Using Arrays, Formats and Macros to Unbundle Medicaid Fraud

Gary McQuown and John Zheng, Data and Analytics Solutions, Fairfax, VA

ABSTRACT

Identifying unbundling fraud in Medicare and Medicaid medical claims data requires that potentially dozens of claims for a recipient be compared to an extensive list of unacceptable claim code combinations under various conditions. If any payment matches the unacceptable claim code combinations, an overpayment has been made. Given the requirement and the characteristics of the data, there are a number of potential issues. While the task was initially performed in Proc SQL, processing time was excessive and greater flexibility was desired. The chosen solution was to combine macros, arrays and formats in a single process to create an extremely efficient and flexible solution to a complex problem. The following discussion is designed for those already familiar with macros, arrays, functions and formats.

INTRODUCTION

Healthcare billing involves the use of various codes to represent every possible procedure that can be performed in a medical setting. This includes what some refer to as “parent” codes that are designed to include inclusive codes referred to as “children”. It is quite common for a healthcare provider to submit multiple codes for multiple services performed on the same recipient on the same day. However if a provider submits a bill for an all inclusive “parent” code, they should not submit a bill for any of the associated “children” codes on the same day.

In another word, unbundling detection compares recipient's claims on a given day to a list of paired parent-child codes. If a claim matches the parent code, other claims billed for the same recipient on the same date of service should not match the child services. For example procedure codes 90853 covers group psychotherapy and is a parent code. Code 99233 covers a subsequent hospital care visit and is a child code paired with 90853 because the provider must visit the recipient to participate in the psychotherapy. However provider have billed and been paid for both these codes. So there was overpayment of $112.34 which is the amount paid for the child code procedure.

The comprehensive code table used is the National Correct Coding Initiative (NCCI) developed by CMS and updated yearly (http://www.cms.hhs.gov/NationalCorrectCodInitEd/). The NCCI table contains over 200,000 pairs of parent-child code combinations representing services ranging from simple evaluations to complex surgical procedures. A small example is given below and a larger example is in appendix (1).

M_CODE	C_CODE	MODIFIER	SOURCE	EFFECTIVE_DATE	DELETE_DATE
59400	C8950	1	ocecce07v132	1/1/2006	12/31/2006
59400	C8952	1	ocecce07v132	1/1/2006	12/31/2006
59400	0021T	1	ocecce07v132	1/1/2003	12/31/2006
59400	36000	1	ocecce07v132	10/1/2002	

Challenges of solving this complex problem include conditional logic implementation, data normalization, large dataset and limited resources etc. To facilitate the matching, the claim-level data needs to be de-normalized / transposed so that all service codes provided to a recipient on a given day are in a single record /row / observation. Conditional logic is then required to match all possible combination of the codes billed to the thousands of parent code combinations that are not allowed. Additional logic is required to insure that the codes used were effective on the date of service. Under

certain conditions, yet more logic is necessary to compensate for modification codes that allow the provider to bill otherwise unacceptable combinations. With a relatively small state such as Mississippi having over 26 million claims for fiscal year 2007, the process also needs to be efficient and simple enough to be modified by support staff.

The following describes the process of combining macros, arrays and formats to detect unbundling fraud. Also detailed are the rationale behind why a particular tool was chosen and the benefits that it provides. There are certainly other ways of completing the task, but given the constraints and conditions the fore mentioned combinations were extremely satisfactory. The appendix also contains the MK_FMT macro.

DATA PREPARATION

Subsetting

The first step to increasing efficiency is to only process the data that is needed. The primary data contains every claim submitted by the providers for the recipients and only claims of either "parent" code or "child" code was billed on days where at least one "parent" was billed. Being able to restrict the data to only those that might be affected by the NCCI combinations greatly reduces processing time.

Formats combined with the data step "where" option were chosen to efficiently and dynamically select only the combinations were a "parent" code was present in the NCCI combinations. We first create the parent code and child code formats dynamically $C_CODE and $P_CODE using CNTLIN option of PROC FORMAT. The primary data was the subset with the "where" option as detailed below.

```
  where=(put(SERVICE_CODE,$P_CODE.)="Y") or
  where=(put(SERVICE_CODE,$C_CODE.)="Y")
```

In order to select only claims with child code billed on the same day with a parent code, we merge the claim table with claims with parent code only with regards to patient, provider and service date.

```
data dsn_claims;
merge  claims(in=ina where=(put(service_code, $P_CODE.)="Y")
         keep=patient_id provider_id service_date service_code)
       claims(in=inb
         where=(put(SERVICE_CODE,$C_CODE.)="Y" or put(SERVICE_CODE, $P_CODE.)="Y"));
by patient_id provider_id service_date;
if ina and inb;
run;
```

After the self-merge, the subset data now only contain claims where any NCCI parent and any NCCI child code have been billed on the same day. The remainder of the process must identify any specific NCCI parent child combinations that are present.

De-Normalization / Transposing

After selecting all the claims with interested service code, the next step is to transpose all the services performed on each recipient each day by each provider onto a single record. PROC TRANSPOSE could restructure data as needed, but it would require extensive renaming and multiple readings of the data. Therefore we chose to use arrays in the data step to de-normalize the data. This powerful and flexible tool allows us to de-normalize the data and process the data within the same data step so that only one read of the data is required.

The following is an example of the data prior to the transposition. There are four claim records for the same recipient (87654888) by the same provider (090151) on the same day (9/30/2004). This data needs to be placed into a single record/row/observation to facilitate processing. However until the data is processed, it is impossible to know the maximum number of claims on a given day. Based on past experience, the maximum number of such claims will be between twenty and forty, but normally in the high twenties.

Recipient _ID	PROVIDER_ID	SERVICE_CODE	SERVICE_DATE	AMOUNT
87654888	090151	59400	9/30/2004	$148.00
87654888	090151	59425	9/30/2004	$11.00
87654888	090151	59426	9/30/2004	$11.00
87654888	090151	59400	9/30/2004	$11.00

After de-normalization the data will look like the following. The maximum number of elements in the array will be equal to the maximum number of NCCI related claims submitted by the same provider for the same recipient on the same day. Other than the de-normalization, the values of the data are unchanged.

S1	S2	S3	S4	S0	a1	a2	a3	a4	Recipient _ID	SERVICE _DATE	PROVIDER _ID
59400	59425	59426	59426	59400	$148	$11	$11	$11	602060888	9/30/2004	090151

The RETAIN statement is a tool to allow SAS® to hold onto the values from prior observations. This is necessary to process claims that are spread over multiple records.

This following code illustrates how the data step de-normalization or transpose is done. Data is first sorted by all criterion variables RECIPIENT _ID PROVIDER_ID SERVICE_DATE SERVICE_CODE. And then the FIRST automatic variable is invoked iteratively, for RECIPIENT _ID, to initialize the counters for each recipient. A counter variable is created to track all criterion variables, with the incrementing of the rest of criterion variables on the values of variable RECIPIENT _ID. The counters, SERVICE_CODE, and AMOUNT are retained for each service. The maximum positions of the one- Dimensional array A{} M{} or S{} are dynamically assigned in a macro variable (&max).

```
data fmt_dsn;
  length S1-S&max. $7. m1-m&max. $2. ;
  retain S1-S&max. m1-m&max. a1-a&max. count 0;
set temp;
by RECIPIENT _ID PROVIDER_ID SERVICE_DATE SERVICE_CODE ;
```

```
array S(*) S1-S&max.;
array A(*) A1-A&max.;
array M(*) M1-M&max.;

/** set all values to missing for each new set of criteria **/
if first.SERVICE_DATE then
      do i = 1 to dim(s);
            S(i) = ' ';    /** service code **/
            A(i) = .;      /** amount **/
            M(i) = ' ';    /** modification codes **/
            count = 0;     /** incremental counter **/
      end;
count = count + 1;
A(count) = AMOUNT;
S(count) = SERVICE_CODE;

/** output only the last record that contains cumulative information **/
if last.SERVICE_DATE or count = dim(s) then output;
```

It is important to use the correct maximum value for the arrays. If the value is too low, important information will be lost. If the value is too high, resources will be wasted and the processing time will be extended. If the elements in the array do not match the data elements present, then an error will result and the process will fail. The solution is to calculate the correct value each time data is processed and to apply that value to the maximum number of elements in the arrays. While it does require an additional reading of the data, the data is restricted to only the by variables.

```
data cnt_dsn;
retain count pcount 0;
set dsn (keep = RECIPIENT _ID PROVIDER_ID SERVICE_DATE SERVICE_CODE);
by RECIPIENT _ID PROVIDER_ID SERVICE_DATE SERVICE_CODE;
if first. SERVICE_DATE then do;
      count = 0; pcount = 0;
end;
count = count + 1;
if first.PROVIDER_ID then pcount = pcount + 1;
if last. SERVICE_DATE and pcount > 1 then output;
run;

/** create a macro variable with the maximum number of claims **/
proc sql;
select max(count) into: mc from temp;
quit;
%let max=%sysfunc(compress(&mc));
```

UNBUNDLING IDENTIFICATION

With the data properly formatted, the next step is to compare all possible combinations of the claims data in the primary data set to the NCCI combinations and to identify any claims that match. The first inclination was to use PROC SQL. While the code for a SQL join was written quickly; the cross table condition EFFECTIVE_DATE < SERVICE_DATE < DELETE_DATE did not allow SAS to optimize the process internally. The result was that this relatively simple SQL statement proved to be very time-consuming for a relatively small amount of data. In the anticipation that much larger data would be processed, we had to look for an alternative. For various reasons we chose to utilize the data step augmented by functions and formats.

One advantage of SQL is the ability to apply a Cartesian join to compare multiple data elements in multiple data sources. This is not a task normally performed in the data step merge. Hashing was an alternative, but the number of diverse users anticipated we would prefer code that was fairly simple, easier to explain and more flexible.

The solution was to eliminate the complexity of matching multiple data elements by concatenating the NCCI combinations into a single variable. A format was then created from the unique “parent” and “child” combinations ($PC_CODE). A Do Loop was then used to cycle through all values to identify any parent codes present, concatenate that parent code with all other codes to match the format of the NCCI codes and finally to use the format to determine if the claims submitted matched any of the NCCI codes.

```
do i = 1 to dim(PC);
  if put(S(i), $P_CODE.) = "Y" then PARENT= S(i);
end;

/** if a parent code is present, create all possible PC combinations */
if PARENT not in ("") then
  do i = 1 to dim(PC);
      if S(i) not in ("") then PC(i) = compress(PARENT ||"_"||S(i) );
/** if any of the combinations match the NCCI format, output the values **/
      if put(PC(i), $PC.) = "Y" then do;
            COL_1 = scan(PC(i),1,"_");
            COL_2 = scan(PC(i),2,"_");
/** only output values where a match was found **/
            output;
   end;
/** reset counter for array **/
count = 0;
```

To “kill two birds with one stone”, we were able to combine the data step transpose and unbundling identification into a single data step to decrease processing time.

The final step of the process is to document the finding. It is important to record the "by group" variables such as the recipient, provider and date of service. Parent and child code combinations as well as the amount of payments should also be reported. The result is a simple report that identifies unbundled claims and the associated overpayments.

CONCLUSION

To summarize, we used a combination of tools to complete the task. For the sake of space and time not everything was discussed in the paper, but sufficient information was given to reproduce the process. While alternative solutions are possible, the solution provided is accurate, efficient, flexible and fairly easy to understand.

- SQL was used to determine the maximum position value of the array
- Macros parameterize the entire process
- Formats create efficient look up tables of parent, child and parent child combinations
- Where is used to subset the data keeping only the data required
- Arrays are used to de-normalize the data and to facilitate the searching.
- Do Loops process the arrays for all combinations of parent and child codes.
- Functions are used throughout in enable or enhance each process.

CONTACT INFORMATION

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

Author Name:	Gary McQuown and John Zheng
Company:	Data and Analytic Solutions, Inc.
Address:	3057 Nutley Street, #602
	Fairfax, VA 22031
Emails:	McQuown@DASconsultants.com and JZheng@DASconsultants.com
Web:	www.DASconsultants.com

The authors would like to acknowledge the Medicaid Integrity Program staff at Centers for Medicare & Medicaid Services and Dr. Dawn Li for their support.

APPENDIX

A1. NCCI table

M_CODE	C_CODE	MODIFIER	SOURCE	EFFECTIVE_DATE	DELETE_DATE
59400	C8950	1	ocecce07v132	1/1/2006	12/31/2006
59400	C8952	1	ocecce07v132	1/1/2006	12/31/2006
59400	0021T	1	ocecce07v132	1/1/2003	12/31/2006
59400	36000	1	ocecce07v132	10/1/2002	
59400	36410	1	ocecce07v132	10/1/2002	
59400	37202	1	ocecce07v132	10/1/2002	
59400	51701	1	ocecce07v132	1/1/2003	
59400	51702	1	ocecce07v132	1/1/2003	
59400	57720	1	ocecce07v132	1/1/1997	
59400	58800	1	ocecce07v132	1/1/1997	
59400	59050	0	ocecce07v132	1/1/1996	
59400	59051	0	ocecce07v132	4/1/2002	
59400	59200	0	ocecce07v132	4/1/1997	
59400	59300	0	ocecce07v132	4/1/2003	
59400	59414	0	ocecce07v132	1/1/1997	
59400	62311	0	ocecce07v132	7/1/2003	
59400	62318	1	ocecce07v132	10/1/2002	
59400	62319	1	ocecce07v132	10/1/2002	
59400	64415	1	ocecce07v132	10/1/2002	
59400	64416	1	ocecce07v132	1/1/2003	
59400	64417	1	ocecce07v132	10/1/2002	
59400	64430	0	ocecce07v132	7/1/2003	
59400	64435	0	ocecce07v132	7/1/2003	
59400	64450	1	ocecce07v132	10/1/2002	
59400	64470	1	ocecce07v132	10/1/2002	
59400	64475	1	ocecce07v132	10/1/2002	
59400	64483	0	ocecce07v132	7/1/2003	
59400	69990	0	ocecce07v132	6/5/2000	
59400	81000	0	ocecce07v132	7/1/2007	
59400	81002	0	ocecce07v132	7/1/2007	
59400	90760	1	ocecce07v132	1/1/2006	
59400	90765	1	ocecce07v132	1/1/2006	
59400	90772	1	ocecce07v132	1/1/2006	
59400	90774	1	ocecce07v132	1/1/2006	
59400	90775	1	ocecce07v132	1/1/2006	

A2. CODE: MK_FMT

```
%macro mk_fmt (dsn=, start=,label=, fmtname=, type=C, Library=work, Other=OTHER);

/** mk_formats.sas

To: Create a format from a SAS data set.

Parms:
      DSN
```

```
      START   =Unique key value  ie. SSN
      LABEL   =Value to be associated with start ie. Full Name with SSN
      FMTNAME     =Name of Format (sans "."), must be seven or less in length
      TYPE  = C or N for Character or Numberic
      Library =libname of Format Library  (default =work)
      Other   =Value to supply for missing (default =OTHER)
      By: wwww.DASconsultants.com

**/

/** note: the following should be used: options fmtsearch=(fmtlib) nofmterr;  **/


DATA temp1 ; set &dsn  end=last;

  if missing(&start) then delete;

    start   =trim(left(&start.));
    fmtname ="&fmtname.";
    type    ="&type.";
    label   =&label;
      output;
    if last then
    do;
      start = "OTHER";
     %if &type = C %then label = "&OTHER"; %else label = . ; ;
        output;
    end;
    keep start fmtname type label;
run;

proc sort data=temp1 nodupkey; by start; run;
proc format cntlin=temp1 library=&library; run;

%mend mk_fmt ;
```

A3. CODE: SAME DAY UNBUNDLING

```
/*************************************************************
*************************************************************
NAME:          OT_MD_UB COMPREHENSIVE UNBUNDLING MOD=0
BY:       www.DASconsultants.com
```

```
VERSION:  1.2
DATE:          01/15/2008
CATEGORY: Unbundling
STATE:    US
*************************************************************
*************************************************************/

%macro OT_MD_UB_COMP_UNBUNDLING0 (st,DSN, out=F:\REPORTS\&ST.\OT);
/** CC_ALL contains all NCCI CPT codes.  **/
data cc_all;
   length MC $11.;
   set tp.cc_ot (where=(MODIFIER in ("0")));
   if DELETE_DATE = . then DELETE_DATE = "&sysdate"d;
   MC = compress(PARENT_CODE||"_"||CHILD_CODE);
run;

/** Create a format for quick retrieveal.  **/
%mk_fmt (dsn=cc_all, start=PARENT_CODE,label="Y", fmtname=UBA, type=C,
Library=work, Other=OTHER);
%mk_fmt (dsn=cc_all, start=CHILD_CODE,label="Y", fmtname=UBB, type=C,
Library=work, Other=OTHER);
%mk_fmt (dsn=cc_all, start=MC,label="Y",    fmtname=MC,  type=C, Library=work,
Other=OTHER);
%mk_fmt (dsn=cc_all, start=MC,label=EFFECTIVE_DATE, fmtname=E_DATE,  type=C,
Library=work, Other=.);
%mk_fmt (dsn=cc_all, start=MC,label=DELETE_DATE,    fmtname=D_DATE,  type=C,
Library=work, Other=.);

/** process only data that contains potentially bundled service codes **/
proc sort data=&st..&st.ot&dsn
      (where=(put(SERVICE_CODE, $UBA.) ="Y" or put(SERVICE_CODE, $UBB.) ="Y" ))

      out=claims_pc;
by PATIENT_ID PROVIDER_ID SERVICE_DATE SERVICE_CODE;
run;

/**keep claims where any NCCI parent and any NCCI child code have been billed on
the same day**/
data claims;
retain count 0;
merge claims_pc(in=ina where=(put(SERVICE_CODE, $P_CODE.)="Y")
```

```
                                keep=PATIENT_ID PROVIDER_ID SERVICE_DATE
SERVICE_CODE)
        claims_pc(in=inb);
by PATIENT_ID PROVIDER_ID SERVICE_DATE SERVICE_CODE;
if ina and inb;
if first.SERVICE_DATE then count = 0;
count = count + 1;
run;

/** determine the max number of OB services per recipient **/
proc sql;
select max(count) into: mc from claims;
quit;
%let max=%sysfunc(compress(&mc));

/** Data Normalization & Unbundling Identification**/
data claims_norm(keep = S1-S&max. M1-M&max. A1-A&max. MC1-MC&max.
PARENT PATIENT_ID PROVIDER_ID SERVICE_DATE PARENT_CODE CHILD_CODE
EFFECTIVE_DATE DELETE_DATE);

length S1-S&max. $7. m1-m&max. $2. MC1-MC&max. $11.;
retain S1-S&max. ;
retain M1-M&max. ;
retain count a1-a&max. 0;
set claims(where=(ADJUSTMENT_INDICATOR = "0")) ;
by PATIENT_ID PROVIDER_ID SERVICE_DATE SERVICE_CODE ;

array S(*) S1-S&max.;
array A(*) A1-A&max.;
array M(*) M1-M&max.;
array MC(*) $ MC1-MC&max.;

*array initialization;
if first.SERVICE_DATE then do i = 1 to dim(s);
      s(i) = ' ';
    a(i) = .;
      m(i) = ' ';
      count = 0;
end;

* Data Normalization;
count = count + 1;
```

```
S(count) = SERVICE_CODE;
A(count) = MEDICAID_AMOUNT_PAID;
M(count) = SERVICE_CODE_MOD;

*Unbundling Identification;
if last.SERVICE_DATE or count = dim(s) then
do;
      *indentify the parent_code;
      do i = 1 to dim(MC);
        if put(S(i), $UBA.) = "Y" then PARENT= S(i);
    end;

      *indentify the valid parent-child code pair;
      if PARENT ^= "" then do i = 1 to dim(MC);
        if S(i) not in ("") then MC(i) = compress(PARENT||"_"||S(i) );
        if put(MC(i), $MC.) = "Y" and
            put(MC(i), $E_DATE.) <= SERVICE_DATE < put(MC(i), $D_DATE.)
            then
        do;
                PARENT_CODE = scan(MC(i),1,"_");
                CHILD_CODE = scan(MC(i),2,"_");
                  EFFECTIVE_DATE = put(MC(i), $E_DATE.);
                  DELETE_DATE = put(MC(i), $D_DATE.);
                  output;
        end;
      count = 0;

      end;
end;
run;

/** calculate OVERPAYMENT **/
data &st..&st.ot&dsn._ub_comp  ;
retain VALID_MOD OVERPAY_MOD;
set claims_norm;

array S(*) S1-S&max.;
array A(*) A1-A&max.;
array M(*) M1-M&max.;

do i = 1 to dim(S);
      if s(i) = CHILD_CODE then do; OVERPAYMENT = a(i); OVERPAY_MOD=M(i); end;
```

```
      if s(i) = PARENT_CODE then VALID_MOD = M(i);
end;

if OVERPAYMENT <= 0 then delete;
run;

%mend OT_MD_UB_COMP_UNBUNDLING0;
```