ABSTRACT
Introducing the %qcomma() macro function. I use it all the time to go from SAS variable names, which are unquoted and space delimited, to an Oracle or DB2 list-of-values, for example, which are single quoted and comma delimited. Clinical treatment group, study number and patient id are examples of user variables which have many values. There may be many patients, each of whom undergo many studies and in each study are experimentally treated in many ways. You might want to analyze only a few patients or report only a few studies. All of this data is usually kept relationally in a big database often in normalized and long & skinny tables, and accessed and managed using SQL. The macro function presented in this article allows seamless transition between SAS and that database.

INTRODUCTION
The SQL standard grammar requires literal strings to be placed between single quote marks. Any text not enclosed by quotes will be interpreted as SQL tokens, keywords or arguments.

Proc SQL; ...WHERE trtgrp IN ('A','B','C') ...; quit;  * OK *;
Proc SQL; ...WHERE trtgrp IN ( A   B   C ) ...; quit;  * Not so good *;

Literal values in SAS don't need to be quoted, and must be separated by spaces not commas.
%macro getdata(testname); %mend;
%getdata(A B C);  * OK *;
%getdata(A,B,C);  * oops. 3 positional parms? No! *

Variable names must not be quoted, and are always separated by spaces.
Data labdat; INPUT A B C P;           * Good *;
Data labdat; INPUT 'A','B','C','D';   * BAD! *

THE OLD WAY OF BUILDING THE BRIDGE BETWEEN SAS AND DBMS
In order to put quotes around each variable name, there is a challenge to do so in SAS. You often copy/paste the names into Excel, replicate quotes and commas in adjacent columns, then copy/paste back into a SAS Proc SQL step IN list remembering to remove that last comma. Exceedingly tedious.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'</td>
<td>'</td>
</tr>
<tr>
<td>2</td>
<td>'</td>
<td>'</td>
</tr>
<tr>
<td>3</td>
<td>'</td>
<td>'</td>
</tr>
</tbody>
</table>

A CHEESEY WAY TO QUOTE
You can use the SQL processor to iterate through your variable names, giving you a chance to append quotes and the comma to each one in a somewhat programmatic fashion.

%local trtgrp;
Proc SQL;
SELECT chr(39) || trtgrp || chr(39) || ',' as trtgrp :into trtgrp
FROM work.mytable;
quit;
%let trtgrp = &trtgrp.' ';  * last element must not be a comma *

CHR(39) is a single-quote mark

This macro variable of quoted values can't be used with any other SAS statement, such as KEEP, DROP, INPUT, PUT, or a BY group, because quotes and commas are not appreciated by SAS. Therefore, you preclude a lot of the functionality of SAS.
SOME APPLICATIONS OF USING THE %QCOMMA() MACRO FUNCTION

WHEN THE DELIMITER IS SPACE
Typical SAS list-of-values use the white space to distinguish elements. Therefore,  
%put %qcomma(one two three four five );  
produces nicely the following result  
'one','two','three','four','five'

EMBEDDED SPACE AS PART OF THE VALUE
Since white-space is usually the element separator, element items having spaces (such as spaces in file names) pose a minor problem  
%put %qcomma(a foo   another bar    third baz);
will return erroneously  
'a','foo','another','bar','third','baz'
To preserve white spaces within each text element value we must choose a different character as our input text separator token. This special character should – actually must – not occur in the text as data.  
%put %qcomma(a foo Y another bar Y third baz, delim='Y');  
correctly produces the desired output of three terms, not six  
'a foo','another bar ','third baz'
Note carefully that, although in this particularly contrived example the upper case 'Y' works as a delimiter, in general it is not good practice to use any alphanumeric character a-z or 0-9 since they can be in the text.

WITH DIFFERENT DELIMITER AND THE SPACE SHOULD NOT BE IGNORED
When we select a non-space character as delimiter, we preserve spaces in the input text element values as shown above. Any white space around the chosen delimiter character, on the other hand, is trimmed and ignored. Thus, both examples  
%put %qcomma(a foo|another bar|third baz, delim='|');  
%put %qcomma(a foo | another bar | third baz, delim='|');  
produce the same output  
'a foo','another bar','third baz'
If one desires leading and trailing element spaces to instead be preserved as well, simply omit the %qcmpres step as noted below in the 'How It Works' section.  
%put %qcomma(a foo | another bar, trim=NO);  
produces  
'a foo ',' another bar'

TYPICAL SAS TO SQL APPLICATION
The real power of the %qcomma() macro function is shown in this example. A SAS variable is assigned space-separated values, perhaps in a macro call.  
%analyze(INR    PT);
Since comma distinguishes positional or keyword parameters in the SAS Macro language, it is clear why SAS chooses space as the item separator for its list-of-values. Most humorous is the statement  
%analyze(INR, PT);  * dont try this at home *;  
Notice the very clean use of %qcomma() in the SELECT statement below. The macro function does all the work of quoting, separating and putting in commas so that the value of macro variable parms is properly specified for the IN list of SQL.  
%macro analyze(parms);
  Proc SQL;
  create table result as (  
    SELECT p.pid, d.parm  
    FROM pats AS p  
    inner join labs AS d ON p.pid = d.pid AND d.parm IN (%qcomma(&parms.))  
  );
  quit;
  Proc sort; data=result; by &parms.; run;
%mend;
%analyze(PT INR);
Proc print data=result; run;
The tables from this example are shown below.

<table>
<thead>
<tr>
<th>WORK.PATS</th>
<th>WORK.LABS</th>
<th>%analyze(INR PT)</th>
<th>WORK.RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>PID</td>
<td>PARM</td>
<td>PID</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>RBC</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>RBC</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>WBC</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>WBC</td>
<td>2</td>
<td>PT</td>
</tr>
<tr>
<td>1</td>
<td>INR</td>
<td>1</td>
<td>PT</td>
</tr>
<tr>
<td>2</td>
<td>INR</td>
<td>2</td>
<td>PT</td>
</tr>
</tbody>
</table>

**HOW IT WORKS**

The `%qcomma()` macro function proceeds through five phases: Scanning, Extraction, Mark-up, Concatenation, and Finalization. The heart of `%qcomma()` is the token scanning of its do-loop. `%qscan` breaks the input text into grammatical pieces and leaves the pieces quoted through next level of resolution.

```plaintext
%do %while(%qscan(&in.,%eval(&i.+1),%str( )) ne );
%if &i. gt 0 %then %do; %let var=%str(%trim(%quote(&var.)),); %end;
%let i = %eval(&i.+1);
%let var=&var.%str(%')%qcmpres(%qscan(&in.,&i.,%str( )))%str(%');
%end;
```

The tokenization process is analogous to that which one sees Java:

```java
StringTokenizer st = new pub.util.StringTokenizer(in, " ");
int tokenNum = 0;
String element;
while(st.hasMoreTokens()) {
    element = st.nextToken();
    if(tokenNum > 0) result += (element + "\",);  
    tokenNum++;
    result += ("'" + element.trim() + "'");
}
```

SAS grammar specifies that anything separated by whitespace is a separate element. Each element is then extracted, one at a time, and prefixed and suffixed with appropriate mark-up blocking characters – for example, the single quote mark chr(39).

These marked-up elements are concatenated back together after first applying the output data separator character, which is usually a comma.

Lastly, the whole thing is %unquote()ed so that it appears as pure text to whomever invoked the `%qcomma()` macro function. Unquoting is very important, as we see in the following example:

```plaintext
%put %nrstr("%qcomma(&myline.)");
```

It should be clear that arithmetic %eval() is required to perform the loop index variable increment. Without %eval the re-assignment of `i` would be treated as character rather than numeric.

You may wish not to surpress leading and trailing blanks around each element. The TRIM=NO option of `%qcomma` simply removes %qcmpres from the mark-up step.

**GENERAL THEORY**

The quoting and delimiting process is really a specialized form of text markup: Encode "just enough" style information with text content so that the text can be picked apart at a later time. By "just enough" we mean keeping the final size as small yet as meaningful as possible. It is senseless to encode an entire sentence of style information where a simple word might do just fine.

%qcomma() is described grammatically like this:
Just like an array is a collection (or group) of variables, a variable may represent more than one value. However, where array variables have natural separation of their name such as suffixed numerals var1, var2, ..., var12, values in a multi-valued lists have no inherent separation. In SAS, white space is the separator. We are free to put as many blank lines and space as we wish between values. There are also no start or end markes in SAS variable name lists.

```
%let parmname = one two
three
four;
```

Both of these, when de-referenced and printed on the Log, produce the same output

```
%put &parmname.;                         one two three four
```

In SQL, comma is the value separator, and there must be only one comma between values. This difference of separator convention is the basis of grammar linguistics and this article.

**CONCLUSION**

When you work with a database you have to speak its language. The list of values is one of the most challenging concepts for the SAS programmer to translate. Using this macro function can help make the task of putting in the quote & comma to satisfy the need of the other DMBS easier. With the %qcomma(), macro function, SAS can truly "speak database."

**REFERENCES**


**ACKNOWLEDGMENTS**

Many thanks to Michael Metts and Paul Dorfman for the clever naming and original idea and ensuing discussion upon which this work is founded. To SAS Technical Support for excellent and timely knowledge helping us understand the subtle nuances of Macro variable quoting. Great appreciation to Na Li for technical review, inciteful examples, and stylistic comments.

**TRADEMARK CITATION**

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

**CONTACT INFORMATION**

Your comments and questions are valued and encouraged. Contact the author at:

Paul D Sherman
Electrical Engineer
335 Elan Village Lane, Apt. 424
San Jose, CA 95134
Home: 408-383-0471
Email: sherman@idiom.com

Web site: http://www.idiom.com/~sherman/paul/pubs
options nosource;
/* ============================================================== */
/* qcomma - single quote delimit a SAS list of values */
/* LIST input list of values, DELIM separated */
/* DELIM= item separators, defaults to whitespace */
/* */
/* 1.0 2005-12-03 pds initial cut */
/* ============================================================== */
%macro qcomma(list, delim=%str( ));
  %local varlist i var;
  %let varlist=&list.;
  %let i=0;
  %do %while(%qscan(&varlist.,%eval(&i.+1),%quote(&delim.)) ne );
    %if &i. gt 0 %then %do; %let var=%str(%trim(%quote(&var.))&datasep.); %end;
    %let i = %eval(&i.+1);
    %let var=&var.%str( %qcmpres(%qscan(&varlist,&i.,%quote(&delim.)))%str( %
    %end;
%unquote(&var.) %* MUST be first and only thing on a line by itself *;
%mend;

EXAMPLES

Proc SQL noprint;
  create table pats (pid integer);
  create table labs (pid integer, parm char(32));
  insert into pats (pid) values (1) values (2) values (3);
  insert into labs (pid, parm)
    values (1, 'RBC')  values (2, 'RBC')
    values (1, 'WBC')  values (2, 'WBC')
    values (1, 'INR')  values (2, 'INR')
    values (1, 'PT')   values (2, 'PT')
  ;
  quit;

%put %qcomma(one two three four five   six);
%put %qcomma(a foo     another bar     third baz);
%put %qcomma(a foo  Y  another bar  Y  third baz, delim=Y);