Paper 2643-2018

Implementing external file processing with no record delimiter via a metadata-driven approach

Princewill Benga, F&P Consulting, Saint-Maur, France

ABSTRACT

Most of the time, we process external files that have fixed width columns, or that contain data separated by a common delimiter, such as comma, tab, pipe, etc. However, there are times where we have to deal with unusual and/or inconsistent file structure and format. This paper will describe an innovative approach to read external files with no available delimiter with which to parse the data. Furthermore, the structure and type of the data can change from one record to another. Each record may have a unique format, sometimes requiring looping constructs to properly set the required data. This case is derived from a real world example in the automobile insurance sector. The solution uses external metadata to define required characteristics of the raw file, along with hash object lookup and advanced INFILE statement processing to achieve the required business outcome.

This paper can be of interest to anyone who needs to read external files whose structure is uncommon.

KEYWORDS

Hash objects, Lookup techniques, external files

INTRODUCTION

SAS provides a rich ecosystem with a series of techniques for reading external files. These files exist in a wide variety of formats and are mostly characterized with a basic structure of records separated by a delimiter. Common delimiters include comma, tab, colon, semicolon, pipe etc. Other delimiters could be specified in SAS using the DLM option in the INFILE statement. These files could easily be read into SAS using the INFILE and INPUT statement with a variety of methods such as pointer controls (absolute, relative, row, and column) and line-hold specifier (single @ and double @@ trailing).

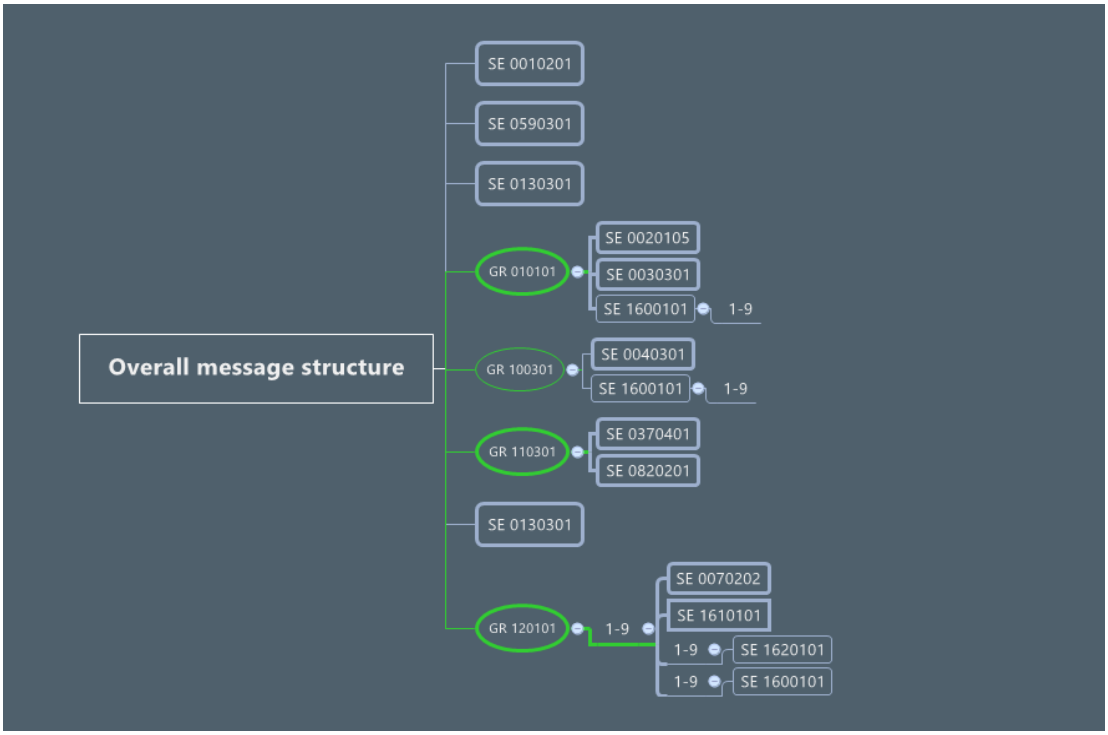
However, there are situations where these out-of the box tools are insufficient or might require complex programming. The power of SAS also comes from the flexibility that it offers programmers to derive specific solutions. The intent of this paper is to demonstrate this power and flexibility of SAS as applied to a real-world business problem.

BACKGROUND

The case that follows is one that I encountered as a SAS programmer on a client side. This client receives an external file from a third party. The file is a text file (.txt) and contains information about accident reports. At this stage, you will say nothing surprising; it shouldn't be an issue to read this in SAS. I would usually agree but I'll say don't judge a book from its cover. So what is in this file? Of course, information about accident reports. But how is the data arranged in the file?

Let us call each record in the file a *message*. This *message* is made up of segments (SE) which are composed with data (elementary, composite, or coded data). Also, these segments can be functionally grouped (GR) as shown in Figure 1 below.

Figure 1: Message structure (partial view)



Segments in the *message* begin with a segment ID (for example 0010201, 0590301...) followed by the data part (value) of the segment. All segment IDs are made up of 7 digits. However, the length of the data part varies across segments. Below in figure 2, is an illustration of Segment IDs (in red ink) from a partial extraction of a *message*. The data portion of segment 0010201 is 40 characters long while that of segment 0592059 is 71 (spaces included). Similarly the data portion of segment 0130301 is 5 characters long and so on for all 53 segments that might make up a *message*.

Figure 2: Content of message (Partial view)

0010201E24053097414A9200001126615040100050605920590301201506121501719
0820150612164639 04958700 0130301 11NN7 0020105 295 B1540248119
150663646835106102 0030301 431721 KATOT MATHY
16001010000320496201560

Furthermore, segments are of different types: mandatory, facultative, repetitive, and non-repetitive. Figure 2 below presents the segment matrix.

Figure 3: Segment Matrix

	Repetitive	**Non-repetitive**
Mandatory	Yes	Yes
Facultative	Yes	Yes

Repetitive or non-repetitive: A repetitive segment is one that will occur several times within the same *message* as compared to a non-repetitive segment that will occur just once. The real number of repetition for a repetitive segment is given by the 5 digits next to the segment ID. For example, from figure 2, the number of repetition for segment 1600101 is 3 (00003).

Mandatory or facultative: A mandatory segment is one that MUST be present in all *messages*. Inversely, a facultative segment MIGHT not be present for certain *messages*.

A metadata file has been used to depict the characteristics of the different segments that might be present in a *message* (Appendix 1). These variables have been constructed to help automate processing of the incoming files. Variables of interest for this business case are:

- **Source**: Describes the type of *message*, either Retour d'Expertise (Expert Return) or Chiffrage (Estimate) represented by RE and CH respectively. Segment IDs for each source will always appear in the order that they are listed in the metadata file.
- **Type**: Could be S for segment or G for group. Segments could occur solely or in groups. Segments belonging to the same group will have the same item string in the path variable. Groups are shown for descriptive purpose; only segments will be considered in the next sections.
- **Occurrence**: Indicates the maximum frequency a segment ID could appear in any given *message*. This is the theoretical maximum for repetitive segments. As already mentioned above, the real frequency is given by the 5 digits next to the segment ID. Occurrence is greater than 1 for repetitive segments and equals 1 for non-repetitive segments. The maximum number of repetitions for a particular segment is the product between the group and the theoretical number of occurrences of the segment ID.
- **Start**: Is used to identify the starting point to extract the data part for a given segment ID. Start is either 8 for non-repetitive segments or 13 for repetitive segments.
- **Seg_length** (Segment length): This is the length of the data portion. It determines the number of characters including blanks to extract after start.
- **Cnt** (Count): Cnt is used as a flag to differentiate between identical segment IDs appearing in different groups.

From the above description, the records in the external file are not arranged as could be found in typical flat files. Lookup techniques such as IF-THEN/ELSE, MERGE, SELECT, INDEX, or ARRAYS would be inefficient or cumbersome to use in this case. Instead, a hash object is used to load the metadata file into memory.

OVERVIEW OF A HASH OBJECT

Since introduced by Paul Dorfman in the early 2000's, much has been written on hash objects (Schacherer, 2015). The SAS hash object is one of the five predefined component objects for use in a DATA Step. It is a powerful, fast, and flexible in-memory table lookup technique. It enables quick, efficient storage, search and retrieval of data based on lookup keys. The hash record is composed of two parts:

- a key part made up of one or more numeric and character columns
- a data part that consists of zero or more character and numeric columns.

SAS provides a set of predefined statements, attributes, methods, and operators to interact with the hash object. To use a hash object, it must first be declared in a DATA step. Once defined, methods are available to perform operations like retrieving, adding, or removing values from the hash object. For more information about using hash objects see the SAS documentation link in the reference section.

SAMPLE CODE

The programming logic consists of loading the metadata file into a hash object in a DATA step. *Segment* and *cnt* are defined as the key while occurrence, start, seg_length are defined as data part. A %dirlist macro is used to retrieve the pathnames of all *message* files in the *message* files directory into a SAS dataset. The dirlist dataset plus a dynamic INFILE statement is used to process all files within a single DATA step. DATA step functions are used to parse the text file, identifying segment IDs, and performing a lookup on the hash table. Further processing to determine the data portion is done using the SUBSTR function, and values of *occurrence, start, and seg_length are* retrieved from the hash table. Once the data portion of a particular segment is extracted, it is truncated from *message* together with the segment ID. This ensures that segment ID is always at the beginning of the *message.* This process is repeated until *message* is empty, that is its length is 0.

Project environment:

```
* root directory;
%let root =C:\SASGF2018;

* libnames;
libname out "&root\out";

*landing zone for incoming files;
filename  in  "&root\in";
```

Import metadata

```
* metadata file is conditionally imported only if doesn't exist;
%macro meta;
  %if %sysfunc(exist(out.meta_tab)) = 0 %then %do;
   proc import
      datafile="&root\metadata\meta_tab.xlsx"
      dbms=xlsx
      out=out.meta_tab (where=(upcase(type)='S'))
      replace;
   run;
  %end;
%mend;

%meta
```

Directory list

```
* Getting the list of files in the landing zone (IN) directory;
* This is a utility macro provided by Scott Bass (my mentor).;
* This macro will create a table listing all files and their characteristics for a given directory.;


%dirlist(
   dir=&root\in
/*,data=filist*/
)
```

Hash and dynamic infile

```
data out.pre_process;
* The variable "msg"(message) represents a record from the input file;
* Records are of different length so the maximum length in SAS is considered for "msg";
* The variable "source" will contain the type of "msg" which can either be CH (Chiffrage) or RE (Retour
* d'expertise);
* The variable "value" will hold the value of each segment extracted from "msg";
* The variable datestamp is the date the message is received;
* This date is included in the filename, position 31-8;

length msg $32767 value $300 source $2;
format datestamp date9.;

*Declaring and Instantiating the Hash Object;

if _n_=1 then do;
   if 0 then set out.meta_tab (
      keep=source segment occurrence start seg_length cnt
   );
   declare hash metadata (
      dataset:"out.meta_tab", hashexp: 16, multidata:"Y"
   );
   metadata.defineKey("segment","cnt");
   metadata.defineData("occurrence","start","seg_length","cnt");
   metadata.defineDone();
end;

* Explicitly set the hash object data variables to missing due to the implied retain from the "if 0 then
* set..." statement;

call missing(of occurrence start seg_length cnt);

* The dirlist dataset contains the files that are processed;
* A dynamic infile statement is use to process the files in the dirlist dataset;
* Each time the filevar= variable changes, a new file will be read;
* The program will loop over the contents of the file so that the filename (read from dirlist) is "frozen";
```

```
* The end= variable will be set to 1 when the last record is read;
* from the current file;

set dirlist (keep = fullname filename);
datestamp = input(substr(filename,31,8),yymmdd8.);
infile=fullname;
infile in lrecl=32767 filevar=infile end=eof;

* A file can contain mix type of records;
* As such, the type of message will be determined inside a DO UNTIL LOOP;

do until (eof);
* An id variable is added to identify each "msg";
      retain id 0;
      id +1;
      input;
      select(substr(_infile_,25,2));
    when ('15') source='RE';
    when ('14') source='CH';
      otherwise   source=' ';
end;

* The continue statement is use to return to the top of the loop if source
* is missing;

if missing(source) then continue;

* The first segment (0010201) is not present in the records;
* The segment number will be added at the beginning of each record;

msg=cat('0010201',_infile_);

* Cnt variable is defined to perform lookup from hash table;

cnt = 1;
do until (lengthn(msg) eq 0);

* The first 7 characters identify a segment;
    call missing(of _freq tot_long);
    segment = substr(msg,1,7);
```

```
* Retrieving Data from the hash object
     rc = metadata.find();

* Check if lookup is successful;
     if (rc = 0) then do;

* Check if there are duplicate values for the segment (key);
       metadata.has_next(result:r);
       do while(r ne 0);
         rc1 = metadata.find_next();
         metadata.has_next(result:r);
       end;

 * Getting the values of non-repetitive segments;
       if occurrence = 1 then do;
         _freq =1;
         value=substr(msg,start,seg_length);
         msg=substr(msg,sum(start,seg_length));
       end;

 * Getting the values of repetitive segments;
       else do;
         _freq = input(substr(msg,8,5),5.);
         tot_long=_freq*seg_length;
         value=substr(msg,start,tot_long);
         msg=substr(msg,sum(start,tot_long));
       end;

 * Setting count = 2 to retrieve next multidata segment for segment;
 * number 0120102;

       if segment in ("0360201","2190101") then cnt = 2;
      end;
     output;
    end;
end;

 * Occurrence, start, cnt, seg_length are dropped because;
 * they are already in;
 * the metadata file;
```

```
* fullnames is dropped as path to file is always the same;
* msg, rc, r, rc1 are working variables so are dropped;

drop fullname _freq occurrence msg start tot_long cnt seg_length rc r rc1;

run;


* Transform "PRE_PROCESS" table to apply business rules;
* The object of of this transformation is to get one observation per "id";

proc sort data=out.pre_process out=temp1;
  by id segment filename datestamp;
run;

* In the below data step, value1 will hold the concatenated values for
* repetitive segments;
* Its length has been set to 500 but there's a risk of truncation.;
* The Maximum theoretical length is from segment 0200301 and is given by:
* Number of Group repetitions*Number of Sub-Group repetitions*Segment
* length (9*999*77 = 890109);
* However, from SAS perspective, we cannot attribut such a length to a
* variable;
* Segment 2190101 has the shortest length, seg_length = 4;

Data temp2 (drop=value);
  length value1 $500;
  set temp1;
  by id segment filename datestamp;
  retain value1 ;
  if first.segment then value1=cat(value);
  else value1=catx(" ",value1,value);
  if last.segment then output;
run;
* The temp2 data set is transposed such that segments (prefix with seg_) becomes columns and
* values of value1 as rows.
* This transformation is required to apply business logic
```

```
Proc transpose data=temp2 out=trans(drop=_name_ ) prefix= seg_;
   variable value1;
   id segment;
   by id source filename datestamp;
run;

* Put value of segment 0070202 into a macro variable.
* This will be use to determine the value of NM_ADD_REP;

Data _null_;
  set temp1 (where=(segment ='0070202' and substr(value,1,2)='02'));
  call symputx(catx('_','seg',put(id,2.)),substr(value,3));
run;

* Identifying segments not present in dataset PRE_PROCESS compared to META_TAB;
* Doing this will help avoid uninitialized variable message on the log;

proc sql noprint;
  select distinct catx('_','seg',segment) into:seg_absent separated by " "
  from out.meta_tab
  where segment not in (select distinct segment from out.pre_process);
quit;

%put "The following segments: &seg_absent are absent from incoming messages";

* Apply Business rules and final table;

Data post_process_0(Keep= SOURCE MISSION REFERENCE
  ACC_REGLEM_DIR_DEF
  CD_CIRC_EXPT

* More variables close to 80 in all;
  );

* attrib ACC_REGLEM_DIR_DEF  format=$1. Label="ACCORD DE REGLEM. DIRECT DEFINITIF ";

* More attribute statement;
```

```
  length &seg_absent $150 x $7;
  set trans;
  call missing(of &seg_absent);
  ACC_REGLEM_DIR_DEF=substr(seg_0110201,19,1);
 if upcase(source)="RE" then do;
        VAL_HT_REMPLACEMENT=input(substr(seg_0160201,24,9),9.2);

* More processing logic;

 end;
 else if upcase(source)="CH" then do;
    CD_EXP =put(substr(seg_2120101,25,14),14.);

* More processing logic;

 end;

  i_0170501=countw(seg_0170501);
 if not missing(i_0170501) then do;
        do j=1 to i_0170501;
    NB_HR_MAIN_OVRE_QLF=sum(NB_HR_MAIN_OVRE_QLF,input(substr(scan
    (seg_0170501,j),2,6),6.2));
      if substr(scan(seg_0170501,j),8,1)='1' then CD_QLF_MES1='1';
      else if substr(scan(seg_0170501,j),8,1)='2' then CD_QLF_MES2='2';
      else if substr(scan(seg_0170501,j),8,1)='3' then CD_QLF_MES3='3';

* More processing logic;

    end;
 end;

     do j=1 to i_0170501;
        if not missing(CD_QLF_MES1) then do;
           if substr(scan(seg_0170501,j),8,1)='1' then  do;
        NB_HR_MAIN_OVRE_MES1 = sum(NB_HR_MAIN_OVRE_MES1,input(substr
        (scan(seg_0170501,j),2,6),6.2));
              mt_mes1= sum(mt_mes1,input(substr(scan
```

```
    (seg_0170501,j),15,9),9.2));
                TX_HRE_MES1= mt_mes1/NB_HR_MAIN_OVRE_MES1;
             end;
       end;

* More processing logic;

* The variable x is defined and SYMGET function is used to get the value of the macro variable defined;
* earlier for segment 0070202;

x= catx('_','seg',put(id,2.));

i_1610101=countw(seg_1610101);
   do j=1 to i_1610101;
      if substr(scan(seg_1610101,j),1,2)='05' then
         NM_ADD_REP=symget(x);
      end;

* More processing logic;

run;

* Add constraints;

proc datasets lib=work nolist;
  modify post_process_0;
  ic create Not null(CD_NA_MISSION);

* More processing logic;
quit;

* Append and backup tables;

%macro append;

* POST_PROCESS table is the final table for Business users so filename is dropped;

%if %sysfunc(exist(out.post_process)) %then %do;
```

```
   data out.post_process;
      set out.post_process post_process_0;
    run;
%end;
%else %do;
  data out.post_process;
    set post_process_0;
  run;
%end;

* The variable FILENAME is backed up for recovery purpose to match incoming messages and their contents;

%if %sysfunc(exist(out.backup_pre_process)) %then %do;
   data out.backup_pre_process;
     set out.backup_pre_process trans;
   run;
%end;
%else %do;
   data out.backup_pre_process;
     set trans;
   run;
%end;

%mend append;

%append;
```

For full code please contact the author.

CONCLUSION

There are a number of traditional techniques that can be used when you need to read an external file. However, these techniques might not be appropriate for all situations. This paper presents a real world case where traditional techniques for reading external files were insufficient. The expectation of this presentation is to encourage innovative approaches of using SAS to deal with real business cases.

REFERENCES

Schacherer, Chris (2015). "Introduction to SAS® Hash Objects." *Proceedings of the SAS Global Forum 2015.* Dallas, TX: SAS Institute, Inc.

Peter Eberhardt, Audrey Yeo, Athene (2015) "The SAS DATA Step: Can you read this into SAS® for me? "Using INFILE and INPUT to Load Data Into SAS®." *Proceedings of SAS Global Forum Paper 3210-2015* Dallas, TX: SAS Institute, Inc.

ACKNOWLEDGMENTS

As a first time SAS Global Forum presenter, I want to thank my mentor Scott Bass for his valuable comments and his support through the entire process. I would also like to thank Peter Eberhardt for his suggestions. I also acknowledge Nancy Moser and the SAS Global Forum support team for their support during the registration process.

RECOMMENDED READING

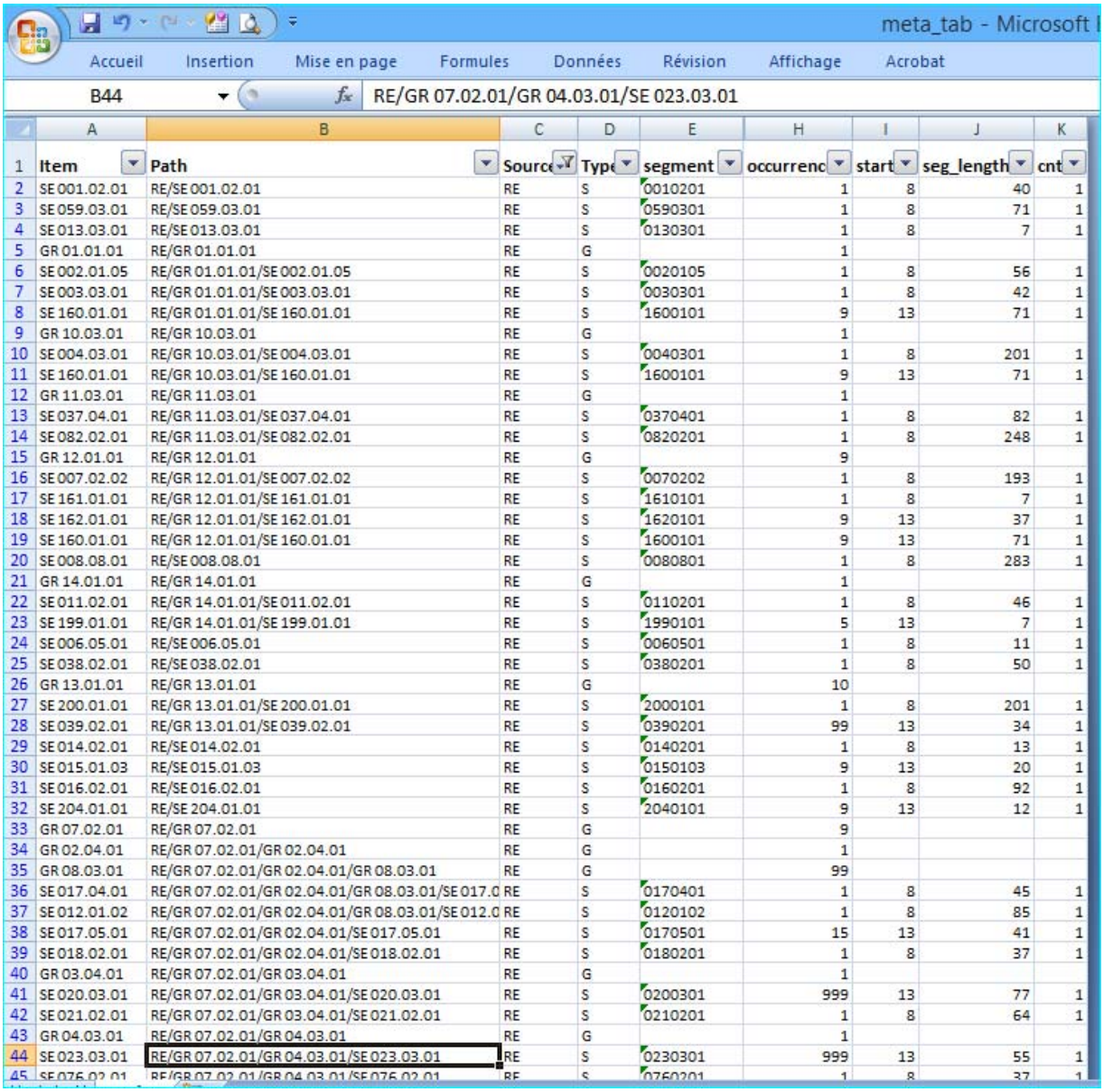
- *SAS® 9.4 Language Reference: Concepts, Sixth Edition: Using the Hash Object*
  http://documentation.sas.com/?docsetId=lrcon&docsetTarget=n1b4cbtmb049xtn1vh9x4waiioz4.htm&docsetVersion=9.4&locale=fr

CONTACT INFORMATION

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

Princewill Benga
F&P Consulting
+33 (0) 6 82 33 25 06
princewill.benga@fnpconsulting.fr
www.fnpconsulting.fr

Appendix 1: Partial view of Metadata file



meta_tab - Microsoft

Accueil | Insertion | Mise en page | Formules | Données | Révision | Affichage | Acrobat

B44 | RE/GR 07.02.01/GR 04.03.01/SE 023.03.01

	A	B	C	D	E	H	I	J	K
1	Item	Path	Source	Type	segment	occurrenc	start	seg_length	cnt
2	SE 001.02.01	RE/SE 001.02.01	RE	S	0010201	1	8	40	1
3	SE 059.03.01	RE/SE 059.03.01	RE	S	0590301	1	8	71	1
4	SE 013.03.01	RE/SE 013.03.01	RE	S	0130301	1	8	7	1
5	GR 01.01.01	RE/GR 01.01.01	RE	G		1			
6	SE 002.01.05	RE/GR 01.01.01/SE 002.01.05	RE	S	0020105	1	8	56	1
7	SE 003.03.01	RE/GR 01.01.01/SE 003.03.01	RE	S	0030301	1	8	42	1
8	SE 160.01.01	RE/GR 01.01.01/SE 160.01.01	RE	S	1600101	9	13	71	1
9	GR 10.03.01	RE/GR 10.03.01	RE	G		1			
10	SE 004.03.01	RE/GR 10.03.01/SE 004.03.01	RE	S	0040301	1	8	201	1
11	SE 160.01.01	RE/GR 10.03.01/SE 160.01.01	RE	S	1600101	9	13	71	1
12	GR 11.03.01	RE/GR 11.03.01	RE	G		1			
13	SE 037.04.01	RE/GR 11.03.01/SE 037.04.01	RE	S	0370401	1	8	82	1
14	SE 082.02.01	RE/GR 11.03.01/SE 082.02.01	RE	S	0820201	1	8	248	1
15	GR 12.01.01	RE/GR 12.01.01	RE	G		9			
16	SE 007.02.02	RE/GR 12.01.01/SE 007.02.02	RE	S	0070202	1	8	193	1
17	SE 161.01.01	RE/GR 12.01.01/SE 161.01.01	RE	S	1610101	1	8	7	1
18	SE 162.01.01	RE/GR 12.01.01/SE 162.01.01	RE	S	1620101	9	13	37	1
19	SE 160.01.01	RE/GR 12.01.01/SE 160.01.01	RE	S	1600101	9	13	71	1
20	SE 008.08.01	RE/SE 008.08.01	RE	S	0080801	1	8	283	1
21	GR 14.01.01	RE/GR 14.01.01	RE	G		1			
22	SE 011.02.01	RE/GR 14.01.01/SE 011.02.01	RE	S	0110201	1	8	46	1
23	SE 199.01.01	RE/GR 14.01.01/SE 199.01.01	RE	S	1990101	5	13	7	1
24	SE 006.05.01	RE/SE 006.05.01	RE	S	0060501	1	8	11	1
25	SE 038.02.01	RE/SE 038.02.01	RE	S	0380201	1	8	50	1
26	GR 13.01.01	RE/GR 13.01.01	RE	G		10			
27	SE 200.01.01	RE/GR 13.01.01/SE 200.01.01	RE	S	2000101	1	8	201	1
28	SE 039.02.01	RE/GR 13.01.01/SE 039.02.01	RE	S	0390201	99	13	34	1
29	SE 014.02.01	RE/SE 014.02.01	RE	S	0140201	1	8	13	1
30	SE 015.01.03	RE/SE 015.01.03	RE	S	0150103	9	13	20	1
31	SE 016.02.01	RE/SE 016.02.01	RE	S	0160201	1	8	92	1
32	SE 204.01.01	RE/SE 204.01.01	RE	S	2040101	9	13	12	1
33	GR 07.02.01	RE/GR 07.02.01	RE	G		9			
34	GR 02.04.01	RE/GR 07.02.01/GR 02.04.01	RE	G		1			
35	GR 08.03.01	RE/GR 07.02.01/GR 02.04.01/GR 08.03.01	RE	G		99			
36	SE 017.04.01	RE/GR 07.02.01/GR 02.04.01/GR 08.03.01/SE 017.0	RE	S	0170401	1	8	45	1
37	SE 012.01.02	RE/GR 07.02.01/GR 02.04.01/GR 08.03.01/SE 012.0	RE	S	0120102	1	8	85	1
38	SE 017.05.01	RE/GR 07.02.01/GR 02.04.01/SE 017.05.01	RE	S	0170501	15	13	41	1
39	SE 018.02.01	RE/GR 07.02.01/GR 02.04.01/SE 018.02.01	RE	S	0180201	1	8	37	1
40	GR 03.04.01	RE/GR 07.02.01/GR 03.04.01	RE	G		1			
41	SE 020.03.01	RE/GR 07.02.01/GR 03.04.01/SE 020.03.01	RE	S	0200301	999	13	77	1
42	SE 021.02.01	RE/GR 07.02.01/GR 03.04.01/SE 021.02.01	RE	S	0210201	1	8	64	1
43	GR 04.03.01	RE/GR 07.02.01/GR 04.03.01	RE	G		1			
44	SE 023.03.01	RE/GR 07.02.01/GR 04.03.01/SE 023.03.01	RE	S	0230301	999	13	55	1
45	SE 076.02.01	RE/GR 07.02.01/GR 04.03.01/SE 076.02.01	RE	S	0760201	1	8	37	1