

# Understanding DataEase

## DQL Exports

Graham Smith  
PLM Consulting, Inc.  
[gsmith@plmconsulting.com](mailto:gsmith@plmconsulting.com)  
October 2006

### Preface

To begin with, it is necessary to understand that (as has been explained elsewhere) DQL's were not planned for in the original specification of DataEase. Once the decision was made to add DQL's to DFW 5, then the challenge became finding a way to incorporate them.

As a matter of historical interest, there was never a specification that called for the inclusion of the DQL Export function, nor was there ever any real plan to include it in DFW 5.x, it just happened. The reasons for it happening are unimportant, but partly include my big mouth. For this reason alone, I feel a kinship to this and a need to find ways to make it work. Unfortunately, like many things that just happen, they don't always happen as expected or desired.

There is a lot that can and probably should be said about this subject, but ultimately it has largely a matter of trial and error for me because the documentation is a bit incomplete. I have some notes from Bill Woods (who wrote this code) that Pete Tabord (the chief developer) sent me that deal with the implementation. They have helped, but the most difficult thing in trying to document how exports work is that there are so many different variations on what people may try and do that there is no way to cover everything.

This, actually, brings up an interesting sidebar. Throughout the history of the DataEase product, the program developers are continually being surprised by the things that the application developers do with the program. Around 1991, Arun Gupta (founder of DataEase) toured a manufacturing plant of W. L. Gore and Associates (makers of GoreTex) and was dumbfounded that they were using DataEase to not only keep track of materials and inventory, but were using it to virtually run the whole manufacturing process. This is one of the reasons that users always seem to be one jump ahead of the program developers and why they are the ones that often are the first to find problems. On several occasions, when reporting a problem, I have been asked, "Yes, but why did you do that in the first place?"

One final note, the examples in this article were prepared and tested using DE6.52 and have not been tested with DE7 because of a bug in that version. While there is no reason to expect that they will not eventually work with DE7, that is not the case with versions prior to DE6.52. The reason is that there were some bug fixes relating to DQL exports in the two 6.5 patches.

### Getting Started

The DQL Export syntax was setup to coincide with the model, which is not a particularly easy task to either explain or understand. It's not too bad when only a single table is involved and there is no grouping, but can start to get confusing when either or both of these are introduced.

The general syntax involves creating a separate section of code after the regular DQL script itself. The DQL script must be complete and include an END which the export script follows. In other words, the export script must be appended to a completed query. Also, nothing can be placed after the export commands. Here is an example of a simple DQL export:

```
for TableA ;
list records
  FieldA in order ;
  FieldB .
end .
export to "filename.txt" .
.items
@f[1, 1]~@f[1, 2]
.end
```

The "export to" line specifies the export file name. This is the only command where the period (.) comes at the end of the line rather than the beginning. Including this line automatically turns off the output to screen or printer.

@f[1,1] denotes the placement of a field where n,n is the table and field. In most cases there will only be a single table, thus most of the time, this will be @f[1,n], with n being a number representing each field in the list records.

It is common practice in an export to include a header row that contains the field names. In a DQL export, any text that appears between the "export to" and the ".items" will show up in that relative position in the file that is created. For example, refer to the following DQL and the output:

```
for Customers ;
list records
  Name in order ;
  State .
end .
export to "customers.txt" .
Customer Name~Customer State
.items
@f[1, 1]~@f[1, 2]
.end
```

```
file: Customers.txt
Customer Name~Customer State
Acme Manufacturing~Delaware
Baker Corporation~Maryland
Compost, Ltd.~Virginia
```

In the above example, the tilde (~) is used as a field separator. Most programs that will be used to import data can accept a wide variety of characters as separators. But you must be sure that you use a character that is not included in the data. The most common field separator is a comma, but because of the possibility of this character being found in the data, it is necessary to enclose the fields with a text delimiter, usually a double quote (“). In that case, your code would look like this:

```
“@f[ 1, 1]”, “@f[ 1, 2]”
```

Since I am basically lazy, I usually opt for something like a ~ or | as a separator since they aren't going to be in the data and I won't have to type in all the annoying quotes. If I need to change it later, I can always do a search and replace (<-- good tip here).

## Multiple Tables

As was previously noted, most often you will only be dealing with one table. This is fortunate because once you get into multiple tables, things start to get complicated. Any DQL that creates multiple “containers” in the format will, by necessity, create multiple “bands” in the export. These bands are represented by the first number in the field designation: @f[BandNbr,FieldNbr]. It is also necessary to identify which band is being referred to by the various items and form header commands. Consider the following:

```
for TableA ;
list records
  FieldA in order ;
  FieldB .
for relatedTableB
list records
  FieldC ;
  FieldD .
end .
end .
export to “filename.txt” .
.items
@f[ 1, 1]~@f[ 1, 2]~@f[ 2, 1]~@f[ 2, 2]
.end
```

If you try and save this DQL, you will get an error telling you “This Export Band requires a TableView name.” What that is saying is that it needs to know which “table” the dot items is referring to. More precisely it needs to know which “level” of the query is being referred to. In this case you would want one row for each record in relatedTableB so you would specify it as .items relatedTableB

While rare, there are also cases where you want separate rows for both tables. In a case like that, you may need something as elaborate as the following:

```

export to "filename.txt" .
. form header TableA
TableA Header Info
. items TableA
@f[1, 1]~@f[1, 2]
. form header relatedTableB
relatedTableB Header Info
. items relatedTableB
@f[2, 1]~@f[2, 2]
. form trailer relatedTableB
relatedTableB Trailer Info
. form trailer TableA
TableA Trailer Info
. end

```

Even if you don't find a need for something this elaborate, it is still worthwhile setting up a test like this because it is very helpful in understanding how these various levels work. In particular, you may find that you can do some things that do not make sense from an organizational perspective but non-the-less still work. An example from the help file follows:

```

export to "filename.txt" .
. form header TableA
TableA Header Info
. items TableA
. form header relatedTableB
@f[1, 1]~@f[2, 1]
. end

```

Exactly what the author expected to accomplish with this is not certain, nor am I entirely certain what the output of a script like this would and should look like, the fact is that it does work. What it points out is that sometimes you can do unexpected things if you let your imagination loose.

It is also worthwhile to mention at this time that the reason that this syntax is so flexible is because the code behind it was placed into the DQL code itself at a point part way between the raw data from the multiview and all subsequent processing in such a way as to just "squirt the stuff out of the middle of the run-time parser (the action stack)". Because of this, it has no knowledge of anything about the formatting in the body and instead relies completely on the nature and order of the fields in the list records. This means that you cannot format the fields in the output. Integers and Fixed Point numbers will have commas and Floating Point numbers won't. The commas will create some problems for some programs when imported, so if you need to format the numbers, you can use NumToText. To convert a number to a float, create a numeric variable named "n" and replace the number field name in the list records with:

```
n + FieldName ;
```

This will cause the export to take on the format of first field in the expression (n) which is a floating point.

A final warning is that the more creative people get with DQL exports, the more likely they are to run across bugs that no one has seen before. For that reason, it is imperative that any new ideas be fully tested before being released upon the end user.

## Exports with Groups

Exporting groups is very similar to exporting multiple tables because adding a group to a DQL will create a second container in the body and thus a second band in the export, but the sequencing of the band numbers is not as clear as it is with multiple tables. See the following DQL for an example.

```
for Inventory ;
list records
  Commodity in groups named "grpComm" ;           --@f[1, 1]
  Name ;                                           --@f[2, 1]
  Siding_Code in groups named "grpSide" ;         --@f[2, 2]
  Shipper_Code in groups named "grpShip" ;        --@f[3, 1]
  Railcar in order .                               --@f[4, 1]
end .
```

The above DQL takes advantage of the ability to “name” the group (thanks to Adrian Jones for this invaluable tip). This makes specifying the band in the export easier. I also put a comment at the end of each field that lists the “table” number and the field number. Take careful note that the numbering of things in groups can be very confusing since it does not proceed in a linear fashion. Following is the syntax for the export section of this DQL:

```
export to "c:\temp\inven.txt" .
. form header grpComm
@f[1, 1]
. form header grpSide
@f[2, 2] @f[2, 1]
. form header grpShip
@f[3, 1]
. items grpShip
@f[4, 1]
. end
```

## The Joy of Numbering

Sorting out the numbering of the bands when there is more than one table involved can be a trial and error proposition. Even though I have done a number of these, I still have to stop and think about it before I get it right. Take a look the following example:

```

for Certificates
with DateIssued between tStartDate to tEndDate ;
list records
  any _Shareholders SSN ;
  any _Shareholders EmpID ;
  any _Shareholders Name ;
  ShareholderNbr in groups with group-totals named "nGrp" .
end .
export to "c:\temp\stocks.csv" .
. form trailer nGrp
"@f[1, 1]", "@f[1, 2]", "@f[1, 3]", @f[3, 1]
. end

```

**There are three tables involved here so why isn't there a @f[2,n] anywhere. Why are the first three fields @f[1,n] rather than @f[2,n] and why is the last field @f[3,n] rather than @f[2,n]?**

**I'm not positive I have the understanding of this completely straight in my head, but what it seems to be is that since the first three fields are related, the values are still counted as if they were a part of the main table. Think of the fields like virtuals where the values are being "pulled into" the table. nGrp, however, starts a new level in the model and it gains a new number because of that. The reason it is numbered 3 rather than 2 is that it is, in fact, the third band in the query because the ANY creates a band. Confusing, isn't it?**

### Other Eccentricities

**Here is another example where things don't behave exactly as you may expect them in to in a DQL export:**

```

for TableA ;
list records
  FieldA in groups named "nGrpA" ;
  FieldB in order .
end .
export to "filename.txt" .
. items TableA
@f[1, 1]
. items nGrpA
  @f[2, 1]
. form trailer nGrpSt
-----
. end

```

**This will produce an output where each group ends with a dotted line. You would expect that this same method could be applied to an example that does not use groups, but you**

would be wrong. There are some unfortunate anomalies which can make some exports an exercise in frustration. This is particularly true of how the dot headers and trailers work. Consider the following:

```
for TableA ;
list records
  FieldA in order ;
  FieldB .
  for relatedTableB
  list records
  FieldC ;
  FieldD .
  end .
end .

export to "filename.txt" .
.items TableB
@f[1, 1]~@f[1, 2]~@f[2, 1]~@f[2, 2]
.form trailer TableA
-----
.end
```

It could reasonably be expected that, based on the grouping export, this export would put a dotted line after each group. But in actual fact, it puts it before the items. Look closely and you will see that each group begins with a dotted line rather than ends with one. This is why, even if it seems silly, I suggest that you create some exports like the one on page 3 to see just what order these various headers and trailers appear in exports. It will save you some frustration down the line if you understand that the sequence of the output is something like this:

```
Header   TableA
Items    TableA
Trailer  TableA
Header   relatedTableB
Items    relatedTableB
Trailer  relatedTableB
Items    relatedTableB
Trailer  relatedTableB
etc
```

Where does the .End go?

I wish I had a good answer for this, but I don't. There has to be a .end in an export, but not only does it not seem to matter where you put it, you can put in several without any apparent change in the output. Consider the above DQL, you can move the .end before the .form trailer or even above the .items and it still behaves just the same. The only thing you

cannot do is to remove it completely. One of these days I'm going to find out why, but not today.

### Changing the DQL Export File Name

I often prefer to set up exports to always go to a specified file in a specified directory. It eliminates problems when users mis-specify the name and then cannot find the export. But there are legitimate cases where the export file name needs to be changed at run time. To do this requires the use of a global variable.

```
define global "gFileName" text 40 .

for TableA ;
list records
  FieldA in order ;
  FieldB .
end .

export to gFileName .
. form header
FieldA~FieldB
. items
@f[1, 1]~@f[1, 2]
. end
```

The trick to this is that the global value has to be set BEFORE the export DQL is run. That means you cannot do this by putting a data-entry form on the DQL itself. One way is to use a Report Ctrl Procedure that gets the file name and passes it onto the export. This can be simple or complex. I have it setup in one app so that the previously used file name is displayed for the user to accept or change.

### Field Length, Suppress Spaces, and Formatting

There are some other parameters that can be added to the fields as well:

- @f[1, 1, 10]      **Creates a fixed length field (no suppress spaces)**
- @f[1, 1, 10, s]      **Creates a max length field with suppress spaces**
- @f[1, 1, 10]\      **When used after the last field on a line, this will suppress the CR**

Including the length sets the display length the field is to occupy, this is how you would create a fixed length format. If the contents of the field exceed the display length, they will be truncated.

Combining both the length and suppress spaces will produce a "max length" field. That is, it will limit the field length to the specified number but will suppress spaces if the field is shorter than that.

There is a type of fixed length format where you do not want the lines to end with a carriage return (CR). In that case, you would want to place a \ after the last field in the output to suppress that CR.

Number fields cannot be formatted. If you put a fixed point field in the output, it will use the formatting of the field from the table. This will include the thousands separator (a common in the US) as well which is often not desired in an export. Two ways to deal with this are to use a numeric variable in place of the field (which will output as a floating point field), or to use the NumToText CDF to format the field to your liking.

### Exporting Summary Information

The export syntax also allows for the inclusion of summary information such as sum, count, min, max, etc. To use these, they must be matched by the inclusion of the same functions in the list records. Here is an example:

```
for TableA ;
list records
  FieldA in order ;
  FieldB : item sum .
end .
export to "filename.txt" .
.items
@f[1, 1]~@f[1, 2]
.form trailer
-----
@f[1, 2 sum]
.end
```

As previously noted, everything after the export to becomes a part of the output. For this reason, you don't actually need to use a .form trailer for this. The following will work just as well:

```
export to "filename.txt" .
.items
@f[1, 1]~@f[1, 2]
.end
-----
@f[1, 2 sum]
```

Be sure and test any export where you use summaries to make sure that you have not mixed things up. To quote from the help file, "No error checking is carried out to verify that the appropriate statistical operator has been used with a particular field...". In other words, make sure that you don't use a count in the body and a sum in the export.