with the data in HdrVect! corresponding to the row calculation stored in BrkVect!. You might also notice that when printing the subtotal line, we added some blanks to OutVect! in order to construct a complete row.

If you want to force a page break but are not concerned with a subheading, or you are including your subheading information in OutVect!, then you can use BrkVect! without HdrVect!. You cannot, however, use HdrVect! alone, because without BrkVect!, DocOut has no idea where to place the HdrVect! data.

Important: The DocOut preview window displays as one continuous grid with a scrollbar. Report headings are at the top of the window, outside of the grid. As such, the preview window will not show subheadings (unless you explicitly repeated them in an OutVect! row), but they will be incorporated into print and PDF output.

TotVect!
If a report contains underlines, as is often the case when printing totals, use TotVect! in conjunction with `columns$[n,10]="T"` to instruct DocOut to underline (in print or PDF output) specified columns. TotVect! is set the same way as BrkVect!; it contains a calculated row count where the underlining should occur. When DocOut processes a row specified in TotVect!, it underlines any column marked with a `"T"` in the `columns$[]` array. Given the following in the `columns$[]` array,

```
columns$[3,0]="Balance",columns$[3,1]="N",columns$[3,2]="15"
columns$[3,3]=m$,columns$[3,10]="T"
```

we can add a line to our customer break routine to underline the customer balance line prior to printing the subtotal as follows:

```
customer_break:
       while old_cust_id$
             TotVect!.addItem(str(OutVect!.size()/(columns+1)))
             OutVect!.addItem("")
             OutVect!.addItem("")
             OutVect!.addItem("Total for: "+old_cust_id$)
             OutVect!.addItem(str(cust_tot))
             break
       wend
```

Special Formatting

In more complex situations such as header/detail reports, you might use a single column for both character and numeric data. In this case, tell DocOut how to position the data you are adding to OutVect!. As you add your data to OutVect!, you can apply bold, italics, or increase/decrease the font size by concatenating a 00 plus these formatting characters:

`B`	bold
`I`	italics
`R`	right justify
`L`	left justify
`C`	center
`+`	increase font size
`-`	decrease font size

For example, this entry makes the customer name appear bold and in italics:

```
OutVect!.addItem(cust_rec.name$")+$00$+"BI
```

When using special formatting, be sure to include this line of code near the end of your program, so DocOut knows to process OutVect! with special formatting in mind:

```
out_action$="FORMAT"
```

Running an Overlay

Frequently, once a report is finished, you would want to continue with an update program. With a report program that prints directly to a printer, you would most likely run the update program directly from the report. We need a different approach when interfacing with DocOut, since the report program itself does not create any output. We don't want to continue with an update until we handed control over to DocOut so the user can view and/or print the report. Use **`next_overlay$`** to specify a program that should be run after DocOut. When DocOut finishes, it runs the Barista public **`bas_process_end`**, which in turn runs **`next_overlay$`**. You can "chain together" several related programs in this manner.

Miscellaneous

Cleanup

Once converted to interface with DocOut, a traditional report program will likely have "leftovers" that you can remove, such as code to increment line counts, print report headings, and open the printer.

Non-tabular Reports

Some reports have a format that does not lend itself to a grid-type display like the Addon Vendor Detail Listing. In this case, set up your **`columns$[]`** array to be a single column and leave the description blank. Then, instead of replacing each item in the print row with an `OutVect!.addItem()` assignment as we saw in the example, dim a string, subscript your column heading and print data into it. Then add the string to OutVect! like this:

```
dim columns$[0,10]
columns$[0,0]="",columns$[0,1]="C",columns$[0,2]="80"
...
dim pbuf$(80)
pbuf$(1,10)="Cust  ID",pbuf$(15,10)="Name",pbuf$(50,10)="Phone",pbuf$(70)
="Balance"
OutVect!.addItem(pbuf$)
...
dim pbuf$(80)
pbuf$(1,10)=cust_rec.cust_id$,pbuf$(15,30)=cust_rec.name$
pbuf$(50,15)=fnphone$(cust_rec.phone$),pbuf$(70)=str(cust_rec.balance:m$)
OutVect!.addItem(pbuf$)
```

Note: The default font for DocOut is proportional. If you use this single-column method for any reports, they will only display and line-up correctly if you change the Print Settings for DocOut to a fixed-width font.

Report Width

DocOut does have some capability to merge columns when creating print or PDF output. Suppose you have a report that lists a customer ID and name on one line, with several periodic amounts on the line(s) below. It is possible that you will run out of room across the page if you size the name column large enough to accommodate the full name always. You have two options. First, you can keep the name column large and let DocOut split the report page in two when it prints or creates a PDF (*this is the same mechanism used by most spreadsheets when the data is larger than the print area*). Alternatively, you can design the report so that the customer name column is under-sized. This appears as a truncation in the DocOut display window, but when DocOut generates the print or PDF output, it allows the customer name to merge into the cells to the right, as long as they are empty.