

CHAPTER 9

SAS Procedures

You can use all the power of SAS as you develop Stored Processes, including the many procedures that provide so much functionality in an easy-to-use package. There are several procedures that are particularly helpful with developing Stored Processes. In this chapter, we will cover

- PROC STP, which allows you to run a Stored Process from other SAS code.
- PROC JSON, which lets you produce JSON-formatted data that is used extensively by JavaScript objects used in building web pages.
- PROC STREAM, which will take some text, resolve any macro variables in it, and send the output on to somewhere, which could be a file or web page.
- PROC HTTP, which can issue HTTP requests and get the response for use in SAS.
- PROC EXPORT, which will export SAS data in various formats including CSV and tab-separated formats.
- PROC IMPORT, which will import data from various supported formats into a SAS table. This is useful to read CSV and other formats of data.

PROC STP

This procedure allows Stored Processes to be executed from a SAS program. This opens up a lot more flexibility and power for the use of Stored Processes. You can execute them in batch, interactively or on servers. It can run locally or on a server, but with its own execution environment, so it has its own work library and so on. If you want to run it on a server, then it needs some additional configuration to work that way.

The following example code in Listings 9-1 and 9-2 shows how to run a stored process from a normal SAS program. This could be run in batch overnight or in fact from anywhere you can execute SAS code.

Listing 9-1. SAS Program to run a stored process from a regular SAS program

```
* connect to metadata server ;
options metaserver=d351tq92 metaport=8561 metauser=phil metapass=goodnight_
for_president ;
* close any open ODS destinations ;
ods _all_ close;
* run Stored Process and put the ODS results into an ODS Item Store ;
proc stp program='/Products/SAS Intelligence Platform/Samples/Sample:
Cholesterol by Sex and Age Group'
odsout=store;
run;
* set the format for graphics we will produce ;
goptions device=png;
* Open an HTML destination ;
ods HTML path="%sysfunc(pathname(work))" file='test.htm' style=HTMLBlue;
* Send the output to the current ODS destination ;
proc document name=&_ODSDOC (read);
replay / levels=all;
run;
quit ;
ods HTML close;
%put _ODSDOC: %superq(_ODSDOC) ;
%put Output has been put into %sysfunc(pathname(work)) ;
```

Listing 9-2. SAS Log produced from running the program

```
1 * connect to metadata server ;
2 options metaserver=d351tq92 metaport=8561 metauser=phil
metapass=XXXXXXXXX;
3 * close any open ODS destinations ;
4 ods _all_ close;
```

```

5 * run Stored Process and put the ODS results into an ODS Item Store ;
6 proc stp program='/Products/SAS Intelligence Platform/Samples/Sample:
  Cholesterol by Sex and Age
6 ! Group'
7         odsout=store;
8 run;

```

NOTE: The Stored Process will execute locally.

NOTE: PROC_STP: ===== Proc STP Execution Starting =====

NOTE: PROC_STP: ===== Stored Process: /Products/SAS Intelligence
Platform/Samples/Sample:
Cholesterol by Sex and Age Group =====

>>> SAS Macro Variables:

```

_CLIENT=PROCSTP TKESTP Windows X64_SRV12 X86_64 6.2
_METAPERSON=phil
_METAUSER=phil@!*(generatedpassworddomain)*!
_ODSDEST=DOCUMENT
_ODSDOC=APSWORK._odsdoc00000001
_RESULT=STREAM

```

```

2 %STPBEGIN;
3
4 proc format;
5     value AgeAtStart low-35 = '< 36'
6                     36-45  = '36 - 45'
7                     46-55  = '46-55'
NOTE: Format AGEATSTART has been output.
8                     56-high = '> 55';
9 run;

```

NOTE: PROCEDURE FORMAT used (Total process time):

```

real time          0.04 seconds
cpu time           0.01 seconds

```

```

9 ! quit;
10

```

CHAPTER 9 SAS PROCEDURES

```
11 title 'Cholesterol by Sex and Age Group';
12 footnote "Generated %sysfunc(datetime()), datetime19.);";
13
14 proc sgpanel data=sashelp.heart;
15     panelby sex / columns=1
16             novarname;
17     hbox Cholesterol / category=AgeAtStart;
18     format AgeAtStart AgeAtStart.;
19 run;
19 ! quit;
```

NOTE: There were 5209 observations read from the data set SASHELP.HEART.

NOTE: PROCEDURE SGPANEL used (Total process time):

| | |
|-----------|--------------|
| real time | 0.16 seconds |
| cpu time | 0.11 seconds |

```
20
21 %STPEND;
```

NOTE: PROC_STP: ===== Stored Process: /Products/SAS Intelligence Platform/
Samples/Sample:

Cholesterol by Sex and Age Group Return Status = 0 =====

NOTE: PROC_STP: ===== Proc STP Execution Ending =====

NOTE: PROCEDURE STP used (Total process time):

| | |
|-----------|--------------|
| real time | 0.60 seconds |
| cpu time | 0.35 seconds |

```
9 * set the format for graphics we will produce ;
10 goptions device=png;
11 * Open an HTML destination ;
12 ods HTML path="%sysfunc(pathname(work))" file='test.htm'
    style=HTMLBlue;
```

NOTE: Writing HTML Body file: test.htm

```
13 * Send the output to the current ODS destination ;
14 proc document name=&_ODSDOC (read);
15     replay / levels=all;
16 run;
```

NOTE: The data set WORK.DATA1 has 4 observations and 21 variables.

NOTE: Format AGEATSTART has been output.

NOTE: There were 4 observations read from the data set WORK.DATA1.

NOTE: PROCEDURE FORMAT used (Total process time):

| | |
|-----------|--------------|
| real time | 0.03 seconds |
| cpu time | 0.00 seconds |

17 quit ;

NOTE: PROCEDURE DOCUMENT used (Total process time):

| | |
|-----------|--------------|
| real time | 1.59 seconds |
| cpu time | 0.37 seconds |

18 ods HTML close;

19 %put _ODSDOC: %superq(_ODSDOC) ;

_ODSDOC: APSWORK._odsdoc00000001

20 %put Output has been put into %sysfunc(pathname(work)) ;

Output has been put into

C:\Users\phil\AppData\Local\Temp\2\SAS Temporary Files_TD11484_D351TQ92_

The SAS Output is written to the location shown in the log, and the directory listing of where PROC STP wrote the output to is also listed in the log. Notice that the PROC DOCUMENT created a graphic file (Figure 9-1) called SGPanel.png (Figure 9-2). The type was set by device= on the goptions statement, and the name defaults to the procedure that was used to produce the graphic. If we have multiple graphics, then they get a sequence number on the end, for example, SGPanel.png, SGPanel1.png, SGPanel2.png, and so on.

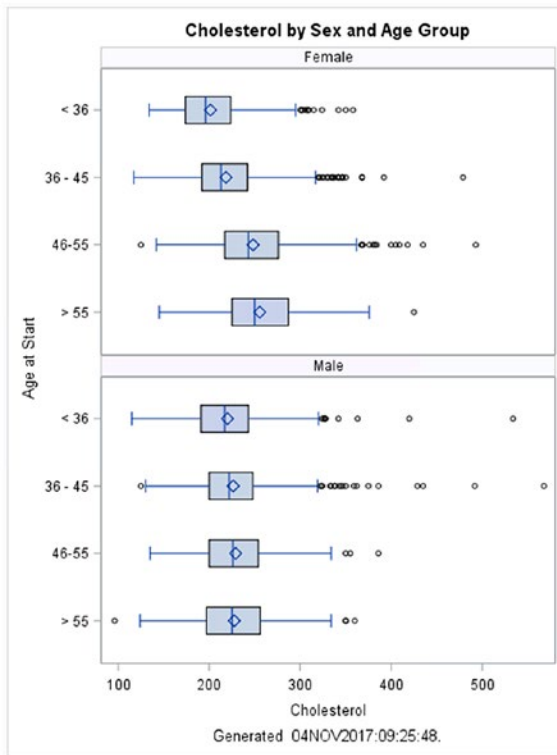


Figure 9-1. Output produced by the previous program

| Name | Date modified | Type | Size |
|--------------------------|------------------|----------------|-------|
| APSWORK1 | 04/11/2017 09:25 | File folder | |
| formats.sas7bcac | 04/11/2017 09:25 | SAS Catalog | 17 KB |
| profile.sas7bcac | 04/11/2017 09:25 | SAS Catalog | 5 KB |
| sasgopt.sas7bcac | 04/11/2017 09:25 | SAS Catalog | 5 KB |
| sasmac1.sas7bcac | 04/11/2017 09:25 | SAS Catalog | 13 KB |
| sasmac2.sas7bcac | 04/11/2017 09:25 | SAS Catalog | 13 KB |
| sasmac3.sas7bcac | 04/11/2017 09:25 | SAS Catalog | 13 KB |
| SASMONO.FOT | 04/11/2017 09:25 | FOT File | 2 KB |
| SASMONOB.FOT | 04/11/2017 09:25 | FOT File | 2 KB |
| sastmp-00000001.sas7bitm | 04/11/2017 09:25 | SAS Item Store | 96 KB |
| sastmp-00000009.sas7bitm | 04/11/2017 09:25 | SAS Item Store | 96 KB |
| SGPanel.png | 04/11/2017 09:25 | PNG image | 24 KB |
| test.htm | 04/11/2017 09:25 | HTM File | 36 KB |

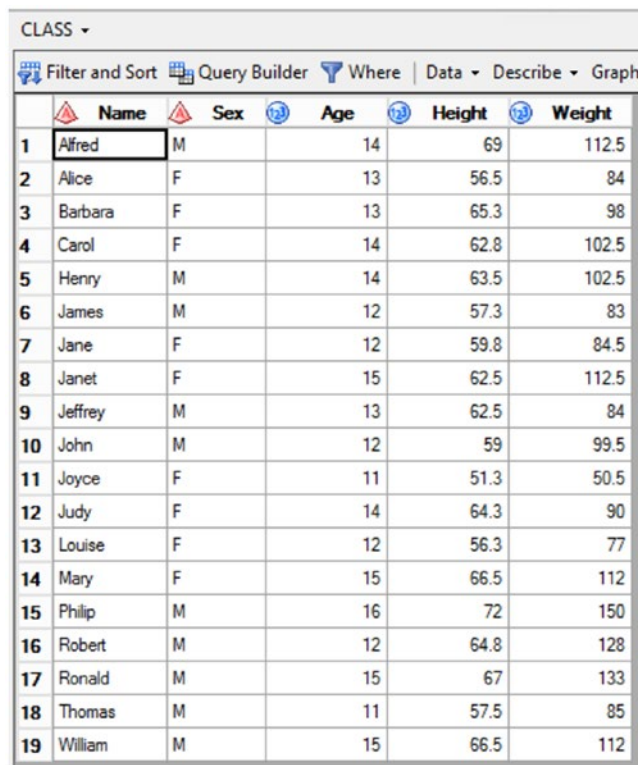
Figure 9-2. Temporary files in work area used by the program run

To read more about the STP procedure in SAS 9.4, you can use this link: <http://support.sas.com/documentation/cdl/en/stpug/68399/HTML/default/p0yy4kd3k4dc03n1mcd76hog6y2u.htm>. Remember that Proc STP is only available from SAS 9.3 onward.

PROC JSON

In SAS 9.4, there is a new procedure called PROC JSON which will create data in JSON format from any data that SAS can read. This enables us to create JSON output to be used in JavaScript objects from virtually any other data source. Some options are provided to customize the JSON produced, which enables very flexible JSON output to be created.

Figure 9-3 shows a table we will use in an example.



| | Name | Sex | Age | Height | Weight |
|----|---------|-----|-----|--------|--------|
| 1 | Alfred | M | 14 | 69 | 112.5 |
| 2 | Alice | F | 13 | 56.5 | 84 |
| 3 | Barbara | F | 13 | 65.3 | 98 |
| 4 | Carol | F | 14 | 62.8 | 102.5 |
| 5 | Henry | M | 14 | 63.5 | 102.5 |
| 6 | James | M | 12 | 57.3 | 83 |
| 7 | Jane | F | 12 | 59.8 | 84.5 |
| 8 | Janet | F | 15 | 62.5 | 112.5 |
| 9 | Jeffrey | M | 13 | 62.5 | 84 |
| 10 | John | M | 12 | 59 | 99.5 |
| 11 | Joyce | F | 11 | 51.3 | 50.5 |
| 12 | Judy | F | 14 | 64.3 | 90 |
| 13 | Louise | F | 12 | 56.3 | 77 |
| 14 | Mary | F | 15 | 66.5 | 112 |
| 15 | Philip | M | 16 | 72 | 150 |
| 16 | Robert | M | 12 | 64.8 | 128 |
| 17 | Ronald | M | 15 | 67 | 133 |
| 18 | Thomas | M | 11 | 57.5 | 85 |
| 19 | William | M | 15 | 66.5 | 112 |

Figure 9-3. sashelp.class table to use in the following example

Here is the Stored Process code to create some JSON:

```
proc json out=_webout;
  export &table / tablename="&table";
run;
```

Here is some of the JSON that is produced by it, when I feed in a parameter of &table=sashelp.class:

```
{"SASJSONExport":"1.0","SASTableData+sashelp.class":[{"Name":"Alfred","Sex":"M","Age":14,"Height":69,"Weight":112.5},{"Name":"Alice","Sex":"F","Age":13,"Height":56.5,"Weight":84},{"Name":"Barbara","Sex":"F","Age":13,"Height":65.3,"Weight":98},{"Name":"Carol","Sex":"F","Age":14,"Height":62.8,"Weight":102.5},{"Name":"Henry","Sex":"M","Age":14,"Height":63.5,"Weight":102.5},{"Name":"James","Sex":"M","Age":12,"Height":57.3,"Weight":83},{"Name":"Jane","Sex":"F","Age":12,"Height":59.8,"Weight":84.5},{"Name":"Janet","Sex":"F","Age":15,"Height":62.5,"Weight":112.5},{"Name":"Jeffrey","Sex":"M","Age":13,"Height":62.5,"Weight":84},{"Name":"John","Sex":"M","Age":12,"Height":59,"Weight":99.5},{"Name":"Joyce","Sex":"F","Age":11,"Height":51.3,"Weight":50.5},{"Name":"Judy","Sex":"F","Age":14,"Height":64.3,"Weight":90},{"Name":"Louise","Sex":"F","Age":12,"Height":56.3,"Weight":77},{"Name":"Mary","Sex":"F","Age":15,"Height":66.5,"Weight":112},{"Name":"Philip","Sex":"M","Age":16,"Height":72,"Weight":150},{"Name":"Robert","Sex":"M","Age":12,"Height":64.8,"Weight":128},{"Name":"Ronald","Sex":"M","Age":15,"Height":67,"Weight":133},{"Name":"Thomas","Sex":"M","Age":11,"Height":57.5,"Weight":85},{"Name":"William","Sex":"M","Age":15,"Height":66.5,"Weight":112}]]
```

You can trim some extraneous information from the JSON using the **nosastags** option on PROC JSON. Here is the output we get if we use NOSASTAGS. Note: In the previous output, I have **bolded** what is dropped by using NOSASTAGS.

```
[{"Name":"Alfred","Sex":"M","Age":14,"Height":69,"Weight":112.5},{"Name":"Alice","Sex":"F","Age":13,"Height":56.5,"Weight":84},{"Name":"Barbara","Sex":"F","Age":13,"Height":65.3,"Weight":98},{"Name":"Carol","Sex":"F","Age":14,"Height":62.8,"Weight":102.5},{"Name":"Henry","Sex":"M","Age":14,"Height":63.5,"Weight":102.5},{"Name":"James","Sex":"M","Age":12,"Height":57.3,"Weight":83},{"Name":"Jane","Sex":"F","Age":12,"Height":59.8,"Weight":84.5},{"Name":"Janet","Sex":"F","Age":15,"Height":62.5,"Weight":112.5},
```



```
{ "Name": "Jeffrey", "Sex": "M", "Age": 13, "Height": 62.5, "Weight": 84 }, { "Name":
"John", "Sex": "M", "Age": 12, "Height": 59, "Weight": 99.5 }, { "Name": "Joyce", "Sex":
"F", "Age": 11, "Height": 51.3, "Weight": 50.5 }, { "Name": "Judy", "Sex": "F", "Age":
14, "Height": 64.3, "Weight": 90 }, { "Name": "Louise", "Sex": "F", "Age": 12, "Height":
56.3, "Weight": 77 }, { "Name": "Mary", "Sex": "F", "Age": 15, "Height": 66.5, "Weight":
112 }, { "Name": "Philip", "Sex": "M", "Age": 16, "Height": 72, "Weight": 150 }, { "Name":
"Robert", "Sex": "M", "Age": 12, "Height": 64.8, "Weight": 128 }, { "Name": "Ronald",
"Sex": "M", "Age": 15, "Height": 67, "Weight": 133 }, { "Name": "Thomas", "Sex": "M",
"Age": 11, "Height": 57.5, "Weight": 85 }, { "Name": "William", "Sex": "M", "Age": 15,
"Height": 66.5, "Weight": 112 } ]
```

You can also lay out the JSON produced in an easier to read form using the **pretty** option on PROC JSON. This makes it far easier to read. Here is the first part of the output produced:

```
{
  "SASJSONExport": "1.0 PRETTY",
  "SASTableData+class": [
    {
      "Name": "Alfred",
      "Sex": "M",
      "Age": 14,
      "Height": 69,
      "Weight": 112.5
    },
    {
      "Name": "Alice",
      "Sex": "F",
      "Age": 13,
      "Height": 56.5,
      "Weight": 84
    }
  ],
}
```

If we use the PRETTY and NOSASTAGS options, then here is the first part of the output produced:

```
[
  {
    "Name": "Alfred",
    "Sex": "M",
    "Age": 14,
    "Height": 69,
    "Weight": 112.5
  },
  {
    "Name": "Alice",
    "Sex": "F",
    "Age": 13,
    "Height": 56.5,
    "Weight": 84
  },
]
```

As we know, PROC JSON can produce JSON data, and there are many JavaScript objects that can use JSON data as input. There is an object called jqGrid which has a URL parameter which lets you point to a data source that is in JSON format. Here is a JavaScript snippet of code which would be used to define where the data is for that object. In this case, I can point the object to the SAS Stored Process Web Application, which will call a Stored Process to provide the JSON data:

```
url: 'http://localhost/SASStoredProcess/do?_program=%2FUser+Folders%2Fphil%2FMy+Folder%2Fjson',
```

In that Stored Process, I can use PROC JSON to produce the JSON data which is needed to feed the object. If you need to customize the standard JSON in order to fit some specific requirements for a JavaScript object, then you are able to use the write statement to write out extra structure to your JSON. In the case of using the jqGrid object, it needs a slightly different JSON layout to standard. I used the code here to get my JSON in the right format for using with the jqGrid object:

```
proc json out=_webout pretty nosastags;
  write open object;
  write values "rows";
  write open array;
  export sashelp.orsales;
  write close;
  write close;
run;
```

Notice in the preceding code that we write to `_webout`, which when used in a Stored Process with the Stored Process Web Application will stream data directly to the browser.

Another useful thing you might do with PROC JSON is that when you are using some kind of static HTML, you might want to get a list of variables that exist in the data table you are using, so you can automatically generate the menus (for instance). The following code in a stored process would get variable name, type, and label from a table, and then send it back to the browser where JavaScript could make use of that data:

```
proc contents data=sashelp.class out=contents noprint;
run;

filename _webout temp;

proc json out=_webout nosastags pretty;
  export contents(keep=name type memlabel) ;
run;
```

The JSON that would be generated is as follows in Listing 9-3.

Listing 9-3. Generated JSON

```
[
  {
    "MEMLABEL": "Student Data",
    "NAME": "Age",
    "TYPE": 1
  },
  {
    "MEMLABEL": "Student Data",
    "NAME": "Height",
```

```

    "TYPE": 1
  },
  {
    "MEMLABEL": "Student Data",
    "NAME": "Name",
    "TYPE": 2
  },
  {
    "MEMLABEL": "Student Data",
    "NAME": "Sex",
    "TYPE": 2
  },
  {
    "MEMLABEL": "Student Data",
    "NAME": "Weight",
    "TYPE": 1
  }
]

```

Here is a full working example in Listing 9-4.

Listing 9-4. Full example

```

var myfilter = "http://<server>/SASStoredProcess/do?_program=<program>%2F
<stored process name>";

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:mso="urn:schemas-
microsoft-com:office:office" xmlns:msdt="uuid:C2F41010-65B3-11d1-A29F-
00AA00C14882">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />

<script type="text/javascript" charset="utf8" src="../../webres/jquery--
3.2.1.min.js"></script>

```

```

<script type="text/javascript" charset="utf8" src="../../webres/jquery-
ui-1.12.1/jquery-ui.js"></script>
<link rel="stylesheet" type="text/css" href="../../webres/jquery-ui-
themes-1.9.2/base/jquery-ui.css" />
<link rel="stylesheet" type="text/css" href="../../webres/DT4/datatables.
min.css"/>
<script type="text/javascript" src="../../webres/DT4/datatables.min.js">
</script>

<script type="text/javascript">
var myfilter = "http://<server>/SASStoredProcess/do?_program=<program>%2F
<stored process name>";
var call = $.ajax({ url: myfilter,
    type: "GET",
    dataType: "json",
});
call.done(function (data,textStatus, jqXHR){
    $('#example2 tbody').off('click', 'tr');
    table2 = $('#example2').DataTable({
        dom: 'l<B>frtip', //'Bfrtip', /*dtsettings,*/
        bLengthChange: true,
        buttons: [
            'excelHtml5'
        ],
        "orderCellsTop": true,
        "bDestroy": true,
        "bProcessing": true,
        "lengthMenu": [[10, 25, 50, -1], [10, 25, 50, "All"]],
        "aaData": data,
        "scrollX": true,
        "aoColumns": [
            { 'sTitle': 'Year','mData': 'Year', 'sClass': 'center_column' }
            ,{ 'sTitle': 'Quarter','mData': 'Quarter', 'sClass': 'center_
column' }
            ,{ 'sTitle': 'Product_Line','mData': 'Product_Line', 'sClass':
'center_column' }
        ]
    });

```

```

    ,{ 'sTitle': 'Product_Category','mData': 'Product_
    Category', 'sClass': 'center_column' }
    ,{ 'sTitle': 'Product_Group','mData': 'Product_
    Group', 'sClass': 'center_column' }
    ,{ 'sTitle': 'Quantity','mData': 'Quantity', 'sClass':
    'center_column' }
    ,{ 'sTitle': 'Profit','mData': 'Profit', 'sClass': 'center_
    column' }
    ,{ 'sTitle': 'Total_Retail_Price','mData': 'Total_Retail_
    Price', 'sClass': 'center_column' }
  ]
  ,"oLanguage": {
    "sSearch": "Search All Columns: "
  }
});
});
call.fail(function (jqXHR,textStatus,errorThrown){
  alert('unable to obtain data from SAS');
});
</script>
</head>
<body>
<table id='example2' class='display' width='100%'><tfoot><tr></tr>
</tfoot></table>
</body>
</html>

```

PROC STREAM

There are various ways that we can get code into a web browser. We could just write a simple file and then load that into the web browser, such as by creating a file report. HTML and opening it. Usually a better way to do this is to use SAS/Intrnet or a Stored Process to stream code to the browser. From a Stored Process, you can do this by writing lines to the `_webout` fileref. This could be done by writing to it from a data step, but you can also use PROC STREAM to do this.

Streaming with a Data Step

The data step can be used to stream by writing to `_webout`, when used from a Stored Process in the SAS Stored Process Web Application. If you use any macro language in what is streamed, then it would not be resolved. For example, the following Stored Process

```
%let name=Phil Mason ;
data _null_ ;
  file _webout ;
  input ;
  put _infile_ ;
  cards ;
<HTML>
<h1>Hello &name</h1>
</HTML>
;;
run ;
```

Listing 9-5 would produce this output (Figure 9-4), when run through the SAS Stored Process Web Application.

Hello &name

Figure 9-4. This is displayed when we view the HTML generated from Listing 9-1

If you want to resolve the macro language before streaming the HTML code out, then you can use something like the `resolve()` function. You would then change your Stored Process code to be like this:

```
%let name=Phil Mason ;
data _null_ ;
  file _webout ;
  input ;
  line=resolve(_infile_) ;
  put line ;
  cards ;
<HTML>
```

```
<h1>Hello &name</h1>
</HTML>
;;
run ;
```

This will produce the following output (Figure 9-5) in the web browser.

Hello Phil Mason

Figure 9-5. Using resolve function means we see this from the HTML generated

There are some problems using the resolve function in the data step, particularly that there is a limitation on size. So the text for each line can only ever fit into the size of a variable, which is 32K maximum. If macro language expands to take more space than that, then it will be truncated. This can lead to unexpected results and errors. There can be additional issues with escaped HTML characters such as & which is not a SAS macro variable, although SAS will think it is.

Streaming with PROC STREAM

Another way to stream data is to use PROC STREAM. The program would look like this:

```
proc stream outfile=_webout;
  BEGIN
  <HTML>
  <h1>Hello &name</h1>
  </HTML>
  ;;;
run;
```

PROC STREAM reads text that appears after the BEGIN statements up to the four semi-colons which indicate the end of input. It then writes the lines to the _webout fileref. As the lines are written, any macro references are resolved, and unlike the RESOLVE() function, there is no limit of data size. This is a hugely powerful facility. In the simplest example, we could have an HTML file where we have a macro variable for the title, which would be replaced as the HTML is streamed to the browser.

A more complex example shows how macro variables and other macro language such as macro functions are all resolved when used within PROC STREAM:

```
%let name=Phil Mason ;
proc stream outfile=_webout ;
BEGIN
<HTML>
<h1>Hello &name</h1>
The time is %sysfunc(time(),time.)
</HTML>
;;;
run ;
```

The output produced by this is displayed in Figure 9-6.

Hello Phil Mason

The time is 15:48:30

Figure 9-6. Output with text resolved from macro function call

An even more complex example shows how all macro language is resolved by PROC STREAM, so even if you use macro programs, they will resolve and what they produce will be included into the stream. If you have a Stored Process with the following code

```
%let name=Phil Mason ;
%macro loop(n) ;
  %do i=1 %to &n ;
    Counting: &i <br>
  %end ;
%mend loop ;
proc stream outfile=_webout ;
BEGIN
<HTML>
<h1>Hello &name</h1>
The time is %sysfunc(time(),time.)
```

```
<p>  
%loop(5)  
</HTML>  
;;;;  
run ;
```

it produces the following output (Figure 9-7).

Hello Phil Mason

The time is 15:55:52

Counting: 1
Counting: 2
Counting: 3
Counting: 4
Counting: 5

Figure 9-7. HTML produced by macro program looping and generating text

Streaming RTF Files with PROC STREAM

PROC STREAM also works well with other kinds of text files, such as RTF files. You could make a letter and save it as RTF and replace certain parts with macro variables, and then by using PROC STREAM, you could effectively carry out a mail merge to produce a customized letter for a set of macro variables.

So if I go into Microsoft Word and make a document like the one shown in Figure 9-8

Dear Sir,

The date is %sysfunc(date()),date.).

Yours sincerely,
&name

Figure 9-8. Document in Microsoft Word with some macro code

I can then save that as an RTF file. This file will have many lines of RTF code, but the lines of interest to use are the ones with the macro statements on them. These are

```
\par The date is %sysfunc(date()),date.).
\par
\par Yours sincerely,
\par &name
```

The following code can be used to read the RTF file in, resolve any macro language, and write it to a new RTF file: “&streamdelim;”

```
%let name=Phil Mason ;
filename oldrtf "F:\letter.rtf" recfm=v lrecl=32767;
filename newrtf "F:\letter1.rtf" recfm=v lrecl=32767;
proc stream outfile=newrtf quoting=both asis;
begin
&streamdelim;
%include oldrtf;
;;;
```

DOSUB

We can also run SAS code while processing the PROC STREAM by using the dosub function with a %sysfunc. SAS code to be run is pointed to by a fileref, and then the dosub uses that fileref. If you have some code that you want to run during PROC STREAM, such as this

```
filename myHTML "temp.txt";
data _null_;
  file myHTML;
  set sashelp.class end=end ;
  if _n_=1 then put '<h1>This is my heading</h1><table>';
  put '<tr><td>' name '</td>' '<td>' age '</td></tr>' ;
  if end then put '</table>' ;
run;
```

you can run the preceding code by pointing to the file it is in (dosub.sas) and using the dosub function to run it in PROC STREAM as follows:

```
filename makeHTML 'c:\test\dosub.sas' ;
filename myHTML "temp.txt";
filename report "report.HTML";

proc stream outfile=report ;
begin
%let abc=%sysfunc(dosub(makeHTML));
%include myHTML;
;;;
```

This runs dosub.sas, which writes HTML to temp.txt. Then in PROC STREAM, we include temp.txt which writes the HTML that the data step generated out to report.HTML. Being able to run code on the fly from within PROC STREAM adds a huge amount of power and flexibility to the use of PROC STREAM.

If you wanted to read a file in to be streamed, without having any macro language resolved, then you can use the readfile keyword. Often this might be used to get some content and put it between PRE tags in HTML, since they are used for pre-formatted content. If you had a log you wanted shown as is in a non-proportional font, then that would be easily done as shown in the code that follows.

For example, the following code uses readfile to read in some text using the exact formatting it had in the file. We then use the PRE HTML tags to enclose that text which indicates that it is pre-formatted text. You can also see that the *&name* which is inside the pre-formatted text is not resolved. However, the **&name** which is inside the text of PROC STREAM is resolved.

```
%let name=Phil ;
filename text temp ;
data _null_ ;
  file text ;
  input ;
  put _infile_ ;
  cards4 ;
Here is a line
And here is the next line
```

```

Here is a macro variable - &name
;;;
run ;

filename dest temp ;
proc stream outfile=dest ;
begin
<PRE>
&streamdelim readfile text ;
</PRE>
My name is &name
;;;

data _null_;
  infile dest ;
  input ;
  put _infile_ ;
run;

```

The code produces the following output in the log:

```

341 %let name=Phil ;
342 filename text temp ;
343 data _null_;
344   file text ;
345   input ;
346   put _infile_;
347   cards4 ;

```

NOTE: The file TEXT is:

```

Filename=C:\Users\phil\AppData\Local\Temp\2\SAS Temporary
Files\_TD10384_D351TQ92_\#LN00216,
RECFM=V,LRECL=32767,File Size (bytes)=0,
Last Modified=28 September 2017 21:46:30 o'clock,
Create Time=28 September 2017 21:46:30 o'clock

```

NOTE: 4 records were written to the file TEXT.

The minimum record length was 80.

The maximum record length was 80.

CHAPTER 9 SAS PROCEDURES

NOTE: DATA statement used (Total process time):

| | |
|-----------|--------------|
| real time | 0.01 seconds |
| cpu time | 0.01 seconds |

```
352 ;;;  
353 run ;  
354  
355 filename dest temp ;  
356 proc stream outfile=dest ;  
357 begin  
358 <PRE>  
359 &streamdelim readfile text ;
```

NOTE: PROCEDURE STREAM used (Total process time):

| | |
|-----------|--------------|
| real time | 0.00 seconds |
| cpu time | 0.00 seconds |

```
360 </PRE>  
361 My name is &name  
362 ;;;
```

```
363  
364 data _null_ ;  
365   infile dest ;  
366   input ;  
367   put _infile_ ;  
368 run;
```

NOTE: The infile DEST is:

Filename=C:\Users\phil\AppData\Local\Temp\2\SAS Temporary
Files_TD10384_D351TQ92_\#LN00217,
RECFM=V,LRECL=32767,File Size (bytes)=358,
Last Modified=28 September 2017 21:46:30 o'clock,
Create Time=28 September 2017 21:46:30 o'clock

```
<PRE>  
Here is a line  
And here is the next line  
Here is a macro variable - &name
```

```
</PRE>My name is Phil
```

NOTE: 6 records were read from the infile DEST.

The minimum record length was 5.

The maximum record length was 80.

NOTE: DATA statement used (Total process time):

real time 0.01 seconds

cpu time 0.01 seconds

You can force a new line in the streamed output by using `newline`. You must have the stream delimiter followed by `newline` in order to do this. There is no other way to be absolutely sure of having a line break in a particular place.

For example, the following code redefines the stream delimiter to be `_delim_` and then uses that with `newline` to make it go to a new line in the output.

```
filename sample temp ;
proc stream outfile=sample resetdelim='_delim_'; begin
Line 1
_delim_ newline;
Line 2

;;;data _null_ ;
  infile sample ;
  input ;
  put _infile_ ;
run ;
```

The log for this is as follows.

```
393 filename sample temp ;
394 proc stream outfile=sample resetdelim='_delim_'; begin
395 Line 1
NOTE: PROCEDURE STREAM used (Total process time):
  real time           0.00 seconds
  cpu time            0.00 seconds

396 _delim_ newline;
397 Line 2
398 ;;;;
```

```
399
400 data _null_ ;
401     infile sample ;
402     input ;
403     put _infile_ ;
404 run ;
```

NOTE: The infile SAMPLE is:
Filename=C:\Users\phil\AppData\Local\Temp\2\SAS Temporary
Files_TD10384_D351TQ92_\#LN00220,
RECFM=V,LRECL=32767,File Size (bytes)=16,
Last Modified=28 September 2017 21:59:28 o'clock,
Create Time=28 September 2017 21:59:28 o'clock

Line 1
Line 2

NOTE: 2 records were read from the infile SAMPLE.
The minimum record length was 6.
The maximum record length was 6.

NOTE: DATA statement used (Total process time):
real time 0.00 seconds
cpu time 0.00 seconds

If we didn't redefine the delimiter, then the code would have been like the following and produced the same result:

```
filename sample temp ;
proc stream outfile=sample ; begin
Line 1
&streamdelim newline;
Line 2
;;;
data _null_ ;
    infile sample ;
    input ;
    put _infile_ ;
run ;
```


If we remove the line which forces the newline, then the code will be as follows:

```
filename sample temp ;
proc stream outfile=sample ; begin
Line 1
Line 2
;;;

data _null_ ;
  infile sample ;
  input ;
  put _infile_ ;
run ;
```

And this produces the following result:

```
451 filename sample temp ;
452 proc stream outfile=sample ; begin
NOTE: PROCEDURE STREAM used (Total process time):
      real time          0.00 seconds
      cpu time           0.00 seconds

453 Line 1
454 Line 2
455 ;;;;

456
457 data _null_ ;
458   infile sample ;
459   input ;
460   put _infile_ ;
461 run ;
```

NOTE: The infile SAMPLE is:

```
Filename=C:\Users\phil\AppData\Local\Temp\2\SAS Temporary
Files\_TD10384_D351TQ92_\#LN00225,
RECFM=V,LRECL=32767,File Size (bytes)=14,
Last Modified=28 September 2017 22:07:29 o'clock,
Create Time=28 September 2017 22:07:29 o'clock
```

Line 1Line 2

NOTE: 1 record was read from the infile SAMPLE.

The minimum record length was 12.

The maximum record length was 12.

NOTE: DATA statement used (Total process time):

real time 0.00 seconds

cpu time 0.01 seconds

You should always check the performance of code using features like dosub, as it can be quite inefficient in some cases and may require some tuning or careful design. Read more about PROC STREAM in the SAS 9.4 documentation here: <http://documentation.sas.com/?docsetId=proc&docsetTarget=p06pqn7v5nkz02n0zkpq7832j1yp.htm&docsetVersion=9.4&locale=en>.

PROC HTTP

The HTTP procedure lets you issue HTTP requests. This means that you can make GET or POST requests as well as other kinds of requests. You send data in the request and can receive a response. Then you'll be able to effectively make a call to a URL using PROC HTTP and get the results of it. You can parse the output returned and extract data from it or do something else with that output. You could call a web service, Stored Process, or virtually any web page.

Example Accessing a Web Page

The simplest usage of PROC HTTP is simply to open a web page and receive the response, which will usually be the HTML. The following code in Listings 9-5 and 9-6 opens the SAS home page and collects the HTML into a temporary file under the fileref resp.

Listing 9-5. SAS Program that opens a web page and writes out response

```
filename resp TEMP;
proc http
  url="http://www.sas.com"
  out=resp;
run;
```

Listing 9-6. SAS Log of Listing 9-5

```

73  filename resp TEMP;
74  proc http
75      url="http://www.sas.com"
76      out=resp;
77  run;

```

NOTE: PROCEDURE HTTP used (Total process time):

| | |
|-----------|--------------|
| real time | 1.87 seconds |
| cpu time | 0.01 seconds |

NOTE: 200 OK

Example Using a Web Service

The following code in Listing 9-7 allocates two temporary files, one which is used as input to a web service and Listing 9-8 is used to receive the output.

Listing 9-7. SAS Program that opens a web service passing in a value

```

dm 'log;clear' ;
filename in temp ;
filename out temp ;
data _null_;
    file in;
    input;
    put _infile_;
    datalines4;
Celsius=0
;;;

proc http
    in=in
    out=out
    url="https://www.w3schools.com/xml/tempconvert.aspx/CelsiusToFahrenheit"
    method="post"

```

```
ct="application/x-www-form-urlencoded"
verbose
;
run;

data _null_ ;
  infile out ;
  input ;
  put _infile_ ;
run ;
```

Listing 9-8. SAS Log from Listing 9-7 that uses a web service

```
84  dm 'log;clear' ;
85
86  filename in temp ;
87  filename out temp ;
88  data _null_;
89    file in;
90    input;
91    put _infile_;
92    datalines4;
```

NOTE: The file IN is:
Filename=C:\Users\phil\AppData\Local\Temp\2\SAS Temporary Files\
TD15020_D351TQ92_\#LN00044,
RECFM=V,LRECL=32767,File Size (bytes)=0,
Last Modified=05 September 2017 21:45:59 o'clock,
Create Time=05 September 2017 21:45:59 o'clock

NOTE: 1 record was written to the file IN.
The minimum record length was 80.
The maximum record length was 80.

NOTE: DATA statement used (Total process time):
real time 0.00 seconds
cpu time 0.01 seconds

```

94   ;;;
95
96   proc http
97     in=in
98     out=out
99     url="https://www.w3schools.com/xml/tempconvert.aspx/
      CelsiusToFahrenheit"
100    method="post"
101    ct="application/x-www-form-urlencoded"
102    verbose
103    ;
URL      = https://www.w3schools.com/xml/tempconvert.aspx/
          CelsiusToFahrenheit
METHOD   = post
CT       = application/x-www-form-urlencoded
In       = C:\Users\phil\AppData\Local\Temp\2\SAS Temporary
Files\_TD15020_D351TQ92_\#LN00044
Out      = C:\Users\phil\AppData\Local\Temp\2\SAS Temporary
Files\_TD15020_D351TQ92_\#LN00045
104 run;

NOTE: PROCEDURE HTTP used (Total process time):
      real time          0.30 seconds
      cpu time           0.03 seconds

NOTE: 200 OK

105
106 data _null_ ;
107   infile out ;
108   input ;
109   put _infile_ ;
110 run ;

```

NOTE: The infile OUT is:

```
Filename=C:\Users\phil\AppData\Local\Temp\2\SAS Temporary Files\  
TD15020_D351TQ92_\#LN00045,  
RECFM=V,LRECL=32767,File Size (bytes)=98,  
Last Modified=05 September 2017 21:45:59 o'clock,  
Create Time=05 September 2017 21:45:59 o'clock
```

```
<?xml version="1.0" encoding="utf-8"?>  
<string xmlns="https://www.w3schools.com/xml/">32</string>
```

NOTE: 2 records were read from the infile OUT.

The minimum record length was 38.

The maximum record length was 58.

NOTE: DATA statement used (Total process time):

```
real time          0.01 seconds  
cpu time           0.01 seconds
```

Note The URL access method on the FILENAME statement provides quite similar functionality to PROC HTTP. It might be a better option if you are considering PROC HTTP. You can read about PROC HTTP in SAS 9.4 here: <https://support.sas.com/documentation/cdl/en/proc/68954/HTML/default/viewer.htm#obdg5vmrpyi7jn1pbgbje2atoov.htm>.

PROC EXPORT

PROC EXPORT takes a SAS table and converts to another format supported.

The converted formats could be

- CSV
- EXCEL
- JMP

These are delimited files, like a CSV, but with another delimiter.

The file produced is written to a fileref specified using OUTFILE. From a Stored Process running through the Stored Process Web Application, we could specify this as

`_webout` in order to send the exported data directly back to the browser. The `REPLACE` parameter can be specified in order to replace any file which is there already. When streaming to the browser, we always need to specify this. For example, to stream CSV data from a specific table back to the browser, we could use a `PROC EXPORT` like this:

```
proc export data=sashelp.orsales outfile=_webout dbms=csv replace;
run;
```

Use Code with a Macro Variable for the Table Name

To generalize this code so it can be used for different tables, we can replace the table name with a macro variable. Here is that code:

```
proc export data=&table outfile=_webout dbms=csv replace;
run;
```

Call Stored Process Passing Parameter for Table

If we now create a Stored Process containing the previous SAS code, then we will be able to call that by using the SAS Stored Process Web Application as follows, remembering to specify a value for the table to be exported. This will then send the table converted to a CSV back to the web browser:

```
http://localhost/SASStoredProcess/do?_program=/User+Folders/phil/My+Folder/
csv&table=sashelp.class
```

Use Code in JavaScript to Feed Objects

If you needed CSV data to feed to a JavaScript object, then you could use a line of JavaScript like the following:

```
$.get('http://localhost/SASStoredProcess/do?_program=/User+Folders/phil/
My+Folder/csv&table=sashelp.class', function(csv)
```

PROC IMPORT

This can be used to read data of various formats into SAS tables. When used with Stored Processes, you could use this to import the data from files uploaded to the server.

Here is some HTML code which will prompt the user for some files to upload to the server. Note that we set the *method* to **POST**, so that we can handle the files being posted in the HTTP request; the alternative would be **GET** but that would not work in all cases. The *enctype* is set so that we can send multiple files in the upload.

```
<HTML>
<form method="post" action="http://d351tq92/SASStoredProcess/do?"
enctype="multipart/form-data">
<input type="hidden" name="_program" value="/User Folders/phil/My Folder/
upload">
Enter CSV to upload and import <input name="file1" type="file"><p>
Enter CSV to upload and import <input name="file2" type="file"><p>
Enter CSV to upload and import <input name="file3" type="file"><p>
Show this many rows <input name="obs" type="text" value="10"><p>
Debug options <input name='_debug' type='text'><p>
<input type="submit" value="Run">
</form>
</HTML>
```

Figure 9-9 is what is displayed when the HTML is used. Notice that when you specify a type of “file”, you get a button which opens a dialog and lets you browse the file system and select a file. When files are uploaded, we get a bunch of automatic macro variables populated which all start with `_WEBIN_`.

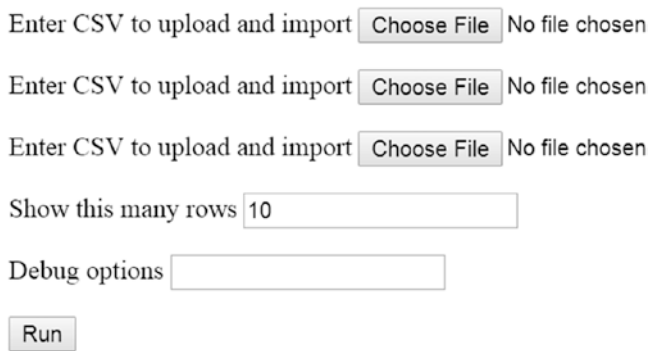


Figure 9-9. Displayed in browser from previous HTML

Here is the Stored Process code which the HTML form calls. The PROC SQL at the start of the code lets us look at the values of the `_WEBIN_` macro variables. We get a

count of the number of files uploaded which is in `_webin_file_count`, and we use that to look through each filename to carry out an import on it. We have some **code** ① to fix up the file format of the CSV so it is ready for PROC IMPORT. PROC IMPORT imports each CSV in and assigns it a table name.

```
proc sql ;
    select * from dictionary.macros
    where name like '_WEBIN_%' ;
quit ;
filename temp temp ;
%macro read_loop ;
    %if &_webin_file_count=1 %then %do ;
        %let _webin_fileref1=&_webin_fileref ;
        %let _webin_filename1=&_webin_filename ;
    %end ;
    %do i=1 %to &_webin_file_count ;
        %let csv_file=%sysfunc(pathname(&&_webin_fileref&i));
        %put &=csv_file ;
        * fix the end of line character for Proc Import ;
        data _null_ ; ①
            infile "&csv_file" sharebuffers termstr=cr ;
            file temp termstr=crlf ;
            input ;
            line=compress(_infile_,'1a'x) ;
            put line ;
        run ;
        filename in "&csv_file" ;
        proc import datafile=temp
            dbms=csv
            replace
            out=file&i ;
            getnames=yes ;
        run ;
        %let dsid=%sysfunc(open(file&i)) ;
        title "%sysfunc(attrn(&dsid,nobs),comma12.) rows imported from CSV
        file: &&_webin_filename&i" ;
```

```

%let dsid=%sysfunc(close(&dsid)) ;
title2 "Table produced: file&i" ;
proc print data=file&i(obs=&obs) ;
run ;
%end ;
%mend read_loop ;
%read_loop

```

Figure 9-10 is the kind of output you get from the PROC SQL, which shows the automatic variables that describe the file being uploaded.

| Macro Scope | Macro Variable Name | Offset into Macro Variable | Macro Variable Value |
|-------------|------------------------|----------------------------|----------------------|
| GLOBAL | __WEBIN_CONTENT_LENGTH | 0 | 2597 |
| GLOBAL | __WEBIN_CONTENT_TYPE | 0 | text/csv |
| GLOBAL | __WEBIN_FILEEXT | 0 | csv |
| GLOBAL | __WEBIN_FILENAME | 0 | Barclays 2017-10.csv |
| GLOBAL | __WEBIN_FILEREF | 0 | #LN00631 |
| GLOBAL | __WEBIN_FILE_COUNT | 0 | 1 |
| GLOBAL | __WEBIN_NAME | 0 | file1 |
| GLOBAL | __WEBIN_STREAM | 0 | __in1 |
| GLOBAL | __WEBIN_STREAM_COUNT | 0 | 1 |

Figure 9-10. Automatic macro variables available relating to reading files into stored processes from a web browser

As this section is about using PROC IMPORT, let me describe its use in this instance. PROC IMPORT specifies the file that is being read in by using the *datafile* option which in our case points to a fileref, although it can also point directly at a file. The *dbms* option specifies what the file format is, and several formats are supported such as delimited files, EXCEL, and more. The *out* option specifies what SAS table to create when the file is imported. The *replace* option specifies that a file should be overwritten if it exists already. You can read about the procedure and options in depth in the documentation.¹

¹<http://documentation.sas.com/?docsetId=proc&docsetTarget=n18jyszn33umngn14czw2qfw7thc.htm&docsetVersion=9.4&locale=en>

Summary

In this chapter, we look at some of the most useful SAS procedures for using with stored processes and building web applications:

- Proc STP allows us to run a stored process from a regular SAS program, meaning it could be run in the background or batch, or we could run several stored processes from a single SAS program.
- Proc JSON allows us to access any data that SAS can access and write JSON data out. There is quite a lot of flexibility available so we can even build quite complex JSON structures.
- Proc STREAM allows us to take a “stream” of text and send it to a destination (e.g., the web browser) and resolve all macro variables and programs as it goes. You can stream all kinds of text, such as HTML, JavaScript, CSS, RTF, CSV, and so on.
- You can use DOSUB to run SAS code while streaming text.
- Proc HTTP lets you issue HTTP GET or POST requests and capture the response for further processing.
- Proc EXPORT will convert data the SAS can access into another format such as CSV or EXCEL.
- Proc IMPORT will read in a range of different data and convert it into a SAS-supported format.