DFW DQL EXPORTING

Special Export

DfW. DQL. Export To. Pretty it is not. Documented it is not (very well). Sentences back in proper word order,* Adrian Jones *finds it ugly – and useful.

IN A COMPETITION FOR "KING Kludge" in DfW, the DQL export command would likely feature high in many programmer's nominations.

The manuals explain quite poorly how the various aspects of this command work, and make misleading comparisons with DOS formatting commands.

Indeed, I understand it was a last-minute addition to the product. Yet even so, DQL exporting is actually quite powerful and fully featured, but suffers from poor explanation. Here I will attempt to put the record straight.

THE BASICS

A simple export from a single table without any grouping or other fancy bits is straightforward. This is the basic outline, which the documentation 'confirms':

```
For MyTable ;
  list records
    FieldA .
end

export to
"c:\path\filename.type" .
.items
@f[1,1]
.end
```

Okay, the "@f[1,1]" bit – which means get the first field from the first mentioned table – looks ugly, but it works. The ".items" marks where the values are to be put. The ".end" … well, it appears to end the items, or maybe the export command itself. But I'll look at what it may be doing in more detail later.

For the moment, bear in mind that you can put more text after the .end, but don't put any other commands here; the export must be the last thing this DQL does.

TABLE NUMBERS

So let's let go of the handrail and peer over the edge by exploring the table numbers. A DQL like this:

```
for Invoices ;
  list records
    InvoiceNo ;
    InvoiceDate ;
    all SeeInvoiceLines
        ProductName ;
    all SeeInvoiceLines Qty .
end
```

is actually referring to two tables – one called Invoices, and another called SeeInvoiceLines (which here is a pre-defined relationship, but could just as well be an ad-hoc) – and would have this export section:

```
export to "myfile" .
.items Invoices
@f[1,1] @f[1,2]
.items SeeInvoiceLines
@f[2,1] @f[2,2]
.end
```

First, we refer to the fields in the second table by the number '2' in the '@f' field description. Second, we NAME the items bands according to the table being referred to. Here, the name of the second table is the relationship name, not the actual table name.

The clue to what is going on – as it ever is with DfW – is to display the body for this procedure and to look at the query by model dialog. Here you will see a crude ERM-like diagram, with each table labelled as above.

(Note also that there is only one '.end'.)

OTHER TABLES

Happy so far? Now let's get confusing!

If you add something like this to the top of the previous DQL:

```
define temp "tCount" .

tCount := count of
    AnotherTable  .

For Invoices, etc, as above
```

and run the export, you'll now find that the file is littered with the words "Invalid Field Number". You'll also see that you now have the invoice details appearing where previously you had the invoice items.

For once, the QBM dialog is

Special Characters

To embed non-ASCII characters and sequences in the value, you an still use the DOS-like method of '@[' followed by the two-character hex value of that character. If you want to chain a series together, put a space between each set of two characters. For example:

```
@[44 69 61 6C 6F 67 75 65]
```

will spell out "Dialogue", with no spaces.

Be careful not to confuse the field format ('@f[') with this hex formatting ('@[' – i.e. no 'f'), or you may end up with invalid table number errors, and sometimes GPFs.

I've not tested this with ESC sequences sent to the printer, as my current printer only works through Windows, but I see no reason why this will not work.

The README has for a while advocated that a space be put between the fields in an export. This is not true, and formats such as:

```
@f[1,1]~@f[1,2]
```

which produce tilde-separated, variable length files quite happily.

If you want fixed length exports, you can add a third number to the field specification to indicate the field length, thus:

```
@f[1,1,20]
```

If the field is text or similar, spaces are added to the right to pad the value. If the field is a number, the spaces are added to the left. If the value is bigger than the specified number of characters, it will be truncated.

END OF THE LINE

The end of line suppress symbol from DOS – '\' – also works, so:

```
.items
@f[1,1] \
.end
```

will produce the output as a single line, with each value separated by a space. Note that the backslash must be the last character on the line, otherwise it is treated as a regular character.

If you want a backslash as the last character, without suppressing the line, code it as it's hexadecimal equivalent ('@[FC]') instead.

more reserved when it comes to the answer. The fact is, "count of AnotherTable" creates a reference to another table, and 'knocks' the table numbering on by one, even though that table does not form part of the list records. So the export section should read instead:

```
export to "myfile" .
.items Invoices
@f[2,1] @f[2,2]
.items SeeInvoiceLines
@f[3,1] @f[3,2]
.end
```

Now try:

```
For Invoices
  list records
    any GetCustomer
        CustomerName ;
    InvoiceNo ;
    InvoiceDate ;
    all SeeInvoiceLines
        ProductName ;
    all SeeInvoiceLines Qty .
  end
```

you'll find that Invoices – and the three fields InvoiceNo, InvoiceDate and 'any GetCustomer CustomerName' – belong to table number 1, but the SeeInvoiceLines fields are moved to table 3. This is because the first tables referred to is Invoices, the second is GetCustomer, and third, SeeInvoiceLines.

It's not a bad idea to add commented-out numbers to the right of your DQL to help clarify what belongs where – and where a table, relationship or group is used.

GROUP WORK

In the last issue, I explored how listing records in groups is tackled (Dialogue Autumn 2000, pages 9-12). Effectively, each group becomes a 'virtual' table, which will also affect the table numbering.

Thus:

```
for Invoices ;
  list records
    CustomerID in groups ;
    InvoiceNo ;
    InvoiceDate .
end
```

has two tables – Invoices and 'group: CustomerID' (see that article for details on how the grouped table is named). CustomerID here belongs to the Invoices table, and InvoiceNo and InvoiceDate to 'group: CustomerID', so the export should look like:

```
export to "Myfile" .
.items Invoices
@f[1,1]
.items group: CustomerID
@f[2,1] @f[2,2]
.end
```

If you add further groups, you'll continue to increase the table count.

For the moment, one final example. What about:

```
for Invoices ;
  list records
    any GetCustomer
        CustomerName ;
    CustomerID in groups ;
    InvoiceNo .
end
```

Invoices is table 1, and contains the fields 'any GetCustomer CustomerName' and CustomerID. GetCustomer is table 2, but doesn't contain any actual fields. Group: CustomerID is table 3, and holds InvoiceNo.

This would still be the case if we'd decided to not group in CustomerID but instead on 'any GetCustomer CustomerName'. This time our grouped table is named 'group: any GetCustomer CustomerName', but it is still table no 3, and still will hold InvoiceNo.

Thus a full working export, making the unlikely assumption that grouping by customer name will provide meaningful data, might look like:

```
for Invoices ;
  list records
    any GetCustomer
        CustomerName in
        groups ;
    InvoiceNo .
end
 export to "Myfile" .
.items Invoices
@f[1,1]
.items group: any
GetCustomer CustomerName
@f[3,1]
.end
```

NOT DOS

Those dot commands look reassuringly DOS-like, don't they? And the nice help file refers to them as having "counterparts in character-based versions of DataEase".

Well, in some ways I suppose '.header', '.items' and '.trailer' are similar to DOS, but it is better to think of them in terms of the form and record objects (or containers) that belong in the body of a DfW document. In other words, the header and trailers are the form objects, and the items, the record object.

They work a bit like this:

```
.header [table/rel/group]
Top of the form object
.items [table/rel/group]
Each record
.trailer [table/rel/group]
Bottom of the form object
```

One key difference is that the '.item' ends the header, and the '.trailer' ends the items. Or rather, the next dot command ends the previous one.

You will also have noticed from previous examples that you can have several '.items', which you cannot in DOS. It's best to think of the .end ending the export command, I suppose in a similar fashion to the way that '.end' ends a DOS output section. It does not end the items.

As we have noticed above, if the DQL lists records from more than one table, you need to add the table name after the header/trailer/items. This name may be the name of the relationship, or the name of the group, as we've discussed.

There is, I think, one bug with regards to grouping. I have not been able to work out how you can have trailer information with a group. Sometimes nothing appears, more often the procedure GPFs.

SUMMARISING

You can use any statistical summary in the export by asking for that statistic in the list records section, and then inserting that word in the export section:

```
for Invoices ;
  list records
    InvoiceTotal : item sum .
end
export to "MyFile" .
.items
@f[1,1] @f[1,1 sum]
.form trailer
TOTAL: @f[1,1 sum]
.end
```

The inclusion of the summary field within the items section will here create a running total.

Note, however, that the length of this summary variable is currently determined by the length of the field it is adding up (the lengths are the same). If the total exceeds this length, the field will contain asterisks instead.

And unfortunately bugs in 5.5 about how temporary variables can be controlled in the body may prevent using temporary values instead, though hopefully these will be sorted out in the next patch.

You must put the summary variable in the trailer – or at least in a place after all its items have listed – otherwise it won't have yet got round to doing the summarising. If you want the total in the header, consider using a 'sum of' relational operator instead.

Finally, as I've just pointed out, the form trailer does not work for groups, which may render this statistical option fairly useless.

CONCLUSION

From the amount of things I've found that can be done with the export command, it must have been pretty extensively developed. But because a lot of how it works is at best poorly explained, and because of the crudeness of some of its syntax, it has in my mind until recently been delegated as something of an after-thought.

In fact, it can prove to be a pretty powerful tool. See Jim Chapman's article in this issue where he uses the export command to create HTML files, for example.

This article has not covered everything to do with exporting – but it is, I hope, enough food for thought. If you have any further ideas or comments on exporting – and on this article – please do get in touch!