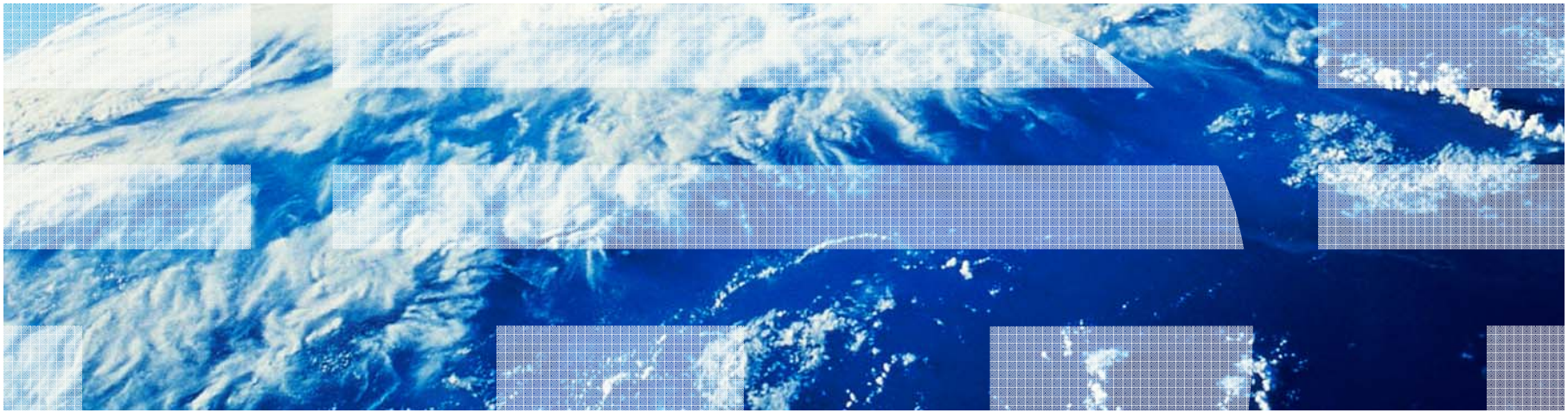
Steven Ruegsegger
IBM Microelectronics, Burlington, VT
ruegsegs@us.ibm.com

Welcome to a Smarter Planet
People want it. We can do it.

Using Datasets to Define Macro Loops and Local Macro Variables



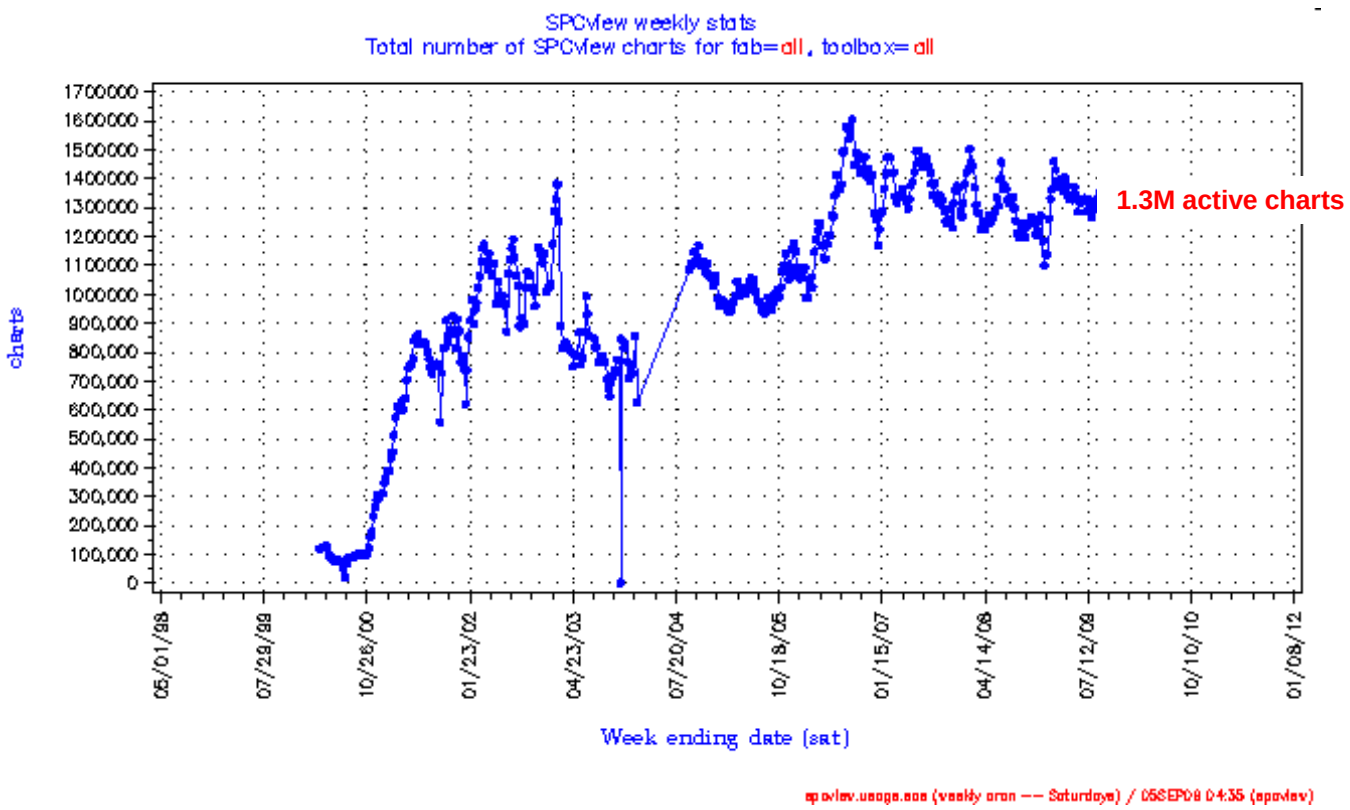
IBM

Problem Statement

- In the IBM Microelectronic factories (fabs), we use SAS in batch mode to update standard output with the newest manufacturing data.
- These daily SAS jobs analyze *1000's of charts*, printing reports and summaries of the alarms.
- The fab is *dynamic* -- part-numbers and customer product-mix change daily.
  - Therefore, the number of charts and population within changes.
- Also, we have several web-applications which create analysis *on-demand*, based on user inputs.
- We programmers can't spend time manually changing our SAS scripts to accommodate the dynamic nature nor the volume of the required analysis.
- Summary: "lots" of analysis is needed from "lots" of *dynamic* data

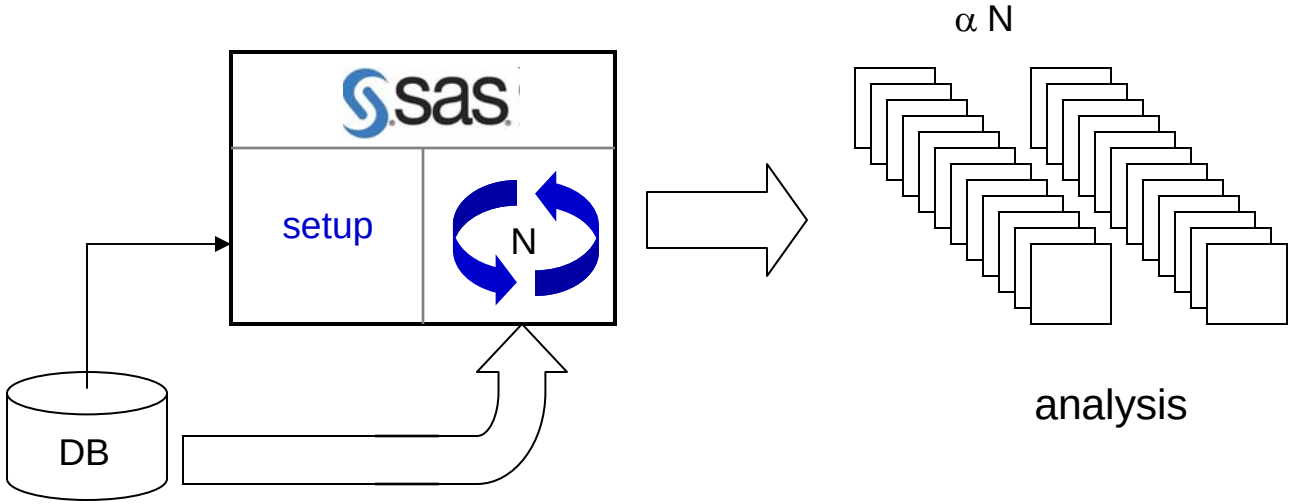
Problem Illustration

- Graph below shows num of charts from one of the several Quality Control apps in the fab
- 1.3M+ charts
- Many "input" sources for all these charts
  - Tool lists
  - Measurement lists
  - Experiments
  - Partnumber lists



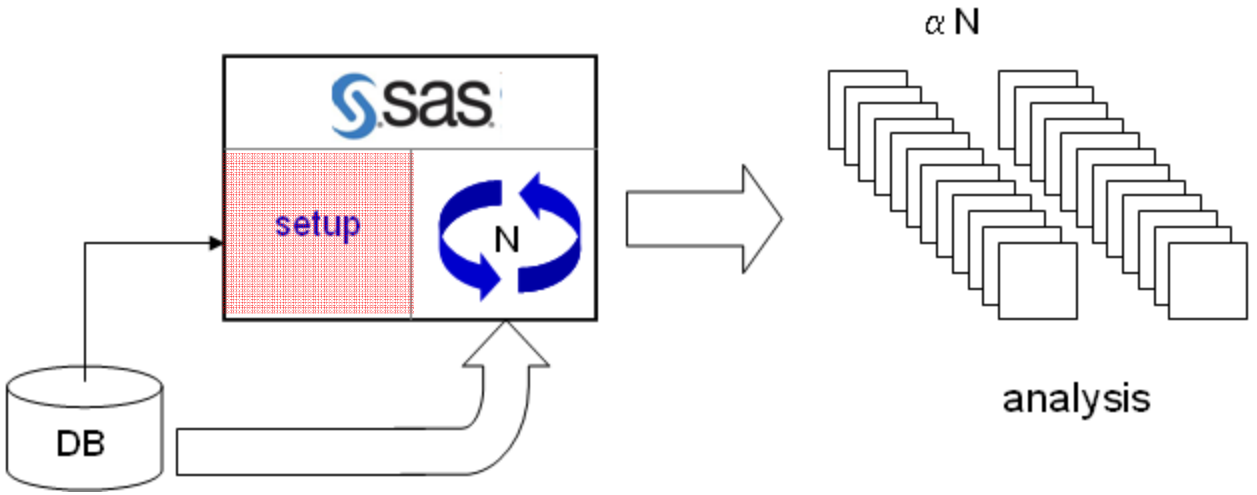
Solution described

- Need a way to loop through actions N times
  - Pull data
  - Make charts
  - Run analysis, summary reports, emails, etc.
- Need to get the loop parameters from a "setup pull"
  - N
  - Data population
  - Measurements to plot, report, analyze, SPC, etc.
- "setup pull" is dynamic
  - From data warehouse
  - From web page



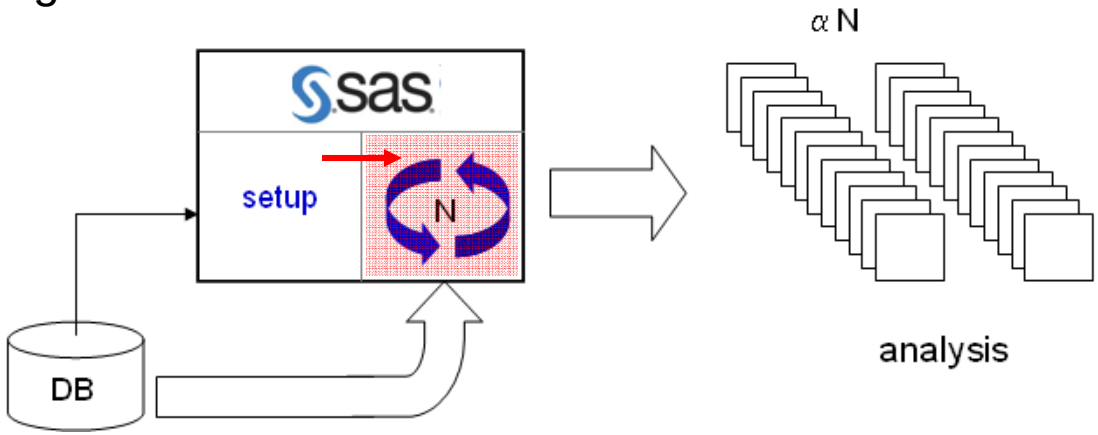
Solution implementation

- Setup pull
  - Creating macro variables from a setup pull is straightforward
  - We can get N, lists of measurements, other loop parameters "easily" from a dataset
    - `call symput`



Solution implementation

- Looping
  - We need a methodology to use the 'loop parameters' from above to define the analysis generation
  - I see 2 solutions, which I've named:
    1. "`&&var&i`" method
    2. "fetch" method
  - The first method has been used a long time at IBM, and I've seen in publications.
  - The second method is, I think, *novel*. It is a little more elegant (IMHO), using more advanced macro language commands. It also has some advantages.



"&&var&i" method

- Overview
  - Defines <u>all</u> macro variables instances *a priori* (up front)
    - *i.e*. all variables from <u>all</u> loops (N)
    - *e.g.* 13 variables in 100 loops = 1300 variable definitions
  - Uses a naming protocol to differentiate a variable for the intended loop
    - format: `&&var&i`
    - "`var`" is the <u>variable</u> differentiator
    - "`i`" is the <u>loop</u> differentiator
    - *e.g.:*

      `&&days&i` – the i^{th} value for "number of days"

      `&&name&i` – the i^{th} value for "parameter name"

      `&&PN&i` – the i^{th} value for "PartNumber"

"&&var&i" method

- *e.g.:*

  `&&days&i` – the ith value for "number of days"

  `&&name&i` – the ith value for "parameter name"

  `&&PN&i` – the ith value for "PartNumber"

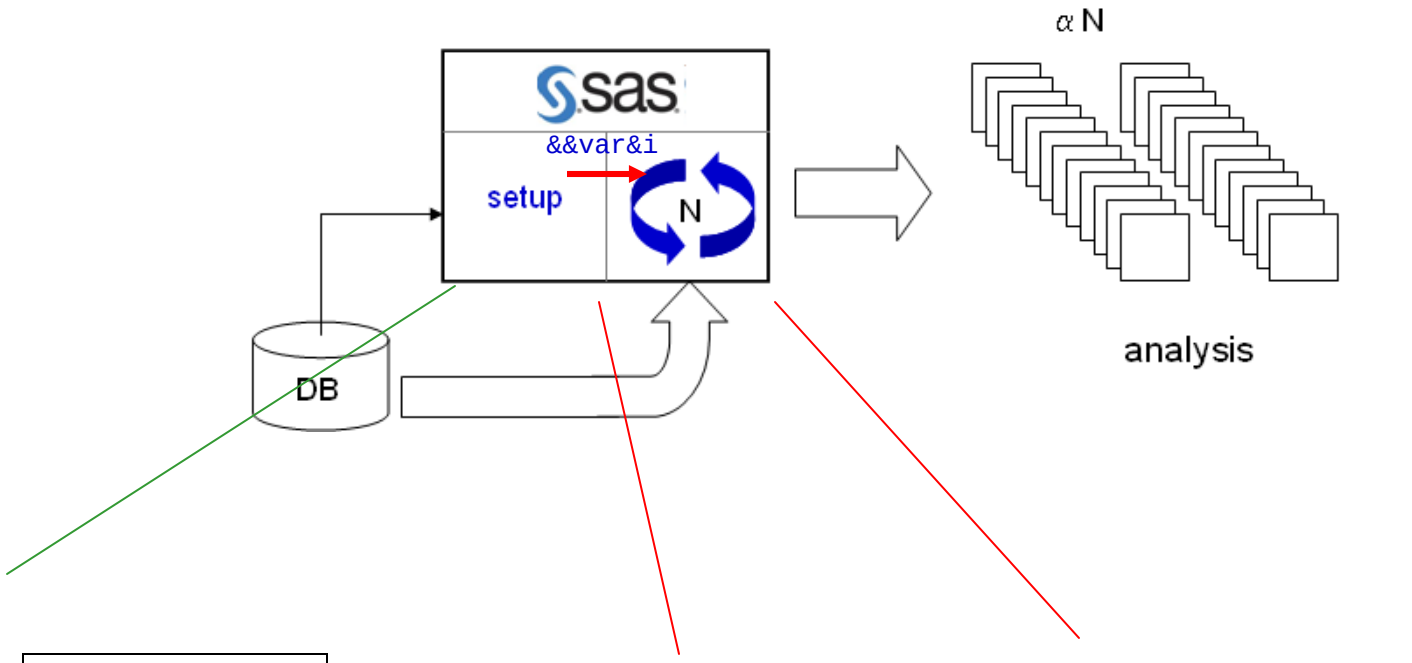
– Obviously, this looks like a *vector* (1-D array) variable
  - Treat like: `days[i], name[i], PN[i]`
  - You can google "sas macro array variable" and find more instruction and examples
    - www2.sas.com/proceedings/sugi29/070-29.pdf

"&&var&i" method

- How does this work?
  - We need to understand 3 rules for SAS macro symbol resolution
    1. Macro resolution is Left to Right
    2. && → &
    3. SAS make multiple passes until no more "&'s"
  - &&days&i
    - %let i = 3;
    - %let days3 = 30;
    - First pass: &&days&i → &days3
      && → &
      &i → 3
    - Second pass: &days3 → 30
  - Use:

```
proc sql;
select * from big_db where days = &&days&i;
quit;
```

"&&var&i" flow



Define all macros with analysis code templates

Setup pull from DB or input from user

Define all macro vars as 1-D arrays in the form `&&var&i`

Run analysis code template for each `i`

analysis

N

end

"&&var&i" method

- Setup pull
  - Define Control Dataset (CD)
    - Each obs is a macro loop
    - Each col will be a macro variable
  - Example:

macro var

ith macro loop

SAS 8: VIEWTABLE: Work.Runme

File Edit View Tools Data Solutions Help

	lotlabel	w_testprogramec	w_test_date	lotlabelq
1	4AXZC00000	0626KFPAWB	07JUL2009	'4AXZC00000'
2	4AXZC0U000	0626KFPAWB	02JUL2009	'4AXZC0U000'
3	4AXZC0W000	0626KFPAWB	01JUL2009	'4AXZC0W000'
4	4BNDC00000	0626KFPAWB	01JUL2009	'4BNDC00000'
5	4BNDC1A000	0805KFPAWA	18AUG2009	'4BNDC1A000'
6	4BNDC1D000	0805KFPAWA	15AUG2009	'4BNDC1D000'
7	4BNDC1DB30	0805KFPAWA	12AUG2009	'4BNDC1DB30'
8	4CH5C00000	0626KFPAWB	04JUL2009	'4CH5C00000'
9	4CH5C11000	0626KFPAWB	09JUL2009	'4CH5C11000'
10	4CH5C15000	0626KFPAWB	10JUL2009	'4CH5C15000'
11	4CH5C16000	0626KFPAWB	10JUL2009	'4CH5C16000'
12	BYDLT00000	0716KFPAWA	16JUL2009	'BYDLT00000'

In this example, each row has inputs for a macro which builds SQL for a datapull, then runs analysis and builds charts.

"&&var&i" method

- Setup
  - Use DATA step to define all variables in the proper naming protocol
    - Use a CALL SYMPUT for each macro "array" variable
    - Use dataset obs counter to define loop iteration
      - SAS DATA block internal var _n_ is macro var &i
  - Also define a macro variable for max number of loops (nobs)

"vector/array" macro var name:
days1, days2, days3, *etc.*

col in the Control Dataset

```
data _null_;
   set controlDS;
   call symput ('days'||left(_n_), days);   * &&days&i;
   call symput ('meas'||left(_n_), meas);
   call symput ('tool'||left(_n_), tool);
   call symput ('usl'||left(_n_), put(usl, best5.2);
   call symput ('lsl'||left(_n_), put(lsl, best5.2);
   call symput ('nobs', _n_);
   run;
```

"&&var&i" method

- Implementing the loop
  - Define macro loop from 1 to &nobs (rows in CD)
  - Use &&var&i throughout the looping macro
    - where "var" is key variable name
    - and "i" is the loop counter

```
%macro runloop;
   %do i %from 1 %to &nobs;
        %put ** loop &i **;
        %prepdata( &&tool&i, &&meas&i, &&days&i );
        %makecharts( &&tool&i, &&meas&i, &&days&i );

        proc print data=report;
             title1 "&&days&i days of &&meas&i from tool &&tool&i";
             where tool eq "&&tool&i";
             run;
        %end;
%mend runloop;
%runloop
```

Another example of "&&var&i" method

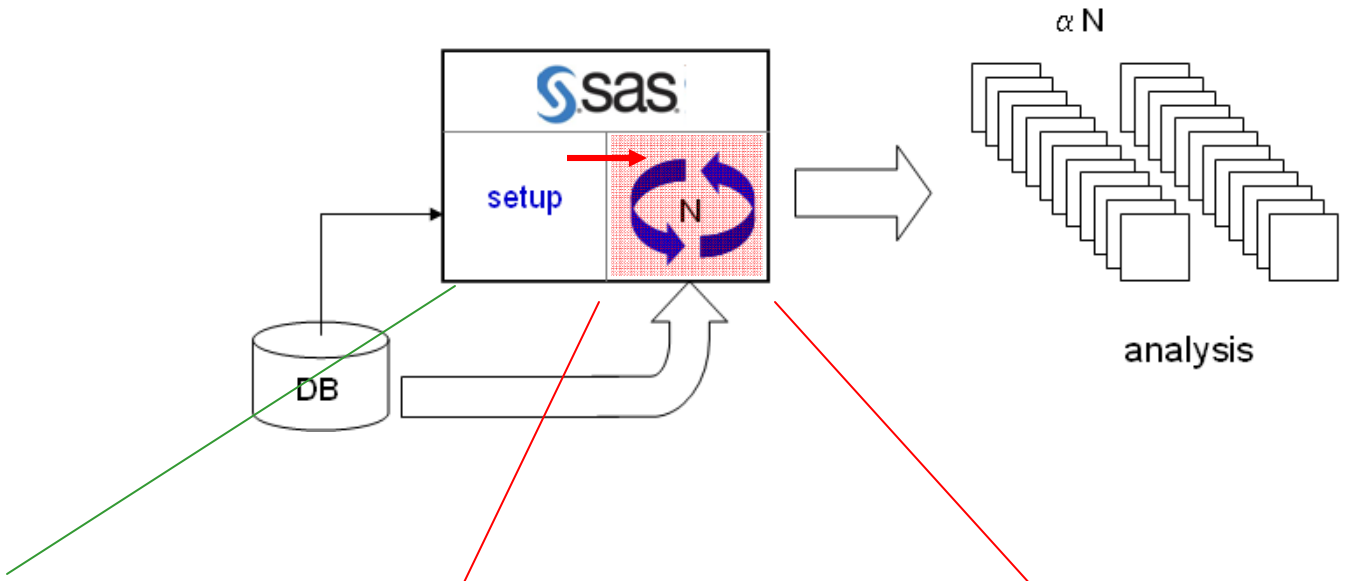
```
%macro run_loops;

    /* setup – Define all variables and total num of loops
             - This can go inside or outside of macro
    */
    data _null_;
        set run_charts;
        call symput ('dsn'||_n_, dsn);
        call symput ('parm'||_n_, parm);
        call symput ('toolset'||_n_, toolset);
        call symput ('nobs', _n_);  * num rows = # iterations;
        run;

    /* analysis loop */
    %do i = 1 %to &nobs;

        /* make charts from this dataset, parm, and toolset */
        %spc_charts(dsn=&&dsn&i, parm=&&parm&i, toolset=&&toolset&i)

        proc print data=alarms;
             title "Alarms from &&parm&i and toolset &&toolset&i";
             var alarm freq;
             run;
        %end;

%mend run_loops;
%run_loops
```

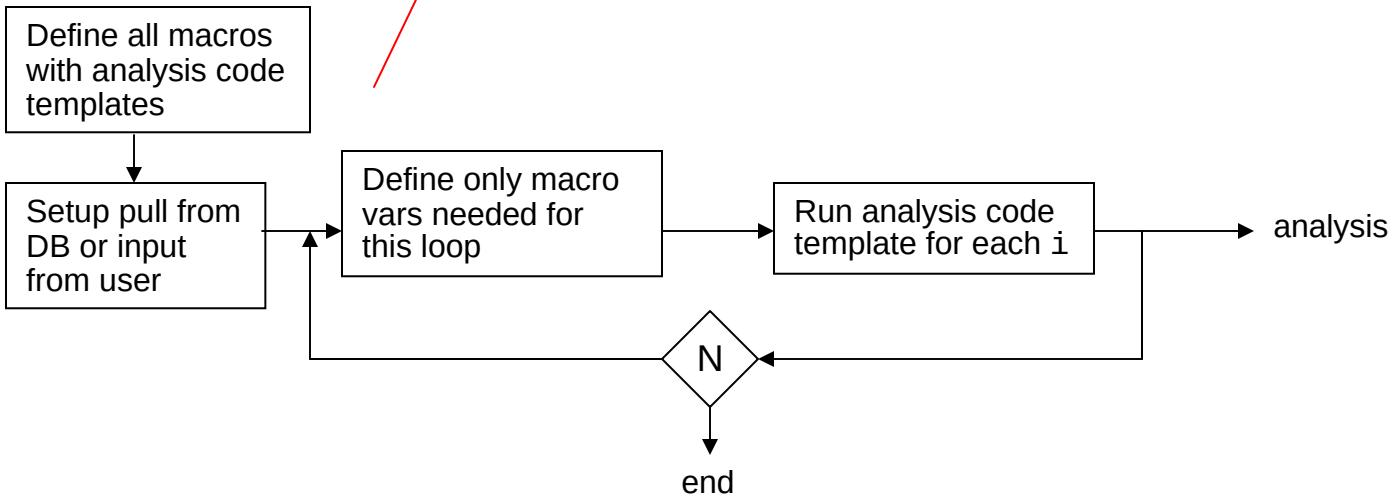
"fetch" method

- Overview
  - I think this is a novel approach. Hope it's useful to you.
  - Again, build a Control Dataset such that
    - all rows are loop iterations
    - all cols are macro variables

- Differences from `&&var&i` method:
  - Each 'set' of macro variables are defined only within their loop iteration
    - *i.e.* It's not the case that all variables from all loops are defined *a priori* (as in the previous method – remember 1300 vars!)
  - This method uses several `%sysfunc` calls to define the macro variables *during* the analysis looping
  - The macro 'array' variables are 'automatically' named from the columns of the control dataset. We don't have to write a `call symput` for each variable set.

"fetch" flow





cmp to slide 10

"fetch" method

- Macro "system" functions to understand:
  - `%sysfunc(open)` - open a dataset for further "macro" reading
  - `%sysfunc(fetch)` - read in a row from an opened dataset
  - `%sysfunc(attrn, <attr>)` - get some attribute of the dataset
  - `%sysfunc(varname, n)` - the name of the nth col in a dataset
  - `%sysfunc(getvarn)` or `%sysfunc(getvarc)`- get the data value
    - n for numeric, c for char
    - 2 different `%sysfunc()`'s based on var type

"fetch" method

- Setup
  - Define the Control Dataset
    - This time, *every* col becomes a macro variable, and the name of the macro variable is the name of the col
    - No need to pre-define all variables (as before). This time, each 'array/set' will be defined *inside* the main loop
    - Use DATA and SQL to create, remove and initialize all columns in the Control Dataset as they will exactly define the macro variables
      - Inner join SQL to define observations
      - Outer join SQLs to add columns
      - DATA block to initialize values, drop cols, and delete rows based on logic.

"fetch" method template

```
%macro runloop;
    %local dsid rc now rows cols;
    %let dsid = %sysfunc(open(new_lots));
    %let rows=%sysfunc(attrn(&dsid,nobs));   %* loops;
    %let cols=%sysfunc(attrn(&dsid,nvars));  %* vars;

    %* loops = rows;
    %do %while (%sysfunc(fetch(&dsid)) = 0);
        %* get vars from cols;
        %do c = 1 %to &cols;
            %local v t;
            %let v=%sysfunc(varname(&dsid,&c));
            %local &v;
            %let t = %sysfunc(vartype(&dsid, &c)); %* N or C;
            %let &v = %sysfunc(getvar&t(&dsid, &c));
            %end;

        %put ** loop # &now of &rows;

        /* lots of code here
           call analysis code template macros */

        %out:
        %end;  %* while fetch loop;
    %let rc = %sysfunc(close(&dsid));
%mend runloop;
%runloop
```

Loop over rows

Nested loop over cols

"fetch" method

- Key points to implementing:
  - Open the Control Dataset (returns an integer we'll use from here on)

```
%let cdid = %sysfunc(open(ControlDS));
```

  - Get size attributes:

```
%let rows=%sysfunc(attrn(&csid, nobs));   %* loops;
%let cols=%sysfunc(attrn(&csid, nvars));  %* vars;
```

  - Start the main loop by looping through the observations (rows) in Control Dataset
    - SAS is now 'pointing' to an observation in a dataset

```
%do %while (%sysfunc(fetch(&cdid)) = 0);
```

  - Loop through columns in Control Dataset

```
%do c = 1 %to &cols;
```

"fetch" method

- Key points to implementing:
  - Within the nested loop over each col of an obs:
    - Get column name (it will be the macro variable name)

```
%let v=%sysfunc(varname(&cdid, &c));
```

    - Get column type (N or C)

```
%let t = %sysfunc(vartype(&cdid, &c)); %* N or C;
```

    - Define a new macro symbol as the name of column set to the value of the column
      - Use getvarc or getvarn to get ith (row) value
      - Define macro var as name of col

```
%let &v = %sysfunc(getvar&t(&cdid, &c));
```

"fetch" method

- Key points to implementing
  - A closer look at that key command:

```
%let &v = %sysfunc(getvar&t(&cdid, &c));
```

col name

col type

  - Examples after 1st macro pass:

```
%let days = %sysfunc(getvarN(&cdid, &c));
%let tool = %sysfunc(getvarC(&cdid, &c));
```

  - Now there are N versions of each macro variable (*e.g.* `&days`)
    → one for each obs in the control dataset:

Making analysis from code template using "fetch" method

```
%macro runloop;
     %local dsid rc now rows cols;
     %let dsid = %sysfunc(open(new_lots));
     %let rows=%sysfunc(attrn(&dsid,nobs));   %* loops;
     %let cols=%sysfunc(attrn(&dsid,nvars));  %* vars;

     %* loops = rows;
     %do %while (%sysfunc(fetch(&dsid)) = 0);
          %* get vars from cols;
          %do c = 1 %to &cols;
               %local v t;
               %let v=%sysfunc(varname(&dsid,&c));
               %local &v;
               %let t = %sysfunc(vartype(&dsid, &c)); %* N or C;
               %let &v = %sysfunc(getvar&t(&dsid, &c));
               %end;

          %put ** loop # &now of &rows;

          /* make charts from this dataset, parm, and toolset */
          %spc_charts(dsn=&dsn, parm=&parm, toolset=&toolset)

          proc print data=alarms;
               title "Alarms from &parm and toolset &toolset";
               var alarm freq;
               run;

          %out:
          %end;  %* while fetch loop;
     %let rc = %sysfunc(close(&dsid));
%mend runloop;
%runloop
```

No longer &&parm&i

cmp to slide 14

Debugging code

- Only run 1 line

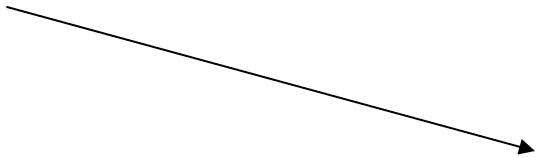
```
%*if &now ne 1 %then %goto out;
```

- Only run particular condition

```
%* if &tool ne X123 %then %goto out;
```

- Print variables and values:

```
%*put ** &v = &&&v;
```



1st pass: `%put days = &days;`
2nd pass: `%put days = 30;`

Control Dataset prep

- You can see how important the Control Dataset is to the "fetch" method
  - All cols become variables
  - Col names become the macro variable names
  - Every row (obs.) becomes an iteration in the control loop
- Therefore, it's very important to build the CD correctly.
- I've developed a 3 stage approach:
  1. Inner Join – defines rows
  2. Left Outer Join – adds other 'peripheral' macro vars
  3. DATA block – final editing, default setting, custom code

Control Dataset prep

- Inner Join
  - Creates the initial 'bare bones' Control Dataset
  - Merges together multiple datasets and only keeps matching criteria

```
proc sql noprint;
  create table runspc as
  select a.parm, a.oper,
         b.parmname, b.parmdesc, b.meastool, b.proctool,
         c.operdesc, c.toolset, c.opertype
  from charts a
  inner join validparms  b on b.parm = a.parm
  inner join validopers  c on c.oper = a.oper
  where c.opertype not in ('EXP','EWR','PCN')
  ;
  quit;
```

- Example:
  - Dataset `runspc` is the Control Dataset
  - The dataset "`charts`" has a list of desired charts from some source
  - We merge with "`validparms`" and "`validopers`" to get our starting list of "to-do's"

Control Dataset prep

- Left Outer Join
  - Adds other variables to the initial list created
  - The "Left Outer Join" ensures that the original list stays "intact"
  - I often use this to add targets and spec limits, if exist

These new cols become macro vars

```
proc sql noprint;
   create table runspc as
   select a.*,
          b.target, b.lsl, b.usl, b.spc, b.lcl, b.ucl, b.spectype,
          c.y1 as ymax, c.y2 as ymin, c.y3 as yby, c.ylog,
          d.customflyer
   from runspc a
   left outer join targets    b on a.parm = b.parm
   left outer join yaxis      c on a.parm = c.parm
   left outer join (select parm, value as customflyer
                    from parmfilter
                    where action eq "filt" ) as d on a.parm = d.parm
   ;
   quit;
```

Control Dataset prep

- DATA block
  - Any final formatting or editting of Control Dataset
  - Format targets as best5.
  - Create default flags
  - Check for valid input
  - *etc.*

```
data runspc;
   set runspc;

   * formatting;
   target = put(target, best5.);
   usl = put(usl, best5.);
   lsl = put(lsl, best5.);

   * conditions;
   * order= (y1 to y2 by y3);
   if y1 < y2 and y3 ne . then y_order_flag = 1 ;
   else y_order_flag = 0;
   if lsl > usl then delete;

   run;
```

This Control Datatset is ready for each col to be a macro var and each obs to be a loop iteration which will call analysis code templates.

Conclusion

- We desire to use SAS to produce *large amounts* of standard output based on changing inputs.
- Defining macro loops from a Control Dataset (CD) is a good way to create the analysis.
  - In a CD, the rows are the macro loops, and the cols are the macro variables
- There are two ways to implement the looping
  - The first method defines all variables up-front, then uses "&&var&i" format during the loops for the i^{th} var.
  - The second method uses %sysfunc calls to define only the macro variables for the current loop iteration.
- Both methods are good methods and you can chose your favorite to implement.
- The paper for this topic is at:
  www.nesug.org/Proceedings/nesug09/bb/bb08.pdf
- This presentation will be updated and available at: http://db.tt/42RTnmo