SAS Macro Quoting

Tim Lepp
Bayer AG
Müllerstr. 178
13353 Berlin
tim.lepp@bayer.com

Zusammenfassung

Haben Sie sich beim Thema Macro Quoting über die Jahre auch eher durchgemogelt? Keine Lust gehabt sich wirklich damit zu beschäftigen? Und mal ehrlich, irgendwie hat man es ja am Ende doch immer hinbekommen.
Auch wenn dies etwas unbefriedigend ist – für den reinen Anwender genügt diese Vorgehensweise vermutlich sogar. Spätestens aber, wenn man wirklich robuste Macros programmieren möchte, muss man sich mit dem Thema intensiver auseinander setzen. Hier wird schnell klar – das Thema ist komplexer, als es auf den ersten Blick erscheint.
Dieser Beitrag gibt nicht nur eine Einführung in die Macro Quoting Funktionen, sondern beschäftigt sich insbesondere mit dem weniger beachteten Timing Aspekt des Quotings.

Schlüsselwörter: Macro Quoting, Maskierung, Makrofunktionen, Programmfluss, Makro Prozessor, %Str, %NrStr, %Quote, %NrQuote, %BQuote, %NrBQuote, %SuperQ, %Unquote

1 Einführung

Was ist das SAS Macro Quoting und warum brauchen wir das? Wenn man Quoting hört, könnte man meinen, es hätte irgendetwas mit Anführungszeichen zu tun. Hat es auch, aber nur im übertragenen Sinne. Bei “normalen“ Programmiersprachen wird mit Anführungszeichen “Text“ von Schlüsselwörtern, Operatoren usw. unterschieden. Die SAS Macro Language betrachtet hingegen alles als Text.

Beispiel:

```
H&M, %Percent, g=9.8
```

Handelt es sich hierbei um

(a) eine Liste von Macro Parametern?
(b) die Macro Variable M, den Macro Call Percent?
(c) einen String von Firmennamen?

Dies ist sogar für den Menschen nicht leicht zu erkennen. Deshalb brauchen wir hier ein anderes Konzept zur Differenzierung, als die Anführungszeichen.

Es gibt natürlich noch weitere Zeichen und Mnemoniken, als die im obigen Beispiel genannten, welche vom Macro Prozessor fehlinterpretiert werden könnten. Eine vollständige Liste ist in Abbildung 1 gegeben.

blank	)	=	LT
:	(	|	GE
¬	+	AND	GT
^	—	OR	IN
~	*	NOT	%
. (comma)	/	EQ	&
'	<	NE	#
"	>	LE	

Abbildung 1: Zeichen und Mnemoniken mit operativer Bedeutung für den Macro Prozessor [1]

Beim SAS Macro Quoting wird nun folgendermaßen vorgegangen: Es werden die in Abb. 1 aufgeführten Zeichen temporär vor dem Macro Prozessor verborgen (maskiert), indem sie mit nicht druckbaren Zeichen im Bereich von 0x01 bis 0x1F ersetzt werden. Dieser Zeichenbereich befindet sich im Bereich der Steuerzeichen und kann deshalb zwischen den Betriebssystemen variieren. Zum besseren Verständnis ist in Abb. 2 ein Beispiel zweier Macro Variablen gegeben. Erstere enthält eine Auswahl an Sonderzeichen unmaskiert. Die zweite die gleichen Sonderzeichen mit der Funktion `%NRBQUOTE` maskiert. Start- und End Bytes (hier 0x06 und 0x08) bezeichnen die Start- und End ID der jeweiligen Quoting Funktion. Werte in eckigen Klammern stellen hexadezimal Werte der jeweiligen Zeichen dar.

UNQUOTED	& % ' " ( ) + - * / < > = ^ ~ ; ,
NRBQUOTED	[06] [0F] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [1A] [1C] [0B] [1F] [0E] [1E] [08]

Abbildung 2: Beispiel zur Funktionsweise des Macro Quotings

2 Macro Quoting Funktionen

Dieser Abschnitt soll einen Überblick, der gängigen Quoting Funktionen mit deren jeweiligen Besonderheiten, geben. Dies ist der Teil, welchen man auch gut in den meisten Paper dargestellt findet. Von daher soll der Themenblock hier kurz gehalten werden.

Generell werden die folgenden Drei Klassen von Zeichen unterschieden, welche vom Macro Prozessor fehlinterpretiert werden können und daher maskiert werden müssen, sofern es sich bei ihnen um Text handelt.

Abbildung 3: Klassen von Sonderzeichen

Bleiben wir bei dem Beispiel aus der Einführung und schauen uns an, was SAS für die verschiedenen Quoting Funktionen daraus macht. Nehmen wir an die Variable **`VAR0`** sei folgendermaßen definiert

```
DATA _NULL_; CALL SYMPUT('var0','H&M, O''Neill, %Percent, g=9.8'); RUN;
```

Nehmen wir weiter an, es seien definiert die Variable **`M`** und das Macro **`%Percent`**

```
%LET M=ennes&Mauritz;
%MACRO Percent(); _Xxxxxxx %MEND;
```

Nun wenden wir die Fünf wichtigsten Quoting Funktionen auf die Variable **`VAR0`** an.

```
%LET var1_STR      =      %str(H&M, O%'Neill, %Percent, g=9.8);
%LET var2_NRSTR    =    %nrstr(H&M, O%'Neill, %Percent, g=9.8);
%LET var3_BQUOTE   =   %bquote(&var0.);
%LET var4_NRBQUOTE = %nrbquote(&var0.);
%LET var5_SUPERQ   =   %superq(var0);
```

Und bekommen folgendes Ergebnis

name Macro Variable Name	value Macro Variable Value
VAR0	H&M, O'Neill, %Percent, g=9.8
VAR1_STR	[01]Hennes&Mauritz[1E] O[11]Neill[1E] _Xxxxxxx[1E] g[1C]9.8[02]
VAR2_NRSTR	[01]H[0F]M[1E] O[11]Neill[1E] [10]Percent[1E] g[1C]9.8[02]
VAR3_BQUOTE	[04]Hennes&Mauritz[1E] O[11]Neill[1E] _Xxxxxxx[1E] g[1C]9.8[08]
VAR4_NRBQUOTE	[06]Hennes[0F]Mauritz[1E] O[11]Neill[1E] _Xxxxxxx[1E] g[1C]9.8[08]
VAR5_SUPERQ	[06]H[0F]M[1E] O[11]Neill[1E] [10]Percent[1E] g[1C]9.8[08]

Abbildung 4: Maskierte Ergebnisse für die Fünf Haupt Quoting Funktionen

In der linken Spalte sieht man den Macro Variablen Namen und in der Rechten den Inhalt, wie er in der *Global Symbol Table* abgelegt ist. Die Werte in eckigen Klammern stellen die in der Einführung erwähnten nicht druckbaren hexadezimalen Zeichen dar.

- %STR
  Maskiert die Klasse A (siehe Abb. 3) und Klasse C, sofern dem zu maskierenden Zeichen ein % als eine Art Escape Character vorangestellt wird. Im obigen Beispiel steht deshalb bei O%‘Neill das % vor dem Hochkomma. Die Funktion hat Schwierigkeiten, wenn es im Argument ungepaarte Klammern oder Anführungszeichen gibt. Nicht selten ist die Sitzung dann nur über einen Session Reset wie-

der herzustellen. Sie arbeitet in der Kompilierungsphase. Was genau dies bedeutet, wird in Abschnitt 4 erläutert.

- %NRSTR
  Maskiert die Klassen A und B. Die Klasse C unterliegt den gleichen Einschränkungen wie bei der %STR Funktion. Die Funktion hat ebenfalls Schwierigkeiten, wenn es im Argument ungepaarte Klammern oder Anführungs-zeichen gibt. Sie arbeitet in der Kompilierungsphase.
- %BQUOTE
  Maskiert die Klassen A und C ohne Einschränkungen. Auch ungepaarte Klammern oder Anführungszeichen werden behandelt. Die Funktion arbeitet in der Ausführungsphase.
- %NRBQUOTE
  Maskiert die Klassen A, B und C. Allerdings ist bei Klasse B zu beachten, dass die Funktion diese Zeichen nicht wie %NRSTR sofort maskiert, sondern die Macro Trigger werden so weit als möglich aufgelöst und nur ganz am Ende, wenn keine Auflösung mehr möglich ist, maskiert. Dies ist gut zu erkennen in Abb.4 – `VAR4_NRBQUOTE`. Die Funktion arbeitet in der Ausführungsphase.
- %SUPERQ
  Maskiert die Klassen A, B und C. Eine Besonderheit ist hier, dass die Funktion als Argument den Namen einer Macro Variablen erwartet, ohne das vorangestellte &. Der Inhalt dieser Macro Variable, welche zudem existent sein muss, wird dann ohne weitere Auflösung maskiert. Hierin unterscheidet sich die Funktion zu %NRBQUOTE. Wenn man die Ergebnisse in Abb. 4 vergleicht, sieht man, dass %SUPERQ die eigentliche Schwesterfunktion zu %NRSTR zur Zeit der Ausführung ist.
- %UNQUOTE
  Demaskiert die Klassen A, B und C. Sie kann hilfreich sein, wenn man z.B. in der Kompilierungsphase etwas vor dem Macro Prozessor verbergen will, was aber in der Ausführungsphase wieder sichtbar sein soll.

Die Sechs eben beschriebenen Funktionen stellen die wichtigsten Quoting Funktionen dar. Mehr braucht man eigentlich nicht. Der Vollständigkeit halber sollen aber auch die restlichen zumindest noch genannt werden.

- %QUOTE / %NRQUOTE
  Die Funktionen maskieren äquivalent zu %STR und %NRSTR nur zur Zeit der Ausführung. D.h. sie haben die gleichen Einschränkungen was die Klasse C angeht und Schwierigkeiten mit ungepaarten Klammern und Anführungszeichen. Das heißt weiter es gibt überhaupt keine Vorteile im Vergleich zur Benutzung von %BQUOTE und %NRBQUOTE. Die Funktionen existieren eigentlich nur noch aus Kompatibilitätsgründen und bedürfen keiner weiteren Beachtung.

- %QSCAN / %QSUBSTR / %QUPCASE / %QSYSFUNC
  Diese Macro Funktionen haben primär natürlich einen anderen Funktionsinhalt, aber maskieren das Ergebnis in gleicher Weise wie %NRBQUOTE.
- %QCMPRES / %QLOWCASE / %QLEFT / %QTRIM
  Hierbei handelt es sich um Autocall Macros. Auch diese haben primär einen anderen Zweck, maskieren aber das Ergebnis wie %NRBQUOTE.

3 Ein Blick hinter die Kulissen

Der folgende Abschnitt soll als generelle Einführung in den Programmfluss dienen und insbesondere die Beziehung zwischen *Macro Prozessor* und *Data Step Compiler* erklären.

Abbildung 5 enthält eine allgemeine Darstellung der am Programmfluss beteiligten Funktionseinheiten. Auf der linken Seite befinden sich die *Input Blöcke*. Das können normale SAS - oder SCL Programme, alle Arten von Batch oder interactive Submits u.ä. sein. Der Code aus diesen Input Blöcken landet sukzessive auf dem *Input Stack*. Dieser wiederum wird Wort für Wort oder Token für Token (das ist die kleinste Bedeutungseinheit innerhalb des Codes) vom *Word Scanner* ausgelesen. Der *Word Scanner* wiederum analysiert den Code und leitet ihn an die jeweiligen Funktionseinheiten zur Kompilierung und Ausführung weiter [3].

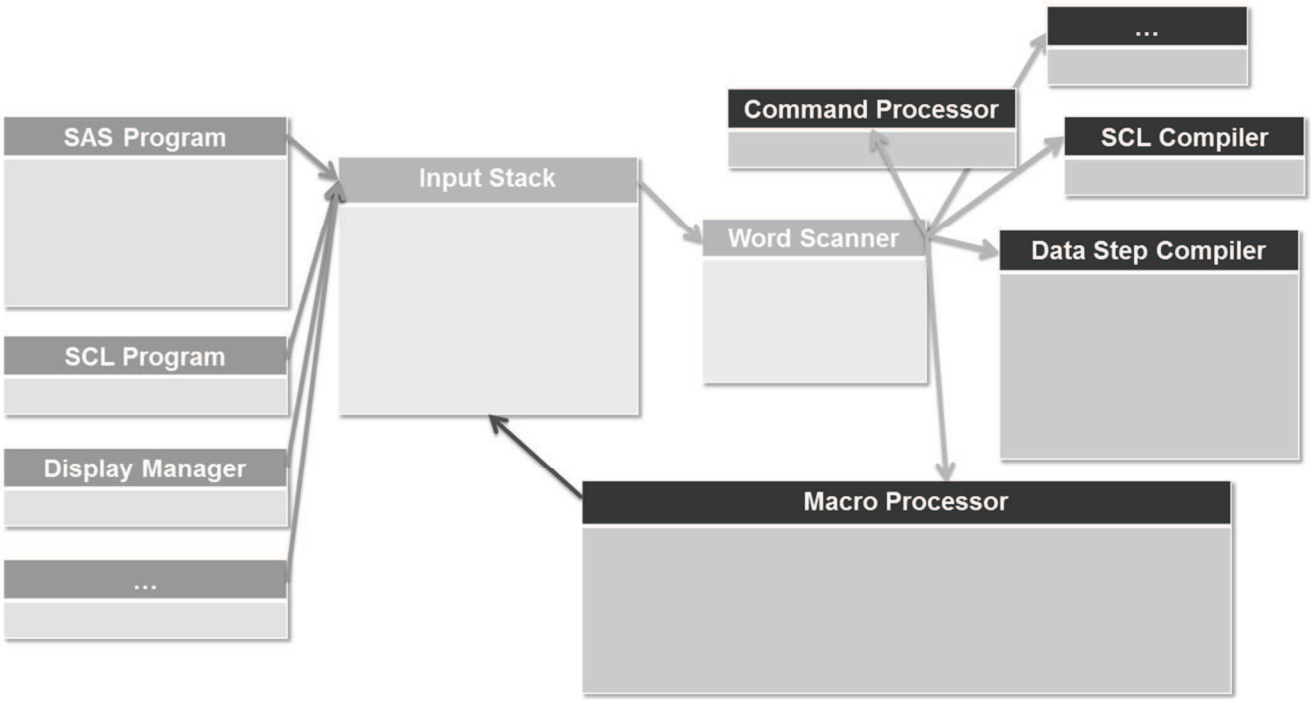

Abbildung 5: Schematische Darstellung des Programmflusses

In der Abbildung fällt auf, dass der *Macro Prozessor* als einzige Funktionseinheit mit dem *Input Stack* rückverbunden ist. Schauen wir uns den Programmfluss an Hand des folgenden kleinen Beispiel Programmes genauer an.

```
DATA today;
     date = "&sysdate9";
RUN;
```

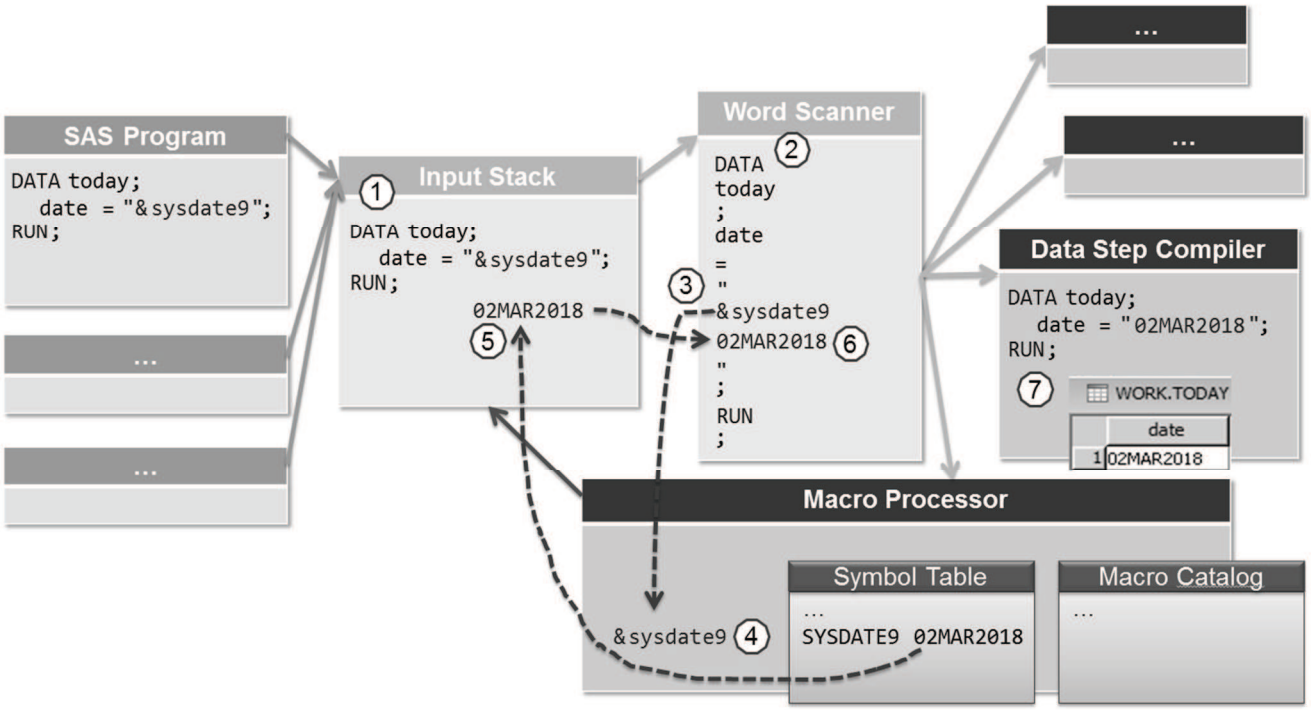

Abbildung 6: Programmfluss am Beispielprogramm

Im ersten Schritt (1) wird unser Beispiel Programm auf den *Input Stack* geladen. Anschließend beginnt der *Word Scanner* seine Arbeit und liest den *Input Stack* Token für Token aus (2). Ausgelesene Token werden vom *Input Stack* entfernt. Der *Word Scanner* erkennt nun die Code Stücken als Data Step Code und leitet diese zur Kompilierung weiter an den *Data Step Compiler*. Dies tut er solange, bis er auf das **`&sysdate9`** stößt. Dieses Token wird als Macro Variablenreferenz erkannt und zur Bearbeitung an den *Macro Prozessor* weitergeleitet (3). Solange der *Macro Prozessor* arbeitet, pausiert der *Word Scanner*. Da unser kleines Testprogramm im Open Code abgeschickt wurde, schaut nun der *Macro Prozessor*, ob er die Macro Variable in der *Global Symbol Table* findet (4). Das tut er, denn es handelt sich hierbei um eine automatische Systemvariable. Den Inhalt dieser Variable sendet nun der *Macro Prozessor* zurück an den *Input Stack*. Und zwar wird diese nicht wie üblich ans Ende, sondern an den Anfang des *Input Stacks* geschrieben (5). D.h. was sich nun noch auf dem *Input Stack* befindet ist folgendes:

```
02MAR2018"; RUN;
```

Jetzt nimmt der *Word Scanner* seine Arbeit wieder auf (6), erkennt den verbleibenden Code als Data Step Code und leitet diesen an den *Data Step Compiler* weiter. Sobald dieser das **`RUN;`** Statement erhält, eine sogenannte Data Step Boundary, heißt dies, dass der Code nun ausgeführt werden kann. Der Datensatz **`WORK.TODAY`** wird generiert (7).

Der Macro Prozessor Zyklus wurde hier anhand einer Macro Variablen Referenz erklärt, aber dies gilt natürlich gleichermaßen für alle Macro Calls / Statements / Funktionen / Variablen und eben auch für die Macro Quoting Funktionen, deren Timing wir uns im Folgenden Abschnitt genauer ansehen wollen.

4 Macro Quoting Timing

Wie wir im 2. Abschnitt erfahren haben, werden die Macro Quoting Funktionen hinsichtlich des Timings in zwei Kategorien unterteilt. Die *Compilation Time* Funktionen und die *Execution Time* Funktionen. In den meisten Papers wird dies als Eigenschaft der jeweiligen Funktion zwar genannt, bleibt aber darüber hinaus unkommentiert, als wäre es völlig klar, wovon hier die Rede ist. Was genau bedeutet dies nun?
Kompilierungs- und Ausführungsphasen gibt es sowohl beim *Macro Prozessor*, als auch beim *Data Step Compiler* und anderen Funktionseinheiten. Hier kann es schon mal die erste Verwirrung geben, worauf sich jene Eigenschaft der Quoting Funktion eigentlich bezieht. Da es sich hier um Macro Funktionen handelt, bezieht sich diese Eigenschaft ausschließlich auf Kompilierungs- und Ausführungsphase des *Macro Prozessors*. Wie wir in Abschnitt 3 gesehen haben, steht dieser aber wiederum in etwas sonderbarer Beziehung zu den anderen Funktionseinheiten. D.h. was man sich an dieser Stelle schon mal merken sollte ist – dass alle Compilation- und Execution Time Quoting Funktionen bereits in der Compilation Phase des Data Steps (oder anderer Funktionseinheiten) zur Anwendung kommen.

Schauen wir uns auch dies an Hand des folgenden kleinen Beispiel Programmes an.

```
%MACRO selectDat(condition);
  %PUT %STR(Selection: KEEP=name team salary; WHERE=&condition.;);
  %UNQUOTE(KEEP name team salary; WHERE &condition.;)
%MEND selectDat;

%LET select=%STR(name="Carter, Joe");

DATA out;
  SET sashelp.baseball;
  %selectDat(%BQUOTE(&select.))
RUN;
```

Die Sinnhaftigkeit dieses Codes muss nicht weiter kommentiert werden, dieser soll nur der Anschauung dienen. In der folgenden Abbildung ist im Prinzip das Schema aus Abb. 6 noch einmal nachgebildet. Die Visualisierung über den *Word Scanner* wird der Übersicht halber aber weggelassen. Die dunklen Markierungen im Code stellen maskierte Zeichen dar.

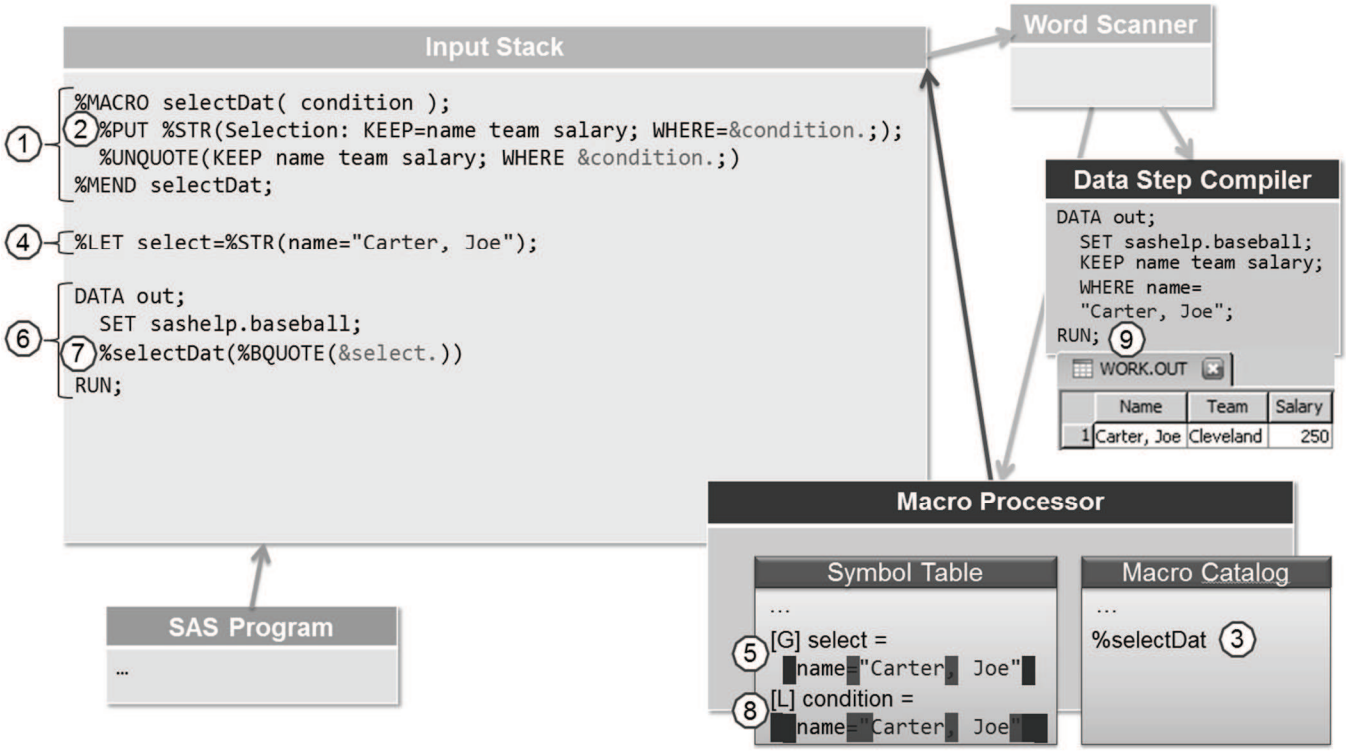

Abbildung 7: Timing des Quotings am Beispielprogramm

Wie wir gelernt haben, wird der *Input Stack* von oben nach unten bearbeitet. Im ersten Schritt wird also das Macro **`selectDat`** definiert (1). Macro Definitionen finden ganz klar zur Kompilierungszeit statt. D.h. **`%STR`** und **`%NRSTR`** Funktionen kommen zum Einsatz. Der Ausdruck (2) wird maskiert zu

```
%PUT %STR(Selection: KEEP=name team salary; WHERE=&condition.;);
```

Das Makro selbst wird dann kompiliert im *Macro Catalog* abgelegt (3).

Schauen wir uns das **`%LET`** Statement (4) an. Macro Statements im Open Code werden vom Gefühl her eigentlich sofort ausgeführt. Aber auch diese Ausdrücke werden vor der Ausführung kompiliert. In diesem Fall kommt wieder die **`%STR`** Funktion zum Einsatz. Der Ausdruck wird maskiert zu

```
%LET select=%STR(name="Carter, Joe");
```

Danach wird das Statement ausgeführt, die Macro Variable **`select`** in der *Global Symbol Table* abgelegt (5).

Nun wenden wir uns dem Data Step (6) zu. Die ersten beiden Zeilen des Data Steps werden vom *Word Scanner* ganz klar als reiner Data Step Code erkannt und an den *Data Step Compiler* zur Kompilierung weitergeleitet. Der Macro Aufruf (7) wird da schon etwas interessanter. Das Macro **`selectDat`** liegt ja quasi schon kompiliert im *Macro Catalog* ab. Trotzdem durchläuft auch dieser Ausdruck Kompilierungs- und Ausführungsphase. Wie üblich kämen in der Kompilierungsphase die **`%STR`** und **`%NRSTR`** Funktionen zum Einsatz. Beide haben wir hier nicht, deswegen machen wir weiter mit der Ausführungsphase. Hier wird nun der Ausdruck von innen nach außen aufgelöst. D.h. zuerst wird die Macro Variablenreferenz **`&select`** aufgelöst. Diese wird aus der *Global Symbol Table* ausgelesen. Der Ausdruck sieht dann folgendermaßen aus:

```
%selectDat(%BQUOTE(name="Carter, Joe"))
```

Nun wird die **%BQUOTE** Funktion auf den bereits maskierten Inhalt angewendet.

```
%selectDat(%BQUOTE(name="Carter, Joe"))
```

Zusätzlich zu den ID Bytes der **%STR** Funktion kommen nun die ID Bytes der **%BQUOTE** Funktion dazu. Und wie wir aus Abschnitt 2 wissen, maskiert die **%BQUOTE** Funktion zusätzlich die Klasse C Character, d.h. es werden nun außerdem die Anführungszeichen maskiert.

Nun wird das Macro selbst ausgeführt. Es wird die Macro Variable **condition** in der *Local Symbol Table* des Macros angelegt (8). Es gibt eine Ausgabe im Log und was am Ende eigentlich als Code von dem Macro übrig bleibt, ist der Ausdruck innerhalb der **%UNQUOTE** Funktion, welcher dann zurück auf den *Input Stack* geschrieben wird.

```
KEEP name team salary; WHERE name="Carter, Joe";
```

Dieser Ausdruck zusammen mit dem **RUN;** Statement wird nun vom *Word Scanner* als Data Step Code identifiziert und an den *Data Step Compiler* übergeben.

Dieser erkennt die Data Step Boundary und führt den kompilierten Code aus (9).

5 Zusammenfassung

Wir haben gesehen, das Thema Macro Quoting ist komplexer, als es auf den ersten Blick erscheint. Die Regeln – also welche Funktion maskiert was? – sind schnell verstanden. Kompliziert wird das Ganze hauptsächlich durch die Timing Komponente und hier insbesondere durch das Zusammenspiel von *Macro Prozessor* und *Data Step Compiler* (oder anderen Funktionseinheiten). Hat man den Kreislauf der Auflösung von Macro Code einmal verstanden, kann man auch das Timing der Quoting Funktionen viel besser einordnen. Ich hoffe, das konnte ich mit diesem Beitrag erreichen.

Zum Schluss noch einmal ein paar Gedankenstützen in stichpunktform.

- %STR / %NRSTR arbeiten vor der Auflösung der Macro Trigger &, %. %BQUOTE / %NRBQUOTE / %SUPERQ und alle anderen Quoting Funktionen arbeiten nach deren Auflösung.
- Zum Maskieren von &Macrovariablen nutzt man %BQUOTE, %NRBQUOTE u. %SUPERQ
- Zum Maskieren von Text Input nutzt man %STR u. %NRSTR
- Compilation und Execution Phase des Data Steps hat mit Compilation / Execution Time der Quoting Funktionen nichts zu tun. Alle Maskierung findet bereits in der Compilation Phase des Data Steps statt.
- %SUPERQ ist die eigentliche Schwesterfunktion zu %NRSTR
- Es gibt kaum eine Situation in der man %BQUOTE der %NRBQUOTE Funktion vorziehen müsste. Beide arbeiten fast identisch und Warnings zu unaufgelösten &.. und %.. gibt es auch bei %NRBQUOTE.
- Wird von einem Ausdruck ein Error im Log generiert, aber den Ausdruck könnte man ohne Probleme im Data Step abschicken – hilft oft ein %UNQUOTE über den gesamten Ausdruck (Es scheint, dass insbesondere maskierte Anführungs-

zeichen hin und wieder nicht korrekt automatisch demaskiert werden, wenn sie die Macro Facility verlassen).

- Es genügt nicht, wenn nur Macro Entwickler sich über das Macro Quoting Gedanken machen, denn Quoting wird oft schon notwendig bei Parameter-übergabe an das Macro oder die Funktion.

Literatur

[1] SAS Institute Inc. 2016. SAS® 9.4 Macro Language: Reference, Fifth Edition. Cary, NC: SAS Institute Inc.

[2] Whitlock, Ian. 2003, A Serious Look at Macro Quoting. Proceedings of the 28th SAS Users Group International (SUGI 28).

[3] Lyons, Lisa. 2004, Going Under the Hood: How Does the Macro Processor Really Work? Proceedings of the 17th NorthEast SAS Users Group (NESUG 17)