Paper A06-2009

When the List Grows Too Long: A Strategy to Utilize Freeform User Input in Your SAS Stored Process Web Applications

Jeffery A. Fallon, Cardinal Health, Dublin, OH

ABSTRACT

In web based reporting applications, we often allow users to select the values for which they want to see results from a list of values. With this approach there are two problems that immediately surface. The first is that lists become unwieldy beyond around 1000 values. The second is that some users have a large number of values that they want to select. Freeform text input is a viable option, but application developers are often reluctant to yield the control that a list gives them over of their application input. In addition, developers are often fearful of what users might throw at their applications. This paper shows developers how to use SAS text processing functions to powerfully transform even the scariest freeform text into application ready input.

OVERVIEW

In this paper, we will examine a method of processing freeform user input in a web based SAS Stored Process as an alternative to providing users with a long exhaustive list of values from which to choose. We will show that this brief, yet robust, technique is ideal for applications in which data are identified by distinctive identifiers.

INTRODUCTION

Consider the following data request.

> *Bob,*
>
> *What's up? Hey, I need you to run me reports for these customers: 37-293 23-297 12-360 17-361*
> *34-364*
> *34-418*
> *53-420*
> *90-424*
> *83-3916*
> *77-4745*
>
> *Oh yeah, also get me '89-669431' and "98-669436".*
>
> *Did you catch those #%$@& Blue Jackets last night? Just when I thought we finally had a professional hockey team.*
>
> *Anyway, now that I think about it, I'll also need 26-383333 and 26-383334.*
>
> *Oh, I almost forgot about the SKUs. I need the reports for these SKUs: 1800074 1800173 1990199*
> *1400207*
> *1050223*
> *1330231*
> *1000363*
> *1099439*
> *1002710*
> *1600736*
>
> *By the way, do you know what I've been doing for fun. . .? Reading Haiku. Seriously. It's very relaxing. You should try it.*
>
> *Oh shoot, I will also need these SKUs: 1781341, 1781358, 1781366, and 1781374.*
>
> *That should do it. Thanks a lot. You're the best.*
>
> *Ms. Irving*

Bob has some work to do, because the reporting application that he will use to run these reports requires him to select the customer numbers and SKUs from a list. Since both number in the tens of thousands, this is no small task.

We can help Bob. We can help him and all of the other people out there with a lot of work to do and not enough time to hunt down and select a handful of numbers from a list of thousands.

It is obvious that the reporting application in question has a serious design flaw. List-based input should only be used when the list is less than a thousand items. Exceptions to this are often made by developers who feel that the control that a list gives them outweigh what they may perceive as a small amount of user inconvenience. There are numerous ways to correct the design so that the application is easier to use. Of course, the most correct design will depend on the type of data that serves as the basis for our reports. For example, if the data are customer or item names, we may best serve by building search functionality into our client. In our example, and any other instance where the identifiers are distinctive—readily distinguished from delimiters and other data (as is often the case with identifying numbers)--we have another option that can greatly benefit users.

What would be most convenient for Bob? The answer is, of course, that Ms. Irving should run her own reports. Unfortunately, that is just not going to happen. What if we just give Bob a text area that he can paste account numbers into? In fact, we will take this to an absurd extreme. We will build the application robust enough to allow him to simply copy the text of Ms. Irving's email and paste it into a text area. The application will return results that match any identifying numbers contained in the email.

FREEFORM INPUT BASICS

In web application development, it is a best practice to restrict the length of all input. Actually, the best practice is to restrict the length twice—once on the client and once on the server. We limit the length on the client page, so that users are unable to enter or paste more data than we are expecting. This is as simple as truncating the length of all text boxes and text areas (make sure that you communicate the limits to your users). Client-side truncation, however, is not sufficient to fully protect our application. One does not have to use the input form that we have provided to call our Stored Process. Anyone can create an input form that will send an HTTP GET or POST to our Stored Process. If they set the same parameters that we do, then our Stored Process will behave pretty much the same way as it does when it is called from our input form. However, we have no control over what they will send. That being the case, we need to also truncate the data on the server.

CUT IT DOWN TO SIZE

The Stored Process Application itself sets the maximum length for input parameters to 65,534 bytes. Any further truncation is up to us. We begin by cutting the input string down to the maximum size we will accept. Remember to communicate this to your users. Here we restrict it to 5000 bytes.

```
data _null_;
 LENGTH input1 $5000 input2 $5000 input3 $5000 input4 $5000 input5 $5000 input6
$5000;
 input1 = SUBSTR(&INPUT_LIST,1,5000);
```

This does not change our sample input.

PROCESS THE DELIMITERS

At my company, there is an application that allows users to generate reports by entering lists of account numbers. However, the values in the list must be delimited by semicolons (;). Not long after this application went into production, someone coded a small web application that transformed tab delimited lists into lists delimited by semicolon. This allowed users to readily transform lists that they created in Microsoft Excel.

Think about your users. What delimiters will they use? What applications might they use to generate their lists of identifiers? When we have identified the list of all likely delimiters, we process them all the same way. We use the TRANSLATE function to replace them all with spaces. In this example, we are providing users with the ability to use tabs ('09'x), commas ('2C'x), carriage returns ('0D'x), and/or line feeds ('0A'x). We also allow spaces as delimiter, but we don't need to translate them.

```
input2 = TRANSLATE(input1, ' ' , '09'x, ' ', '2C'x, ' ', '0D'x, ' ', '0A'x);
```

Your users may need a different set of delimiters. Whatever you allow, make sure that you provide for it in the translate function.

After we process our delimiters, our input looks like this:

> *Bob What's up? Hey I need you to run me reports for these customers: 37-293 23-297 12-360 17-361 34-364 34-418 53-420 90-424 83-3916 77-4745 Oh yeah also get me '89-669431' and "98-669436". Did you catch those #%$@& Blue Jackets last night? Just when I thought we finally had a professional hockey team. Anyway now that I think about it I'll also need 26-383333 and 26-383334. Oh I almost forgot about the SKUs. I need the reports for these SKUs: 1800074 1800173 1990199 1400207 1050223 1330231 1000363 1099439 1002710 1600736 By the way do you know what I've been doing for fun. . .? Reading Haiku. Seriously. It's very relaxing. You should try it. Oh shoot I will also need these SKUs: 1781341 1781358 1781366 and 1781374. That should do it. Thanks a lot. You're the best. Ms. Irving*

GET RID OF EXTRANEOUS CHARACTERS

All of the training in the world will not prevent users from entering extraneous characters. Quotation marks can be particularly problematic at times. Fortunately, the COMPRESS function allows us to easily purge any undesirable characters from our input.

```
input3 = COMPRESS(input2, '-0123456789 ', "K");
```

The third argument of the COMPRESS function is a modifier which changes the default behavior of the function. A value of "K" reverses the action of the function, keeping the characters in the second argument instead of eliminating them. Take care to not eliminate anything that is a legitimate part of the value. In our example, we only retain hyphens, digits and spaces. Spaces are particularly important to retain at this point--they are our delimiter.

After the COMPRESS function executes, our sample input looks like this:

37-293 23-297 12-360 17-361 34-364 34-418 53-420 90-424 83-3916 77-4745 89-669431 98-669436 26-383333 26-383334 1800074 1800173 1990199 1400207 1050223 1330231 1000363 1099439 1002710 1600736 1781341 1781358 1781366 1781374

NORMALIZE SPACE

We are getting there. After we translated all delimiters to spaces, we introduced a number of extraneous spaces to the input. Now, we will use the COMPBL function to replace multiple consecutive spaces with a single space and the STRIP function to remove all leading and trailing spaces.

```
input4 = COMPBL(input3);
input5 = STRIP(input4);
```

This leaves our input as:

37-293 23-297 12-360 17-361 34-364 34-418 53-420 90-424 83-3916 77-4745 89-669431 98-669436 26-383333 26-383334 1800074 1800173 1990199 1400207 1050223 1330231 1000363 1099439 1002710 1600736 1781341 1781358 1781366 1781374

FINALIZE THE LIST

We are almost done. However, what we do next will vary a bit depending upon whether the identifiers that we are trying to match are character or numeric variables. In the real application, upon which the example is based, both customer numbers and SKUs are actually character fields. The final step for processing character fields is to translate each space into a double quoted comma and space. We do this with the TRANWRD function.

```
input6 = TRANWRD(input5, ' ','", "');
```

This leaves us with a list of quoted, comma-delimited, strings—except for the first and last values.

37-293", "23-297", "12-360", "17-361", "34-364", "34-418", "53-420", "90-424", "83-3916", "77-4745", "89-669431", "98-669436", "26-383333", "26-383334", "1800074", "1800173", "1990199", "1400207", "1050223", "1330231", "1000363", "1099439", "1002710", "1600736", "1781341", "1781358", "1781366", "1781374

The list itself is not quoted. Is that a problem? Not really. We will put this value back into the "INPUT_LIST" macro variable.

```
call SYMPUT("INPUT_LIST ",STRIP(input6));
run;
```

When we call it in our subsequent code, a quoted reference "&INPUT_LIST" will supply the leading and trailing quotation marks.

If your identifiers are numeric, then you don't need the quotes at all. Let us return to the TRANWRD function call. Simply omit the double quotes from the replacement argument of the function.

```
input6 = TRANWRD(input5, ' ',', ');
```

Remember to use an unquoted reference (&INPUT_LIST) to the macro variable in your subsequent code.

FINAL THOUGHTS

It is worth mentioning that these lines of code can be structured as a series of nested functions for more compact code.

```
data _null_;
    LENGTH input $5000;
    input = tranwrd(strip(compbl(compress(translate(
    substr("&INPUT_LIST",1,5000),' ','09'x,' ','0A'x,' ','0D'x,' ','2C'x),
    '-0123456789 ',"K"))),' ','", "');
    call symput("INPUT_LIST ",strip(input));
run;
```

Finally, in the example, we ended up with a list which contained both customer and item numbers. Can we query both at the same time? It is possible. The actual answer depends upon how your data is structured. In the business application upon which the example was based, it was not. The interface was such that there were separate web pages to search by customer number or item number. In the case of an email like the one we processed, the user would have to paste it into both input pages in order to get both types of results. As long as there is no overlap between the sets of identifiers, there is no real issue with throwing item numbers against the customer table and customer numbers against the item table. Even if there is overlap, you will simply match both types of entities with a single value of identifier. Beyond a small amount of extra processing, I see no harm. Most of the time, users will input the type of identifier that they are looking for into the correct input. For my part, I accept brevity of code as elegance of approach, at least in this case. If this offends your sensibilities as a programmer, then you have more coding to do. Unfortunately, such extended validation is outside of the scope of this paper. As a start, I would suggest investigating regular expressions as a tool to further edit the data.

CONCLUSION

We have explored a method to process freeform text as an input to SAS Stored Process web applications. We have seen that the method is a good alternative to list input when the number of records is large and the data have distinctive identifiers. Furthermore, the technique that we have used does not greatly increase the number of lines of code in our Stored Process.

CONTACT INFORMATION

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

Jeffery A. Fallon
Cardinal Health
7000 Cardinal Place
Dublin, OH 43017
614-553-3643
jeffery.fallon@cardinalhealth.com