SCSUG 2018

Binning Bombs When You're Not a Bomb Maker: A Code-Free Methodology To Standardize, Categorize, and Denormalize Categorical Data Through Taxonomical Control Tables

Troy Martin Hughes

ABSTRACT

Benefits of categorical data include the structure imposed through standardized values and the analytic opportunity this facilitates; without rules, only nominal, unstructured data exist. However, when quality assurance is not enforced through data entry constraints, the inclusion of acronyms, synonyms, abbreviations, and other spelling variations impedes understanding and analysis. The construction of a taxonomy, thus, is required to bin categorical data into values that remove ambiguous and unwanted variation. While taxonomies often are implied through static if-then-else logic or text parsing scripts, these rudimentary approaches can be difficult to design and to interpret. Moreover, analysts' taxonomical views may differ, requiring competing taxonomies to be maintained concurrently by developers who may lack subject matter expertise. This paper introduces a SAS macro that empowers subject matter experts—not SAS developers—to build, validate, maintain, and modify taxonomies without the need for code revision. Through dynamic control tables, hierarchical taxonomical models are constructed that automatically standardize and categorize data into tiered structures and that produce HTML contingency tables useful for model validation. Moreover, categorized data automatically are denormalized into a flat file to facilitate further analysis.

INTRODUCTION

You hear a siren approaching and watch a red, multi-ton vehicle race by with emergency lights flashing; a woman remarks to her young son "There goes a fire engine!" while he exclaims "Fire truck!" Has this been a semantically confusing exchange, or a successful use of synonyms? To many people—likely including this pair—fire engine and fire truck represent interchangeable terms. As a firefighter, however, each has a distinct technical connotation that must be used appropriately to avoid confusion. An engine maintains a several hundred gallon water tank and the ability to pump water, whereas a truck has ladders and tools for search, rescue, and extrication. Aerial trucks represent one category of fire truck, some of whose subcategories include towers, tillers, and hook and ladders. Further complexity is added when colloquial terms replace technical ones; both wagon and pumper, for example, are used synonymously to describe fire engines, even by firefighters. When a fire apparatus taxonomy is constructed that includes all subcategories, synonyms, and spelling variations, it comprises a hierarchical structure of more than 50 terms, each of which a layman might refer to simply as a "fire truck." Thus, to ensure semantic clarity and analytic integrity, raw categorical values must be mapped accurately within a taxonomy.

The analytic value of categorical data is improved further when data integrity constraints are enforced during data entry; however, if the quality of data cannot be guaranteed—as is often the case with third-party data—this burden must be borne later in the data life cycle by developers. Unstandardized categorical responses typically are resolved through if-then-else conditional logic or text parsing. This logic can be complicated where taxonomies or analytic requirements differ; one analyst might want to examine aerial trucks separately from fire engines, while a second analyst might combine towers and tillers into a single category. Thus, the process that standardizes and categorizes data—often termed "binning"—must remain flexible to meet the disparate needs of all data consumers. Once data are binned, a common final step of categorical data transformation is denormalization into a flat file; each categorical field is transposed into multiple numeric fields that represent frequency counts of unique values. Refined flat files are preferred for certain analyses because the data can be analyzed directly without further need for cleaning or transformation and because flat files inherently have a unique key by which they can be joined.

The Joint Improvised Explosive Device Defeat Organization (JIEDDO) is charged with analyzing and defeating improvised explosive devices (IEDs) and the insurgents that use them to kill, maim, and terrorize Coalition Forces (CF) and civilians[1]. IEDs consistently are the leading cause of CF casualties in Afghanistan and have wounded and killed tens of thousands of US service members[2]. Because over 80 percent of IEDs contain homemade explosives (HME) as opposed to military-grade, commercial, or other charges[3,4], chemical analysis is a critical component to defeating bombs and bomb makers. A primary obstacle to analysis, however, is the complexity of chemical taxonomies; one scientist might require two precursor chemicals to be binned separately, while another might prefer to reference the chemicals interchangeably. Furthermore, chemical abbreviations, acronyms, and spelling variations may represent valid nomenclature to chemists but utterly baffle developers who, while savvy software engineers, are not scientists. Thus, just as laymen often cannot recognize the sometimes subtle variations between fire apparatuses, developers are ill-equipped to build and maintain complex taxonomies having thousands of lines of static conditional logic.

The dynamic solution implemented at JIEDDO efficiently standardizes, categorizes, and denormalizes categorical data—all without the necessity of building or maintaining static code. A taxonomy still is required but, unlike legacy if-then-else code, it is maintained within a control table—also called a decision table—that dynamically transforms data. The simplicity of the control table empowers non-developers to build and maintain schemas through modifications to an ASCII text file. Once a schema is created, the taxonomy is represented through an HTML report useful for model validation. As a schema changes, modifications are reflected in this taxonomical report and are incorporated automatically into both the standardized data set and the refined flat file. In addition, multiple control tables can be maintained simultaneously, thus enabling competing, concurrent taxonomies. Most importantly, the binning of bombs is placed back into the capable hands of chemists and scientists and, because the solution is scalable and generalizable, it can be applied to categorical data of varying content, quantity, and quality.

INTRODUCTION TO HOMEMADE EXPLOSIVES (HME)

JIEDDO leads the global effort to defeat IEDs and the terrorists who manufacture, distribute, and emplace them. HME, in the most basic sense, consists of the pairing of an oxidizer and a fuel which, when mixed, form the explosive. In Afghanistan, fertilizer-based oxidizers are most common, with the majority derived from ammonium nitrate and calcium ammonium nitrate; aluminum is a common fuel. Ammonium nitrate and aluminum (ANAL), for example, is the explosive formed from those respective compounds. While the expansive HME taxonomy is sensitive, this paper presents an abridged taxonomy derived from JIEDDO's unclassified Homemade Explosive (HME)/Bulk Explosive (BE) Recognition Guide[5] to demonstrate the binning methodology, depicted in Table 1:

Oxidizer
ammonium nitrate (fertilizer, prills)
calcium ammonium nitrate (fertilizer, prills)
potassium chlorate
Fuel
aluminum powder
urea nitrate

Table 1. Abridged Taxonomy of HME Precursor Materials

Given data that are structured but inconsistent, the first step in building a taxonomy is data standardization—the removal of unwanted variation or noise. Because NH4NO3 is the chemical formula for ammonium nitrate, the two terms are synonymous. Ammmonium nitrate, similarly, also represents the same chemical, albeit entered by an overzealous typist who keyed "M" one too many times. Data entry constraints could have eliminated these discrepancies but, because the terms should not be distinguished analytically, they should be binned into a single, standardized value. The following conditional logic accomplishes this task, by creating the new field Chemical1:

```
if chemical in ('ammonium nitrate','ammmonium nitrate','NH4NO3') then
    chemical1='ammonium nitrate';
```

This can be an arduous process because each observed value must be accounted for in the model. Text parsing represents one method to expedite the process, but fidelity may be compromised because as new values appear, they may be binned automatically albeit erroneously into the wrong category.

DATA STANDARDIZATION

Standardization, as utilized herein, refers not to the statistical connotation but rather to the removal of ambiguity, unwanted variation, or "noise" from the data. A standardized categorical variable is one for which distinct values represent truly distinct constructs (i.e., construct validity) and for which synonymous values are represented by a single construct (i.e., signal-to-noise ratio.) For example, to maintain construct validity, because ammonium nitrate and ammonium nitrite represent distinct constructs, two separate entries should be maintained in the taxonomy. Noise in the data results when variations of the same value occur and are not modeled appropriately; for example, ammonium nitrate and its chemical nomenclature NH4NO3 should be indistinguishable in the model. Thus, to reduce noise and build useful data constructs, categorical values must be portrayed accurately by the model.

Text parsing scripts (e.g., PERL) are a common solution that often efficiently can bin data with significantly less code. The complexity of writing—and, worse, deciphering—scripts are primary detractors from their implementation. Step away from a PERL script for a few weeks or months and you will find yourself starring into the eyes of a stranger. In a complex taxonomy, a more straightforward approach may be required to facilitate schema creation and validation,

which often can be accomplished only by subject matter experts, and not necessarily by developers maintaining the code. Thus, at the heart of the JIEDDO requirement was a solution that would allow subject matter experts to build, validate, and maintain their own taxonomies.

A taxonomical control table offers the greatest flexibility to standardize and categorize data because binning occurs without direct code modification. Acting as a lookup table, the control table prescribes how a data set should be transformed. Table 2 represents the control table in Table 1 and, when a value appearing in the right-hand column is observed in the raw data, the value is transformed to the standardized value in the left-hand column:

Chemical1 (standardized value)	**Chemical (raw value)**
ammonium nitrate	ammonium nitrate
	ammonium nitrate fertilizer
	ammonium nitrate prills
	ammmonium nitrate
	NH4NO3
calcium ammonium nitrate	calcium ammonium nitrate
	calcium ammonium nitrate fertilizer
potassium chlorate	potassium chlorate
	KClO3
aluminum	aluminum
	aluminum powder
urea nitrate	urea nitrate
	urea

Table 2. Abridged Taxonomy of HME Precursor Materials (Standardized)

Table 2 can be represented with the following conditional logic code:

```
if chemical in ('ammonium nitrate','ammonium nitrate fertilizer',
    'ammonium nitrate prills','ammmonium nitrate','NH4NO3') then
    chemical1='ammonium nitrate');
else if chemical in ('calcium ammonium nitrate',
    'calcium ammonium nitrate fertilizer') then chemical1=
    'calcium ammonium nitrate';
else if chemical in ('potassium chlorate','KClO3') then chemical1=
    'potassium chlorate';
else if chemical in ('aluminum','aluminum powder') then chemical1='alumnium';
else if chemical in ('urea nitrate','urea') then chemical1='urea nitrate';
```

The macro solution implemented, thus, dynamically creates the above code directly from the control table, without the necessity for creating or maintaining complicated and lengthy if-then-else or other logic or text-parsing code. This improvement is demonstrated below in the Binning Bombs section.

DATA CATEGORIZATION

While the minimal amount of data cleaning and standardization has been accomplished, additional hierarchical levels may lend themselves to further categorization. For example, scientists may be interested in binning all oxidizers (or all fertilizer-based oxidizers) together. Table 3 demonstrates the expanded taxonomy that has been standardized and categorized:

Chemical1 (category)	**Chemical2 (category)**	**Chemical3 (refined value)**	**Chemical (raw value)**
Oxidizer	Fertilizer-Based	ammonium nitrate	ammonium nitrate
			ammonium nitrate fertilizer

			ammonium nitrate prills
			ammmonium nitrate
			NH4NO3
		calcium ammonium nitrate	calcium ammonium nitrate
			calcium ammonium nitrate fertilizer
	Other-Based	potassium chlorate	potassium chlorate
			KClO3
Fuel		aluminum	aluminum
			aluminum powder
		urea nitrate	urea nitrate
			urea

Table 3. Abridged Taxonomy of HME Precursor Materials (Standardized and Categorized)

Given the above taxonomy, the first chemical 'ammonium nitrate' can be binned through the following code:

```
if chemical in ('ammonium nitrate','ammonium nitrate fertilizer',
           'ammmonium nitrate','ammonium nitrate prills','NH4NO3') then do;
    chemical1='oxidizer';
    chemical2='fertilizer-based';
    chemical3='ammonium nitrate';
    end;
```

The resultant data set will contain the original field Chemical as well as new fields Chemical1, Chemical2, and Chemical3 that contain the standardized, categorized values. Analysts are thus able to analyze data at any desired specificity—the original raw value, the refined value, or one of its superordinate categorical groups. With each additional category or hierarchical level, however, the amount of required code grows. Thousands of lines of static code would be required to bin data according to the actual unabridged HME taxonomy; an automated solution would be necessary to accomplish this feat.

DATA DENORMALIZATION

Denormalization, as referenced herein, is the transformation of a data set into a flat file having one or more primary keys which facilitate both analysis and joining to other data sets. Denormalization combines observations that share a common value, essentially transforming a foreign key into an unduplicated primary key. One observation in a flat file thus represents summations (or other transformations) performed across multiple observations in the raw data. Many transformations are facilitated by numeric data, which permit simple, quantitative analyses. Table 4 represents a fictitious data set in which data are grouped by the field Date_Discovered, and in which the field Pounds_of_HME will be summed across groups:

Date_Discovered	Pounds_of_HME
10-20-2013	50
10-20-2013	20
10-21-2013	10
10-21-2013	20
10-22-2013	343

Table 4. Normalized Data Representing HME Seizures

Table 5 represents the resultant flat file, denormalized by Date_Discovered, with the field Pounds_of_HME summed into the new field Total_Pounds_of_HME:

Date_Discovered	Total_Pounds_of_HME
10-20-2013	70
10-21-2013	30
10-22-2013	343

Table 5. Denormalized Data Representing HME Seizures

Denormalization is less straightforward, however, where categorical data are involved. Table 6 represents a separate fictitious data set that includes type of HME precursor seized and the Afghanistan province in which it was encountered:

Province	HME
Zabul	ammonium nitrate
Zabul	ammonium nitrate
Zabul	potassium chlorate
Kandahar	urea nitrate
Kandahar	urea nitrate
Helmand	potassium chlorate

Table 6. Normalized Data Representing HME Precursor Seizure Locations

In order to denormalize data presented in Table 6 by Province, an analyst must determine whether only the nominal properties of the HME field should be maintained, or whether the magnitude also should be included. In other words, is it important to know only that ammonium nitrate was seized in Zabul, or is it necessary also to know how many times it was seized in Zabul? The analyst must also determine whether the flat file should represent each categorical variable in a single (often concatenated) field, or whether multiple fields can be utilized. In the former case, HME observations in Zabul might be represented in a comma-delimited format as “ammonium nitrate, potassium chlorate”; in the latter case, new numeric fields Ammonium_Nitrate and Potassium_Chlorate might be created and ascribed the values of 2 and 1, respectively, to denote magnitude. Several methods exist for transforming categorical data into flat files; while none are as straightforward as the quantitative approaches available for numeric data, categorical transformation is successful when performed consistently and in support of the analytic intent.

Given the precursor data in Table 6, the flat file structure that was required is demonstrated in Table 7, and represents both the type and magnitude of chemicals encountered, denormalized by Province:

Province	ammonium_nitrate	potassium_chlorate	urea_nitrate
Zabul	0	1	0
Kandahar	0	0	2
Helmand	0	1	0

Table 7. Denormalized Data Representing HME Precursor Seizure Locations

The static code to produce the flat file (Flat) presented in Table 7 could resemble the following, given the data already are sorted by Province, and given all values of HME precursors are known:

```
data Flat (drop HME);
    set NotSoFlat;
    by province;
    length ammonium_nitrate potassium_chlorate urea_nitrate $50;
    if first.province then do;
          ammonium_nitrate=0;
          potassium_chlorate=0;
          urea_nitrate=0;
          end;
    if HME='ammonium nitrate' then ammonium_nitrate+1;
    else if HME='potassium chlorate' then potassium_chlorate+1;
    else if HME='urea nitrate' then urea_nitrate+1;
```

```
    if last.province then output;
    retain ammonium_nitrate potassium_chlorate urea_nitrate;
run;
```

While this code can be shortened somewhat through array processing and other efficiency measures, a static code solution defies efficiency, remaining lengthy and difficult to maintain. If a new spelling variation is encountered in the raw data, it must be binned appropriately or it will not be parsed through the above code. If a new chemical is encountered, denormalization code must be modified to create an additional field in the flat file. Moreover, because values often may contain spaces and other non-alphanumeric characters, the developer is required manually to name each numeric field in the flat file. A communication barrier also may exist between developers and subject matter experts; if a developer must first interpret a taxonomy in order to translate it into code, analysts or subject matter experts likely will want to validate the code and results to ensure conformity to the taxonomy. This validation can be taxing when a taxonomy is not explicitly defined and can only be inferred through hundreds or thousands of lines of conditional logic code. A dynamic, robust solution is required and is presented in the following sections.

CONSTRUCTING A TAXONOMICAL CONTROL TABLE

The first step toward an efficient solution is the construction of a control table that represents the standardized and categorized taxonomy. The control table is used to create dynamic, data-driven processing that prescribes all transformations and which produces HTML reports and contingency tables useful for taxonomical model validation. The following ASCII text file (chem_abridged.txt) represents a simplified taxonomy and control table, derived from Table 2:

```
<LIB> CHEMICALS
<TAB> CHEM_ABRIDGED
<VAR> CHEM
<1> Oxidizer
<2> Fertilizer-Based
<3> ammonium nitrate
ammonium nitrate
ammonium nitrate prills
ammonium nitrate fertilizer
ammmonium nitrate
NH4N03
<3> calcium ammonium nitrate
calcium ammonium nitrate
calcium ammonium nitrate fertilizer
<2> Other-Based
<3> potassium chlorate
potassium chlorate
KClO3
<1> Fuel
<3> aluminum
aluminum
aluminum powder
<3> urea
urea
urea nitrate
```

Within the ASCII control table, the following structure and tags are prescribed:

- All entries should be left-justified.
- <LIB> (required on the first row) represents the SAS library containing the data set being processed (i.e., standardized, categorized, and denormalized.)
- <TAB> (required on the second row) represents the SAS data set containing the data set being processed.
- <VAR> (required on the third row; additional instances can occur later in the control table) represents the categorical field being processed. An infinite number of variables can be defined in a single control table, so long as each variable occurs in the referenced SAS data set.

- <1>…<N> (at least one <1> is required per superordinate <VAR>; numbers can be repeated, and can increment to infinity) each discrete number represents a categorical level identified in the data, with higher numbers representing subcategories of the numbers they follow. In the example above, <1> Oxidizer represents a first order category, and <2> Fertilizer-Based a subcategory subsumed entirely within <1> Oxidizer. Although any higher number represents a subcategory of the previous (smaller) number, the higher number is not required to increment by 1, as evidenced in the jump from <1> Fuel to <3> aluminum. Because categorical tags eventually are transformed into SAS variables during denormalization, they 1) cannot be repeated within a single control table, and 2) must contain 30 characters or less. Tag names, however, can contain spaces and other non-alphanumeric characters that improve readability.
- Entries containing none of the above tags represent raw values observed in the data set. Successive entries subordinate to a single numeric tag indicate that those values are interchangeable (e.g., synonyms, abbreviations, spelling errors and variations) and should be mapped to the same data construct. For example, because “urea” and “urea nitrate” are subsumed under the same tag <3> urea, the chem_abridged taxonomy indicates that both terms are being treated interchangeably as the single term Urea. Capitalization is irrelevant, so including values “aluminum” and “Aluminum” would be redundant because both raw values would be identified and treated interchangeably.

The ASCII control table should be created in an empty directory, which should be ascribed the SAS library name BINNING through the LIBNAME statement. Because the BINNING library ultimately may contain control tables for multiple data sets or projects, a generic location is preferred rather than a project-specific one. Once the control table is created and the BINNING library is established, the macro %IMPORT_BINNING (see Appendix A) should be invoked with the following parameters:

```
libname binning [INSERT REFERENCE TO LIBRARY PATH];
%let path=%sysfunc(pathname(binning));
%import_binning(intab=&path.\chem_abridged.txt, outtab=chem_abridged, lev=5);
```

The above invocation of %IMPORT_BINNING locates the ASCII control table (chem_abridged.txt), and creates the derivative SAS control table (BINNING.chem_abridged) and the taxonomical report (chem_abridged.html) that can be utilized to demonstrate or validate the model. The LEV parameter indicates that the process will search for no more than five hierarchical levels, which is appropriate because the highest numeric tag found in the ASCII control table is <3>. The chem_abridged.HTML report is demonstrated below in Table 8 (and mimics the standardized and categorized schema shown in Table3):

Level 1	**Level 2**	**Level 3**	**Value**
Oxidizer	Fertilizer-Based	ammonium nitrate	ammonium nitrate
			ammonium nitrate prills
			ammonium nitrate fertilizer
			NH4NO3
			ammmonium nitrate
		calcium ammonium nitrate	calcium ammonium nitrate
			calcium ammonium nitrate fertilizer
	Other-Based	potassium chlorate	potassium chlorate
			KClO3
Fuel		aluminum	aluminum
			aluminum powder
		urea	urea
			urea nitrate

Table 8. Standardized and Categorized HME Precursor Chemical Taxonomy Report

MODIFYING A TAXONOMICAL CONTROL TABLE

While there are numerous advantages to creating an explicit taxonomy (as opposed to an implicit taxonomy containing if-then-else or other conditional logic), one of the greatest is the ease with which modifications can be made. For example, after reviewing the taxonomical report depicted in Table 8, a scientist might note that powdered sugar—identified in the HME Recognition Guide as a fuel source—is absent. Furthermore, an analyst familiar with the raw data might report that an overzealous—and, potentially underfed—typist has keyed “Aluminumnumnum” in place of aluminum. Rather than troubling a developer with a request to modify the code to accept these additional cases, the analyst or scientist directly can modify the ASCII control table. Communication and collaboration also are maximized because the control tables—coded in simple ASCII text—can be emailed without concern for formatting challenges or other limitations. The below control table includes the scientist’s and analyst’s recommendations that incorporate sugar and aluminumnumnum:

```
<LIB> CHEMICALS
<TAB> CHEM_ABRIDGED
<VAR> CHEM
<1> Oxidizer
<2> Fertilizer-Based
<3> ammonium nitrate
ammonium nitrate
ammonium nitrate prills
ammonium nitrate fertilizer
ammmonium nitrate
NH4N03
<3> calcium ammonium nitrate
calcium ammonium nitrate
calcium ammonium nitrate fertilizer
<2> Other-Based
<3> potassium chlorate
potassium chlorate
KClO3
<1> Fuel
<3> aluminum
aluminum
aluminum powder
aluminumnumnum
<3> urea
urea
urea nitrate
<3> sugar
powdered sugar
sugar
```

After the revised ASCII control table has been saved, the %IMPORT_BINNING macro can be run again with identical parameters, which will produce an updated SAS control table and its respective HTML taxonomical report. This is an acceptable level of overhead if taxonomies are fairly stable and modifications uncommon. However, in more dynamic environments in which multiple taxonomies actively are being developed and modified daily, a preferred approach requires invocation of the %IMPORT_BINNING_ALL macro (see Appendix B), as demonstrated below:

```
%import_binning_all;
```

The %IMPORT_BINNING_ALL macro requires no parameters, and ingests all ASCII control tables located in the SAS BINNING library, producing all respective SAS control tables and HTML reports. Thus, in a more dynamic environment, this macro can be scheduled to run on an hourly, daily, or as-needed basis to ensure currency of all taxonomies. The %IMPORT_BINNING parameter LEV is set to a default of five in the macro invocation, but can be adjusted to support more complex, hierarchical environments. By scheduling %IMPORT_BINNING_ALL to run at regular intervals, both developers and analysts can be ensured that only the most current taxonomies are being utilized.

A second advantage of creating taxonomical control tables is the ability to maintain concurrent, competing models without the frustration of code upkeep. Because the above control table can be edited with ease, multiple versions can be maintained that highlight disparate viewpoints. And, with each new model that is produced, the accompanying HTML report can be used to demonstrate the strengths, weaknesses, and differences of those analytic perspectives. For example, if a subsequent viewpoint emerged that powdered sugar was not to be considered a fuel, the taxonomy and all derivative data and reports could be modified by removing three lines of text from the ASCII control table; in the traditional, static coding paradigm, the equivalent change would require substantial, entangled code revision. And, in those instances in which a shared taxonomy cannot be agreed upon, multiple versions can be maintained and generated with ease and—you guessed it—without code revision.

The third major advantage of control table implementation already has been stated, but its impact not fully explored—the ability to modify a taxonomy, its data, and reports without code modification. While the time savings aspect is attractive, a greater allure can be the ability to maintain robust, dynamic production-level code that does not require frequent—or any—revision. Because the data-driven processes are directed by an external control table, that ASCII table can be modified at will while SAS code continues to run in the background. In a production environment in which SAS jobs are deployed to and scheduled on a central server, code modification is not only inefficient but moreover can cause process interruption for minutes, hours, or days. And, as with any surgery there exists the potential for infection to enter the body, with any code revision, there exists the possibility that a rogue character or illogical statement will cause production to grind to a halt.

BINNING BOMBS: APPLYING A TAXONOMY TO RAW DATA

The taxonomy (chem_abridged) has been created and modified and now is ready to be operationalized on fictitious HME precursor chemical data. In the below data set, RecNo represents the record number identifying a particular seizure of chemicals; Province represents the province in which the seizure occurred, and; Chem represents the precursor chemical. Because each seizure can occur in only one province, all respective values of Province will be identical for each unique value of RecNo. The below code creates the raw data set (work.chem_data):

```
data chem_data;
    infile datalines delimiter=',';
    length RecNo $20 Province $ 40 Chem $50;
    input RecNo $ Province $ Chem $;
    datalines;
1,Zabul,ammonium nitrate
2,Kandahar,ammonium nitrate
3,Helmand,calcium ammonium nitrate
3,Helmand,sugar
3,Helmand,urea
4,Ghazni,calcium ammonium nitrate
4,Ghazni,ammmonium nitrate
4,Ghazni,potassium chlorate
5,Zabul,ammonium nitrate crystals
6,Kandahar,aluminumnumnum
;
run;
```

In order to operationalize a taxonomy and bin a data set, the %BINMEPLEASE (see Appendix C) macro should be invoked, which contains the following parameters:

- BINTAB – the binning (i.e., control) table that contains the taxonomy, which should include both the BINNING library and data set name.
- INTAB – the data set containing the categorical data to be processed.
- OUTTAB – the name of the refined (i.e., standardized and categorized) data set (which will be utilized later as the input data set during the denormalization process %FLATTEN.)
- VARLIST – the space-delimited list of all categorical fields that are being binned; note that each field must have a corresponding variable (as indicated by a <VAR> tag) in the referenced SAS control table.

- LEV – number representing the highest hierarchical level to be searched and binned; note that because a control table can contain schemas for multiple fields, the highest hierarchical level of ALL fields must be utilized as the LEV parameter.
- PRINTFREQ – YES to produce an optional HTML contingency table that demonstrates the relative frequencies of all values and categorical levels observed in the data, as well as highlights values that appear in raw data but not within the taxonomy.

To apply the taxonomy (chem_abridged) to the data set (chem_data), the following parameters should be specified:

```
%binmeplease (bintab=binning.chem_abridged, intab=chem_data,
    outtab=chem_data_binned, varlist=chem, lev=5, printfreq=yes);
```

The binned data set (chem_data_binned) produced by the %BINMEPLEASE macro is identical to the original data set (chem_data), with the exception of three fields (Chem1, Chem2, and Chem3) that have been added. Chem1 corresponds to the highest-order category (e.g., Oxidizer or Fuel), Chem2 to the second-order category (e.g., Fertilizer-Based), and Chem3 to the standardized value with unwanted variation removed. In analyses that require use of the Chem field, it is thus appropriate to instead run the analysis on the standardized Chem3 field, or optionally on one of the categorical fields (e.g., Chem1 or Chem2.) To reduce confusion between the raw and refined data, the raw data field (Chem) optionally can be deleted once a taxonomy has been adopted and validated.

The contingency table report demonstrates identical information as produced in the previous taxonomical report (chem_abridged.html) by the %IMPORT_BINNING macro, but additionally includes frequency counts for values, summarized at each categorical level. This critical information immediately illustrates frequently occurring values in the data set as well as values identified in the taxonomy but not found in the data set. Furthermore, values that appear in the data set but are not identified in the taxonomy appear at the bottom of the report in red highlighting to alert analysts that the values (likely) should be included in the ASCII control table. This quality control measure is meant to ensure that analytic integrity is never compromised simply because of spelling variation or shifting notation in the raw data. The contingency table for the above macro invocation is demonstrated below in Table 9 (and includes the updated sugar construct):

Level 1	**Count**	**Level 2**	**Count**	**Level 3**	**Count**	**Value**	**Count**
Oxidizer	6	Fertilizer-Based	5	ammonium nitrate	3	ammonium nitrate	2
						ammonium nitrate prills	
						ammonium nitrate fertilizer	
						NH4NO3	
						ammmonium nitrate	1
				calcium ammonium nitrate	2	calcium ammonium nitrate	2
						calcium ammonium nitrate fertilizer	
		Other-Based	1	potassium chlorate	1	potassium chlorate	1
						KClO3	
Fuel	3			aluminum	1	aluminum	
						aluminum powder	
						aluminumnumnum	1
				urea	1	urea	1
						urea nitrate	

				sugar	1	powdered sugar	
						sugar	1
VALUE NOT IN SCHEMA	1					AMMONIUM NITRATE CRYSTALS	1

Table 9. HME Precursor Chemical Taxonomical Contingency Table

The Table 9 contingency table illustrates that a new value “ammonium nitrate crystals” has appeared, which an analyst aptly will bin into the “ammonium nitrate” construct through a simple modification to the ASCII control table. Table 9 also illustrates the improved signal-to-noise ratio observed in the standardized value column (i.e., Level 3) as compared to the welter of spelling errors, variations, and chemical formulae observed in the raw value column. With the Chem field (i.e., Value column in report) removed from the data set and its values more aptly classified as constructs and categories Chem1, Chem2, and Chem3, distraction is reduced and analytic clarity and integrity are improved.

DENORMALIZATION DONE RIGHT

The final (optional) step in categorical data transformation is denormalization to a flat file. Table 7 and its respective code illustrate the methodology adopted that produces numeric fields representing frequencies of observed values; the %FLATTEN macro performs this transformation automatically and contains the following parameters:

- INTAB – the data set that contains categorical data to be flattened.
- OUTTAB – the resultant data set (i.e., flat file.)
- ID – the field by which data are related or grouped that will become the primary key in the flat file.
- VARLIST – the space-delimited list of all variables being flattened; applying the asterisk (*) immediately after a variable creates a wild card that indicates all fields containing the root concatenated with a number. In the above example, the field Chem is binned into three standardized, categorized fields Chem1, Chem2, and Chem3; in order to include all three fields in the VARLIST parameter, entering Chem* is sufficient. This could also be represented by conventionally specifying VARLIST=chem1 chem2 chem3.
- KEEPLIST – a space-delimited list of fields that correspond to the ID field and which will be retained in the flat file. In the above data set, RecNo 3 always corresponds to Helmand Province; thus, to retain the Province field in the flat file, Province should be indicated in the KEEPLIST parameter.

Thus, invocation of the %FLATTEN macro to denormalize the data set is:

```
%flatten(intab=chem_data_binned, outtab=chem_flat,id=RecNo,
    varlist=Chem*, keeplist=Province);
```

Table 10 illustrates the fields present in the flat file chem_flat after the %FLATTEN macro has performed the above denormalization:

Field	Description
recno	record number (primary key)
province	maintained because it was in the KEEPLIST parameter of the %FLATTEN macro
oxidizer	denormalized from field chem1
fuel	denormalized from field chem1
fertilizer_based	denormalized from field chem2
other_based	denormalized from field chem2
ammonium_nitrate	denormalized from field chem3
calcium_ammonium_nitrate	denormalized from field chem3
potassium_chlorate	denormalized from field chem3

aluminum	denormalized from field chem3
urea	denormalized from field chem3
sugar	denormalized from field chem3

Table 10. Fields Present in Flat File (chem_flat) produced by %FLATTEN macro

The above example first relied on the macro %BINMEPLEASE to implement a taxonomy in order to standardize and categorize data. Some categorical data, however, are of impeccable quality and do not require standardization; other data sets are categorical but not hierarchical, and thus require no categorization. In both of these instances, binning data through the %BINMEPLEASE macro may not be warranted; however, in each scenario, the %FLATTEN macro still can be applied directly to the raw data set to transform data. Regardless of whether %FLATTEN is implemented independently or after %BINMEPLEASE, the space-delimited list of categorical variables must contain values that do not exceed 30 characters. Care also should be exercised in establishing that all variables in the KEEPLIST parameter are 100 percent correlated to the ID variable; otherwise, a value at random could be chosen to represent the observation in the flat file that truly is not indicative of all observations having that ID in the raw data.

The macro %CONVERT is called by the macro %FLATTEN to transform text into SAS V7-acceptable variable names; spaces and other non-alphanumeric characters are transformed into underscore characters, with consecutive non-alphanumeric characters transformed into a single underscore character. For example, in the data set above, the value “ammonium nitrate” would be represented by the numeric field Ammonium_Nitrate in the flat file. Note that because all non-alphanumeric values resolve to underscores, multiple values can resolve to identical field names, which would cause logic errors. For example, other values that would resolve to the field Ammonium_Nitrate include “ammonium_nitrate”, “ammonium (nitrate)”, and “ammonium$*@!nitrate”. Thus, if data ultimately are intended to be denormalized into a flat file, this decision should be made before taxonomy creation so that appropriate, unique names are given to all categories and refined values.

The addition of hierarchical variables in the flat file provides analysts the flexibility to examine a problem from multiple perspectives, effectively zooming in or out to the desired level of specificity. All oxidizers are binned together in the Oxidizer field, but ammonium nitrate (a subset) also can be isolated by utilizing the Ammonium_Nitrate field, without the necessity to produce an additional data set. Care should be exercised, however, to ensure that values are not doubly counted; because Ammonium_Nitrate represents a subset of the Oxidizer category, the two fields should not be summed. To eliminate the potential for this confusion, the macro %FLATVARS (see Appendix E) can be invoked, which reads a taxonomy and returns a non-duplicative, space-delimited variable list of all fields at a specific hierarchical level.

The macro %FLATVARS is invoked with the following parameters:

- BINLIB – is the static SAS binning library (defaulted to BINNING) that is created.
- BINTAB – the binning (i.e., control) table data set name.
- VAR – the categorical variable that was denormalized; this should correspond to the <VAR> tag in the control table. In the current example, this would correspond to Chem.
- LEVEL – the hierarchical level for which a variable list is being created. In the current example, because the variable Chem has three levels indicated in the control table, valid levels for the LEVEL parameter would be 1, 2, or 3.

To produce the list of all refined values (i.e., the highest number for each category), invoke the following macro statement to produce the variable list “aluminum sugar urea ammonium_nitrate calcium_ammonium_nitrate potassium_chlorate”:

```
%flatten(binlib=binning, bintab=chem_abridged, var=chem, level=3)
```

To produce the field list for the lowest level of categorical field values, invoke the following macro statement to produce the variable list “aluminum sugar urea fertilizer_based other_based”:

```
%flatten(binlib=binning, bintab=chem_abridged, var=chem, level=2)
```

To produce the field list for the highest level of categorical field values, invoke the following macro statement to produce the variable list “fuel oxidizer”:

```
%flatten(binlib=binning, bintab=chem_abridged, var=chem, level=1)
```

The denormalization methodology demonstrated in this solution represents only one of a myriad of techniques that could be implemented to flatten data dynamically. With little effort expended to modify the logic in the %FLATTEN macro, code could be re-engineered to produce other flattened values that are concatenated, truncated, or otherwise transformed to answer additional analytic needs. One planned upgrade to the current solution will offer the ability to transform weighted categorical fields. For example, if an additional field in the data set (chem_data) were to include pounds of HME seized, this value could be incorporated into a flat file, but currently would have to utilize a separate process to perform the quantitative summation. This summation subsequently would need to be joined to the flat file in a separate process. A more integrated and elegant approach would recognize one or more numerical values as weights ascribed to a categorical field, and allow those numerical values to be summed during the automated denormalization process. Notwithstanding, the current methodology provides an adequate, automated solution that dynamically transforms large quantities of HME and other data.

CONCLUSION

Analysts sometimes eschew categorical data as an unwieldy burden. Even before the complexities of categorical data statistical analysis can be grappled, challenges to data clarity and conformity abound. In order to remove unwanted variation and noise, and especially in support of hierarchical data, an explicit taxonomy can facilitate a clear description and understanding of a comprehensive data model. Moreover, a taxonomical control table can prescribe data transformation—including standardization, categorization, and denormalization—through data-driven, macro-implemented processes. Through only the following two macro invocations, and without the necessity to maintain or modify static code, thousands of lines of code dynamically are generated, parsed, and efficiently perform complex data transformations:

```
%binmeplease (bintab=binning.chem_abridged, intab=chem_data,
    outtab=chem_data_binned, varlist=chem, lev=5, printfreq=yes);
%flatten(intab=chem_data_binned, outtab=chem_flat,id=RecNo,
    varlist=Chem*, keeplist=Province);
```

Most remarkable, although this continuing example transforms fictitious data utilizing an abridged taxonomy, identical code will bin and denormalize classified data utilizing the actual, unabridged HME precursor chemical taxonomy. The methodology implemented at JIEDDO, thus, represents a scalable, portable solution that can be applied to categorical data of diverse content, quantity, and quality, and that has standardized a once laborious process.

ACKNOWLEDGMENTS

Special thanks to the employees of CACI-Wexford and JIEDDO for their assistance, encouragement, and perseverance toward a noble mission.

CONTACT INFORMATION

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

Name: Troy Martin Hughes
E-mail: troymartinhughes@gmail.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

APPENDIX A. IMPORT_BINNING MACRO

```
* This macro imports a single binning schema to be used to standardize and categorize
data;
* It creates the table BINNING.TABLE to be used in later processes;
* An HTML report automatically is output that depicts the binning structure, and which
can be used to validate the schema;
* The schema is a text file located in the BINNING library (which must be assigned);
* The first line of the schema should contain <LIB> followed by the library to which it
conforms;
* The second line of the schema should contain <TAB> followed by the table to which it
conforms;
* The third line of the schema should contain <VAR> followed by the first variable
being analyzed;
* The fourth line of the schema should contain <1> followed by the highest order
categorical grouping for that variable;
* The fifth and following lines contain additional numbers (in brackets, as above),
additional variables, or values;
* Values do not contain brackets, but instead should be written flush against the left
margin;
* Categorical groups can contain spaces, capitalization, and alphanumeric and other
characters to improve readability;
* Categorical groups cannot contain percent signs or ampersands, or exceed 30
characters in length;
* Categorical groups cannot be repeated within a single variable or within a single
binning schema;
* Categorical groups should NOT be repeated even between separate schemas, if the
tables they represent are intended to be merged;
* Values can be repeated in a binning schema, if they occur under different variables;


%macro import_binning (intab= /* text file (including full path, file name, and .TXT
              extension) */,
       outtab= /* table name given to SAS binning table, typically identical to text
              file, but having no extension */,
       lev= /* number of hierarchical levels total that will be searched--5 or less
              is typical */);
* create the binning data set from the binning schema;
data temp;
       infile "&intab" dlm="@";
       input blah :$50.;
run;
data binning.&outtab (drop=blah maxlist varlist vars max);
       length lib tab var $40 varlist maxlist $1000 value $100 vars 8;
       array level {&lev} $40 lev1-lev&lev;
       set temp end=eof;
       order=_n_;
       if _n_=1 then do;
              vars=0;
              maxlist='';
              end;
       if upcase(substr(blah,1,5))='<LIB>' then lib=
              strip(substr(blah,6,length(blah)-5));
       else if upcase(substr(blah,1,5))='<TAB>' then tab=
              strip(substr(blah,6,length(blah)-5));
       else if upcase(substr(blah,1,5))='<VAR>' then do;
              vars+1;
```

```
            if vars>1 then maxlist=catx(' ',maxlist,strip(put(max,$10.))); *saves the
                  maximum number of levels encountered for the PREVIOUS variable;
            max=0;
            var=strip(substr(blah,6,length(blah)-5));
            varlist=catx(' ',varlist,var);
            end;
      %do i=1 %to &lev;
            else if upcase(substr(blah,1,3))="<&i>" then do;
                  if max<&i then max=&i;
                  lev&i=strip(substr(blah,4,length(blah)-3));
                  %do j=%eval(&i + 1) %to &lev;
                        lev&j='';
                        %end;
                        end;
                  %end;
      else do;
            value=blah;
            output;
            end;
      if eof then do;
            maxlist=catx(' ',maxlist,strip(put(max,$10.)));
            call symput('lib',strip(lib));
            call symput('tab',strip(tab));
            call symput('varlist',strip(varlist));
            call symput('maxlist',strip(maxlist));
            end;
      retain lib tab var max lib tab varlist maxlist lev1-lev&lev vars;
run;
* produce the HTML report that demonstrates the binning hierarchy;
ods listing close;
ods noproctitle;
ods html path="%sysfunc(pathname(binning))" file="&outtab..htm" style=sketch;
%let i=1;
%do %while(%length(%scan(&varlist,&i))>1);
      proc report data=binning.&outtab (keep=var value
            lev1-lev%scan(&maxlist,&i)) nocenter nowindows missing;
            where var="%scan(&varlist,&i)";
            title1 h=2 "field: %upcase(%scan(&varlist,&i))";
            title2 h=2 "for use with %upcase(&lib..&tab)";
            column lev1-lev%scan(&maxlist,&i) value;
            %do b=1 %to %scan(&maxlist,&i);
                  define lev&b / group "Level &b" order=data;
                  %end;
            define value / display "Value" order=internal;
      run;
      %let i=%eval(&i+1);
      %end;
ods html close;
ods listing;
run;
proc sort data=binning.&outtab;
      by var lev1-lev&lev;
run;
%mend;
```

APPENDIX B. IMPORT_BINNING_ALL MACRO

```
* This macro automates the import of all binning tables that are found in the BINNING
library;


%macro import_binning_all;
%let path=%sysfunc(pathname(binning));
filename indata pipe "dir %quote(&path) /a-d /b";
run;
data x;
       length tab $100;
       infile indata truncover;
       input tab $100.;
       if upcase(scan(tab,-1,.))='TXT';
       tab=substr(strip(tab),1,length(tab)-4);
run;
proc sql;
       select tab
       into :tablist separated by '*'
       from x;
       quit;
run;
%let k=1;
%do %while(%length(%scan(&tablist,&k,*))>1);
       %let tab=%scan(&tablist,&k,*);
       %import_binning(intab=&path.\&tab..txt, outtab=&tab, lev=5);
       %let k=%eval(&k+1);
       %end;
%mend;
```

APPENDIX C. BINMEPLEASE MACRO

```
* This macro ingests a binning schema and utilizes it to create a hierarchical table;
* The output table contains all original observations and all original variables;
* The output table additionally contains newly created standardized and categorized
variables;
* The new variables are incremented by one depending on their hierarchical level, and
follow the respective binning schema;
* Values not found in the schema will NOT be included in final output (as a newly
created variable);
* An optional contingency table can be printed that demonstrates the frequency of
observed values, as well as values outside schema definition;


%macro binmeplease (bintab= /* the SAS binning table used to parse the data */,
      intab= /* the SAS table that is being parsed, in LIBNAME.MEMNAME format */,
      outtab= /* the SAS table that is output, in LIBNAME.MEMNAME format */,
      varlist= /* space-delimited list of fields in INTAB that will be parsed (fields
            must also occur in BINTAB table) */,
      lev=5 /* maximum number of hierarchical levels to search, typically defaulted
            to 5 */,
      printfreq=yes /* YES to print a frequency report that demonstrates the number
            of observations within each level by group */);
%let i=1;
%do %while(%length(%scan(&varlist,&i))>1);
      %if &i=1 %then %let varlistc=%upcase("%scan(&varlist,&i)");
      %else %let varlistc=&varlistc,"%upcase(%scan(&varlist,&i))";
      %let i=%eval(&i+1);
      %end;
%let i=%eval(&i-1);
data _null_;
      set &bintab end=eof;
      by var lev1-lev&lev;
      where upcase(var) in(&varlistc);
      array binme {&i} $32000 bin1-bin&i;
      length maxlist $1000;
      if _n_=1 then i=0;
      if first.var then do;
            i+1;
            binme{i}= 'if ' || strip(var) || ' in ("' || strip(value) || '"';
            max=0;
            end;
      else if %do j=1 %to %eval(&lev-1);
                  first.lev&j or
            %end;
                  first.lev&j then do;
            binme{i}=strip(binme{i}) || 'else if ' || strip(var) || ' in
                  ("' || strip(value) || '"';
            end;
      else binme{i}=strip(binme{i}) || ',"' || strip(value) || '"';
      if %do k=1 %to %eval(&lev-1);
            last.lev&k or
            %end;
            last.lev&k then do;
                  binme{i}=strip(binme{i}) || ') then do;';
                  %do m=1 %to &lev;
                        if ^missing(lev&m) then do;
                              binme{i}=strip(binme{i}) || strip(var) || "&m='"
                                    || strip (lev&m) || "'; ";
                              if max<&m then max=&m;
```

```
                                    end;
                              %end;
                        binme{i}=strip(binme{i}) || 'end;';
                        call symput(var,strip(binme{i}));
                  end;
      if last.var then do;
            maxlist=strip(maxlist) || ' ' || strip(var) || '1-' || strip (var) ||
                  strip(put(max,$10.));
            call symput(strip(var)||'m',max);
            end;
      if eof then call symput('maxlist',strip(maxlist));
      retain i binme maxlist max;
run;
data &outtab;
      set &intab;
      length &maxlist $40;
%let i=1;
%do %while(%length(%scan(&varlist,&i))>1);
      %let var=%scan(&varlist,&i);
      &&&var;
      %let i=%eval(&i+1);
      %end;
run;

* optionally produce a frequency report that demonstrates frequency of observations for
each hierarchical level;
%if %upcase(&printfreq)=YES %then %do;
      %let i=1;
      ods listing close;
      ods noproctitle;
      ods html path="%sysfunc(pathname(binning))" file="&outtab._FREQ.htm"
            style=sketch headtext="<div align=left>";
      %do %while(%length(%scan(&varlist,&i))>1);
      %let var=%scan(&varlist,&i);
      %let lev=&&&var.m;
      proc freq data=&intab (keep=&var) noprint;
            tables &var/out=x;
      run;
      data x;
            set x;
            length valueU $100;
            valueU=upcase(&var);
      run;
      proc sort data=x;
            by valueU;
            where ^missing(valueU);
      run;
      data bin;
            set &bintab;
            length valueU $100;
            where upcase(var)="%upcase(&var)";
            valueU=upcase(value);
      proc sort data=bin out=bin2;
            by valueU;
      data xx;
            merge bin2 x;
            by valueU;
            if missing(order) then order=99999;
            if missing(lev1) then lev1='VALUE NOT IN SCHEMA';
            if missing(value) then value=valueU;
      run;
      proc sort data=xx;
            by lev1-lev&lev;
```

```
      run;
* compute frequency sums for all respective levels;
data
%do j=1 %to &lev;
      freq&j (keep=lev1-lev&j cnt&j)
      %end;;
      array lev {&lev} $40 lev1-lev&lev;
      set xx;
      by lev1-lev&lev;
      array cnt {&lev} 8 cnt1-cnt&lev;
      if missing(count) then count=0;
      %do k=1 %to &lev;
            if missing(lev&k) then cnt&k=.;
            else do;
                  if first.lev&k then do;
                        do l=&k to &lev;
                              cnt{l}=0;
                              end;
                        end;
                  cnt&k=cnt&k+count;
                  end;
            if last.lev&k then output freq&k;
            %end;
      retain cnt1-cnt&lev;
run;
data freqs;
      set freq1;
run;
%if %eval(&lev>1) %then %do;
      %do m=2 %to &lev;
            data freqs;
                  merge freqs freq&m;
                  by lev1-lev%eval(&m-1);
            run;
            %end;
      %end;
* rejoin frequencies to produce report;
proc sort data=xx;
      by lev1-lev&lev;
data rpt;
      merge xx(in=a) freqs(in=b);
      by lev1-lev&lev;
      if order<99999 then inschema=1;
            else inschema=2;
run;
proc report data=rpt nocenter nowindows missing;
      title1 h=2 "incidence of field: %upcase(&var)";
      title2 h=2 "in table: %upcase(&intab)";
      title3 h=2 "binning schema: %upcase(&bintab)";
      column inschema
            %do j=1 %to &&&var.m;
                  lev&j cnt&j
                  %end;
            order value count dummy;
      define inschema / group noprint '';
            %do j=1 %to &&&var.m;
                  define lev&j / group "Level &j";
                  define cnt&j / group "Count" format=comma15.;
                  %end;
      define order / display noprint '';
      define value / display "Value";
      define count / display "Count" format=comma15.;
      define dummy / computed noprint '';
```

```
	compute dummy;
		if order=99999 then do;
			call define (_row_, 'style','style=[backgroundcolor=light red]');
			end;
		endcomp;
	run;
	%let i=%eval(&i+1);
	%end;
ods html close;
ods listing;
%end;
%mend;
```

APPENDIX D. CONVERT AND FLATTEN MACROS

```
* The CONVERT macro converts a string (including spaces, alphanumeric, and other
characters) and creates a SAS V7-acceptable variable name;
* The macro fails if the string contains either a percent or ampersand, or if the
input string exceeds 32 characters;
* CONVERT is called by the FLATTEN macro, not called directly;

%macro convert(convertword= /* the word being transformed into a SAS V7 variable
      name */);
%if %sysfunc(nvalid(&convertword,v7)) %then %let newconvertword=&convertword;
%else %do;
      %if %eval(%sysfunc(anyalpha("&convertword"))^=2) %then %let
            convertword=_&convertword;
      %let convertword=%lowcase(&convertword);
      %let converti=1;
      %let convertvalue=1;
      %let newconvertword=;
      %do %while(&converti <= %length(&convertword));
            %let convertpos=%substr(%bquote(&convertword),&converti,1);
            %if ("&convertpos">="a" and "&convertpos"<="z") or ("&convertpos">="0"
                        and "&convertpos"<="9") or ("&convertpos"="_") %then %do;
                  %let newconvertword=
                        &newconvertword%substr(%bquote(&convertword),&converti,1);
                  %let convertvalue=1;
                  %end;
            %else %do;
                  %if &convertvalue=1 %then %let newconvertword=&newconvertword._;
                  %let convertvalue=0;
                  %end;
            %let converti=%eval(&converti+1);
            %end;
      %end;
&newconvertword
%mend;



* The FLATTEN macro inputs a table having a single key by which data are sorted and
grouped and creates numeric variables representing frequencies of categorical values;
* All fields representative of only a single observation should be excluded in output;
* All fields that are perfectly correlated to the group variable should be included in
the KEEPLIST parameter for output inclusion;
* The INTAB table can be the same OUTTAB table from the BINMEPLEASE macro, but does
not have to be;
* If the INTAB table contains fields that are incremented, an asterisk after the field
base will act as a wild card for inclusion;
* For example, if BINMEPLEASE produced the variables var1, var2, and var3, then VAR*
should be included in the VARLIST to reference all three variables, respectively;

%macro flatten (intab= /* table that is being flattened, in TABLE or LIBRARY.TABLE
            format */,
      outtab= /* table that is output as a flattened file, in TABLE or LIBRARY.TABLE
            format */,
      id= /* the unique ID by which the output data are being grouped/summarized */,
      varlist= /* the space-delimited list of categorical fields that are being
            transformed into numeric count variables */,
      keeplist= /* the list of additional fields that are also grouped by primary
            ID, thus they can be included in flat file */);
* determine if a wildcard has been used to represent all levels of a specified value;
```

```
%let i=1;
%let newvarlist=;
%do %while(%length(%scan(&varlist,&i,,S))>1);
	%let var=%scan(&varlist,&i,,S);
	%if "%substr(&var,%length(&var),1)"="*" %then %do;
		%let var=%substr(&var,1,%length(&var)-1);
		* determine the maximum number of levels that are utilized for the
			specific field;
		proc sql;
			select count(*)
			into :maxlev
			from dictionary.columns
			where memname="%upcase(&intab)" and
				upcase(substr(name,1,length(name)-1))="%upcase(&var)";
			quit;
		run;
		%let newvar=;
		%do j=1 %to &maxlev;
			%let newvar=&newvar &var&j;
			%end;
		%let var=&newvar;
		%end;
	%let newvarlist=&newvarlist &var;
	%let i=%eval(&i+1);
	%end;
%let varlist=&newvarlist;
* retrieve distinct lists of values for each variable;
%let i=1;
proc sql;
	%do %while(%length(%scan(&varlist,&i))>1);
		%let var=%scan(&varlist,&i);
		select distinct &var
		into :&var separated by "`"
		from &intab
		where not missing(&var);
		%let i=%eval(&i+1);
		%end;
	quit;
run;
* create variable list to be used in retain statement;
%let newvallist=;
%let i=1;
%do %while(%length(%scan(&varlist,&i))>1);
	%let var=%scan(&varlist,&i);
	%let j=1;
	%do %while(%length(%qscan(%bquote(&&&var),&j,`))>1);
		%let val=%qscan(%bquote(&&&var),&j,`);
		%let newval=%convert(convertword=&val);
		%let newvallist=&newvallist &newval;
		%let j=%eval(&j+1);
		%end;
	%let i=%eval(&i+1);
	%end;
* create transposed data set;
proc sort data=&intab;
	by &id;
run;
data &outtab (drop=&varlist);
	set &intab (keep=&id &varlist &keeplist);
	by &id;
	if first.&id then do; * initialize all counters at every new GROUP variable;
	%let i=1;
```

```
      %do %while(%length(%scan(&varlist,&i))>1);
            %let var=%scan(&varlist,&i);
            %let j=1;
            %do %while(%length(%qscan(%bquote(&&&var),&j,`))>1);
                  %let val=%qscan(%bquote(&&&var),&j,`); /
                  %let newval=%convert(convertword=&val);
                  %let j=%eval(&j+1);
                  &newval=0;
                  %end;
            %let i=%eval(&i+1);
            %end;
            end;
      %let i=1;
      %do %while(%length(%scan(&varlist,&i))>1); * increment values;
            %let var=%scan(&varlist,&i);
            %let j=1;
            %do %while(%length(%qscan(%bquote(&&&var),&j,`))>1);
                  %let val=%qscan(%bquote(&&&var),&j,`);
                  %let newval=%convert(convertword=&val);
                  %if &j=1 %then %do;
                        if &var="&val" then &newval=&newval+1;
                        %end;
                  %else %do;
                        else if &var="&val" then &newval+1;
                        %end;
                  %let j=%eval(&j+1);
                  %end;
            %let i=%eval(&i+1);
            %end;
      if last.&id then output;
      retain &newvallist;
run;
%mend;
```

APPENDIX E. FLATVARS MACRO

```
* macro FLATVARS produces a space-delimited list of fields that are produced when a
binning schema is transposed (i.e., flattened);
* for example, if the level 1 values of variable BATTERY included D CELL, C CELL, and
9 volt, FLATVARS would return "D_CELL C_CELL _9_volt";
* if a value (i.e., observation) does not contain data at a specific level, the value
from the next highest available value is selected;
* this ensures that no matter what level is selected, a sequence of fields will be
chosen that represents--but does not duplicate--every observation;
* the global macro FLATVARS is created as the list of numeric variables, and is
intended to be used as input for statistical functions or in KEEP or DROP statements;

%macro flatvars (binlib=BINNING /* binning schema library, which typically should be
              BINNING */,
       bintab= /* binning schema data set*/,
       var= /* variable name */,
       level= /* number of the hierarchy level that is desired */);
%global flatvars;
%let flatvars=;
* determine the maximum number of levels present;
proc sql;
       select count(*)
       into :maxlev
       from dictionary.columns
       where libname="%upcase(&binlib)" and memname="%upcase(&bintab)" and
              upcase(substr(name,1,3))="LEV";
       quit;
run;
* create space-delimited list of numeric variables in level;
%let maxlev=&maxlev;
%if %eval(&level > &maxlev) %then %goto err;
%let varlist=;
data _null_;
       array lev{&level} $100 lev1-lev&level;
       set &binlib..&bintab end=eof;
       by lib tab var lev1-lev&maxlev;
       where upcase(var)="%upcase(&var)";
       length temp $100 varlist $10000;
       if _n_=1 then varlist='';
       if first.lev&level and ^missing(lev&level) then do;
              temp=lev&level;
              end;
       else if missing(lev&level) then do;
              i=&level;
              do until(^missing(lev{i}) or i=0);
              i=i-1;
              end;
              %do i=1 %to &level;
                     if i^=0 then do;
                            if i=&i and first.lev&i then temp=lev&i;
                            end;
                     %end;
              end;
       if ^missing(temp) then varlist=catx('`',strip(varlist),strip(temp));
       if eof then call symput('vars',strip(varlist));
       retain varlist;
run;
%let i=1;
%do %while(%length(%qscan(%bquote(&vars),&i,`))>1);
```

```
      %let var=%qscan(%bquote(&vars),&i,`);
      %let flatvars=&flatvars %convert(convertword=&var);
      %let i=%eval(&i+1);
      %end;
%err:;
%mend;
```

REFERENCES

[1] Counter-Improvised Explosive Device Strategic Plan 2012 – 2016. JIEDDO. Retrieved from https://www.jieddo.mil/content/docs/20120116_JIEDDO_C-IEDStrategicPlan_web-MED.pdf.

[2] The Spreading Threat of Roadside Bombs and Other Explosive Devices, April 25, 2013. Retrieved from http://www.hstoday.us/briefings/correspondents-watch/single-article/the-spreading-threat-of-roadside-bombs-and-other-improvised-explosive-devices/cc19223abe4d54b1b1c2c6e8c94f88df.html.

[3] Statement by Lieutenant General Michael D. Barbero before the United States House of Representatives Committee on Appropriations Subcommittee on Defense. September 20, 2012. Retrieved from https://www.jieddo.mil/content/docs/20120920_JIEDDO_Statement_for_the_Record.pdf.

[4] Counterterrorism: U.S. Agencies Face Challenges Countering the Use of Improvised Explosive Devices in the Afghanistan/Pakistan Region, July 12, 2012. Retrieved from http://www.gao.gov/products/GAO-12-907T.

[5] Homemade Explosive (HME)/Bulk Explosive (BE) Recognition Guide, JIEDDO. Retrieved from https://www.jieddo.mil/content/docs/HMEGuide_Final_v3_HR_U.pdf.