PharmaSUG2011 - Paper PO12

Processing the RefSeq and CCDS Annotation Datasets Using the SAS System: Creation of Gene Reference

Kevin Viel, Saint Joseph's Translational Research Institute, Atlanta, GA
Shannon Grabich, Kennesaw State University, Kennesaw, GA

ABSTRACT

A source for the human genome data is the UCSC Genome Browser, in particular the RefSeq and CCDS annotation databases, which are indispensible to biomedical researchers. The goal of this paper is to assist SAS® programmers in their efforts to obtain a local copy of the reference human genome, the RefSeq file, CCDS file, and to use these to create SAS datasets of reference sequences via SAS data steps. Further, using tools available in BASE SAS, such as perl regular expressions (PRX), allows investigation of the DNA and protein sequences. The creation of gene reference sequence is an integral part of a suite of SAS programs that creates a genomics database. Finally, this paper describes a minor modification of the genome-wide peptide search initially described by Burroughs et alia.

INTRODUCTION

The reference human genome is ostensibly "complete"[1]. Nearly 30,000 proteins and their transcription variants are documented. These data are indispensible to biomedical researchers especially in this era of next generation sequencing; whole-genome sequencing data will affordably become routine components of medical records and are already essential to many research projects. A source for the approximately three billion base-pairs of chromosomal DNA sequence and annotation data for these proteins is the UCSC Genome Browser[2]. In particular the RefSeq[3] and CCDS[4] (consensus coding sequencing) databases provide annotated DNA sequence data of proteins.

The **goal** of this paper is to assist investigators in their efforts to obtain a local copy of reference human genome and both the RefSeq and CCDS base files and to use these files to create (gene) reference sequences SAS® datasets using SAS data steps. Since these files and their archived generational releases are publicly available, their use in research projects can create standardized reference sequence files that minimize confusion and error between groups of researchers. Further, using tools available in BASE SAS®[5], such as perl regular expressions (PRX), allows investigation of the DNA sequences that might not be easily performed using National Center for Biotechnology Institute (NCBI)[6] tools, such as BLAST[7]. The creation of gene reference sequence is an integral part of a suite of SAS programs that create a genomics database of DNA sequence from Ab1 files (Paper AD16 in these Proceedings). We encourage the reader to peruse the introduction of that paper for a short review of DNA and genetics. The creation of reference sequences allows the creation of reference amplicons and provides genic details of those loci, such as whether they are within the coding sequence and whether nucleotide substitutions might create non-synonymous single nucleotide polymorphism (ns-SNP). Finally, this paper describes a minor modification of the genome-wide peptide search initially described by Burroughs et alia[8] to illustrate that the SAS system can be an emerging tool in the field of bioinformatics.

GENOMIC DATA SETS

The users must obtain the files refGene.txt.gz[9], ccdsGene.txt.gz[10], and chromFa.zip[11]. They are not small. We have confined our attention to the "main" fasta files of the chromFa folder, that is, those with names like "chr16.fa". Fasta files are text files for DNA or RNA, with a line length of typically of 60 or 80 bytes and one header line prefixed by the symbol ">".

Table 1 describes the contents of the fasta files for Chromosomes 01-22, X, and Y. The nucleotide composition, which one can determine using the FREQ procedure, reveals that a sizeable portion of each chromosome is actually nucleotides of unknown identity (N). These N may in very large contiguous blocks, some of which start (PADDING_N) the sequence.

We convert these fasta files to SAS data sets in which each base constitutes an observation and the sole variable. Each base can be "indexed" by its observation number, i.e. using the POINT= option to the SET statement. One alternative would otherwise require a computation of observation number and position

Table 1. Details of the chromosomes contained in chromFa.zip

Chromosome	A	C	G	T	N	Padding_N	Largest_N_Repeat	Size
01	26.3	18.9	18.9	26.3	9.6	10,000	21,050,000	249,250,621
02	29.2	19.7	19.7	29.3	2.1	10,000	3,000,000	243,199,373
03	29.6	19.5	19.5	29.7	1.6	60,000	3,000,000	198,022,430
04	30.3	18.8	18.8	30.3	1.8	10,000	3,000,000	191,154,276
05	29.7	19.4	19.4	29.7	1.8	10,000	3,000,000	180,915,260
06	29.5	19.4	19.4	29.5	2.2	60,000	3,100,000	171,115,067
07	28.9	19.9	19.9	28.9	2.4	10,000	3,000,000	159,138,663
08	29.2	19.6	19.6	29.2	2.4	10,000	3,000,000	146,364,022
09	25.0	17.6	17.6	25.0	14.9	10,000	18,150,000	141,213,431
10	28.3	20.1	20.1	28.3	3.1	60,000	3,200,000	135,534,747
11	28.4	20.2	20.2	28.4	2.9	60,000	3,100,000	135,006,516
12	28.8	19.9	19.9	28.9	2.5	60,000	3,000,000	133,851,895
13	25.5	16.0	16.0	25.5	17.0	19,020,000	19,020,000	115,169,878
14	24.2	16.8	16.8	24.4	17.8	19,000,000	19,000,000	107,349,540
15	23.0	16.8	16.8	23.0	20.3	20,000,000	20,000,000	102,531,392
16	24.0	19.5	19.6	24.2	12.7	60,000	11,100,000	90,354,753
17	26.1	21.8	21.8	26.1	4.2	0	3,000,000	81,195,210
18	28.8	19.0	19.0	28.8	4.4	10,000	3,100,000	78,077,248
19	24.3	22.8	22.9	24.4	5.6	60,000	3,100,000	59,128,983
20	26.2	20.8	20.9	26.5	5.6	60,000	3,100,000	63,025,520
21	21.7	14.9	14.9	21.5	27.1	9,411,193	9,411,193	48,129,895
22	17.7	16.3	16.3	17.6	32.0	16,050,000	16,050,000	51,304,566
X	29.4	19.2	19.2	29.5	2.7	60,000	3,100,000	155,270,560
Y	12.9	8.6	8.7	13.0	56.8	10,000	30,000,000	59,373,566

within the line, but the use of the POINT= option would then require computations, involving, for instance, the MOD() function. We did not find this to be an appealing advantage.

These fasta files can be read with the following data step, which can be incorporated into a macro:

```
Data hg19.Chr&Chr. ( PW   = &PW.
                     Drop = __Line
                   ) ;
  Length Base $ 1 ;
  Infile "~\hg19\chromFa\chr&Chr..fa"
         FirstObs = 2
         Length   = Len
         ;
  Input __Line $Varying256. Len ;
  Do _n_ = 1 to Len ;
    Base = Substr( __Line , _n_ , 1 ) ;
    Output ;
  End ;
Run ;
```

The “&Chr.” that appears in the two-level name of the DATA statement is a macro variable. The authors also keep the password in a macro variable, but it is not intended as a security measure. Note that the authors end the macro variables with a period, which is not required, unless the variable to be resolve is “embedded” in a greater string. The fasta (fa) file indicated in the INFILE statement illustrates just such a need. The first period is needed to delineate the macro variable name from the second component, which happens to be a period in this case.

Each line of the fasta file is INPUT (read) using the $VARYINGd. informat. Use of this format requires that the length of the line is provided, hence the LENGTH= options to then INFILE statement. Each base of the line is then abstracted (SUBSTR) and output as an observation.

The refGene.txt and ccdsGene.txt contain data concerning genes. We chose to use the refGene.txt file because it contains data on the non-coding portions of the genes, too. "Genes" may be a variety of loci, some predicted and others may be transcript variant. Whereas some genes are well documented, others in the file may be subject to change. The following data step will read the refGene.txt file, but the ccdsGene.txt file has a similar structure:

```
Data hg19.refGene_Feb2009 ( PW = &PW. ) ;
  Infile " \UCSC Genome Browser\hg19\refGene.txt"
  DSD Delimiter = "09"x LRecL     = 5000 FirstObs  = 2 ;
  Length bin_num            8 /* Indexing field to speed chromosome range queries */
         name           $   15 /* Name of gene (usually transcript_id from GTF)    */
         chrom          $    5 /* Reference sequence chromosome or scaffold        */
         strand         $    1 /* + or - for strand                                */
         txStart            8 /* Transcription start position                     */
         txEnd              8 /* Transcription end position                       */
         cdsStart           8 /* Coding region start                              */
         cdsEnd             8 /* Coding region end                                */
         exonCount          8 /* Number of exons                                  */
         exonStarts    $ 2000 /* Exon start positions                             */
         exonEnds      $ 2000 /* Exon end positions                              */
         score              8 /* No Description provided by UCSC                 */
         name2          $    8 /* Alternate name (e.g. gene_id from GTF)           */
         cdsStartStat   $    8 /* enum('none','unk','incmpl','cmpl')               */
         cdsEndStat     $    8 /* enum('none','unk','incmpl','cmpl')               */
         exonFrames     $  500 /* Exon frame s, or -1 if no frame for exon   */
         ;
  Input  bin_num name chrom strand txStart txEnd cdsStart cdsEnd exonCount
         exonStarts exonEnds score name2 cdsStartStat cdsEndStat exonFrames ;
Run ;
```

The UCSC Genome Browser is an excellent resource and provides a schema for the tables it provides (http://genome.ucsc.edu/cgi-bin/hgTables). **Table 2** displays an observation from the hg19.refGene_Feb2009 data set. The libname hg19 indicates that these data pertain to February 2009 Human GRCh37 assembly. It is essential to use the corresponding files for each assembly to obtain valid results. Thankfully, the structure of the reference sequences generated by the macro described in this paper makes it easy to compare the amino acid sequence to that of a gene of interest to provide the user some assurance.

Table 2. The observation in hg19.refGene_Feb2009 data set corresponding to NM 153763.

bin_num	name	chrom	strand	txStart	txEnd	cdsStart	cdsEnd
178	NM_153763	chr1	+	110754064	110766704	110754121	110766656
exonCount	**exonStarts**	**exonEnds**	**score**	**name2**	**cdsStartStat**	**cdsEndStat**	**exonFrames**
2	110754064, 110765585,	110754799, 110766704,	0	KCNC4	cmpl	cmpl	0,0,

A few features of these data needed for the generated of the reference sequence are worth emphasizing. First, the chromosome (CHROM) on which the gene is located will determine which reference chromosome file we must use to obtain the reference sequence for this gene. The second important variable is STRAND. As described in AD16 of these Proceedings, DNA is double stranded, but the strands are reverse complimentary. Genes can be on one strand or the other; the "-" strand indicates that the reverse compliment of the bases present in the chromosome reference file must be used. This means, that the starts and ends, for instance, TXSTART, must also be "reversed". The "start" in one strand is the reverse of the "end" in another). For instance, consider the sequence AGC, the reverse compliment of which is TCG. A is the first base of the sequence, but for a gene on the minus-strand, the base corresponding to A is the last base of reverse compliment, T. Written as a sequence of letters to represent nucleotides, that appears incorrect, however, the nucleotides on a strand on covalently bonded and we speak, by convention, of DNA in the 5'-3' orientation. To emphasize this, we sometimes write 5'-AGC-3', the reverse compliment of which is 5'-CGT-3'. One might sometime refer to transcription as occurring telomeric-to-centrometric or vice versa to describe this concept, but "+" and "-" are more economical as values in a database. Note that the exon starts and ends are provided as a comma-delimited list that pairs by relative position in the lists. If

this gene were on the minus-strand, the first base of Exon 01 would be 110766704, the "last" base of the second (final) exon, instead of 110754064, the first base of the first exon. Finally, note the transcription starts (TxStart) does not necessary correspond to translations start (cdsStart, or coding sequence Start). Further, as stated, on the minus-strand TxEnd is the first base of the mRNA (transcription) product. By convention, we numbered the first base that is transcribed as +1 (transcription), but the base immediate 5' to it (the last base of the promoter) is numbered -1; nucleotide number 0 does not exist, which requires special care, especially since we index this data set by NN (nucleotide number). Lastly, the UCSC Genome Browser employs a zero index, instead of a one, in reference to the sequences (discussed here http://genome.ucsc.edu/FAQ/FAQtracks.html#tracks1).

OVERVIEW OF THE MACRO

The macro, GENE_ANNOTATE, consists of a SQL procedure and a data step. It requires the two SAS datasets described above (hg19.refGene_Feb2009 and one of the chromosome files, for instance hg19.ChrX). The SQL procedure simply determines and returns the chromosome on which the gene is located as a macro variable. The data set uses the information in hg19.refGene_Feb2009 (Table 2) to create a reference sequence file of the gene by reading the corresponding bases from the reference chromosome file. The gene sequence is flanked by 50,000 base-pairs (bp) of the 5' and 3' genomic DNA. The sequence is annotated and, if the base is in an exon, it is capitalized. Note, however, that not all exonic bases contribute to the coding sequence, for instance, the 3' UTR (Untranslated Region).

EXPLANATION OF SAS CODE

The MACRO statement that indicates the start of the macro code and names the macro uses **keyword parameters** to give the user flexibility (Lines 1-3). When the user calls the macro, he or she must provide values for the parameters, GENE and NM. Whereas GENE can be any SAS name the user desires, NM is the RefSeq accession number. The gene name is a good choice for GENE. For instance, the gene that encodes coagulation Factor VIII, the protein that is absent or deficient in the bleeding disorder Hemophilia A, is *F8 (*NM_000132, note that by convention, gene names are italicized).

Two sections of code (Line 5 and Lines 6-11) appear multiple times throughout the macro. Whereas these lines appeared as "wall-paper" in the original macro, for economy of space the authors replaced them with macro variables. The first macro variable CM (for call missing) is an entire SAS statement consisting of the call routine CALL MISSING, which sets the values of the variables CODONIC_ NUCLEOTIDE, CDS_NUCLEOTIDE, CODON, and EXONIC_NUCLEOTIDE to missing. Note that the semi-colon ending the statement is also part of the macro variable value, but that we used to %STR() function to mask it. Otherwise, it would be taken to end the macro assignment statement and the second semi-colon would become SAS code written by the macro (a null statement). The second macro variable COMP_BASE consists of two functions and their arguments. The entire value is actually an argument to a function at various points in the data step. Note the distinction between LINE and STATEMENT; a SAS statement may run over several lines. Both of these macro variables are LOCAL to this macro.

The SQL procedures (Lines 13-18) is another way to assign a value to a macro variable (CHROM). This is accomplished by using INTO with the macro variable prefixed by a colon. Usually, if multiple values exist, then you can delimit them using "SEPARATED BY", but in this case, it effectively trims the value. Care must be taken when assigning the keyword parameter NM, since we use the EQT (EQual Trim, which is the SQL equivalent to "=:"). Our intention was to avoid issue with generational accession numbers, for instance NM_000132.1, but one may have a devastating problem if one provides an accession number that is too short, for instance, NM_00013 (note, for instance, that F9 has the accession number NM_000133). We also allow flexibility with the password (PW), which is a GLOBAL variable. However, the intention is not security.

The DATA statement (LINES 20-24) begins the data step, sets the data set password, indicates that the only variable to appear in the output data set are those listed in the KEEP= data set option (Lines 21-22), and finally creates a simple index that must have unique values in the output data set. The resulting data set is a permanent data set in the SEQ library. The user is free to change this or even to make the first level of the name into a macro variable.

The LENGTH statement (Lines 26-29) represents the first variables encountered during compilation and thus they appear as the first three variables in the data set. NN (Nucleotide Number) is numeric and has a length of 8. Eight is the default length of numeric variables, but this assures that NN is the first variable of the dataset. BASE is the nucleotide read from the reference chromosome dataset. LOCATION is a description of the genic (pertaining to the gene) nature of the base, for instance "Exon 01". As the data step loops, it reset variables in its program data vector to missing by default. The RETAIN overrides this action for NN and sets its value to -50001. This is intentional even though the first NN of the data set will be -50000.

The SET statement in Lines 33-35 reads the observation pertaining to the RefSeq accession of interest (Table 2). The values of the variables it provides drive the remaining statements. Base on the value

of STRAND, either Lines 37-171 or Lines 174-331 are executed. The authors avoided using ELSE IF for two reasons: the computational require of two exclusive IF statements is negligible and indenting made the code more difficult to read on a smaller screen. In either set of IF-THEN-DO-END statements, the approach is the same, but must account for the reverse compliment for the minus-strand.

First it acquires the sequence for the promoter (NN = -50000 to NN = -1). Then it obtains the sequence for the gene. Then it acquires the sequence for the 3' genomic DNA. The DO-Loop on Lines 41-48 cycles through the promoter if the gene is on the plus-strand. Note that START, which is computed as 50000 bp 5' to TXSTART is offset by +1 as described above. Consider a number example: TXSTART = 50000 (and the first base of transcript would be 50001 since it is zero-indexed). This would mean that __POINT cycles from 50001-50000=1 to 50000. Note that the difference between 50000 and 1 is 49,999 but the number of bases is 50000 (50000-1+1). The variable P (Line 42) is the absolute position of the base in the reference chromosome sequence. It is the same as __POINT, but since __POINT is used in the POINT= option to the SET statement, it is not allowed to be in the output data set. Each time a base is added to the reference gene data set, we increment the value of NN, which is the relative position of the base in the reference gene sequence. Two separated reference gene sequences may contain the same bases, but their NN may be different, whereas P will be constant, allowing us to later determine that these bases are, indeed, the same. To indicate that these bases are not exonic, we convert them to lower case (Line 45). Finally, we designate this region as the "Promoter", but it could also be called "5' genomic".

After we have acquired the promoter, the data step will now be within the region designated as the gene, that is beyond TXSTART (or TXEND, for minus-strand genes). The first and last exons are handled differently from the others (Lines 54-76 and 134-159) simply because these may have non-coding bases. At this point, the last value for NN was -1. Line 54 sets it to 0, since it will be incremented in the loop (Line 58) and no further checks are needed with this construct.

The start and end limits for the loops are taken from the pairs of values in EXONSTARTS and EXONENDS. Notice, however, that we account for the zero-index by adding one where appropriate (Line 51). Since these bases are now exonic bases, we capitalize them (Line 59) and begin a tally of exonic nucleotides using the temporary variable __EXONIC_NUCLEOTIDE. This allows us to set the values of EXONIC_NUCLEOTIDE and variable in the output data set to missing when the location is not exonic (intronic or genomic DNA). Until the loop reaches the CDSSTART, no other annotation or enumeration occurs and the location remains "5' UTR" (Line 62, beyond the transcription start site but before the translation start site). After this point, we begin a rolling tally of the coding sequence nucleotides and the codons (Lines 63-72). We should not that the location is set to Exon 01 (line 65), but some genes actually have translation start points after Exon 01 and this macro should not be used to generate their reference sequences without modifications. Codons are a triplet of bases, so when the value of __CODONIC_NUCLEOTIDE reaches four, then it is reset to one (Line 68). The permanent variables are assigned their values from their corresponding temporary variables (Lines 69-71). CODON is synonymous with Amino Acid number. Instead of counting the number of codons output, we calculate it from the CDS_NUCLEOTIDE (Line 71). The CEIL() function returns the value of the next greatest integer for decimal numbers or the integer itself. Finally, regardless of location, if the base is within a pair of EXONSTARTS and EXONENDS, we enumerate it (Line 73). Since we are still within Exon 01, the value of INTRONIC_NUCLEOTIDE is missing (Line 74). This variable is important because it allows us to determine how close to an exon-intron junction a base may be, an essential piece of information when studying splice-sites and variants that may affect them. By having a running tally (enumeration) of the bases specific to a region, we can also report the sizes of them without counting them, but rather by just determining the maximum values of those tallies.

After the first exon, we can enter a Do-Loop (Lines 78-115) that cycles through the second through penultimate exons (pairs of EXONSTARTS-EXONENDS values). First, however, the INTRON_START value is assigned of the base immediate 3' to the last value of EXON_END (Line 77). First the bases corresponding to the intron are abstracted, followed by those for the next exon. At this point, it may be obvious that a codon can split between two exons. One part of the codon can be literally 20,000 bp from the other part. This is why we keep a running total of the number of bases in a codon in a temporary variable.

The bases abstracted from the Do-Loop on Lines 78-115 were for the penultimate exon. The next bases will be for the last intron, the limits of which are assigned on Lines 114 and 119. Those for the last exon are assigned on Lines 117 and 118. The potential presence of a 3' UTR distinguishes the last exon. If the position of the base is beyond CDSEND by less than TXEND, then these bases are no longer coding bases, although they are still exonic. This situation is handled in Lines 150-159. After the last base of the last exon is abstracted, we abstract 50000 bp of the flanking 3' genomic DNA (Lines 161-170).

If the gene is on the minus-strand, then the process is similar to what has been described above, but the NN will be in the opposite direction than P. For instance, when P increases, NN will decrease. The sequence of abstraction remains the same: Promoter, Exon 01, Intron 01, … , Exon N, 3'UTR, 3' Genomic. This is accomplished in Lines 174-331.

GENOME-WIDE PEPTIDE SCAN

Using the refGene_Feb2009 and Chr__ datasets, one can obtain the amino acid sequences of the known (documented) proteins of the human genome. Indeed, by obtaining the related data sets for other species such as the mouse, one has quite a bioinformatics arsenal at hand. Burroughs *et alia* describe a peptide scan of the genome that is possible to repeat using the datasets created in this paper. Burroughs *et alia* generated all of the overlapping nonamer (peptides of length 9) and queried them for matches. To do this, one would need only to cycle through the exonic bases of the genes and translate the codons to their corresponding amino acids. For a peptide length of interest, one then can obtain the list of overlapping peptides for a gene. That list is not insignificant. For a peptide length of nine, one will generate approximately 14 million nonamers, approximately 12 million of which are distinct.

The definition of distinct depends on one's purpose. From an immunologic perspective, and exact match may not be necessary, but the degree of mismatch that might be "tolerated" before two peptides might generate different immune responses is not well known. For that matter, the lengths of peptides that are important are only bounded with regards to MHC Class I complexes since their grooves are closed and thought to bind peptides no longer than nonamers. In this respect, if one uses these peptides to generate regular expressions that allow a one base mismatch between the pattern and the target, one may approach the matching a bit more robustly. For example, consider the Factor VIII peptide MQIELSTCF, one of the nine regular expression patterns that allow a one base mismatch might be /MQI.LSTCF/. The SAS coding technique in AD16 demonstrate one way to create such regular expression. The author has done this for a variety of proteins of interest and has programs available. Indeed, the next important phase of this area of research will be to obtain the peptides from the Human Microbiome Project (http://commonfund.nih.gov/hmp/) and repeat the scans-and endeavor that will require immense computation resources, but may be just as immense rewarding.

CONCLUSION

Understanding the genomic data set publicly is an essential tool in biomedical research. This paper briefly describes three types of data sets and the resources to obtain local copies. This paper describes a macro that creates annotated reference sequence files. This files by themselves provide a rich data set for the study of genes, but when used in conjunction with other programs in a suite of programs written by one of the authors (KRV), they become a foundation to a genomic database (paper AD16 of these Proceedings). These genomics data are at the heart of the revolution of personalized medicine and pharmacogenetics. Being adept at using them will be a necessity.

REFERENCES

1. National Human Genome Research Institute (http://www.genome.gov/11006943)
2. Fujita et alia. The UCSC Genome Browser database: update 2011. Nucleic Acids Res. 2010 Oct 18. http://genome.ucsc.edu/
3. http://www.ncbi.nlm.nih.gov/RefSeq/
4. http://www.ncbi.nlm.nih.gov/projects/CCDS/CcdsBrowse.cgi
5. SAS® 9.1.3 Language Reference: Dictionary, Fifth Edition. Copyright © 2002–2006, SAS Institute Inc., Cary, NC, USA. All rights reserved.
6. http://www.ncbi.nlm.nih.gov/
7. http://blast.ncbi.nlm.nih.gov/blast_overview.shtml
8. Burroughs et alia. "Discriminating self from nonself with short peptides from large proteomes." Immunogenetics. 2004 Aug;56(5):311-20.
9. http://hgdownload.cse.ucsc.edu/goldenPath/hg19/database/refGene.txt.gz
10. http://hgdownload.cse.ucsc.edu/goldenPath/hg19/database/ccdsGene.txt.gz
11. http://hgdownload.cse.ucsc.edu/goldenPath/hg18/bigZips/chromFa.zip

ACKNOWLEGEMENTS
This work was supported in part by the following grants: NIH-5RC2HL101851-02, NIH-5R01HL072533-04, and NIH-5K08HL071130-05.

Contact Information

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

Name: Kevin Viel
Enterprise: Saint Joseph's Translational Research Institute
Address: 5671 Peach-Dunwoody Road, NE Suite 330
City, State ZIP: Atlanta, GA 30342
Work Phone: 678-843-6076
Fax: 678-843-6153
E-mail: kviel@sjha.org, genepistat@gmail.com
Web: www.sjtri.org

```
%Macro Gene_Annotate ( Gene =
                     , NM   =
                     ) ;

  %Let CM = Call Missing(Codonic_nucleotide,CDS_nucleotide,Codon,Exonic_nucleotide) %Str(;);
  %Let Comp_Base = Translate( Upcase( Base )
                            , "a" , "T"
                            , "c" , "G"
                            , "g" , "C"
                            , "t" , "A"
                            ) ;

  Proc SQL NoPrint ;
    Select Chrom Into : Chrom Separated By ""
    From hg19.RefGene_Feb2009  ( PW = &PW. )
    Where Name eqt "&NM."
    ;
  Quit ;

  Data Seq.&Gene. ( PW    = &PW.
                    Keep  = NN Base Location P Exonic_nucleotide Intronic_nucleotide
                            Codonic_nucleotide CDS_nucleotide Codon
                    Index = ( NN / Unique )
                  ) ;

    Length NN                    8
           Base                $  1
           Location            $ 20
           ;

    Retain NN -50001 ;

    Set hg19.RefGene_Feb2009 ( PW = &PW.
                               Where = ( Name = "&NM." )
                             ) ;

    If Strand = "+"
    Then
      Do ;
        Start = TxStart - 50000 ;           /* Promoter */
        Do __Point = Start + 1 To TxStart  ; /* TxStart has offset of 0 */
          P = __Point ;
          Set hg19.&Chrom. ( PW = &PW. ) point = __Point ;
          NN + 1 ;
          Base = LowCase( Base ) ;
          Location = "Promoter" ;
          Output ;
        End ;
        /* Exons and Introns         */
        /* Add the offset of one (1) */
        Exon_Starts = Input( Scan( exonStarts , 1 , "," ) , 15. ) + 1 ;
        Exon_Ends   = Input( Scan( exonEnds   , 1 , "," ) , 15. )     ;
        /* Exon 01 */
        NN = 0 ;
        Do __Point = Exon_Starts To Exon_Ends ;
          P = __Point ;
          Set hg19.&Chrom. ( PW = &PW. ) point = __Point ;
          NN + 1 ;
          Base     = UpCase( Base ) ;
          __Exonic_nucleotide  + 1 ;
          /* 5' UTR */
          If __Point <= CDSStart Then Location = "5' UTR" ;
           Else
             Do ;
               Location = "Exon 01" ;
               __CDS_nucleotide     + 1 ;
               __Codonic_nucleotide + 1 ;
               If __Codonic_nucleotide = 4 Then __Codonic_nucleotide = 1 ;
               Codonic_nucleotide  = __Codonic_nucleotide       ;
               CDS_nucleotide      = __CDS_nucleotide           ;
               Codon               = Ceil( CDS_nucleotide / 3 ) ;
             End ;
           Exonic_nucleotide   = __Exonic_nucleotide        ;
           Intronic_nucleotide = .                          ;
           Output ;
        End ;
        Intron_Starts = Exon_Ends + 1 ;
        Do Exon = 2 To CountC( exonStarts  , "," ) - 1 ;
          Exon_Starts = Input( Scan( exonStarts , Exon , "," ) , 15. ) + 1 ;
          Exon_Ends   = Input( Scan( exonEnds   , Exon , "," ) , 15. )     ;
          Intron_Ends = Exon_Starts - 1 ;
          /* Previous Intron */
          __Intronic_nucleotide = 0 ;
          Do __Point = Intron_Starts To Intron_Ends ;
            P = __Point ;
            Set hg19.&Chrom. ( PW = &PW. ) point = __Point ;
            NN + 1 ;
            Base     = LowCase( Base ) ;
            Location = "Intron " || Put( Exon , Z2. ) ;
            &CM.
            __Intronic_nucleotide + 1 ;
            Intronic_nucleotide = __Intronic_nucleotide ;
            Output ;
          End ;
          /* Exon */
          __Exonic_nucleotide = 0 ;
          Do __Point = Exon_Starts To Exon_Ends ;
            P = __Point ;
            Set hg19.&Chrom. ( PW = &PW. ) point = __Point ;
            NN + 1 ;
            Base     = UpCase( Base ) ;
            Location = "Exon " || Put( Exon , Z2. ) ;
            __Exonic_nucleotide  + 1 ;
            __CDS_nucleotide     + 1 ;
            __Codonic_nucleotide + 1 ;
            If __Codonic_nucleotide = 4 Then __Codonic_nucleotide = 1 ;
            Codonic_nucleotide  = __Codonic_nucleotide       ;
            CDS_nucleotide      = __CDS_nucleotide           ;
            Codon               = Ceil( CDS_nucleotide / 3 ) ;
            Exonic_nucleotide   = __Exonic_nucleotide        ;
            Intronic_nucleotide = .                          ;
            Output ;
          End ;
          Intron_Starts = Exon_Ends + 1 ;
        End ; /* Cycled through second to last exonStarts-exonEnds pairs */
        /* Last intron and exon */
        Exon_Starts = Input(Scan(exonStarts, CountC(exonStarts , "," ), "," ) , 15. ) + 1 ;
        Exon_Ends   = Input(Scan(exonEnds  , CountC(exonStarts , "," ), "," ) , 15. )     ;
        Intron_Ends = Exon_Starts - 1 ;
        /* Last Intron */
        __Intronic_nucleotide = 0 ;
        Do __Point = Intron_Starts To Intron_Ends ;
          P = __Point ;
          Set hg19.&Chrom. ( PW = &PW. ) point = __Point ;
          NN + 1 ;
          Base     = LowCase( Base ) ;
          Location = "Intron " || Put( Exon , Z2. ) ;
          &CM.
          __Intronic_nucleotide + 1 ;
          Intronic_nucleotide = __Intronic_nucleotide ;
          Output ;
        End ;
        /* Last Exon */
        __Exonic_nucleotide = 0 ;
        Do __Point = Exon_Starts To Exon_Ends ;
          P = __Point ;
          Set hg19.&Chrom. ( PW = &PW. ) point = __Point ;
          NN + 1 ;
          Base     = UpCase( Base ) ;
          Location = "Exon " || Put( CountC( exonStarts  , "," ) , Z2. ) ;
          __Exonic_nucleotide  + 1 ;
          __CDS_nucleotide     + 1 ;
          __Codonic_nucleotide + 1 ;
          If __Codonic_nucleotide = 4 Then __Codonic_nucleotide = 1 ;
          Codonic_nucleotide  = __Codonic_nucleotide       ;
          CDS_nucleotide      = __CDS_nucleotide           ;
          Codon               = Ceil( CDS_nucleotide / 3 ) ;
          Exonic_nucleotide   = __Exonic_nucleotide        ;
          Intronic_nucleotide = .                          ;
          Output ;
        End ;
        /* 3' Genomic */
        Do __Point = TxEnd + 1 To TxEnd + 49999 ;
          P = __Point ;
          Set hg19.&Chrom. ( PW = &PW. ) point = __Point ;
          NN + 1 ;
          Base     = LowCase( Base ) ;
          Location = "3' Genomic" ;
          &CM.
          Intronic_nucleotide = . ;
          Output ;
        End ; /* Cycled through __Points */
      End ; /* Strand = + */

    /*************************************************************************/
    If Strand = "-"
    Then
      Do ;
         Start = TxEnd + 50000 ;                /* Promoter           */
         Do __Point = Start To TxEnd + 1 By -1 ; /* Reverse Orientation */
           P = __Point ;
           Set hg19.&Chrom. ( PW = &PW. ) point = __Point ;
           /* Compliment */
           Base = LowCase( Translate( Upcase( Base )
                                    , "a" , "T"
                                    , "c" , "G"
                                    , "g" , "C"
                                    , "t" , "A"
                                    )
                         ) ;
           NN + 1 ;
           Location = "Promoter" ;
           Output ;
         End ;
        /* Exons and Introns */
        /* Add the offset of one (1) */
        Exon_Starts = Input( Scan( exonStarts , -2 , "," ) , 15. ) + 1 ;
        Exon_Ends   = Input( Scan( exonEnds   , -2 , "," ) , 15. )     ;
        Intron_Ends = Exon_Starts - 1 ;
        /* Exon 01 */
        NN = 0 ;
        Do __Point = Exon_Ends To Exon_Starts By -1 ;
          P = __Point ;
          Set hg19.&Chrom. ( PW = &PW. ) point = __Point ;
          /* Compliment */
          Base = Upcase( &Comp_Base. ) ;
          NN + 1 ;
          __Exonic_nucleotide  + 1 ;
          /* 5' UTR */
          If __Point > CDSEnd Then Location = "5' UTR" ;
           Else
             Do ;
               Location = "Exon 01" ;
               __CDS_nucleotide     + 1 ;
               __Codonic_nucleotide + 1 ;
               If __Codonic_nucleotide = 4 Then __Codonic_nucleotide = 1 ;
               Codonic_nucleotide  = __Codonic_nucleotide       ;
               CDS_nucleotide      = __CDS_nucleotide           ;
               Codon               = Ceil( CDS_nucleotide / 3 ) ;
             End ;
           Exonic_nucleotide   = __Exonic_nucleotide        ;
           Intronic_nucleotide = .                          ;
           Output ;
        End ;
        Do __Exon = CountC( exonStarts  , "," ) - 1 To 2 By -1  ;
          Exon_Starts = Input( Scan( exonStarts , __Exon , "," ) , 15. ) + 1 ;
          Exon_Ends   = Input( Scan( exonEnds   , __Exon , "," ) , 15. )     ;
          Exon = CountC( exonStarts  , "," ) - __Exon + 1 ;
          Intron_Starts = Exon_Ends + 1 ;
          /* Previous Intron */
          __Intronic_nucleotide = 0 ;
          Do __Point = Intron_Ends To Intron_Starts By -1 ;
            P = __Point ;
            Set hg19.&Chrom. ( PW = &PW. ) point = __Point ;
            /* Compliment */
            Base = Lowcase( &Comp_Base. ) ;
            NN + 1 ;
            Base     = LowCase( Base ) ;
            Location = "Intron " || Put( Exon - 1 , Z2. ) ;
            &CM.
            __Intronic_nucleotide + 1 ;
            Intronic_nucleotide = __Intronic_nucleotide ;
            Output ;
          End ;
          /* Exon */
          __Exonic_nucleotide = 0 ;
          Do __Point = Exon_Ends To Exon_Starts By -1 ;
            P = __Point ;
            Set hg19.&Chrom. ( PW = &PW. ) point = __Point ;
            /* Compliment */
            Base = Lowcase( &Comp_Base. ) ;
            NN + 1 ;
            Base     = UpCase( Base ) ;
            Location = "Exon " || Put( Exon , Z2. ) ;
            __Exonic_nucleotide  + 1 ;
            __CDS_nucleotide     + 1 ;
            __Codonic_nucleotide + 1 ;
            If __Codonic_nucleotide = 4 Then __Codonic_nucleotide = 1 ;
            Codonic_nucleotide  = __Codonic_nucleotide       ;
            CDS_nucleotide      = __CDS_nucleotide           ;
            Codon               = Ceil( CDS_nucleotide / 3 ) ;
            Exonic_nucleotide   = __Exonic_nucleotide        ;
            Intronic_nucleotide = .                          ;
            Output ;
          End ;
        End ; /* Cycled through second to last exonStarts-exonEnds pairs */
        /* First intron and exon */
        Exon_Starts = Input( Scan( exonStarts , 1 , "," ) , 15. ) + 1 ;
        Exon_Ends   = Input( Scan( exonEnds   , 1 , "," ) , 15. )     ;
        Intron_Starts = Exon_Ends + 1 ;
        /* Last Intron */
        __Intronic_nucleotide = 0 ;
        Do __Point = Intron_Ends To Intron_Starts By -1 ;
          P = __Point ;
          Set hg19.&Chrom. ( PW = &PW. ) point = __Point ;
          /* Compliment */
          Base = Lowcase( &Comp_Base. ) ;
          NN + 1 ;
          Base     = LowCase( Base ) ;
          Location = "Intron " || Put( Exon , Z2. ) ;
          Codonic_nucleotide  = . ;
```

```
                CDS_nucleotide      = . ;
                Codon               = . ;
                Exonic_nucleotide   = . ;
                __Intronic_nucleotide + 1 ;
                Intronic_nucleotide = __Intronic_nucleotide ;
                Output ;
              End ;
              /* Last Exon */
              __Exonic_nucleotide = 0 ;
              Do __Point = Exon_Ends To Exon_Starts By -1 ;
                P = __Point ;
                Set hg19.&Chrom. ( PW = &PW. ) point = __Point ;
                /* Compliment */
                Base = Lowcase( &Comp_Base. ) ;
                NN + 1 ;
                Base      = UpCase( Base ) ;
                Location = "Exon " || Put( CountC( exonStarts  , "," ) , Z2. ) ;
                __Exonic_nucleotide  + 1 ;
                __CDS_nucleotide     + 1 ;
                __Codonic_nucleotide + 1 ;
                If __Codonic_nucleotide = 4 Then __Codonic_nucleotide = 1 ;
                Codonic_nucleotide  = __Codonic_nucleotide       ;
                CDS_nucleotide      = __CDS_nucleotide           ;
                Codon               = Ceil( CDS_nucleotide / 3 ) ;
                Exonic_nucleotide   = __Exonic_nucleotide        ;
                Intronic_nucleotide = .                          ;
                If TxStart < __Point < CDSStart + 1
                Then
                  Do ;
                    Location            = "3' UTR" ;
                    Codonic_nucleotide  = . ;
                    CDS_nucleotide      = . ;
                    Codon               = . ;
                    Intronic_nucleotide = . ;
                  End ;
                Output ;
              End ;
              /* 3' Genomic */
              Do __Point = TxStart To TxStart - 50000 By -1 ;
                P = __Point ;
                Set hg19.&Chrom. ( PW = &PW. ) point = __Point ;
                /* Compliment */
                Base = Lowcase( &Comp_Base. ) ;
                NN + 1 ;
                Base     = LowCase( Base ) ;
                Location = "3' Genomic" ;
                &CM.
                 Intronic_nucleotide = . ;
                Output ;
              End ; /* Cycled through __Points */
            End ; /* Strand = "-" */
          Stop ;
        Run ;

    %MEnd Gene_Annotate ;
```