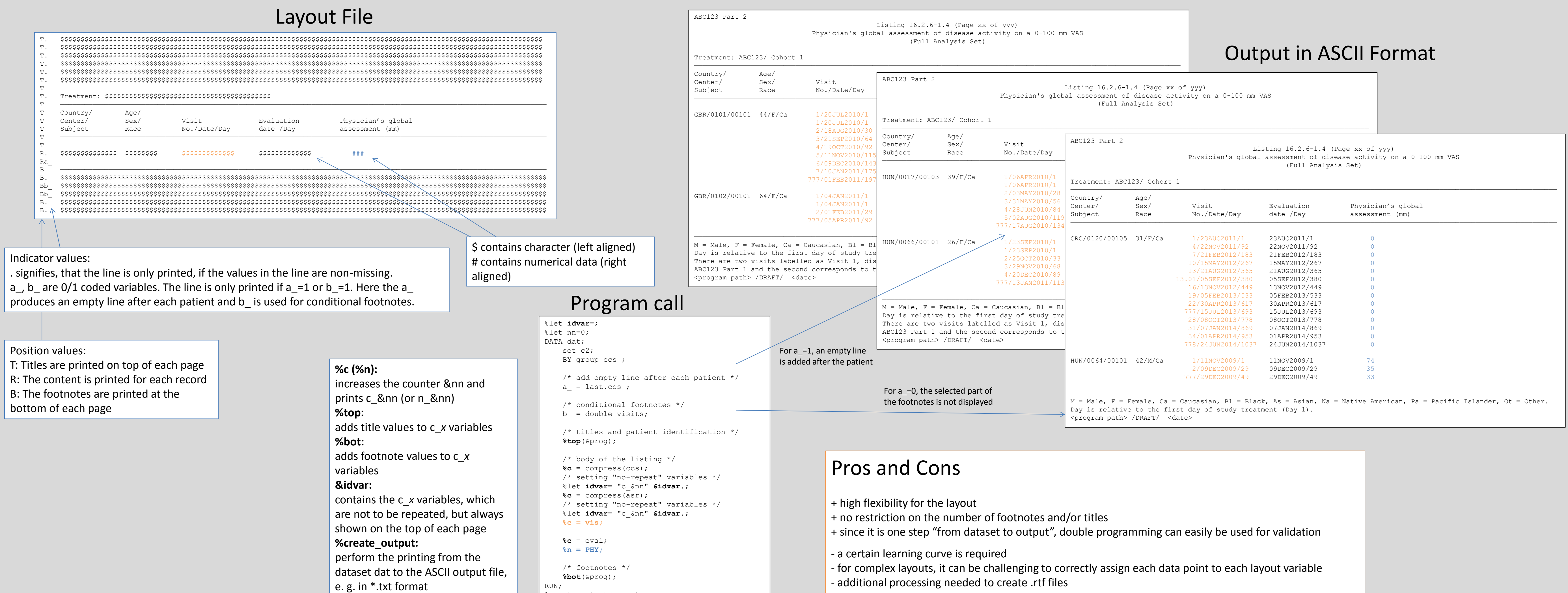


Christoph Baumer, Syneos Health
christoph.baumer@syneoshealth.com

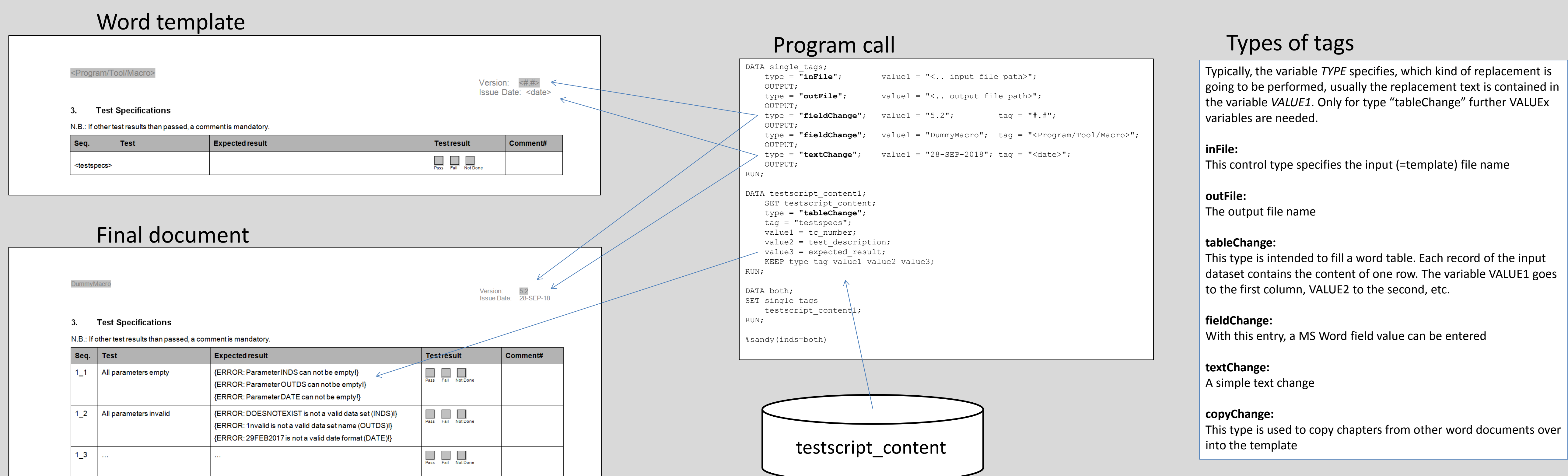
Statistical Outputs

An ASCII Layout File is used for creation of statistical outputs. Each entity of \$ or # sequences is replaced with content of the output dataset. The clinical data has to be prepared, so that each variable to be printed is assigned to a certain variable, e. g. variable c_12 will go to the 12th variable representation of the layout file. In addition, there are position values and indicator values provided with the layout.



MS Word Documents

The creation of template based MS Word documents works mainly with “tags”, keywords enclosed in angle brackets (<>). A typical use of Word template based file creation is the creation of validation documentation. The SAS log(s) can be read into a SAS dataset which can be transferred to a MS Word document. In our implementation, we are using SAS together with Java / Aspose.Words for Java.



SAS Programs

Using templates for SAS programs becomes useful, when you want to run the same program code under different conditions, e. g. for different datasets or different variables. The approach can be as simple as storing the template program in a variable or, in case of longer programs, in an actual ASCII file. The to-be-substituted part is displayed as \$1\$ in the examples below for easy numbering, but instead, one could also use more meaningful names such as e. g. \$DATASET\$. This technique gives you a lot of control over the actual program, the code to be executed stays much more readable than a series of CALL EXECUTE calls combined with concatenated strings.

SQL query for number of observations by subject

```
DATA null ;
    LENGTH cmd cmd_exec $1000. ds $40.;
    CALL EXECUTE("PROC SQM %1;
cmd = 'select $1$' usubjid, count(*) from results,$1$ group by usubjid';
/* macro variable %DS is a list of the datasets of interest */
DO ds = %ds.;
    cmd_exec = TRANWRD(cmd,"$1$",ds);
    CALL EXECUTE(cmd_exec);
END;
RUN;
QUIT;
```

In this example, the whole “template” is contained in the variable *cmd* and the dynamic part ‘\$1\$’ is replaced with a dataset name and then executed using *CALL EXECUTE*

Compare all datasets which are present in two libraries

```

MACRO EXEC;
  PROC COMP BASE=results..$1$ COMP=refdata.$1$ LISTALL;
  RUN;
  MEND;
PROC SQL NOPRINT;
CREATE TABLE comp_tables AS
  SELECT * FROM dictionary.tables a
    INNER JOIN dictionary.tables b
    ON a.memname EQ b.memname AND
    a.blibname EQ 'RESULTS' AND b.blibname EQ 'REFDATA';
QUIT;
DATA _null_;
  LENGTH cmd_exec $1000.;
  SET comp_tables;
  cmd = 'exec';
  cmd_exec = TRANWRD(cmd,'$1$',STRIP(memname));
  rc = DOSUB(cmd_exec);
RUN;

```

Here, the program code is stored in a macro, which is loaded into the dataset variable *cmd*, followed by the appropriate replacement. Note that double quotes must be avoided within the macro. Execution of the program code is done via DOSUBL. This approach is expandable in a very easy manner, you can just add additional (dummy) code to the %exec macro definition, e. g. prior sorting of the comparison datasets.

Same comparison using actual files

```
DATA tmp1;
  INFILE '<template_file_path>'
    LRECL=1048 TRUNCOVER;
  INTCV 8 1-500;
  txt = _INFILE_;
RUN;

PROC SQL NOPRINT;
  CREATE TABLE both AS
    SELECT TRUNC(txt,50) '51'; b.mname; AS cmd
    FROM tmp1 a FULL JOIN cmd b ON a EQ
    ORDER BY b.mname;
QUIT;

filename x temp;
DATA null_;
  SET both;
  FILE x;
  PUT cmd;
RUN;
run;
```

In this approach, the PROC COMPARE code is stored in an actual file, which is read into the dataset *tmp1*. Using the FULL JOIN with “1 EQ 1”, the dataset names are merged to each record and the *cmd* variable is created already in the SQL procedure.

The content of the resulting dataset is written to a temporary file and executed via file include (%INC). For debugging, it is recommended to use a permanent file location for the file *x*.