NONMEM Users Guide - Part IV

NM-TRAN Guide

September 2020

by

Alison J. Boeckmann

Stuart L. Beal

Lewis B. Sheiner

NONMEM Project Group
University of California at San Francisco

ICON plc, Gaithersburg, Maryland

Preface to 5th Edition

The appearance of this 5th edition of the NM-TRAN Guide coincides with the appearance of NM-TRAN 7.5.0 and NONMEM 7.5.0. It contains changes since the 4th. edition, which appeared with NM-TRAN 7.4.2 and NONMEM 7.4.2.

Significant changes since the previous edition are marked with bars.

This version supports the new features of NONMEM and PREDPP and NM-TRAN Versions 7.5.0.

Abbreviated code:

PRED_IGNORE_DATA, PRED_IGNORE_DATA_TEST feature.
See Chapter IV Section E.6

New output file:

FDATA.csv
See Chapter II .B.1

Compartment Name Substitution

Compartment names defined in $MODEL are automatically available for substitution without requiring $ABBR REPLACE records. This is called "implicit" compartment name replacement.
See Chapter V, C.4. $MODEL Record

Symbolic Label Substitutions at $THETA, $OMEGA, and $SIGMA records

The $THETA record may be used to specify a symbolic label substitution and initial values for a THETA.
See Chapter III B.9

The $OMEGA record may be used to specify symbolic label substitution for an ETA and initial values for $OMEGA
See Chapter III B.10

The $SIGMA record may be used to specify symbolic label substitution for an EPS and initial values for $SIGMA
See Chapter III B.11

This preface mentions some of the most important changes to NM-TRAN, but is not complete. NONMEM Users Guide VIII and on-line help and Introduction to NONMEM 7 should also be consulted.

Table of Contents

I. Introduction

NM-TRAN stands for **N**ON**M**EM **Tran**slator, a preprocessor to NONMEM which translates user-inputs into i) a NONMEM data set, ii) a NONMEM control stream, and iii) various subroutines which must be included in a NONMEM load module. It is a separate computer program which is written in FORTRAN 90/95, and one *precedes* a NONMEM run by first running it. This document describes NM-TRAN and how to use it. In order to read this document the reader should be familiar with the concepts and nomenclature associated with the statistical models expressible in NONMEM. This familiarity can be obtained by reading modeling discussions in Guides I and VI. At the same time attention should also be paid to material describing the concepts and nomenclature associated with NONMEM data records and data items (especially Guide I, section B.1) and PREDPP event records and data items (especially chapters I, II, and V of Guide VI), and to the concepts and nomenclature associated with the various kinds of NONMEM output. However, material in these guides describing how control records, file records, and user-supplied subroutines are constructed may be skipped. Beginning NONMEM users who desire to analyze pharmacokinetic data will find it particularly helpful to first read Guide V. That document is written especially for the beginning user and from the point of view that the user is going to use NM-TRAN. Much of NM-TRAN, as well as much about NONMEM modeling, is described there.

The inputs for NM-TRAN include a data set, the NM-TRAN data set , but this data set need not be formatted quite as rigidly as a NONMEM data set. NM-TRAN translates the NM-TRAN data set into a NONMEM data set. The part of NM-TRAN which performs this translation is called the Data Preprocessor .

The inputs for NM-TRAN also include a control stream, the NM-TRAN control stream , but, again, the language for this control stream is much more user-friendly than the fixed-field numerical-code type language used for a NONMEM control stream. NM-TRAN translates the NM-TRAN control stream into a NONMEM control stream.

The NM-TRAN control stream can (optionally) also include abbreviated FORTRAN codes from which various completely FORTRAN-coded NONMEM subroutines are generated. Thus, for example, from one such abbreviated code a PRED subroutine can be generated which computes the partial derivatives of the statistical model with respect to η and ε random variables and stores them in the G and H arguments of the PRED routine. The abbreviated code itself does not directly involve partial derivatives. In effect, NM-TRAN performs symbolic differentiation, and this ability probably represents its most useful purpose. An abbreviated code also allows the statistical models to be represented in a natural and perspicuous symbolic way, wherein the η and ε variables are explicitly expressed. Indeed, one need not even understand the allusions that have just been made to partial derivatives and G and H arrays; it is enough to understand that a statistical model may be represented in a natural way and that from this representation NM-TRAN automatically generates information required by NONMEM.

As powerful a device as is NM-TRAN abbreviated code, such code is still somewhat limited, and certain complicated subroutines which could be useful cannot be generated from it alone. It will probably be adequate, though, for the purposes of any beginning NONMEM user.

It should be emphasized that NONMEM can be used without NM-TRAN. NM-TRAN simply makes the user's tasks easier, and we strongly recommend the use of this preprocessor, especially for beginning NONMEM users.

Here follows an example of an NM-TRAN control stream; it is meant to be used along with the example of an NM-TRAN data set shown in Appendix II. This NM-TRAN control stream is recorded on the NONMEM distribution medium as CONTROL4; see Guide III. NM-TRAN will translate the data set and control stream to a NONMEM data set, a NONMEM control stream, and a completely coded PRED subroutine. The effect of using these three NONMEM inputs in a NONMEM run will be to produce essentially the same output obtained from using the NONMEM control stream and PRED subroutine shown in Figures 75 and 74 of Guide I. That is, the effect will be to produce the same data analysis for

the population theophylline data as that presented and detailed in chapter F of that guide. The data set, control stream, and PRED routine produced by NM-TRAN are given in Appendix III. They differ somewhat from the data set, control stream and PRED routine of Figures 75 and 74. (In fact, the NONMEM control stream of Fig. 75 contains the data itself, rather than the data being in a separate file. NONMEM data can be embedded in a NONMEM control file.) However, the NONMEM output is the same, no matter whether NM-TRAN is used or not. Subroutine PRED shown in Appendix III is a simplified version of the subroutine PRED generated by NONMEM 7.3's NM-TRAN.†

```
$PROB  THEOPHYLLINE   POPULATION DATA
$INPUT      ID DOSE TIME CP=DV WT
$DATA       THEO

$PRED
;THETA(1)=MEAN ABSORPTION RATE CONSTANT (1/HR)
;THETA(2)=MEAN ELIMINATION RATE CONSTANT (1/HR)
;THETA(3)=SLOPE OF CLEARANCE VS WEIGHT RELATIONSHIP (LITERS/HR/KG)
;DOSE=WT-ADJUSTED DOSE (MG/KG)
;DS=NON-WT-ADJUSTED DOSE (MG)
   IF (DOSE.NE.0) THEN
      DS=DOSE*WT
      W=WT
   ENDIF
   KA=THETA(1)+ETA(1)
   KE=THETA(2)+ETA(2)
   CL=THETA(3)*W+ETA(3)
   D=EXP(-KE*TIME)-EXP(-KA*TIME)
   E=CL*(KA-KE)
   F=DS*KE*KA/E*D
   Y=F+EPS(1)

$THETA  (.1,3,5) (.008,.08,.5) (.004,.04,.9)
$OMEGA BLOCK(3)  6 .005 .0002 .3 .006 .4
$SIGMA  .4

$EST     MAXEVAL=450  PRINT=5
$COV
$TABLE          ID DOSE WT TIME
$SCAT           (RES WRES) VS TIME BY ID
```

Much of the remainder of this document is devoted to describing the language illustrated in this example.

An NM-TRAN control stream includes control type information. It also can include information indicating that a FORTRAN-coded PRED routine is being supplied by the user, or it can include an abbreviated code from which a PRED routine can be generated (as in the example). However, NM-TRAN is also designed to make the use of PREDPP easier. PREDPP is a special, but elaborate, PRED routine that has

† In order to obtain subroutine PRED for Appendix III, the record *$ABBR NOFASTDER DERIV2=NO*, was added to the control stream so that code for eta first partial derivatives is not collected and performed separately, and code for eta second derivatives is not generated. Other statements usually produced by NM-TRAN (such as those for NMPRD4; see Chapter II) were deleted because they are not needed for this example. Other code not relevant to the Estimation Method of figure 75 was also omitted for the sake of clarity.

been developed to assist with the task of analyzing pharmacokinetic data; see Guide VI. So, the NM-TRAN control stream can also include special control type information which can be used only when PREDPP is used and which facilitates the use of PREDPP, and it can also include abbreviated codes from which routines for PREDPP, which are otherwise user-supplied (INFN, MODEL, PK, ERROR, DES, AES), can be generated.

It may also contain abbreviated code from which a user-supplied MIX subroutine can be generated, and control record information from which a user-supplied PRIOR subroutine can be generated. This is independent of the choice of $PRED vs. PREDPP.

Accordingly, this document is divided into a few major parts which separate PREDPP considerations from general NONMEM considerations (see the Table of Contents).

II. General Considerations

II.II.A. Generated Subroutines and nmfe

A subroutine may be generated from an abbreviated code in an NM-TRAN control stream, as explained in chapter I. Such a routine is called a generated subroutine , or a generated code . With NONMEM 7 this is the only choice because NM-TRAN Library subs are no longer supported.

Typically, NM-TRAN is run using shell script nmfe74 and batch file nmfe74.bat (nmfe stands for "Non-Mem Front-End"; "74" indicates the version of NONMEM). These are supplied with NONMEM. NM-TRAN creates certain files for use by nmfe. However, any user-written or third-party interface may be used. It may (or may not) invoke NM-TRAN before running NONMEM. It may use any of the files created by NM-TRAN.

II.II.B. Files

II.II.B.1 NM-TRAN Files

NM-TRAN takes its input from two files: one file contains the NM-TRAN control stream, and one file contains the NM-TRAN data set. The control stream points to the name of the file containing the data set. For more information about how to run NM-TRAN, see Guide III. NM-TRAN outputs several files with names as shown:

	File Name	Content
1.	FDATA	NONMEM data set
2.	FCON	NONMEM control stream
3.	FSUBS	generated and user subroutines
4.	FSTREAM	NONMEM file stream
5.	FREPORT	report file

FDATA and FCON are needed in a NONMEM run and have already been mentioned in chapter I. If NM-TRAN actually modifies the NM-TRAN data set, then the modification, the NONMEM data set, is contained in FDATA.

With NONMEM 7.5, an additional file, FDATA.csv is produced that outputs the contents of its input data file (typically FDATA) in a comma delimited file format, so you can check how NONMEM interprets the input data. The records in FDATA.csv may differ from those in FDATA in the following cases. If REPL/REPL_ is used, the replicated form of the data will appear in FDATA.csv. Also, records excluded by PRED_IGNORE_DATA will not be present in FDATA.csv.

If, though, NM-TRAN does not need to modify the NM-TRAN data set, then the NONMEM data set is simply identified with the NM-TRAN data set and is found in whatever file contains the latter. FSUBS contains the FORTRAN-coded subroutines generated from abbreviated codes. With NONMEM 7, FSUBS contains additional subroutines; see below. All subroutines in FSUBS must be compiled and the resulting object modules used in the NONMEM load module.† FSTREAM is also needed in the NONMEM run. The NONMEM file stream is described in Guide I, section B.3. Sometimes a NONMEM run does not need a file stream (see Guide I, section C.3.1). However, whenever NM-TRAN is used before NONMEM, a NONMEM file stream is needed. (This is because with NM-TRAN, the NONMEM data set is never embedded in the NONMEM control stream, and the NONMEM file stream is needed to point

† The NONMEM executable is also referred to as the NONMEM load module. This is a different usage of the word "module" than the Fortran 90 MODULE's discussed below; the latter will always have the word MODULE in upper-case.

to the file containing the NONMEM data set.) FREPORT contains a list of all the routines which must be present in a NONMEM load module in order to implement the NONMEM run specified by the NM-TRAN inputs. It may be useful to the user to have this information, but this file is not needed by NONMEM. For more about how this information could be useful, see Guide III, which discusses the shell script nmfe74 and batch file nmfe74.bat.

Examples of FDATA, FCON, FSTREAM, FREPORT, and FSUBS are given in Appendix III. These examples result from using the NM-TRAN control stream shown in chapter I along with the NM-TRAN data set shown in Appendix II.

FSUBS contains several subroutines and MODULES in addition to those generated for the user's abbreviated code, as follows.

SUBROUTINE MUMODEL2 (nm71)

The MUMODEL2 subroutine is based on the PRED or PK subroutine in FSUBS. It contains only statements (if any) for the MU_ model, which is used in the new (Bayseian) methods of NONMEM. This subroutine is frequently called during the Estimation Step, more often than PRED or PK. The fewer code lines that MUMODEL2 has to go through to evaluate all the MU_s the more efficient.

SUBROUTINE FSIZESR (nm72)

Starting with NONMEM 7.2, dynamic memory allocation is performed in NONMEM. Subroutine FSIZESR contains the constants for the allocation in this run. The $SIZES record can be used to override some of the values in FSIZESR. Constants set to 0 cannot be determined or are not given by NM-TRAN and will default to the values hard-coded in resource/SIZES.f90†

MODULES NMPRD4, PRINFN, and DECLAREVARIABLES

NMPRD4 is an area of storage that is defined by NONMEM. It may be used by user-supplied and generated code, and also by NONMEM. Variables stored in this area can be displayed by NONMEM in tables and scatterplots. It is possible for the user to set aside a portion of NMPRD4 for variables defined in user-supplied code, or for variables directly controlled by the user (see COMACT, COMRES, COMSAV in Chapters III.B.7 and IV.E.2). The implementation of NMPRD4 changed with NONMEM 7.

PRINFN is an area of storage that may be used by user-supplied and generated code, but not by NONMEM. PRINFN is generated only if an $INFN block of abbreviated code is present and defines variables. The implementation of PRINFN changed with NONMEM 7.

DECLAREVARIABLES is an area of storage that is defined using the $ABBREVIATED DECLARE control record. (nm73)

See Section D, below, for more information.

SUBROUTINE THETAISUB and SUBROUTINE THETARSUB (nm73)

These subroutines implement the $THETAI and $THETAR blocks of code, respectively, if present in the NM-TRAN control stream. If either or both blocks are not present, the corresponding subroutine is not present in FSUBS.

Several other files are also generated by NM-TRAN.

FSUBS_MU.F90

This file contains the MUMODEL2 portion of FSUBS (the MU_ model statements). File FSUBS_MU is produced for easy reading by the user, and is not used by the NONMEM system. It allows the user to see easily how the MU_ model is implemented in generated code. File FSUBS_MU may be ignored.

†The NONMEM installation directory contains several subdirectories. One is resource, in which the file SIZES.f90 may be found. Another is util, which contains utility files such as nonmem_reserved_general (see Chapter IV.J.4).

FSIZES

File FSIZES contains the same values as the FSIZESR routine in FSUBS. File FSIZES is produced for easy reading by the user, and is not used by the NONMEM system. File FSIZES may be ignored.

PRSIZES.f90 (nm72)

PRSIZES.f90 contains sizes for compiling PREDPP or a user-supplied or generated PRED ("pr recompile" in nmfe74). This is discussed in Guide VI, Chapter VII.A.

FMSG

This file contains normal and warning and error messages arising from the program's efforts to translate the NM-TRAN data set and control stream. The messages are also displayed at the terminal by NM-TRAN. When errors are detected, Files 1-5 may be incomplete, and NONMEM should not be run. Even if there are no fatal errors and NM-TRAN and NONMEM seem to terminate normally, the warning messages in FMSG should be reviewed, because they may describe situations in which the NONMEM results may be incorrect.

FORIG, FREPL

When the $ABBR REPLACE statement is present in the NM-TRAN control stream, NM-TRAN produces two files:

FORIG - Contains the original (pre-replacement) abbreviated code and $TABLE and $SCATTER records.

FREPL - Contains the new (post-replacement) abbreviated code and $TABLE and $SCATTER records.

These may be helpful for debugging.

FSUBS2, FSUBS.F90

FSUBS2 and FSUBS.F90 are copies of FSUBS.

thetair.f90

This file contains the subroutines THETAISUB and THETARSUB, if present in FSUBS.

II.II.B.2 NONMEM Files

NONMEM creates a number of files during the run.

INTER

If an MSF is output, and intermediate output with iteration summaries is requested (i.e., $ESTIM options MSFI and PRINT are used), then NONMEM also writes the parameter estimates of these summaries to the console and to file INTER, which now exists after the run terminates. INTER now include lines giving the parameter estimates in their natural unscaled space (NPARAMETR) as well as the UCP values (PARAMETER) (nm72).

Additional output files are produced by NONMEM. These provide a more efficient way of way of extracting numerical results from the analysis. Names of the files start with "root", where root is the root name (not including extension) of the NM-TRAN control stream file. The files have names with extensions (suffixes) .ext, .cov, .coi,
.cor, .phi, .phm, .grd, .shk, .shm, .cnv, smt, .rmt, etc. The file root.ext is called the raw output file. The root name may be specified by $ESTIMATION record option FILE. (nm72).

II.II.C. Data Set Translation - The Data Preprocessor

II.II.C.1. Fieldless Format

A NM-TRAN data set is much like a NONMEM data set. In particular, each data record consists of a sequence of data items, these data items are of the the same types across all data records (unless some

data records serve as continuations for others), and the data records are grouped into individual records.† An example of a NM-TRAN data set is given in Appendix II, and the first two individual records of this data set are these:

```
1       4.02       0.          .74      79.6
1         .        0.25       2.84        .
1         .        0.57       6.57        .
1         .        1.12      10.5         .
1         .        2.02       9.66        .
1         .        3.82       8.58        .
1         .        5.1        8.36        .
1         .        9.05       6.89        .
1         .        7.03       7.47        .
1         .       12.12       5.94        .
1         .       24.37       3.28        .
2       4.4        0.         0.        72.4
2         .         .27       1.72        .
2         .         .52       7.91        .
2         .        1.         8.31        .
2         .        1.92       8.33        .
2         .        3.5        6.85        .
2         .        5.02       6.08        .
2         .        7.03       5.4         .
2         .        9.         4.55        .
2         .       12.         3.01        .
2         .       24.3         .90        .
```

This same data set could be used as a NONMEM data set; it conforms to all the requirements of a NONMEM data set. A dot (surrounded by blanks) would be interpreted in a NONMEM run as a data item which is 0. Dots can substitute for 0 data items in order to improve the readability of the data set. Fields of blanks can also be used for the same purpose; see, for example, the NONMEM data set in Appendix III. This is because a FORTRAN format specification for the data set is always supplied in the NONMEM control stream. (Such a format specification can also be supplied in the NM-TRAN control stream, and when it is, fields of blanks can be used in the NM-TRAN data set). However, there are other NM-TRAN data sets which are convenient to use, but cannot be used as NONMEM data sets; NM-TRAN must be used to translate these to NONMEM data sets.

In general, use of NM-TRAN simplifies things for the user. For one thing, it is not required that a FORTRAN format specification be supplied in the NM-TRAN control stream. Without a format specification there really need be no fields of fixed lengths, even though in the example the data items have been lined up (in informal fields of fixed lengths) to improve readability. NM-TRAN understands two data items in a data record to be separated by any number of blanks or by a comma. It recognizes and translates the dot to a field of blanks (or to a field consisting of a specified character preceded by blanks).

The question "When Must a Format Specification be Included or Omitted?" is discussed in NONMEM Users Guide - Part V Introductory Guide, Chapter 6, Section 10.4.

---

† Note the difference between a *data record* (which is a single record in a data file) and an *individual record*, which is a group of contiguous data records having the same value for the ID data item and presumably containing data from the same individual. (When the data are not population, then the all the data records comprise a single individual record.)

Another way to format these data records for NM-TRAN is this:

```
1,4.02,0.     .74     79.6
1,,0.25      2.84
1,,0.57      6.57
1,,1.12     10.5
1,,2.02      9.66
1,,3.82      8.58
1,,5.1       8.36
1,,9.05      6.89
1,,7.03      7.47
1,,12.12      5.94
1,,24.37      3.28
2,4.4,0.    0.       72.4
2,,.27      1.72
2,,.52      7.91
2,,1.        8.31
2,,1.92      8.33
2,,3.5       6.85
2,,5.02      6.08
2,,7.03      5.4
2,,9.        4.55
2,,12.        3.01
2,,24.3        .90
```

Beginning with NONMEM 7.1, The maximum number of characters in a data item is given by constant SDF in resource/sizes.f90. This is 24. With previous versions, the maximum number was 12.

Notice that two successive commas act as a single dot, and a record may terminate with a data item which is not the last one if the subsequent data items in the record can be represented by dots.

Such items are referred to as null data items. They are replaced by a field of blanks in the NONMEM data set. Null data items can be instead be replaced with one consisting of a specified alphanumeric character. The NULL option of $DATA may be used to specify a different character (e.g., NULL=. or NULL=0). See Chapter III.B.5.

A file saved from Excel as a csv file ("comma separated values") may used as an NM-TRAN input file.

Starting with NONMEM vi.2.0, a tab character may also be used to separate data items. E.g., let T stand for the tab character.
"1T2" and "1,2" are read as "1 2"
"1TT2" and "1,,2" are read as "1 0 2"

Thus, a file saved from Excel as a txt file ("Text (tab delimited)" or "Windows Formatted Text") may also be used as an NM-TRAN input file.

Starting with NONMEM VI.2.0, MS-DOS formated files can be read in Unix. These are control files and data files having a carriage return character at the end of the line. Such characters may appear as "^M" in NONMEM output.

Useful information for operating systems as of 1992 can be found in the Introduction to Version VI (intro.pdf). This information may be relevant to more recent operating systems. In particular, see
Section 26. "NM-TRAN datafiles - Ill-formed files"
Section 29. "NM-TRAN datafiles: Tab and ^M".

Null data items as described above consist of a single dot (.) or consecutive commas or consecutive tabs.

It may happen that a record in the NM-TRAN data file is totally blank. NM-TRAN will produce an error message unless $DATA option BLANKOK is used. In this case, there is no error message and the corresponding record in the NONMEM data set will contain null values.

II.II.C.2. Day-Time Translation

Another way to format these data records for NM-TRAN is this:

```
1       4.02       9:00        .74      79.6
1        .         9:15       2.84        .
1        .         9:34       6.57        .
1        .        10:07      10.5         .
1        .        11:01       9.66        .
1        .        12:49       8.58        .
1        .        14:06       8.36        .
1        .        18:03       6.89        .
1        .        16:02       7.47        .
1        .        21:07       5.94        .
1        .        33:22       3.28        .
2       4.4        8:00       0.        72.4
2        .         8:16       1.72        .
2        .         8:31       7.91        .
2        .         9:00       8.31        .
2        .         9:55       8.33        .
2        .        11:30       6.85        .
2        .        13:01       6.08        .
2        .        15:02       5.4         .
2        .        17:00       4.55        .
2        .        20:00       3.01        .
2        .        32:18        .90        .
```

Here the third data item of each record, which is a value of time in hours, has been expressed as a clock time, rather than as a relative time. NM-TRAN can accept clock time; NONMEM (and PREDPP) cannot. NM-TRAN translates a clock time to a relative time. Information in the NM-TRAN control stream indicates that translation of this type should be performed; see section III.B.2. When this is done, all times in the NM-TRAN data set are assumed to be clock times. A clock time is either of form hr:min, as in the example, or hr.fr, where fr is a decimal fraction of the hour to two digits, using a military clock in both cases. E.g. 2:45PM can be expressed either as 14:45 or 14.75.
With NONMEM 7.3, values may also have the form hh:mm:ss (i.e., hours:minutes:seconds). For example, 8:45:29.

NM-TRAN provides five digits (xxxxx.xx), which allows relative time to be at most approximately 4166 days. Furthermore, when the $DATA record includes the option WIDE, six digits (xxxxxx.xx) are provided.

NM-TRAN makes a distinction between population data and single-subject data; see section C.4. If the data are population data, then the clock time in the first data record of an individual record is translated to relative time 0. If the data are single-subject data, then the clock time in the first data record of the data set is translated to relative time 0. If PREDPP is used, then a time on a reset or reset-dose event record is translated to relative time 0, whether the data type is population or single-subject.

Yet another way to format these data records for NM-TRAN is this:

```
1      4.02     10/1       9:00       .74     79.6
1       .       10/1       9:15      2.84       .
1       .       10/1       9:34      6.57       .
1       .       10/1      10:07     10.5        .
1       .       10/1      11:01      9.66       .
1       .       10/1      12:49      8.58       .
1       .       10/1      14:06      8.36       .
1       .       10/1      18:03      6.89       .
1       .       10/1      16:02      7.47       .
1       .       10/1      21:07      5.94       .
1       .       10/2       9:22      3.28       .
2      4.4      10/1       8:00      0.        72.4
2       .       10/1       8:16      1.72       .
2       .       10/1       8:31      7.91       .
2       .       10/1       9:00      8.31       .
2       .       10/1       9:55      8.33       .
2       .       10/1      11:30      6.85       .
2       .       10/1      13:01      6.08       .
2       .       10/1      15:02      5.4        .
2       .       10/1      17:00      4.55       .
2       .       10/1      20:00      3.01       .
2       .       10/2      08:18       .90       .
```

Here calendar dates have been included, and clock time has been treated modulo 24 hours. These dates are called date data items . Information in the NM-TRAN control stream indicates that translation with this type of data should be performed; see section III.B.2. Information in the NM-TRAN control stream also indicates that the date data items should not appear in the NONMEM data set; only the relative time data items should appear where clock time data items appear. More generally, instead of month followed by day in date data items, day can follow month, and year can also be included at either the beginning or end of the data items. Or, only day need be given. Again, the choice of format in this regard is specified in the control stream. Regardless of format, with any date data item any non-numeric character, *except* comma or blank, can separate the date fields. Here are some examples of possibilities: 10/1, 10-1, 10-1-1986, 10-1-86, 86-10-1. Leap years are recognized; they are years 4, 8, 12, etc. If year is omitted from the format, the year is assumed to be year 0, which is not a leap year. When the date data items only give days (months and years are missing), then these data items can be any positive or negative integers, and for the purpose of computing relative times, they are ordered in the same way as are the integers.

If year is recorded with four digits, it is always processed correctly and the value of LAST20 is of no consequence. The following discussion only affects years that are recorded with two digits. The year 1900 was not a leap year, but the year 2000 was a leap year. A constant LAST20 in resource/TRGLOBAL.f90 affects how two digit years are interpreted.
One or two digit years > LAST20 are assumed to be in the 1900's,
One or two digit years <= LAST20 are assumed to be in the 2000's.
The default value of LAST20 is 50. Two digit years are interpreted as follows:
00-50 = 2000-2050
51-99 = 1951-1999

The option LAST20 on the $DATA record can be used to change how two digit years are interpreted. For example, LAST20=-1 can be used when two digit years span the year 2051. All two digit years will be

assumed to be in the same century.

The TRANSLATE option of the $DATA record was new to NONMEM V and has been expanded with NONMEM 7.3. Any values may be given for dividing TIME and II values, and any precisions may be requested for the result. An example is:

```
$DATA TRANSLATE (TIME/0.01/6)
```

which divides TIME values by 0.01, and writes 6 digits to the right of the decimal. TRANSLATE may be used to convert hours to days. E.g., TIME/24.000 or TIME/24.00. With NONEM 7.3, new choices are TIME/F, TIME/F/D and II/F, II/F/D. F ("factor") and D ("digits") may be integer or real values. This is independent of day-time translation. That is, TRANSLATE may be specified whether or not day-time translation occurs, and is applied by NM-TRAN after day-time translation. The user must insure that the units of PK parameters such as CL or K are consistent with the units of TIME. See Guide VIII on-line help description for $DATA TRANSLATE for more information.

II.II.C.3. Included Comments

Comments can be included in NM-TRAN data sets. E.g.

```
         1       .       12.12      5.94       .
         1       .       24.37      3.28       .
C This next individual may be an outlier
         2      4.4       0.        0.       72.4
         2       .         .27      1.72       .
```

If requested, any (FORTRAN) record with a designated character in position 1 is ignored, i.e. it does not appear in the NONMEM data set. The character used in position 1 must be the same for all comment records; it is specified in the $DATA record using IGNORE=C (see section III.B.5).

Note that the IGNORE and ACCEPT options of the $DATA record can also be used to select records according to the values of specified data items. This is also discussed in section III.B.5.

II.II.C.4. Generated ID Data Items

II.II.C.4.1. Background

NONMEM data records are grouped into contiguous sets of records called individual records . With population data, where there are multiple observations from multiple subjects, an individual record contains the data records associated with a given subject. Each data record of the individual record has the same ID data item. NONMEM can also be used to analyze data from a single subject. With such data, and when each observation consists of a single number, an individual record is simply any group of contiguous data records including only one observation record and having the same ID data item if the number of data records in the group is two or more. When a multivariate observation is present, the individual record containing it includes several observation records, each containing one element of the observation.

The inclusion of ID data items in a data set with single-subject data is a little unnatural, and it is not commonly required by computer programs that are meant to be used only with single-subject data, With single-subject data, NM-TRAN automatically generates the ID data items required in a NONMEM data set. Such ID data items are called generated ID data items . Usually, this is advantageous. There are situations, though, where the generated ID data items are not appropriate, and then generated ID data items should be disallowed; see below. Most notably, such a situation occurs when multivariate observations

are included in the data set. The mechanism for disallowing generated ID data items is given in sections C.4.2 and III.B.2.

NM-TRAN "recognizes" the difference between population data and single-subject data .†
(NONMEM itself also makes this distinction, which it makes available to PRED via reserved variable IPS.)
A data set consisting of population data is called a population data set . A data set consisting of single-subject data is called a single-subject data set .† The only consequence of NM-TRAN's ability to recognize this difference between data types is to allow ID data items to be automatically generated with single-subject data. NM-TRAN infers from information in the control stream whether the data are of one type or the other. The way this is done is outlined next. For details, see section C.4.2.

Both NONMEM and NM-TRAN explicitly recognize two types of random variables, η-variables and ε-variables. These two types are nested, i.e. for any set of fixed values for the η-variables, the ε-variables can assume different values, but not conversely. The ε variables can only occur along with η variables, and then they represent random intraindividual effects, while the η variables represent random interindividual effects. If control stream information indicates that ε variables occur in the statistical model, then population data are inferred. To infer population data when the ε variables occur is consistent with the fact that with population data there are *both* random intersubject and random intrasubject effects in the statistical model; then the interindividual (intraindividual) effects are identified with the intersubject (intrasubject) effects. (However, for noncontinuous population data see the discussion below.)

If population data are not inferred, then only η variables occur in the model. These variables, occuring by themselves, are nonnested variables. They can represent either intersubject effects or intrasubject effects. (The terms 'intrasubject' and 'intraindividual' are used interchangeably, but the term 'interindividual' is reserved; see below.) With single-subject data there is no subject-to-subject variability, and the η variables represent intrasubject effects. When each observation comes from a different subject, either the data are regarded as population data and ε variables are used, or more usually, intrasubject variability is not distinguishable from intersubject variability, and η variables only are used to represent the one type of random effect. As long as only η variables occur in the statistical model and the observations are taken to be statistically independent, it really makes no difference whether the η variables are regarded as representing intersubject or intrasubject random effects. Statistically, the observations being modeled can be regarded as arising from a single subject, or each observation can be regarded as coming from a different subject. The NONMEM convention is to take a "middle position". The term individual is used to mean an individual observation in the *population of observations*. Thus no matter how the data actually arise, the η variables can be regarded as representing interindividual effects. Moreover, in deference to the often-occuring pharmacokinetic study where data arise from a single-subject, when the data are not recognized as population data, they are called single-subject data, even when different observations actually arise from different subjects. This convention conforms with a convention of PREDPP, whereby when only nonnested random variables occur, the data are regarded as single-subject data (Guide VI, section IV.A).

With single-subject data, each observation must be in a *different* individual record (this requirement is a consequence of the fact that the η variables represent interindividual effects). Accordingly, if one wants to extract an individual record from a population data set and use this record as a single-subject data set (a new NM-TRAN data set), the ID data items in this record must be changed since they are all equal. In this case one might allow NM-TRAN to generate new ID data items for the single-subject data set. At the same time one might (but one need not) instruct NM-TRAN to exclude the old ID data items from the NONMEM data set (see discussion of the DROP attribute in section III.B.2). Whether or not one does the latter, NONMEM will be properly instructed to use only the new ID data items.

---

† In the first edition of this guide and in certain other NONMEM Users Guides, the terms individual data and individual data set are used instead of single-subject data and single-subject data set.

There are single-subject data sets where generated ID data items should be disallowed. One example is where there are multivariate observations. Each element of the multivariate observation must be placed on a different data record, but each of these records must be included in the same individual record. Were generated ID data items allowed, a multivariate observation would span more than one individual record. In this example, the user must include appropriate ID data items in the NM-TRAN data set.

Note that sometimes the data are population data in the sense that multiple observations are obtained from multiple subjects, but ε variables are not used in the statistical model. This situation arises when, for example, the data are categorical, rather than continuous. Then η variables still represent random interindividual effects, and random intraindividual variablity exists, but it is expressed without the use of ε variables. If option LIKELIHOOD or -2LL is used on the $ESTIMATION record, NM-TRAN recognizes this as odd-type data and does not generate the ID data item. See II.II.D below.

Note that NONMEM can also analyze a population data set as if the data from each individual were single-subject data. ("POPULATION WITH UNCONSTRAINED ETAS"); see Section III.B.10 (nm73).

II.II.C.4.2. Implementation

NM-TRAN infers that the data are population data when at least one of the following is true of the control stream:

1. An abbreviated code is present which uses `EPS`'s.
   See section IV.A
2. An abbreviated code is present which uses `ETA`'s, and an abbreviated code is present (perhaps the same one) which uses `ERR`'s.
   See section IV.A
3. A $SIGMA record is used.
   See section III.B.11.
4. A $MSFI record is used with the option `NPOPETAS`=n, where n is positive.
   See section III.B.12.
5. With PREDPP: no $PK record is used, and a $OMEGA record precedes a $ERROR record.
   See section V.C.6
6. An $ESTIMATION record includes the option LIKELIHOOD or -2LOGLIKELIHOOD, and an abbreviated code is present which uses `ETA`'s.†
7. The $ESTIMATION record includes the option LIKELIHOOD or -2LOGLIKELIHOOD, and an $OMEGA record is used.†

When none of the above is true, the data are inferred to be single-subject data. In this case NM-TRAN generates ID data items unless this is disallowed by the presence of the reserved labels `L1` or `L2` on the $INPUT record. Generated ID data items have values 1 and 2.‡

When each data record includes an actual observation, then the generated ID data items alternate between 1 and 2 with every data record. More precisely, when MDV (missing dependent variable) data items do not appear in the NM-TRAN data set, or when NM-TRAN does not automatically include them in the

† This option is used with non-continuous observed responses. If the data are population, The data are referred to as "odd-type data". See II.II.D below.

‡The rule can be described as follows: When the data are single-subject and the L2 data item is defined to NM-TRAN, then the L2 data item is identified to NONMEM as the ID item. No L2 item is identified to NONMEM. The ID item may be defined to NM-TRAN, but it is not identified to NONMEM as such. The variables L2 and ID (if defined) may be used in abbreviated code, and then they refer to the corresponding items of the data record. However, if the user uses the label L1 instead of the label ID, or uses the label L1 as a synonym with the ID label, then NM-TRAN does not change the designations: the items labeled L1 are taken to be the ID data items, and the items labeled L2 are taken to be the L2 data items.

NONMEM data set (see section V.B), then the generated ID data items alternate between 1 and 2 with every data record. Suppose, though, that either MDV data items appear in the NM-TRAN data set, or are automatically included by NM-TRAN in the NONMEM data set. Then the generated ID data items alternate between 1 and 2, remaining constant over a group of contiguous data records with MDV=1.

For an example, see Appendix IV.

II.II.D. Odd-type Data (nmv)

By default, objective functions used with NONMEM are least-squares type functions, and as such, need not be considered to be related to likelihood functions. However, NONMEM allows data to be analyzed with an objective function equal to -2log likelihood for the data, and this is particularly necessary when the intraindividual distributions of the data are non-continuous (discrete) and/or very asymmetrical. Categorical data is an example of non-continuous data. (For the purposes of this discussion, such data are called "odd-type" data.) To accomplish this, the value of F returned by the PRED routine with an observation may be set to the value of the conditional likelihood of η for the observation, given the values of the model parameters θ , rather than set to a prediction for the observation. The matrices G and H should be the usual derivatives of the expression used for F. The η's, if any, are understood to be population etas. No ε variables may be used.

This different use of F is signaled by using the option LIKELIHOOD on the $ESTIMATION record. It is assumed that first-order type approximations are not being used, and so the LAPLACIAN option should also appear. Alternatively, F may be set to the value of -2log conditional likelihood of Î· for the observation. This is signaled by using the option -2LOGLIKELIHOOD (or simply -2LL) in the $ESTIMATION record.

Odd-type data may also be simulated. For this purpose, PRED must return the value of a simulated observation as the DV data item (rather than in the argument F, as with standard-type data), a technique that will work even when the data are of standard-type.

A data set may contain both predictions and likelihoods; see `F_FLAG` in chapter IV.

See Guide VIII and on-line help for "Logistic regression example" (logit2.exa). See also example10.ctl and example10l.ctl in the NONMEM examples directory for simultaneous analysis of predictions and likelihoods.

II.II.E. Implementation of NMPRD4, PRINFN and DECLAREVARIABLES in NONMEM 7

The user of NM-TRAN need not understand how MODULES NMPRD4, PRINFN, and DECLAREVARIABLES are implemented in FSUBS. The following is supplied for readers who are curious or who need to understand the implementation.

II.II.E.1 Implementation of NMPRD4 in NONMEM 7

NMPRD4 is an area of storage that is used by user-supplied and generated code, and also by NONMEM. Variables stored in this area can be displayed by NONMEM in tables and scatterplots. In NONMEM IV through NONMEM VI, NMPRD4 is a named FORTRAN COMMON declared in both NONMEM and FSUBS. The declaration in NONMEM is

```
COMMON /NMPRD4/ VRBL(LNP4)
```

The allocation LNP4 can only be changed by re-compiling all of NONMEM and PREDPP and NM-TRAN.

The same COMMON may be defined in other subroutines with different names for the variables. A variable listed in a given location is the same variable, even though the name differs. Thus, a PK generated subroutine may contain a list such as

```
COMMON/NMPRD4/KA,K,CL,...
```

Within the generated PK subroutine, variable names KA, K, etc. are used. These variables are the same variables that are known to NONMEM as VRBL(1), VRBL(2), VRBL(3),...

Another way that user code can refer to positions in NMPRD4 is as elements of array COM, e.g., `COM(1),COM(2),...`

The use of COM variables is described in Chapter IV.E.3.

Starting with 7.1, NMPRD4 is a FORTRAN MODULE. Starting with 7.2, it is also dynamically allocated as needed for the current problem. With 7.2 and higher, the declaration in NONMEM is in resource/GLOBAL.f90:

```
MODULE NMPRD4
 REAL(KIND=DPSIZE), ALLOCATABLE, TARGET :: VRBL(:)
END MODULE
```

The VRBL array is allocated by NONMEM according to the value of LNP4 in resource/SIZES.f90 (or as specified by the $SIZES record).

A user-written PRED code need only contain `USE NMPRD4, ONLY: VRBL`

Because NMPRD4 is a Fortran MODULE, however, the names must match. The generated code would have to refer to user variables as VRBL(1), VRBL(2), etc.

In order to use alternate names for a variable in a MODULE, the variables must be given attributes TARGET (for the real name) and POINTER (for the alternate name) and the association operator "=>" must be used: pointer=>target. E.g., `KA=>COM(00001)` The subroutine must execute the association statement explicitly. Pointer KA may now be used as an alias to target COM(1).

An example may help illustrate the NM-TRAN generated code.

Suppose
`$ABBR COMRES=5`
is present in the control stream CONTROL5.

With NONMEM VI, FSUBS contains

```
COMMON/NMPRD4/BBBBB0(0005),KA,K,CL,SC,Y,A00031,A00032,A00035
COMMON/NMPRD4/A00040,A00041,A00034,BBBBBB(01984)
DIMENSION COM(002000)
EQUIVALENCE(BBBBB0(1),COM(1))
```

User-defined variables KA,K, etc., and NM-TRAN generated variables such as A00031 (which contain the values of their eta derivatives), are listed. BBBBBB is a "filler" to pad the size of the COMMON to size LNP4, which (using the default value of LNP4) is 2000. The array BBBBB0 reserves the 5 positions in NMPRD4, as requested. Variables COM(n) are alternative names for positions in NMPRD4. They may be used in abbreviated and generated code, and may be displayed in tables.

With NONMEM 7, The code is more complicated.

FSUBS contains MODULE NMPRD4P, and an included subroutine ASSOCNMPRD4, of which the important code follows:

```
MODULE NMPRD4P
USE NMPRD4,ONLY: VRBL
REAL(KIND=DPSIZE), DIMENSION (:),POINTER ::COM
REAL(KIND=DPSIZE), POINTER ::KA,K,CL
 ...
SUBROUTINE ASSOCNMPRD4
COM=>VRBL
KA=>COM(00001);K=>COM(00002);CL=>COM(00003)
 ...
```

```
END SUBROUTINE ASSOCNMPRD4
END MODULE NMPRD4P
 ...
IF (ICALL <= 1) CALL ASSOCNMPRD4
```

This code defines array VRBL in MODULE NMPRD4. COM is defined as a pointer (alias), and variables KA etc. are also defined as pointers. The ASSOCNMPRD4 subroutine assigns COM as an alias for VRBL, and KA, etc. as aliases for COM(1), etc.

In FSUBS, a subroutine ASSOCNMPRD4 is called at ICALL 0 (Run Initialization) and ICALL 1 (Problem Initialization).
`CALL ASSOCNMPRD4` ASSOCNMPRD4 performs the associations for that subroutine.

With this code, COM(i) can be used in abbreviated and generated code, as with earlier versions.
Within the generated PK subroutine, variable names KA, K, etc. are used, as with previous versions of NONMEM.

In Chapter IV Section IV.IV.I the use of verbatim code is described. Code from the FIRST block is inserted between the last declaration and the first executable statement.

II.II.E.2. Implementation of PRINFN in NONMEM 7

PRINFN is an area of storage that may be used by both user-supplied and generated code, but not by NONMEM. It is part of generated code if there is an $INFN block that defines Initialization-Finalization variables.

Prior to NONMEM 7, it was a named COMMON PRINFN, defined in FSUBS in subroutines such as INFN, PK and ERROR.

With NONMEM 7, it is implemented much like NMPRD4. MODULE PRINFN is defined in resource/GLOBAL.f90:

```
MODULE PRINFN
  REAL(KIND=DPSIZE), TARGET, DIMENSION (DIMTMP):: ITV
END MODULE PRINFN
```

DIMTMP is set in SIZES.f90. The default value is 500. It may be increased using $SIZES.

Suppose this code is used:

```
$ABBR COMRES=1
 ...
IF (ICALL.EQ.1) SUM=0
```

FSUBS will contain:

```
MODULE INFNP
USE PRINFN, ONLY: ITV
REAL(KIND=DPSIZE), DIMENSION (:),POINTER ::TLCOM
REAL(KIND=DPSIZE), POINTER ::SUM
CONTAINS
SUBROUTINE ASSOCPRINFN
TLCOM=>ITV
SUM=>TLCOM(00001)
END SUBROUTINE ASSOCPRINFN
END MODULE INFNP
```

Each subroutine in FSUBS contains the same code, e.g.,

```
SUBROUTINE INFN(ICALL,THETA,DATREC,INDXS,NEWIND)
USE INFNP
```

```
IF (ICALL <= 1) CALL ASSOCPRINFN
```

In each subroutine, variables TLCOM and SUM may be used.

II.II.E.3. Implementation of DECLAREVARIABLES in NONMEM 7 (nm73)

The implementation of DECLAREVARIABLES is much simpler. Variables are listed as defined. For example, suppose the following control record is present with $PRED (with PREDPP, A is reserved and a different name would have to be used to avoid a compiler error):

```
$ABBR DECLARE A,B(10),C(1,NO-1),INTEGER I J
```

The generated subroutine contains the following, which declares the variables and initializes them to zero:

```
MODULE DECLAREVARIABLES
USE SIZES, ONLY: DPSIZE,ISIZE,NO
SAVE
REAL(KIND=DPSIZE) :: A=0.0D+00
REAL(KIND=DPSIZE) :: B(10)=0.0D+00
REAL(KIND=DPSIZE) :: C(1,NO-1)=0.0D+00
INTEGER(KIND=ISIZE) :: I=0
REAL(KIND=DPSIZE) :: J=0.0D+00
END MODULE DECLAREVARIABLES
```

Each subroutine in FSUBS contains the same code:

```
USE DECLAREVARIABLES
```

III. Control Records

III.III.A. Record Syntax - General

A partial listing of NM-TRAN control record names and options is given in Appendix I. A complete listing of all the control records can be found in NONMEM Users Guide V Appendix 3. NONMEM Users Guide VIII and on-line help describe all the options. General rules for constructing these names and options are given in this section. The reader may refer to the example of the control records given in chapter I. It is not possible for NM-TRAN to generate a NONMEM control stream which contains syntax errors. Errors in an NM-TRAN control stream are reported in a report file; see Chapter I (FMSG).

1. Record names begin with `$`. With NONMEM 7.2 and higher, both lower and upper case may be used for record names and all user-defined and reserved words in the control stream. Upper-case is used in this guide for clarity.

   E.g. `$OMEGA`

2. Option names on records may also be upper-case or lower-case.

   E.g. `SIGDIGITS`

3. The order of the records and the order of options within records is immaterial except where noted in the particular discussion of the record or option.

4. With record names and options prior to NONMEM 7, initial substrings of record or option names, and of length 3 or more, are recognized as abbreviations.

   E.g. `$COVARIANCE` or `$COV` or `$COVAR`

   An abbreviation is a type of alias for the record or option name. For any particular record or option, there may also be other acceptable aliases; these are noted as part of the description of the particular record or option. All aliases may be abbreviated according to the convention just described.

   There are exceptions such as the `NOABORTFIRST` option of $THETA. With NONMEM 7, some new record names (e.g., $ANNEAL) and options (e.g. $ESTIMATION record option ISAMPLE) cannot be abbreviated.

5. Text after a semicolon is regarded as a comment.

   E.g. `$THETA  7  ;Mean Clearance`

6. Blank-line records and records containing only comments can be included for clarity or readability.

7. With NONMEM VI 2.0 and later versions, the length of a single record of the NM-TRAN input file is at most 160 characters. (Previously, it was 80 characters.) With NONMEM 7.3, the maximum length is given by FSD in resource/SIZES.f90 (FSD=67000 with NONMEM 7.3).

   For readability, a record can be continued to form a contiguous block of records .

E.g.

```
$THETA   7   ;Mean Clearance
        20   ;Mean Volume
```

The record name, or an alias for it, physically appears only in the first record of a contiguous block. This name or alias is "understood" to appear in each continuation record of the block. Also, a record can be continued by a series of contiguous blocks, no two of which need to be contiguous to each other. In general, all the information from all records which use (or are understood to use) the same name, or use (or are understood to use) an alias for this name, is regarded as coming from a single record with that name. If this is ordered information, the ordering is determined by the ordering of the separate records.

E.g.

```
$THETA  7
        20
$OMEGA  .5  4
$THETA  .7
$OMEGA  .005
```

is equivalent to

```
$THETA 7 20 .7
$OMEGA .5 4 .005
```

The $ESTIMATION record, like the $TABLE and $SCATTER records, is an exception, as noted in the descriptions of these records.

With NONMEM 7.3 and higher, & may be used at the end of any line of the control stream to indicate that the line is to be continued, including control records as well as abbreviated code. (This is FORTRAN 90-style continuation.) If the ampersand at the end of a line is not to be interpreted as a continuation marker, but as a part of the record, then, place a ; after it. For example, an option of the $TABLE record may terminate with &.

```
FORMAT=s1PE15.8:160& ;
```

8. Options can be separated by commas and/or any number of spaces.

   E.g. `MAXEVAL=400    SIGDIGITS=4, PRINT=5`

9. An option of the form A=B must be contained on a single record and may contain spaces around =. A is called the option name , and B is called the option value .

   E.g. `MAXEVAL = 400`

   With an option of form A=B, the = can be omitted.

   `MAX = 300` or `MAX 300`

10. Filenames on control records may consist of as many characters as fit on a single line. A filename may not contain embedded spaces. If it contains commas, semicolons, or parentheses, or if it starts with an equal sign, then it must be surrounded by single quotes ' or double quotes ". An option name may not be used as a filename unless it is surrounded by quotes.
11. The numerical values in $THETA, $OMEGA, and $SIGMA may be each up to 30 characters long, and may be described in E field notation.

In the descriptions of the particular records, which follow in later sections, square brackets are used to surround an option or group of options, none of which need actually appear in the record. If they surround a group of options, a vertical line is used to separate these options in the description, and at most one of the options may be selected to actually appear in the record. If none are selected to appear, then the default option , indicated in boldface (if there is an option so indicated), is understood to apply.

E.g. [**`UNCONDITIONAL`**|`CONDITIONAL`] indicates a choice between the options `UNCONDITIONAL` and `CONDITIONAL`, and if neither are selected to appear in the record, it is understood that the first option applies.

III.III.B. Record Syntax - Specific Records

Specific NM-TRAN control records are described in the next subsections. The first 17 records ($PROBLEM thru $SCATTERPLOT) were part of NONMEM IV, and are discussed in detail. Options are listed as of NONMEM 7.4. Options added with NONMEM V and later are not always discussed in this document, but are discussed in the help/Guide VIII documents. The remaining records (starting with B.18. $SUPERPROBLEM Record) is a listing of record types introduced with NONMEM V and later, with brief descriptions. All of these records are optional. They are listed in order in which they were added to NONMEM, with some exceptions. For additional information, see Guide VIII and on-line help.

III.III.B.1. $PROBLEM Record

```
$PROBLEM  text
```

E.g.
```
$PROB   THEOPHYLLINE    POPULATION DATA
```

The text becomes a heading for the NONMEM printout.

This record is required. Only $SIZES and $SUPERPROBLEM may precede a $PROBLEM record. (See III.B.28 and III.B.18 below.) A $PROBLEM record other than the first one marks the beginning of another problem specification.

The text must be contained on a single record, and only the first 72 characters of text (starting with the second character after the record name) are used in the heading. Spaces *and* semicolons in the text are included "as is". The text is optional.

III.III.B.2. $INPUT Record

`$INPUT` item$_1$ item$_2$ item$_3$ $\cdots$

E.g.
```
$INPUT       ID DOSE TIME CP=DV WT
```

The items define the data item types that appear in the NM-TRAN data records, as well as the order of their appearance.

This record is required, and it must precede any other NM-TRAN control record in the problem specification that refers to specific data item types.

Each item has form B or A=B, where A and B are data item labels. Each data item label consists of letters (A-Z) and numerals (0-9), but it must begin with a letter. Starting with NONMEM 7.1, the underscore character _ may be used in a data item label. Starting with NONMEM 7.1, the maximum number of characters in a label is given by SD in resource/SIZES.f90. The default value is 20. With previous versions, the maximum number of characters is 4. The labels may be used in subsequent NM-TRAN control records, and they will be used as labels for data items in NONMEM output.

NM-TRAN recognizes certain reserved labels:
`ID`, `L1`, `L2`, `DV`, `MDV`, `TIME`, `DATE`, `DAT1`, `DAT2`, `DAT3`, `DROP`
`RAW_` (nmv), `MRG_` (nmv), `RPT_` (nmvi), `REPL_` (nm75)

The RAW_ data item identifies template records for which NONMEM computes and displays raw-data averages. With this feature, the TEMPLT variable may be used in abbreviated code, and the $OMIT record may be used.
The MRG_ data item identifies records for which NONMEM computes and displays marginal quantities (expectations).
The RPT_ data item identifies NONMEM's repeat data item. It is used to mark a data record as a repetition base. (Another way of doing this is via global "Repetition Variables" RPTI, RPTO, RPTON, PRDFL in abbreviated code.)
The REPL_ data item identifies NONMEM's replication data item. NONMEM replicates subjects from the template data set at the start of the problem. REPL_ may be used with the $DATA ... REPL option.

By using `ID`, `L1`, `L2`, `DV`, `MDV`, `RAW_`, `MRG_`, `RPT_`, or `REPL_` as B, the user defines the NONMEM data item type whose name corresponds to the label. The NONMEM data item type whose name corresponds to `ID` is the same NONMEM data item type whose name corresponds to `L1`, but `L1` has another special meaning. Use of `L1` not only defines the `ID` data item, it *also* supresses automatic generated ID data items (see section II.C.4).

By using `TIME` as B, the user defines a time data item type; time data items can be recognized as clock times and translated to relative times (see section II.C.2 and the discussion below).

By using `DATE`, `DAT1`, `DAT2`, or `DAT3` as B, the user defines a date data item type (see section II.C.2 and the discussion below). Usually, date data items should not appear in the NONMEM data set (see below).

By using `DROP` as B, the user defines a data item type which will not appear in the NONMEM data set, e.g. `DATE=DROP`. This is the one label that can be used more than once in the $INPUT record. Ignoring items with this label, the total number of items in a NM-TRAN data record cannot exceed a certain limit, and, if the data set is single-subject, this limit includes generated ID data items (which actually do not appear in the NM-TRAN data set).

Starting with NONMEM 7.2, this limit is given by PD in resource/SIZES.f90. The default value is 50. PD may be changed using the $SIZES record. PD also specifies the maximum number of data items in the NONMEM data set. With previous versions, the maximum number of items in both NM-TRAN and NONMEM data sets was 20.

If the user prefers to label a NONMEM data item type or a time data item type with a label (A) other than the reserved one (B), he may use the item A=B. In this case A is called a synonym for B. Alternatively, he may use the item B=A, i.e. the order of the labels A and B are reversed in the item so that the reserved label comes first. Either of the labels A and B may be used in subsequent control records of the problem specification. However, only the synonym is used as a label in NONMEM output. Both A and B can be reserved labels. Thus one type of data item can serve simultaneously as another type, e.g. `ID=TIME`. Another example of this is `DATE=DROP`.

The items `DROP`, `A=DROP`, or `DROP=A` are all equivalent. The label `SKIP` acts just as does the label `DROP`. The label `DROP` may not be used when the $DATA record contains a format specification (see section B.5).

Time data items are translated from clock times to relative times when at least one time data item contains a colon (:). Time data items are also translated from clock times to relative times when the reserved label `DATE`, `DAT1`, `DAT2`, or `DAT3` is used. In this case the order of the fields (day, month, year) must be the same across all the date data items. This order corresponds to which label is used:

Label	Order
`DATE`	month day year
`DAT1`	day month year
`DAT2`	year month day
`DAT3`	year day month

If only one field is used, it is assumed to be the day field. If two fields are used, they are assumed to be month and day fields. When two or more fields are used, the user should be careful to use, for example, `DATE=DROP`; otherwise, use of the nonumeric separator will raise an error during the NONMEM run.

If the data set is single-subject, ID data items are automatically generated, unless the reserved label `L1` is used. `A=L1` or `L1=A` can also be used. Generated ID data items are assigned the label `.ID.` (i.e. `ID` surrounded by dots). This label can be used in subsequent NM-TRAN control records of the problem specification.

Changes to the $INPUT record may cause changes to generated codes. In this case care should be taken in using the previous load module.

`INPT` is an alias for `INPUT`.

III.III.B.3. $INDEX Record

`$INDEX` [$label_1$|$value_1$] [$label_2$|$value_2$] [$label_3$|$value_3$] ...

E.g.
```
$INDEX    DOSE  1
```

The labels are those of data items, as established by the $INPUT record. The values are integers no larger than the number of (non-`DROP`) items in the $INPUT record. Either (i) the index of the data items with label $label_I$, or (ii) $value_I$, whichever is chosen, is stored in INDXS(I). (The index of a data item is its position in the NONMEM data record.) For the above example, and using the $INPUT record shown in chapter I: INDXS(1)=2, and INDXS(2)=1.

This record is optional, and it need not appear unless some complete FORTRAN-coded subroutines are developed by the user, and at least one of these subroutines makes explicit use of the INDXS array.

INDXS is an array which is one of the arguments to certain user-supplied subroutines; see Guide I, section C.4.1, or Guide VI, sections III.C, IV.B, and VI.A. The array cannot be used with an abbreviated code. Therefore, the $INDEX record is not of interest to users of NM-TRAN who only use abbreviated codes.

In the above description INDXS(I) should be understood to always mean the Ith element of the INDXS array that is available to a user-written routine. With PREDPP the routine actually has access to only part of a larger INDXS array, only to elements 12-50 of the larger array. NM-TRAN stores the indices of certain data items of use to PREDPP in elements 1-11, but PREDPP renumbers elements 12-50 to 1-39 before passing INDXS as an argument to the routine.

`$INDXS` is an alias for `$INDEX`.

Commas between labels and values are optional.

III.III.B.4. $CONTR Record

```
$CONTR    DATA=([label1|0] [label2|0] [label3|0])
```

E.g.

```
$CONTR     DATA=(0,TYPE)
```

This record defines one to three types of data items defined in the $INPUT record to be made available to subroutines CONTR, CCONTR, and MIX in the DATA array.

This record is optional and need only appear if the CONTR, CCONTR, or MIX subroutine is user-supplied and uses data items stored in the DATA array.

$label_i$ is the label (or synonym) given in the $INPUT record for the ith data item type.

The CONTR and CCONTR subroutines are used to define an objective function which differs from the default (ELS) objective function. The MIX subroutine is used to describe the mixing parameter of a mixture model; see Guide VI, section III.L.2. See also $MIX in Chapter IV.L.
These routines are called with individual records. An array DATA is available in a NONMEM MODULE and changes value with each individual record. It contains up to three types of data items occuring in each observation record of an individual record. For any individual record, the data item occuring in the Ith observation record of the individual record, having the Jth label occuring in the DATA option of the $CONTR record, is available in DATA(I,J). If a `0` is used in the DATA option instead of a label, then a 0 is found in DATA(I,J).

Commas between labels and `0`'s are optional.

III.III.B.5. $DATA Record

$DATA [filename|*] [(format)] [*IGNORE*=c_1] [*NULL*=c_2]
[*IGNORE=(*list*)*...|*ACCEPT=(*list*)*...]
[*PRED_IGNORE_DATA*]
[**NOWIDE**|*WIDE*] [*CHECKOUT*]
[*RECORDS*=n_1|*RECORDS*=label]
[*LRECL*=n_2] [**NOREWIND**|*REWIND*]
[*NOOPEN*] [*LAST20*=n_3] [*TRANSLATE=(*list*)*]
[*BLANKOK*]
[*MISDAT*=r...]
[*REPL*=n...]

E.g. `$DATA DATAFILE`

This record gives the name of the file containing the NM-TRAN data set.

This record is required with the first problem specification, and with any problem specification where the NM-TRAN data set differs from that of the preceding problem specification.

The filename must be the first option on the record.

If the $DATA record is missing from the problem specification, the problem uses the same NONMEM data set as that used in the previous problem. With the previous problem the user might have modified data items from those found in the NONMEM data set at the beginning of that problem (e.g. the modification may have occurred in the simulation step, or in the initialization/finalization step - see Guide II, section D.2.2, Guide VI, section VI.A), and then the data set used in the current problem (NONMEM's internal copy of the data set) contains the modified data items. Using an asterisk in the $DATA record, in place of the file name, has the same effect as omitting the $DATA record. However, when one wants to use the `CHECKOUT` option, as well as use the NONMEM data set from the previous problem, the asterisk must be used. In this case no other options should be used.

The format refers to a FORTRAN format specification to be used by NM-TRAN to read the NM-TRAN data records. Note that this specification is to be enclosed in parentheses. Format codes F, E, and X may be used, but not I. The format will also be used by NONMEM to read the NONMEM data records, after it had been modified to account for generated data items. If a format is provided, the label DROP cannot be used, and the WIDE and NULL options may not be used. If a format is omitted, the NM-TRAN records can still be read, and a format specification which is appropriate for reading the NONMEM data records will be generated. In this case a NONMEM data set is found in FDATA (see section II.B).

The IGNORE option specifies records to be dropped from the NONMEM data set.

> When `IGNORE=`c_1 is coded and the character c_1 appears in column one of a (FORTRAN) record of the data set, this record does not appear in the NONMEM data set. Such records can serve as comment records in the NM-TRAN data set (see section II.C.3). The character c_1 can be any character other than a blank or semicolon. In the $DATA record it can be enclosed by single quotes or by double quotes, in which case a semi-colon is permitted. The default is `IGNORE=#`. That is, in the absence of IGNORE option, any record whose first character is # is treated as a comment record.
>
> IGNORE=@ signifies that any data record having an alphabetic character or @ as its first non-blank character (not just in column 1) should be ignored. Alphabetic characters are the letters A-Z and a-z. This permits a table file having header lines to be used as an NM-TRAN data set.

`IGNORE=(list)` is new with NONMEM 7.

"List" is a list of one or more data item labels, with logical operators and values, of the form "label=value", "label.EQ.value", "label.NE.value", "label.GT.value", "label.GE.value", "label.LT.value", and "label.LE.value". (FORTRAN 90 logical operators such as '==' '/=' '<' '<=' '>' '>=' " may also be used.) With NONMEM 7.3, "label.NEN.value" and "label.EQN.value" are permitted. (There is no FORTRAN 90 operator for this comparison.) If the logical operator is omitted, the default is "=". With each data record, the value of the data item with the given label and the value in the list are compared according to the logical operator, and if result is "true", the record is ignored, i.e. it is not included in the NONMEM data set (see example below). Such records are called "dropped records". With "=", "==", "/=', ".EQ." and ".NE.", the value in the data record and the value in the list are compared as character strings. Otherwise, they are converted to numeric and compared numerically. (This is the case with .NEN. and .EQN.) This comparison is made prior to time translation. Hence, the TIME item cannot be compared numerically if it contains non-numeric characters such as ":".

Note: if the data file is a table file from a previous NONMEM run, values that had been integers (0,1,..) in the original data file will be real values (0.000E+00, 1.000E+00, ...) in the table file. A comparison for equality or inequality should now be for the real value. E.g.
`IGNORE=(OCC==1.000E+00).`

A data item label along with a logical operator and value is called a condition. A list may contain several conditions; these should be separated by commas, and the list should be enclosed in parentheses. Up to 100 different conditions altogether can be specified. Multiple IGNORE options with different lists may be used. A list may span one or more NM-TRAN records. The use of "=" after IGNORE is optional, but parentheses are required with this form of IGNORE. Values may be alphabetic or numeric, and may optionally be surrounded by single quotes ' or double quotes ". Quotes are required if a value contains special characters such as =. However, a value may not contain spaces or commas. No format specification is permitted with this form of IGNORE.

A data item type may be dropped from the NONMEM data set by means of the DROP or SKIP synonym on the $INPUT record, after records are dropped due to a condition based on the data item type. E.g.,

```
$INPUT ... GEN=SKIP ...
$DATA file IGNORE=(GEN='M')
```

Records having GEN equal to 'M' will be dropped, and the GEN data item type will then be omitted from the NONMEM data set. A dropped data item may be any alphanumeric string (without a data item delimiter - a blank or a comma).

If there is more than one condition, then records satisfying at least one of these conditions will be dropped. In effect, the conditions for dropping a record are connected by the implied conjunction ".OR.". E.g.

```
IGNORE=(GEN.EQ.1,AGE.GT.60).
```

Records having GEN equal to 1 or AGE greater than 60 are dropped. All others are accepted.

`ACCEPT=(list)`

The ACCEPT list option is identical to the IGNORE list option, except that it specifies conditions for acceptance of records. An ACCEPT list cannot be used together with an IGNORE list.

E.g.

```
ACCEPT=(GEN.EQ.1,AGE.GT.60).
```

Records having GEN equal to 1 or AGE greater than 60 are accepted. All others are dropped.

Suppose it is desired that records be dropped that satisfy the logical ".AND." of several conditions. This can be implemented by using an ACCEPT list with the negations of the conditions. For example, suppose that records to be ignored are those having GEN=1 .AND. AGE > 60. This may

be done as follows:

```
ACCEPT=(GEN.NE.1,AGE.LE.60)
```

It is possible to implement more complicated expressions, e.g., with nested parentheses.
See Guide VIII Ignore_accept example.

The PRED_IGNORE_DATA option is new to NONMEM 7.5. Data records may be dropped using by PRED or PREDPP using the PRED_IGNORE_DATA block of abbreviated code. See Chapter IV.E.6. This option informs NONMEM that a PRED_IGNORE_DATA pass through the data set is required. If the abbreviated code uses variables PRED_IGNORE_DATA or PRED_IGNORE_DATA_TEST, this option is supplied. The explicit option is needed if the use of the PRED_IGNORE_DATA variables occurs in user-written or verbatim code, so that NMTRAN is unaware of it.
(See Guide Introduction_7 "PRED_IGNORE_DATA Feature (NM75)")

Option `NULL=`c_2 specifies the character used for null data values in the NONMEM data set. Every occurrence in the NM-TRAN data set of a dot surrounded by blanks/commas/tab characters, or of consecutive commas or consecutive tab characters, is replaced by a null data item in the NONMEM data set. Null data items are also supplied for data items that are missing from the end of an NM-TRAN data record (there may be several missing from the end of any given record) but that are defined in the $INPUT record. This null data item consists of the specified character c_2, and it occupies the last position in the field of the NONMEM data set into which it is placed. The character c_2 can be enclosed by single quotes or by double quotes. The default for c_2 is a blank.

The option `NOWIDE` specifies that the FORTRAN records of the NONMEM data set, generated by NM-TRAN, are to consist of up to 80 characters. In order to accomplish this NM-TRAN may need to suppress space between fields or generate multi-line NONMEM data records. This is the default. The option `WIDE` specifies that single-line NONMEM data records are to be generated if possible. These records always include at least one space between fields. With this option the records can be up to 300 characters. (Note that they will be multi-line if 300 characters is too few.) They comprise a NONMEM data set which may be more legible than one obtained without using `WIDE`, and which can be processed by a program other than NONMEM. With this option, there will be no FINISH (FIN) record in the NONMEM data set. (A FINISH record is generated with there are more than 99999999 records in the NONMEM data set.) `WIDE` also provides an extra character for elapsed times, so that they can accommodate the number of hours in (approximately) 4166 days with a leading space. Six digits (xxxxxx.xx) are provided.

The option `CHECKOUT` specifies that NONMEM is to run in data checkout mode. In this mode, tables and scatterplots can be requested, so that the data (as "understood by NONMEM") can be examined, but no computations involving the model are performed, so that this examination cannot be hindered by problems with the logic in user-written routines or abbreviated codes. These routines and codes need only be syntactically correct. The predictions, residuals, weighted residuals, `ETA`'s and PRED-defined items that are displayed in tables and scatterplots (see sections B.16 and IV.F) are all 0. `CHECKDATA` can be used as an alias.

When the `RECORDS=`n_1 option is used, the number n_1 is the number of data records to be read from the NM-TRAN data set. Comment records are not counted. If NM-TRAN does not drop any records from its data set (see IGNORE list and ACCEPT list), then n_1 is also the number of records written to the NONMEM data set. If NM-TRAN drops records, then the total number of records written to the NONMEM data set is n_1 minus the number of dropped records. This number must be no more than 99,999,999. When the option is omitted, then there is no limit to the number of data records in the NM-TRAN data set. In this case the file is read to a FINISH record (see below), or to the physical end of file, whichever comes first. However, if there are 100,000,000 or more records, then it is suggested that a FINISH record be used since in this case and when, moreover, the NONMEM data set happens to coincide with the NM-TRAN data set, the NONMEM data set needs a FINISH record.

RECORDS=label

If the option is coded as RECORDS=label, where label is a data item label, NM-TRAN understands the data records for the problem to start with the first data record of the NM-TRAN data set (at the place where the file is positioned before data records are read; see the NOREWIND option), and to include as well, those and only those subsequent contiguous data records having the same value of the data item as does the first record. It counts the total number of these data records, minus any comment or dropped records, and puts this number in the NONMEM control file.

In particular, the ID label may be used (or alternatively, the option may be coded RECORDS=IR, RECORDS=INDREC, or RECORDS=INDIVIDUALRECORD). If a label other than ID is used, the $INPUT record must precede the $DATA record. If the data are single-subject data, the ID data items used to determine the data records for the problem are those labeled ID (not .ID.).

If there is more than one problem specification with a $DATA record that includes an option of the form RECORDS=label, then either none of these $DATA records may also include a format specification, or all of them must include the same format specification.

The LRECL option is only needed when the format is omitted, and when either (i) the operating system (e.g. IBM/CMS) raises a fatal error when a FORTRAN program tries to read more characters from a logical record than the number of characters in the record, or (ii) the operating system imposes a maximum record length which is smaller than 999 characters (e.g. CRAY/CTSS). The number n_2 is the number of characters in the NM-TRAN data records.

NOREWIND specifies that the file is not to be rewound before it is read, and REWIND specifies that the file is to be rewound. These options are ignored if used on the $DATA record appearing in the first problem specification of the NM-TRAN control stream, or on a $DATA record appearing in a subsequent problem specification when this record contains a file name different from that contained on the $DATA record of the prior problem specification. In these cases the file is automatically rewound. The REWIND and NOREWIND options are significant only when there are multiple problem specifications in the NM-TRAN control stream, and when the $DATA records appearing in two consecutive problem specifications, corresponding to two problems A and B, contain the same file name. In this situation:

When the REWIND option is used on the the $DATA record for problem B, the first NM-TRAN data set on the file is re-used for problem B. If NM-TRAN does not modify this data set, then an instruction to rewind the (same) file is also contained in the NONMEM control stream problem specification for problem B. If NM-TRAN does modify the data set, then the NONMEM data set for problem B is placed on FDATA after the last NONMEM data set already present on FDATA, and no instruction to rewind FDATA is contained in the NONMEM control stream problem specification for problem B.

If the NOREWIND option is used on the $DATA record for problem B, or neither option is used, then the file is not rewound, and the NM-TRAN data set on this file that follows the one used for problem A is used for problem B. In this case note that the $DATA record with problem A must have contained the RECORDS option or the NM-TRAN data set used for problem A must end with a FINISH record. Also in this case, no instruction to rewind the file containing the NONMEM data set for problem B (whether this file is the file named in the $DATA record or whether it is FDATA) is contained in the NONMEM control stream problem specification for problem B.

NOOPEN specifies that the file is not to be opened by NM-TRAN. It permits a data file to be created by one problem and used in a subsequent problem of the same run.

Translation of NM-TRAN data records is a slower process than is translation of NM-TRAN control records. However, usually during a data analysis, changes are made to the NM-TRAN control records between runs, but not to the NM-TRAN data records. With large data sets once translation of the data records has been performed successfully, the output in FDATA can be stored (in a file of a different name) and used with subsequent NM-TRAN runs. In a subsequent run, use the format specification and

value for n_1 found in the problem summary pages of the NONMEM output from the first run. If data items were dropped or generated, then use the list of "LABELS FOR DATA ITEMS", found in this earlier output, for the list of labels needed in the $INPUT record. If the data set is single-subject, and ID data items were generated, use L1 for the label of these data items.

`LAST20`, `BLANKOK`, `TRANSLATE`

See Chapter II

`MISDAT`=r...(nm74) Defines one or more particular numerical values to indicate a missing data value in the data set, which is displayed on $TABLE outputs, but is safely interpreted as 0 by other steps of NONMEM.

`REPL=n` (NM75) When the REPL=n option of $DATA is coded, the NONMEM data set is considered to be a template data set. NONMEM itself replicates the template data set n times at the start of the problem to create an expanded NONMEM data set. The REPL option may be used with the REPL_ data item. If both are used the REPL_ data item applies first, and the REPL option applies second. The REPL option and REPL_ data item are meant to be used with $SIMULATION or $DESIGN.
(See Guide Introduction_7 "$DATA REPL (NM75)")

See Introduction to NONMEM 7 and Guide VIII and on-line help.

`$INFILE` is an alias for `$DATA`.

When a format is omitted, a FINISH record consists of the characters FIN appearing *anywhere* in the record (the other characters are all blank). When a format is provided, a FINISH record must have the form described in Guide II, section D.2.3.

III.III.B.6. $SUBROUTINES Record

$SUBROUTINES [subname1 = name1] [subname2 = name2] ...
[*SUBROUTINES*=kind]

E.g.
```
$SUBROUTINES    PRED=mypred
```

This record gives names associated with any subroutines which are user-supplied, i.e. for which abbreviated codes are not given. User-supplied FORTRAN codes are copied to the FSUBS file. The names and subnames are also listed in the FREPORT file. This file is needed for documentation purposes, and it also can be used as input to a program such as nmfe that creates system commands for running NONMEM.

This record is only required with the first problem specification, and then only if the record contains some option. It applies to all problem specifications in the control stream, and it must not appear with a problem specification other than the first.

A subname may be chosen from the list: `PRED`, `CRIT`, `CONTR`, `CCONTR`, `CONPAR`, `USMETA`, `SPTWO`, `MIX`, `PRIOR`

The names on the right are the names of files containing FORTRAN 90/95 code. These follow the usual rules for filenames on control records; see A.10 above. An extension such as ".f" or ".f90" may be part of the file name for descriptive purposes, but the file itself should contain FORTRAN 90/95 code. A file may contain more than one FORTRAN subroutine. A name could be the same as the subname, but a subname is generic, and a name which is more specifically related to the actual code comprising the subroutine can be more useful. One subname-name pair should be used for each subroutine which is user-supplied.

If the PRED subroutine is not user-supplied, so that `PRED` is not used as a subname, then an abbreviated code must be given for $PRED (unless PREDPP is used; see chapter V). Abbreviated code can also be given for $MIX; see Chapter IV.L. The $PRIOR record can be used to provide instructions for a generated PRIOR subroutine in FSUBS; see B.23 below. FORTRAN code must be provided if the remaining subnames are user-supplied.

There may exist one or more "other" user-supplied subroutines used by one of the subroutines listed above. The special subname `OTHER` can be used in conjunction with such a routine; it is set to a name of a file to be read by NM-TRAN. (A subroutine B, called by subroutine A, can be included in the file containing A. In this case `OTHER` would not be set to a name for B.) Unlike the other subnames, `OTHER` can appear up to forty times in the $SUBROUTINES record, and used for up to forty unique file names. Here are some examples.

The `OTHER` option is used to supply the FORTRAN code for user-written functions, e.g., the reserved functions FUNCA, FUNCB, FUNCC, etc., or, with NONMEM 74, functions declared using the $ABBR FUNCTION option. Suppose there is one such file and its name is funcfile. (File funcfile may contain more than one FUNCTION.) It should be listed on the $SUBROUTINES record. E.g.,
```
$SUBROUTINES ... OTHER=funcfile
```

Another example of the use of `OTHER` has to do with the NONMEM subroutine CONSTRAINT. The default version found in NONMEM's source directory (source/CONSTRAINT.f90) performs simulated annealing, and uses information from the $ANNEAL record. See section B.29. below. The OTHER option is not needed if the default version is to be used. See anneal.ctl in NONMEM's example directory.

A user-written subroutine CONSTRAINT may be used instead to provide any kind of constraint pattern on any parameters. Example9.ctl in NONMEM's examples directory contains

```
$SUBROUTINES ... OTHER=aneal.f90
```

File aneal.f90 contains an alternative version of subroutine CONSTRAINT and is discussed in Introduction to NONMEM 7 "Example 9: Simulated Annealing For Saem using Constraint Subroutine."

If the SUBROUTINES option is not used, it is understood to be present with the option "kind" equal to DOUBLE. DP and D are aliases for DOUBLE. SUBS is an alias for SUBROUTINES. The SUBROUTINES option may be coded with the part SUBROUTINES= missing.

E.g.

```
$SUB  DP
```

III.III.B.7. $ABBREVIATED Record

$ABBREVIATED [*COMRES*=$n_1$] [*COMSAV*=$n_2$]
[*DERIV2=NO*] [*DERIV2=NOCOMMON*] [*DERIV1=NO*]
[*FASTDER* | *NOFASTDER*]
[*CHECKMU* | *NOCHECKMU*]
[*DES=COMPACT* | *DES=FULL*]
[*REPLACE* left_string = right_string] ...
[*DECLARE* [type] [DOWHILE] name [(dimension [,dimension])] ...
[*PROTECT*]
[*FUNCTION function_name(input_vector_name,dimension[,usage])*]
[*VECTOR input_vector_name(dimension)*]

E.g. `$ABBREVIATED COMRES=2`

Optional. May be used when $PK, $ERROR, or $PRED abbreviated code is present. Must precede all blocks of abbreviated code. appear. Then it only should appear with the first problem specification, and before any abbreviated code. With NONMEM 7.4, may also be used when there is no abbreviated code. For example, $ABBR REPLACE may be used for label substitution in NONMEM report files.

Abbreviated code is described in Chapters IV and V.

`COMRES` ("common reserve") and `COMSAV` ("common save")

One use of the option `COMRES` involves the presence of both abbreviated and user-supplied subroutines. Values for variables defined in abbreviated codes may be displayed in tables and scatterplots. These variables are listed (i.e. their values are stored) in a NONMEM MODULE NMPRD4 in the generated subroutine. (With versions of NONMEM prior to nm7, NMPRD4 was a named FORTRAN common, hence the use of options "COM...".) User-supplied FORTRAN routines may also list variables whose values are to be displayed in NMPRD4. The first n_1 positions in NMPRD4 are reserved for the use of these routines; generated subroutines will not list variables defined in abbreviated codes in these positions. If there are no abbreviated codes, not only is `COMRES` not needed for this purpose, it may not be used. All variables listed in NMPRD4 must be double precision. The number n_1 can be nonnegative, and if the option is not used, it is understood to be 0. It can also be -1, in which case no variables defined in abbreviated codes are listed in NMPRD4. In this regard, see also section IV.H.

All variables defined in an abbreviated code are listed in NMPRD4, whether or not their values are displayed. Thus with PREDPP, where there may be more than one abbreviated code, each such variable is recognized as the same variable in all abbreviated codes in which it is used. If it desireable to avoid this type of global definition, then the option value -1 can be used. See also section IV.H.

The option `COMSAV` may be used when the variable `COMACT` is used in abbreviated code (see section IV.E.2). It defines the SAVE region of NMPRD4. In this case the option `COMRES` should also be used, and n_2 must satisfy $0 \le n_2 \le n_1$.

The next two options primarily concern ways to avoid changing and recompiling NM-TRAN or NONMEM source code when NM-TRAN produces error messages indicating that certain internal table sizes are found to be too small for a given problem; see Guide III. In this regard, the option `COMRES=-1` can also be useful (see also section IV.H). With NONMEM 7, NM-TRAN allocates most internal arrays dynamically as needed. The $SIZES record can be used to make other arrays larger. Internal table sizes are unlikely to be exceeded. However, when there is a great deal of abbreviated code, the generated code and the NONMEM load module may be very large, which can slow the execution of NM-TRAN and NONMEM. The following options may still be useful.

`DERIV2=NO` and `DERIV2=NOCOMMON`

When the option `DERIV2=NO` is used, code to compute second-partial derivatives with respect to η variables is not generated from abbreviated codes. These derivatives are only needed when the Laplacian estimation method is used (see section B.14). So in order to save CPU time one might be tempted to use this option when the Laplacian method is not used. However, when the Laplacian method is not used, the code computing the second derivatives is never executed. Therefore usually, there is little reason to include the option. It is generally recommended that the option not appear, so that with generated subroutines, a load module that computes these derivatives can also be used in a subsequent run which uses the Laplacian method. If it does appear, if PREDPP is not used, and if the Laplacian estimation method is requested in a subsequent run using the same load module, then, in effect, the first-order conditional estimation method is used in that run, although using somewhat more computer time than is necessary. However, care should be taken to avoid this situation. If the option appears, and PREDPP is used, then the Laplacian estimation method must not be requested in a subsequent run using the same load module, even when all second-partial derivatives that might be computed are uniformly zero (in which case code computing these zeros is actually not generated whether or not the option appears).

Values of second-partial derivatives of a variable defined in an abbreviated code are stored in other variables defined in the generated routine. Normally, these variables are listed are also listed in NMPRD4 (see above). When the option `DERIV2=NOCOMMON` is used, these variables are not listed in NMPRD4. In this case these variables are not displayable. In this case also, if PREDPP is used, *no variable defined in the abbreviated code for PK*, may be referenced in the abbreviated code for ERROR.

In the generated subroutine, NM-TRAN collects (re-arranges) all code for second derivative computation so that only a few tests (for MSEC==1; see Chapter IV.E.4) are needed to compute them when NONMEM indicates that it needs second-partial derivatives, and skip them otherwise.

`FASTDER` and `NOFASTDER` (nm72)

Code to compute first-partial derivatives with respect to η variables is always generated from abbreviated codes. These derivatives are almost always needed by classical NONMEM methods with population data. However, NONMEM does not always use these derivatives for the newer Bayesian methods. Since NONMEM 7.2, NM-TRAN collects (re-arranges) all code for first derivative computation so that only a few tests (for FIRSTEM==1; see Chapter IV.E.4) are needed to compute them when NONMEM indicates that it needs first-partial derivatives, and skip them otherwise. This is explicitly requested by option `FASTDER`, which is the default. The collection (re-arrangement) of first derivative code can be prevented with option `NOFASTDER`.

`DERIV1=NO` (nm74)

With NONMEM 7.4, `DERIV1=NO` prevents the computation of first derivatives.

`CHECKMU` and `NOCHECKMU` (nm73)

Abbreviated code may contain statements for the MU model, which is used in NONMEM 7 EM (Expectation Maximization) methods and Gibbs sampling methods. See Chapter IV K.3. With NONMEM 7.3, NM-TRAN checks the MU model statements and issues warning messages if they appear to contain mistakes. This can take a long time for large control streams. For some models, NM-TRAN may be unable to do so, or may issue inappropriate warnings. Option `NOCHECKMU` can be used to prevent NM-TRAN from attempting to check the MU model statements. Option `CHECKMU` requests that MU model statements be checked, and is the default. Neither option affects the generated code.

`DES=FULL`

Requests that arrays of the DES routine are stored in non-compact form. With $ESTIMATION METHOD=COND LAPLACIAN, the option NUMERICAL is also required. `DES=FULL` is the default with ADVAN9 and ADVAN15 and ADVAN17. (Prior to NONMEM 7.4, FULL was required with ADVAN13.)

`DES=COMPACT`

Arrays of the DES routine are stored in compact form. Required with Laplacian method; optional otherwise. This is the default, except with ADVAN9 and ADVAN15 and ADVAN17.

`$ABBREVIATED REPLACE` option (nm73)

Any character string may be replaced in abbreviated code. In particular, this allows for symbolic labeling of thetas, etas, and epsilons. For example,

```
$ABBR REPLACE ETA(CL)=ETA(5)
```

The characters `ETA(CL)` are replaced by `ETA(5)` where ever they appear in abbreviated code.
See Guide VIII for more features of $ABBR REPLACE.
See $OMEGA and $SIGMA record for an alternate way of naming the ETA and EPS using NAMES and VALUES options.

With NONMEM 74, use of this option may affect the NONMEM report file, even when there is no abbreviated code.

Either ETA(CL) or ETA5 may be listed in the $TABLE or $SCATTER records, and the label in the NONMEM report file is "`ETA(CL)`". This is called label substitution.

The $ESTIMATION, $TABLE, $SCATTER, and $DEFAULT records all have an option `NOSUB` that can be used to control label substitution for that portion of the report. In general, with `NOSUB=0`, label substitution occurs. E.g., the label in the NONMEM report file is "`ETA(CL)`". This is the default. With `NOSUB=1`, label substitution is turned off. E.g., the label is "`ETA5`".

Label substitution is never made in the additional output files *.ext, ,phi, etc., to maintain their third party software readability.

Compartment names may be explicitly replaced. For example,

```
$ABBR REPLACE A(DEPOT)=A(1)
$ABBR REPLACE DADT(DEPOT)=DADT(1)
  ...
$DES
 DADT(DEPOT)=-KA*A(DEPOT)
```

This is called "explicit" compartment name substitution. With NONMEM 7.5, compare the "implicit" compartment name substitution feature of the $MODEL record.

The REPLACE option also allows replacement with selection. This provides a compact way of writing complicated abbreviated code. Here is an example of replacement with selection by data item. Suppose OCC is a data item.

```
$ABBR REPLACE THETA(OCC)=THETA(4,7)
```

Supppose the abbreviated code is:

```
TVCL=THETA(OCC)
```

The generated code is:

```
IF (OCC==1) TVCL=THETA(4)
IF (OCC==2) TVCL=THETA(7)
```

Here is an example of replacement with selection by data item and parameter:

```
$ABBR REPLACE THETA(SID_KA)=THETA(4,6)
$ABBR REPLACE THETA(SID_CL)=THETA(5,7)
  ...
$PK
KA=THETA(SID_KA)
```

```
CL=THETA(SID_CL)
```

The generated code is

```
IF (SID==1) KA=THETA(4)
IF (SID==2) KA=THETA(6)
IF (SID==1) CL=THETA(5)
IF (SID==2) CL=THETA(7)
```

A short-hand notation may be used to describe a series of values, e.g.,

```
$ABBR REPLACE THETA(SID_KA)=THETA(,4 to 13 by 3)
```

is equivalent to

```
$ABBR REPLACE THETA(SID_KA)=THETA(4,7,10,13)
```

`$ABBREVIATED DECLARE` option (nm73)

Integers and arrays may be declared and used in abbreviated code:

```
$ABBR DECLARE DOSE(100),DOSETIME(100)
$ABBR DECLARE INTEGER I
$ABBR DECLARE DOWHILE I
```

One or names may be coded. They are referred to as declared variables. If INTEGER or DOWHILE is coded, the type of the variable is integer. Otherwise, the type of the variable is double precision. If one or two dimensions are declared, the variable being declared is an array. Declared variables are global, i.e., are defined in all blocks of abbreviated code. Declared variables that are not INTEGER or DOWHILE will be random variables if they are assigned in a statement whose right-side involves ETA's or EPS's. Declared variables are not known to or used by NONMEM or PREDPP.

See IV.K.2 for an example.

`$ABBREVIATED FUNCTION` option (nm74)

In NONMEM 7.4 the $ABBR FUNCTION option allows user-defined function names and user-defined argument vector names. The dimensions of the argument vector and the maximum number of times a given function name may appear in abbreviated code is user-specified.

See Chapter IV Section IV.J.7 $ABBR FUNCTION and $ABBR VECTOR

`$ABBREVIATED VECTOR` option (nm74)

In NONMEM 7.4 the $ABBR VECTOR option allows user-defined vector names to be defined independently of any function.

See Chapter IV Section IV.J.7 $ABBR FUNCTION and $ABBR VECTOR

`$ABBREVIATED PROTECT` option (nm74)

With NONMEM 7.4, a series of routines are available that protect against domain violations, divide by zero, and floating point overflows. Each of these routines start with the letter P, followed by the name of the mathematical operation they are to perform. For example, PLOG is the protective code routine that performs the LOG operation. With $ABBR PROTECT, NMTRAN will automatically replace all relevant function names with the P name.

See Chapter IV Section IV.J.6. PROTECT functions

III.III.B.8. $PRED Record

`$PRED`
: the abbreviated code

This record gives an abbreviated code for the PRED routine. The syntax of an abbreviated code is described in chapter IV.

This record is optional. If it appears, it must be with the first problem specification, and only with this problem specification.

III.III.B.9. $THETA Record

$THETA $value_1$ [$value_2$] [$value_3$] ...
[($value_k$)xn]
[label=value... FIXED]
[NAMES (label ...)value ...]
[*NUMBERPOINTS*=n]
[**ABORT**|*NOABORT*|*NOABORTFIRST*]

E.g. `$THETA (.1,3.,5.) (.008,.08,.5) (.004,.04,.9)`

This record gives initial estimates for θ's, as well as bounds on the final estimates.

This record is required only if the statistical model contains θ parameters (most models do). When a $MSFI record appears in the problem specification, the $THETA record should not appear.

A value has one of four forms:

1 init [`FIXED`]
2 (low, init, [up] [`FIXED`])
3 (low,,up)
4 (value)xn

where init is the initial estimate, and low and up are lower and upper bounds respectively. The lower bound can be `-INF`, i.e. $-\infty$, and the upper bound can be `INF`, i.e. ∞, unless form 3 is used, in which case both bounds must be finite numbers. Form 3 is used when the user requires some help in obtaining an initial estimate for the parameter. Usually, though, the user should be able to develop a reasonable initial estimate, and when he can, there is some savings in computation time. An initial estimate equal to 0 is not allowed, unless the `FIXED` option is used. Use of this option indicates that the final parameter estimate is to be constrained to equal the initial parameter estimate. If this option is used with form 2, then low, init and up must all be equal.

Another example:

```
$THETA  3 FIXED  (-INF,.08,.5) (.004,,.9)
```

where the three forms used are 1,2 and 3, in that order.

Parentheses around init with form 1 are optional. The designation `INF` can also be coded `INFINITY`, `INFIN`, or `1000000`. The character '+' can precede `INF`, as can the character '−'. Integers need not have decimal points.

With NONMEM 7.2, form 4 may be used. Any initial value or group of initial values may be enclosed in parentheses and followed by "xn", which means to replicate the values within parentheses n times ("repeated value"). The values within the parenthesis may have any of the above forms. For example, the following two are equivalent:

```
$THETA 2 2 2 2 (0.001,0.1,1000) (0.001,0.1,1000) (0.001,0.1,1000)
       (0.5 FIXED) (0.5 FIXED)
$THETA (2)x4 (0.001,0.1,1000)x3 (0.5 FIXED)x2
```

If form 3 is used, a search for an initial estimate is undertaken by NONMEM (not NM-TRAN). A number of points in a subspace of the θ parameter space will be examined. This subspace consists of the multidimensional rectangle formed by the lower and upper bounds for all parameters whose values are of form 3. The number of points examined will be automatically determined by NONMEM, or it can be specified by the number n with the `NUMBERPOINTS` option. The options `ABORT` and `NOABORT` and `NOABORTFIRST` apply during the search. For information concerning these option, see section IV.G.

Aliases for `NUMBERPOINTS` are: `NUM`, `NUMPTS`, `NUMBERPTS`. This option can occur at the end of the record, or at the beginning, or between two values. `THTA` is an alias for `THETA`.

With NONMEM 7.4, when initial thetas are to be estimated, evaluations can now be done for FOCE and LAPLACE, not just for FO.

Commas between values are optional, except with form 3.

Records $THETAI and $THETAR may be used to generate subroutines that transform the initial and final values of THETA. See Chapter IV Sections M and N.

With NONMEM 7.5, the $THETA record may specify symbolic label substitution. For example,

```
$THETA CL=(0.0,7.0) V1=(5.0 fixed)
```

This is equivalent to

```
$ABBR REPLACE THETA(CL)=THETA(1)
$ABBR REPLACE THETA(V1)=THETA(2)
$THETA (0.0,7.0) (5.0 fixed)
```

The symbolic subscript may be used for THETA in abbreviated code, and will also identify this element of THETA in the NONMEM output. (Only the first 9 characters of the label will appear).

With NONMEM 7.5, the $THETA record can define one or more thetas and initial values in a compact way. For example,

```
$THETA NAMES(V1,CL,Q,V2) (0.0,7.0) (0.0,7.0) (0.0,7.0) 7
```

III.III.B.10. $OMEGA Record

```
$OMEGA [DIAGONAL(n) | BLOCK(n) | BLOCK(n) SAME(m) | BLOCK SAME(m)]
       [[value1]  [value2]  [value3] ...
       [(value,value...) xn]
       [BLOCK(n) VALUES(diag,odiag)]
       [label=value] ...
       [BLOCK(n) [NAMES (label1,...,labeln)] [VALUES (diag,odiag)]
       [FIXED] [UNINT]
       [VARIANCE | STANDARD] [COVARIANCE | CORRELATON] [CHOLESKY]
```

E.g.

```
$OMEGA BLOCK(3)  6. .005 .3 .0002 .006 .4
```

This record gives initial estimates for elements of the Ω matrix, i.e. the variances and covariances of the η variables in the statistical model. Constraints on Ω are also indicated.

This record should appear only if the statistical model contains η variables. If it appears, then there must be one such record corresponding to each block of Ω, and the order of these records in the control stream must correspond to the order of the blocks. The values in an $OMEGA record are the initial estimates for the elements of the corresponding block. Under some circumstances $OMEGA records may or may not appear, and the equivalent effect is achieved in either case (see below). When a $MSFI record appears, no $OMEGA records should appear. When PREDPP is used, and a $PK record does not appear, while a $ERROR record does appear, certain care must be taken with the $OMEGA record; see section V.6.

Ω can be considered to be in block diagonal form with blocks B_1, B_2, ..., B_m (submatrices of Ω):

$$\begin{pmatrix} B_1 & 0 & & 0 \\ 0 & B_2 & & 0 \\ & & \cdots & \\ 0 & 0 & & B_m \end{pmatrix}$$

This form need not be unique, and there may be only one block (which is most usual). The following description applies to the ith block, $i = 1, \cdots, m$.

If the `DIAGONAL` option is used, it must precede the values. In this case B_i is constrained to be diagonal, and the values are the initial estimates of its diagonal elements given in the diagonal order. The number n is the dimension of B_i. In addition, a final estimate of an individual element of a diagonal block can be constrained to be equal to the initial estimate of the element by using the `FIXED` option. The value giving the initial estimate should be coded with any one of the forms:
init `FIXED`
(init `FIXED`)
(`FIXED` init)

If the `BLOCK` option is used, it must precede the values. In this case the form of B_i is unconstrained, and the values are the initial estimates of its lower triangle elements given in row-wise order. The number n is the dimension of B_i. If `FIXED` option is used, *all* the final estimates of the elements of B_i are constrained to be equal to the initial estimates of these elements.

If the `BLOCK(n) SAME` or `BLOCK SAME` option is used, B_i is constrained to be equal to B_{i-1}. In this case i must be greater than 1, the number n (if it is given) must be the dimension of B_{i-1}, and values are not given.

With NONMEM 7.3, SAME(m) is permitted . If m is present, then this record is equivalent to m identical records without (m). E.g.,

```
$OMEGA BLOCK(2) SAME(3)
```

is equivalent to

```
$OMEGA BLOCK(2) SAME
$OMEGA BLOCK(2) SAME
$OMEGA BLOCK(2) SAME
```

If some of the values in a record are omitted (other than a record with the `BLOCK(n) SAME` or `BLOCK SAME` option), then *all* values in the record must be omitted, and NONMEM will try to obtain initial estimates for the elements of the block. In this case, if the `DIAGONAL` option is used, it must appear explicitly in the record. Often, though, the user should be able to develop reasonable initial estimates, and when he can, there may be a little savings in computation time.

If no $OMEGA records appear, if no $MSFI record appears, but η variables are used in an abbreviated code, then it is assumed that a record

```
$OMEGA  DIAGONAL(n)
```

where n is the largest index used with an η variable in all abbreviated codes, might have equivalently been used. (If, though, with PREDPP an abbreviated code for PK is not used, while an abbreviated code for ERROR is used, see section V.6.)

With the `BLOCK` option, the `FIXED` option can occur anywhere among the list of values. These values (of the lower triangle) are given in row-wise order, i.e. $B_{i,11}$, $B_{i,21}$, $B_{i,22}$, ..., $B_{i,n1}$, $B_{i,n2}$, ..., $B_{i,nn}$.

Commas between values are optional.

An initial estimate of an element of a diagonal block must be >0, or it can be 0 if the `FIXED` option is used with it. An initial estimate of a non-diagonally-constrained block must be positive definite, or it can be (uniformly) 0 if the `FIXED` option is used with it. (In any case the initial estimates of Ω and Σ cannot both be 0 unless the Simulation Step is the only step implemented.)

The NM-TRAN translation of $OMEGA records is such that new blocks in addition to B_1, B_2, etc. may be created. This should be transparent to the user, except that he will see additional blocks listed in the problem summary output by NONMEM.

With NONMEM V, an initial estimate of a diagonal block of either the OMEGA or SIGMA matrices may have a band symmetric form, in which case the final estimate has the same form. That is, given the diagonal and a group of contiguous subdiagonals symmetrically ocurring across the diagonal, the elements off both the diagonal and the subdiagonals are constrained to be zero. To specify the initial estimates of such a block, the initial estimates of those elements that are to be constrained to 0 should be given as 0, while all other initial estimates should be given as nonzero. E.g., with these structures for $OMEGA BLOCK(3), the 0's are preserved:

```
 x
 0x
 00x

 x
 xx
 0xx
```

With NONMEM 7.3 if the initial estimate of a block is not positive definite because of rounding errors, a value will be added to the diagonal elements to make it positive definite. A message in the NONMEM report file will indicate that this was done. E.g.,

DIAGONAL SHIFT OF 1.1000E-03 WAS IMPOSED TO ENSURE POSITIVE DEFINITE-NESS

With NONMEM 7.3, (value)xn is permitted, so that repeated inputs of $OMEGA may be entered easily. Any initial value or group of initial values may be enclosed in parentheses and followed by "xn" which means to replicate the items n times ("repeated values"). The item to be repeated must always be in parentheses, and the xn must always be immediately after the item, not before it (4x(0.2) is not permitted). Here is an example:

```
$OMEGA BLOCK(6)
0.1
0.01 0.1
(0.01)x2 0.1
(0.01)x3 0.1
(0.01)x4 0.1
(0.01)x5 0.1
```

With NONMEM 7.3, new options are available.

`$OMEGA BLOCK(n) VALUES`(diag,odiag)

This supplies initial values for a block such that the initial estimates of the diagonal elements are all the same, specified by "diag", and the initial estimates of the off-diagonal elements are all the same, specified by "odiag". If present, `VALUES` must follow `BLOCK`.

The above example could be coded

```
$OMEGA BLOCK(6) VALUES(0.1,0.01)
```

For fixed block (such as for omega priors):

```
$OMEGA BLOCK(6) FIX VALUES(0.15,0.0)
```

The following options may follow `VALUES` or be placed between `BLOCK` and `VALUES`.

`VARIANCE` indicates that all initial estimates given for diagonal elements are understood to be initial estimates of variances of etas. This is the default.

`STANDARD` indicates that all initial estimates given for diagonal elements are understood to be initial estimates of standard deviations of etas. May also be coded `SD`.

`COVARIANCE` indicates that all initial estmates given for off-diagonal elements are understood to be initial estimates of covariances of etas. This is the default.

`CORRELATON` indicates that all initial estmates given for off-diagonal elements are understood to be initial estimates of correlations of etas.

`CHOLESKY` indicates that the block is specified in its Cholesky form.

Options `VARIANCE` or `STANDARD` may be combined with `COVARIANCE` or `CORRELATON`.

Note that NONMEM converts all initial estimates to variance and covariances. The values desplayed in the NONMEM report and in the raw and additional output files are always variances and covariances.

With NONMEM 7.5, the $OMEGA record may specify symbolic label substitution. For example,

```
$OMEGA label=value (NM75)
```

This is a compact method of defining an ETA (an element of OMEGA) specifying its initial estimate, and specifying a label for the subscript for this element of OMEGA. The label may be used as a subscript for

ETA in abbreviated code, and will also identify this element of OMEGA in the NONMEM output. If new $OMEGA records change the ordering, the abbreviated code does not have to be changed. For example, suppose the first element of OMEGA that is defined happens to be

```
$OMEGA ECL=.4
```

The NONMEM report will describe the relationship, e.g.,

```
LABELS FOR ETAS
ETA(1)=ETA(ECL)
```

and ETA(CL) rather than ETA1 will appear in the NONMEM report. The abbreviated code can use this symbolic subscript instead of the numeric subscript. Then, these take effect on both ETA's and MU_'s.

For example, suppose the following code is present for the first elements of THETA and ETA. Note that $OMEGA and $THETA records must be placed ahead of any records that use the symbolic label.

```
$THETA CL=(0.0,7.0)
$OMEGA ECL= 0.3
$PK
MU_ECL=THETA(CL)
CL=EXP(MU_ECL+ETA(ECL))
```

This is equivalent to

```
$THETA (0.0,7.0)
$OMEGA .3
$PK
MU_1+THETA(1)
CL=EXP(MU_1+ETA(1))
```

Another example defines symbolic labels for a block of OMEGA:

```
$OMEGA BLOCK(4)
ECL=  0.3
EV1=  0.01 0.35
EQ=   0.01 0.01 0.54
EV2=  0.01 0.01 0.01 0.67
```

Or, for diagonals,

```
$OMEGA
ECL= 0.3
EV1= 0.35
EQ=  0.54
EV2= 0.67
```

With NONMEM 7.5, Symbolic label substitution may be specified for an entire block using the NAMES option.

$OMEGA *BLOCK(*n*) NAMES* (label$_1$,...,label$_n$) *VALUES* (odiag,diag) (NM75)

This is a compact way of defining one or more etas with labels and, when combined with VALUES, with initial values. For example

```
$OMEGA BLOCK(4) NAMES(ECL,EV1,EQ,EV2) VALUES(0.03,0.01)
```

This is equivalent to

```
$OMEGA BLOCK(4)
ECL=  0.03
EV1=  0.01 0.03
EQ=   0.01 0.01 0.03
EV2=  0.01 0.01 0.01 0.03
```

If both are present, VALUES() must come after NAMES().

SPECIAL CASE (NONMEM 7.3)

If all diagonal elements of $OMEGA are "1.0E+06 FIXED", then NONMEM describes the data as
ANALYSIS TYPE: POPULATION WITH UNCONSTRAINED ETAS

Structurally NONMEM sees the analysis as population, but mathematically, the population density portion of the total likelihood is not included. This allows a population data set to be analyzed as if the data from each individual were single-subject data. Furthermore, some theta parameters could be shared across subjects ("pooled fit parameters"), whereas etas are free to fit each individual without any population constraint. Parallelization is possible.

III.III.B.11. $SIGMA Record

```
$SIGMA [DIAGONAL(n) | BLOCK(n) | BLOCK(n) SAME(m) | BLOCK SAME(m)]
       [[value1]  [value2]  [value3] ...
       [(value,value...) xn]
       [BLOCK(n) VALUES(diag,odiag)]
       [label=value] ...
       [BLOCK(n) [NAMES (label1,...,labeln)] [VALUES (diag,odiag)]
       [FIXED] [UNINT]
       [VARIANCE | STANDARD] [COVARIANCE | CORRELATON] [CHOLESKY]
```

E.g.

```
$SIGMA BLOCK(3)  6. .005 .3 .0002 .006 .4
```

This record gives initial estimates for elements of the Σ matrix, i.e. the variances and covariances of the ε variables in the statistical model. Constraints on Σ are also indicated.

This record should appear only if the statistical model contains ε variables. If it appears, then there must be one such record corresponding to each block of Σ, and the order of these records in the control stream must correspond to the order of the blocks. The values in an $SIGMA record are the initial estimates for the elements of the corresponding block. Under some circumstances $SIGMA records may or may not appear, and the equivalent effect is achieved in either case (see below). When a $MSFI record appears, no $SIGMA records should appear.

Σ can be considered to be in block diagonal form with blocks B_1, B_2, ..., B_m (submatrices of Σ):

$$\begin{pmatrix} B_1 & 0 & & 0 \\ 0 & B_2 & & 0 \\ & & \cdots & \\ 0 & 0 & & B_m \end{pmatrix}$$

This form need not be unique, and there may be only one block (which is most usual). The following description applies to the ith block, $i = 1, \cdots, m$.

If the `DIAGONAL` option is used, it must precede the values. In this case B_i is constrained to be diagonal, and the values are the initial estimates of its diagonal elements given in the diagonal order. The number n is the dimension of B_i. In addition, a final estimate of an individual element of a diagonal block can be constrained to be equal to the initial estimate of the element by using the `FIXED` option. The value giving the initial estimate should be coded with any one of the forms:
init `FIXED`
(init `FIXED`)
(`FIXED` init)

If the `BLOCK` option is used, it must precede the values. In this case the form of B_i is unconstrained, and the values are the initial estimates of its lower triangle elements given in row-wise order. The number n is the dimension of B_i. If `FIXED` option is used, *all* the final estimates of the elements of B_i are constrained to be equal to the initial estimates of these elements.

If the `BLOCK(n) SAME` or `BLOCK SAME` option is used, B_i is constrained to be equal to B_{i-1}. In this case i must be greater than 1, the number n (if it is given) must be the dimension of B_{i-1}, and values are not given.

With NONMEM 7.3, SAME(m) is permitted . If m is present, then this record is equivalent to m identical records without (m). E.g.,

```
$SIGMA BLOCK(2) SAME(3)
```
is equivalent to
```
$SIGMA BLOCK(2) SAME
$SIGMA BLOCK(2) SAME
$SIGMA BLOCK(2) SAME
```

If some of the values in a record are omitted (other than a record with the `BLOCK(n) SAME` or `BLOCK SAME` option), then *all* values in the record must be omitted, and NONMEM will try to obtain initial estimates for the elements of the block. In this case, if the `DIAGONAL` option is used, it must appear explicitly in the record. Often, though, the user should be able to develop reasonable initial estimates, and when he can, there may be a little savings in computation time.

If no $SIGMA records appear, if no $MSFI record appears, but ε variables are used in an abbreviated code, then it is assumed that a record

```
$SIGMA  DIAGONAL(n)
```

where n is the largest index used with an ε variable in all abbreviated codes, might have equivalently been used. (If, though, with PREDPP an abbreviated code for PK is not used, while an abbreviated code for ERROR is used, see section V.6.)

With the `BLOCK` option, the `FIXED` option can occur anywhere among the list of values. These values (of the lower triangle) are given in row-wise order, i.e. $B_{i,11}$, $B_{i,21}$, $B_{i,22}$, ..., $B_{i,n1}$, $B_{i,n2}$, ..., $B_{i,nn}$.

Commas between values are optional.

An initial estimate of an element of a diagonal block must be >0, or it can be 0 if the `FIXED` option is used with it. An initial estimate of a non-diagonally-constrained block must be positive definite, or it can be (uniformly) 0 if the `FIXED` option is used with it. (In any case the initial estimates of Ω and Σ cannot both be 0 unless the Simulation Step is the only step implemented.)

The NM-TRAN translation of $SIGMA records is such that new blocks in addition to B_1, B_2, etc. may be created. This should be transparent to the user, except that he will see additional blocks listed in the problem summary output by NONMEM.

With NONMEM V, an initial estimate of a diagonal block of either the OMEGA or SIGMA matrices may have a band symmetric form, in which case the final estimate has the same form. That is, given the diagonal and a group of contiguous subdiagonals symmetrically ocurring across the diagonal, the elements off both the diagonal and the subdiagonals are constrained to be zero. To specify the initial estimates of such a block, the initial estimates of those elements that are to be constrained to 0 should be given as 0, while all other initial estimates should be given as nonzero. E.g., with these structures for $SIGMA BLOCK(3), the 0's are preserved:

```
 x
 0x
 00x

 x
 xx
 0xx
```

With NONMEM 7.3 if the initial estimate of a block is not positive definite because of rounding errors, a value will be added to the diagonal elements to make it positive definite. A message in the NONMEM report file will indicate that this was done. E.g.,

```
DIAGONAL SHIFT OF  1.1000E-03 WAS IMPOSED TO ENSURE POSITIVE DEFINITE-
NESS
```

With NONMEM 7.3, (value)xn is permitted, so that repeated inputs of $SIGMA may be entered easily. Any initial value or group of initial values may be enclosed in parentheses and followed by "xn" which means to replicate the items n times ("repeated values"). The item to be repeated must always be in parentheses, and the xn must always be immediately after the item, not before it (4x(0.2) is not permitted). Here is an example:

```
$SIGMA BLOCK(6)
0.1
0.01 0.1
(0.01)x2 0.1
(0.01)x3 0.1
(0.01)x4 0.1
(0.01)x5 0.1
```

With NONMEM 7.3, new options are available.

`$SIGMA BLOCK(n) VALUES`(diag,odiag)

This supplies initial values for a block such that the initial estimates of the diagonal elements are all the same, specified by "diag", and the initial estimates of the off-diagonal elements are all the same, specified by "odiag". If present, `VALUES` must follow `BLOCK`.

The above example could be coded

```
$SIGMA BLOCK(6) VALUES(0.1,0.01)
```

For fixed block (such as for SIGMA priors):

```
$SIGMA BLOCK(6) FIX VALUES(0.15,0.0)
```

The following options may follow `VALUES` or be placed between `BLOCK` and `VALUES`.

`VARIANCE` indicates that all initial estimates given for diagonal elements are understood to be initial estimates of variances of etas. This is the default.

`STANDARD` indicates that all initial estimates given for diagonal elements are understood to be initial estimates of standard deviations of etas. May also be coded `SD`.

`COVARIANCE` indicates that all initial estmates given for off-diagonal elements are understood to be initial estimates of covariances of etas. This is the default.

`CORRELATON` indicates that all initial estmates given for off-diagonal elements are understood to be initial estimates of correlations of etas.

`CHOLESKY` indicates that the block is specified in its Cholesky form.

Options `VARIANCE` or `STANDARD` may be combined with `COVARIANCE` or `CORRELATON`.

Note that NONMEM converts all initial estimates to variance and covariances. The values desplayed in the NONMEM report and in the raw and additional output files are always variances and covariances.

The symbolic label substitution feature is new with NONMEM 7.5.

`$SIGMA label`=value *`(NM75)`*

It is similar to the symbolic label substitution feature for $OMEGA. It is a compact method of defining an EPS (an element of SIGMA) specifying its initial estimate, and specifying a label for the subscript for this element of SIGMA. The label may be used as a subscript for EPS in abbreviated code, and will also identify this element of SIGMA in the NONMEM output. If new $SIGMA records change the ordering, the abbreviated code does not have to be changed. For example, suppose the first element of SIGMA

that is defined happens to be

```
$SIGMA RSW=0.6
```

The NONMEM report will describe the relationship, e.g.,

```
LABELS FOR EPS
EPS(1)=EPS(RSW)
```

and EPS(RSW) rather than EPS1 will appear in the NONMEM report. The abbreviated code can use this symbolic subscript instead of the numeric subscript.

As with $OMEGA, $SIGMA and $THETA records (if elements of THETA are used) must be placed ahead of any records that use the symbolic label.

Another example defines symbolic labels for a block of SIGMA:

```
$SIGMA BLOCK(2)
RSW=  0.3
EX=   0.01 0.35
```

Or, for diagonals,

```
$SIGMA
RSW= 0.3
EX= 0.35
```

With NONMEM 7.5, Symbolic label substitution may be specified for an entire block using the NAMES option.

$SIGMA *BLOCK(*n*) NAMES* (label$_1$,...,label$_n$) *VALUES* (odiag,diag) (NM75)

This is a compact way of defining one or more εs with labels and, when combined with VALUES, with initial values. For example

```
$SIGMA BLOCK(2) NAMES(RSW,EX) VALUES(0.03,0.01)
```

This is equivalent to

```
$SIGMA BLOCK(2)
RSW=  0.03
EX=   0.01 0.03
```

If both are present, VALUES() must come after NAMES().

III.III.B.12. $MSFI Record

$MSFI filename [**NORESCALE**|*RESCALE*] [*NPOPETAS[*=n]]
[*ONLYREAD*] [*NOMSFTEST*|**MSFTEST**]
[*VERSION*=n]
[*NEW*]

E.g.
```
$MSFI   MSF13
```

This record gives the name of a Model Specification File to be input. Such a Model Specification File is a file output by a previous NONMEM run, which contains certain model information pertaining to that run. It also contains other information which allows (i) the minimization search in that run, if terminated unsuccessfully because the limit on the allowable number of objective function evaluations was attained, to be smoothly continued in the current run, and (ii) Covariance, Table, and Scatterplot Steps in the current run, which follow the successful termination of that search (in the current run or the previous run) to be implemented.

Starting with NONMEM V, a MSFI may be used to repeat the Estimation Step using a method other than the one used to write the MSF. This is possible if the MSFI contains the results of a search that terminated successfully.

This record is required only if a Model Specification File is to be input. With PREDPP, if a $PK record is not used, while a $ERROR record is used, the $MSFI record must precede the $ERROR record.

The filename must be the first option on the record.

If the search is continued in the current run (see section B.14), use of the `NORESCALE` option means that the search is continued just as it would have been continued in the previous run had the limit on the number of function evaluations not been attained. Use of the `RESCALE` option means that before the search is continued, the final estimates of the UCP (Unconstrained Parameters) from the previous run are rescaled so that they are all 0.1. (See Guide I, section C.3.5.1, where the UCP are referred to as STP. For repeating and rescaling, see Guide II, section F, where the UCP are referred to as RCP).

When the $MSFI record is used in a problem specification, $THETA, $OMEGA, and $SIGMA records should not appear for that specification.

The number n with the `NPOPETAS` option is the number of η variables used with population data. When n is 0, then the data are regarded as single-subject data (see section II.C.4). It is a good practice to include the `NPOPETAS` option; it is a simple thing to do. However, the `NPOPETAS` option is only needed when the data should be regarded as population data and: (i) a $PRED record is not used and the label L1 is not used in the $INPUT record (or if PREDPP is used, (ii) $PK and $ERROR records are not used and the label L1 is not used in the $INPUT record, or (iii) a $PK record is not used, while a $ERROR record is used).

`POPETAS` or any of its abbreviations are aliases for `NPOPETAS`.

With NONMEM VI and 7.x, other options are available.

`ONLYREAD NOMSFTEST MSFTEST VERSION NEW`

These are described in Guide VIII On-line help

III.III.B.13. $SIMULATION Record

```
$SIMULATION  (seed1 [seed2] [NORMAL|UNIFORM|NONPARAMETRIC] [NEW]) ...
             [SUBPROBLEMS=n] [ONLYSIMULATION] [OMITTED]
             [REQUESTFIRST] [REQUESTSECOND] [PREDICTION|NOPREDICTION]
             [TRUE=INITIAL|FINAL|PRIOR]
             [BOOTSTRAP=n [REPLACE|NOREPLACE] [STRAT=label] [STRATF=label]]
             [NOREWIND|REWIND] [SUPRESET|NOSUPRESET]
             [RANMETHOD=[n|S|m|P] ]
             [PARAFILE=[filename|ON|OFF]
```

```
E.g.  $SIMULATION     (889215690) (2239177789 UNIFORM)
```

This record requests that the Simulation Step be implemented.

This record is optional.

Data are generated according to a statistical model. The DV data items of (NONMEM's internal copy of) the data set are replaced by generated DV items. The model is that specified in the code for PRED (for PK, ERROR, etc. if PREDPP is used). The data are simulated using the parameter values given as initial estimates; see sections B.9-11. Initial estimates must be given for all parameters of the model.

The model used for data simulation may be complicated. It may involve covariables and random interindividual and intraindividual random effects. It may be the very model used for data analysis. One reason to simulate with such a model is to explore the information content of data obtained according to some particular design. The design is encoded into the data set. The Simulation and Estimation Steps, and possibly the Covariance Step too, are implemented, so that one can assess how well the true parameter values of the data analysis model can be estimated. This technique can be used during the course of a data analysis when the adequacy of the design has become suspect. However, it can also be used before the data are actually obtained to explore design choices.

If only a simulation of the structural part of the model is desired, i.e. the model without statistical components, the model need not involve random effects. In this case the $SIMULATION record need not even appear. If it does appear, the DV items are replaced with generated DV items based only on the structural model. If it does not appear, the DV items are not replaced, but prediction items, i.e. predictions returned by PRED, may still be displayed. To simulate only the structural part of the model when a full statistical model has already been specified, *fix* the initial estimates of the variances of the random effects, i.e. the initial estimates of Ω and Σ, to zero.

Data can be simulated with one model and analyzed with another. There are two approaches. With the first approach, data are simulated in the Simulation Step and output in the Table Step (see section B.16). The table serves as the data set for a subsequent NONMEM run. (The DROP label can be used with the prediction, residual, and weighted residual data items of the table; see section B.2.) In the subsequent run the data are analyzed using the analysis model. With the second approach, codes for both models are given in PRED, but with any particular call to PRED, one or the other code is executed according to the value of a special variable (ICALL) that signals whether PRED is being called during the Simulation Step or a data analysis step (see section IV.D). The advantage of the first approach is that it is a bit more flexible. The advantage of the second is that data sets and the analyses performed on them can be more easily replicated; see the discussion below concerning the option SUBPROBLEMS.

A random source is an infinite sequence of pseudo-random numbers. At most 10 random sources can be defined for a single problem. The sources are numbered according to the ordering of their definitions in the record. The information coded within each set of parentheses defines the attributes of a single random source. A random source can be used with the problem for which it is defined. The same source

can be defined for different problems. A source defined for one problem can be continued in a subsequent problem; see below. Unless a random source is explicitly defined for a given problem, it cannot be used with that problem.

When the model, i.e. *either* the simulation model or the data analytic model, uses η variables, or both η and ε variables, and the Simulation Step is implemented, at least one random source must be defined (unless the variances of these variables are 0; see below). When the model is a mixture model and the Simulation Step is implemented, at least one random source must be defined. The first source is used by NONMEM to generate realizations of the η and ε variables and/or to randomly mix individuals into different subpopulations according to the mixing parameter. Only the `NORMAL` attribute can be used with this source, i.e. during simulation the η and ε variables are understood to be normally-distributed. (Mixing, though, does not involve normal pseudo-random numbers.) The `NEW` attribute cannot be used with any source other than this first source (see below).

The remaining defined random sources are used exclusively by the PRED subroutine (by the PK and ERROR subroutines if PREDPP is used). If no η variables appear in the model and a mixture model is not used, then *all* defined random sources can be used by PRED. On the other hand, in this case, no sources need even be defined. If η (and possibly ε) variables appear in the simulation model, but the initial estimates of Ω (and Σ) are 0, and if a mixture model is not used, then again, no sources need be defined. A random source not used by NONMEM per se is called a user random source . Such a source can use either the `NORMAL` or `UNIFORM` attribute (see below).

Numbers from user random sources are obtained via the NONMEM utility RANDOM. An abbreviated code or a user-supplied FORTRAN code can use RANDOM by executing the FORTRAN statement

```
CALL RANDOM (K,R)
```

Each time RANDOM is called with `K` set to the index of a given user source, the routine returns the next number from that source. This number is returned in `R`, and is always a single-precision number.

Seed1 and seed2 together initiate the random source. Seed2 is used only seldomly (see below), and each seed is an integer between 0 and 21474836447. Two sources are the same if they are initiated with the same seeds.

Suppose that the current problem specification is not the first problem specification in the control stream, that the source is the ith source, S_i, and that the last problem to use as many as i sources was problem number m. Then seed1 can be -1, indicating that S_i is the continuation of the ith source, R_i, defined with problem m. That is, if the last number used from R_i was x_k, then S_i is the infinite tail sequence of R_i starting with x_{k+1}.

Seed2 is used when it is desired that the current problem make use of a continuation of a random source defined in a preceding NONMEM run. Examine the NONMEM output from the Simulation Step of the last problem in the earlier run using this source. Two ending seeds are printed in this output; often the second seed is 0. The continuation in the current problem is defined using these two seeds as seed1 and seed2. When the second seed is 0, it need not be given.

Use of the `NORMAL` option means that the numbers of the source are to be pseudo-normal with mean 0 and variance 1 (unless the source is the first and used to generate η and ε realizations, in which case the variance-covariance of these variables is that specified in the $OMEGA and $SIGMA records). Use of the `UNIFORM` option means that the numbers of the source are to be pseudo-uniform on the interval [0,1].

During the Simulation Step PRED has access to simulated values for the η variables via the NONMEM utility routine SIMETA. When PRED is called with a data record from a given individual record, PRED can execute the abbreviated code or FORTRAN statement

```
CALL SIMETA (ETA)
```

which results in appropriate values being stored in the one-dimensional ETA array; see section IV.A. These values will arise from a multivariate normal pseudo-random distribution with mean 0 and variance-covariance as specified with the $OMEGA record. By default, no matter how many times this statement is executed, as long as the individual record is the same, the same values are stored. If, though, the NEW option is used, each time the statement is executed, new values are stored. Thus, for example, when PRED is called with the first data record of an individual record, PRED can in turn call SIMETA multiple times until values are obtained such that none are larger than 5 in absolute value, i.e. values can be obtained from a truncated distribution. (To pursue this example, see section IV.I.)

During the Simulation Step PRED has access to simulated values for the ε variables via the NONMEM utility routine SIMEPS. When PRED is called, it can execute the abbreviated code or FORTRAN statement

```
CALL SIMEPS (EPS)
```

which results in appropriate values being stored in the one-dimensional EPS array; see section IV.A. These values will arise from a multivariate normal pseudo-random distribution with mean 0 and variance-covariance as specified with the $SIGMA record. By default, no matter how many times this statement is executed within the same call to PRED (or more precisely, as long as PRED is being called with a data record from the same level-two record; see Guide I, section B.1), the same values are stored. If, though, the NEW option is used, each time the statement is executed, new values are stored.

Values of η's and ε's are obtained by calls to SIMETA and SIMEPS occuring in the generated subroutine. When the data are population data and the Simulation Step is implemented, SIMETA is called once per individual record, and SIMEPS is called once every call to PRED (once every call to ERROR if PREDPP is used). When the data are single-subject data and the Simulation Step is implemented, SIMETA is called once every call to PRED (once every call to ERROR if PREDPP is used). These calls are implemented so that even if the Simulation Step is not implemented, the load module resulting from using an abbreviated code for PRED (for PK or ERROR if PREDPP is used) can be reused with a run implementing the Simulation Step. For multiple calls to SIMETA or SIMEPS making use of the NEW option, additional calls can be used either in PRED (or PK or ERROR) or in the abbreviated code.

If the SUBPROBLEMS option is used, the entire problem is repeated n times in succession. Each repetition of the problem is called a subproblem . Each subproblem includes the Simulation Step, and any other steps requested in the problem specification, but each of the random sources *are continued* from subproblem to subproblem. This allows the effects of sampling variability to be directly assessed. If n = 0 or 1, the result is the same as if the SUBPROBLEMS option is omitted.

By default, when the Simulation Step is implemented, various statistics used for data analysis are always computed from the simulated data. These are the value of the objective function at the initial parameter estimates and if a table is requested, the weighted residuals based on these estimates. The data may be simulated in a way that gives rise to a problem with computing these statistics. For example, perhaps the simulation model is not appropriate for data analysis using the NONMEM default objective function. For this reason, or for some other, the Simulation Step can be implemented so that these statistics are not computed. This is accomplished by including the option ONLYSIMULATION. In this case the Estimation and Covariance Steps are not implementable in the problem.

If the option ONLYSIMULATION is used, a PRED-defined item (see section IV.F) that depends on values of η's and/or ε's is displayed in tables and scatterplots using simulated values for the η's and ε's. See sections B.16 and IV.F for a description of the appropriate label to use. Starting with NONMEM VI, simulated values of an η variable are displayable by using ETA labels in $TABLE or $SCATTERPLOT records.

Otherwise, the item is displayed either using zero values, or if it depends on η's and conditional estimates are available, it is displayed using conditional estimates for the η's (see section B.14).

Another way to display a quantity computed in PRED that uses simulated values of η's and/or ε's is to store the quantity in the data array; data items are always displayable.
Transgeneration of data items is allowed during the Simulation Step. (Transgenerated items are stored in the internal copy of NONMEM's data set.) This is a way, therefore, to display the quantity even when ONLYSIMULATION is not used. Starting with NONMEM V, one can store quantities in the data array using abbreviated code (see section IV.A). For earlier versions, either a user-supplied PRED must be used (for an example with a user-supplied PK, see Guide VI, Figure 2 and the accompanying discussion in section III.L.1), or verbatim code must be used (for an example of transgeneration using verbatim code, see section IV.I). These examples are still of interest.

If the OMITTED option is used, the Simulation Step is not implemented, even though the $SIMULATION record appears. When used, no other option should be used.

When the Simulation Step is implemented, initial estimates must explicitly appear for all parameters of the statistical model.

Seed1 must occur first among the attributes of a random source. All the other attributes can occur in any order. Any two attributes can be separated by spaces or a comma.

SIML is an alias for SIMULATION.

REQUESTFIRST and REQUESTSECOND

REQUESTFIRST
: NONMEM sets a variable IFIRSTEM in Module ROCM_INT (referenced as FIRSTEM in abbreviated code) informing PRED whether or not PRED needs to compute first-partial derivatives with respect to eta. Normally, during the Simulation Step, these derivatives are not needed, either by NONMEM or by the user. However, the user may want the first-partial eta derivatives of a PRED-defined item and may want FIRSTEM to reflect this. With the REQUESTFIRST option, the FIRSTEM variable is set so to inform PRED that the derivatives need to be computed. In this case, if an abbreviated code is used to compute the PRED-defined item, the item should not be computed within a simulation block, because NM-TRAN does not provide derivatives for PRED-defined items in a simulation block.

REQUESTSECOND
: NONMEM sets a variable ISECDER in Module ROCM_INT (referenced as MSEC in abbreviated code) informing PRED whether or not PRED needs to compute second-partial derivatives with respect to eta. Normally, during the Simulation Step, these derivatives are not needed, either by NONMEM or by the user. However, the user may want the second-partial eta derivatives of a PRED-defined item and may want the MSEC variable to reflect this. With the REQUESTSECOND option, the MSEC variable is set so to inform PRED that the derivatives need to be computed. In this case, if an abbreviated code is used to compute the PRED-defined item, the item should not be computed within a simulation block, because NM-TRAN does not provide derivatives for PRED-defined items in a simulation block. REQUESTSECOND implies REQUESTFIRST.

PREDICTION
: Permitted only with ONLYSIM, and is the default.
With or without ONLYSIM, unless the NOPREDICTION is used, the simulated observation is taken to be the quantity to which the Y variable (with NM-TRAN abbreviated code) or F variable (with a user-supplied PRED or ERROR routine) is set. In a simulation block, the DV variable may be directly set to the simulated observation, but the Y (or F) variable should also be set to this observation. E.g., if a line of code DV=... is used in a simulation block, be sure to follow this line with the additional line Y=DV.

With the Simulation Step, PRED may return the simulated observation as the DV data item, rather than in the argument F. With odd-type data the simulated observation must be returned as the DV data item.

`NOPREDICTION`

Permitted only with `ONLYSIM`.

Indicates that the simulated observation will be taken to be the value to which the DV variable is set. The code `Y=...` is permitted inside or outside a simulation block, but if such code appears in a simulation block, be sure to also include e.g. `DV=Y`. Also, etas (if any) are understood to be population etas, even if epsilons do not appear.

With NONMEM VI and 7, other options are available.

III.III.B.14. $ESTIMATION Record

```
$ESTIMATION [METHOD=kind] [NOINTERACTION|INTERACTION] [NOLAPLACIAN|LAPLACIAN]
            [NOPOSTHOC|POSTHOC] [SIGDIGITS=n1] [MAXEVALS=n2] [PRINT=n3]
            [ABORT|NOABORT] [MSFO=filename] [NOREPEAT|REPEAT] [OMITTED]
            [PREDICTION|LIKELIHOOD|-2LOGLIKELIHOOD]
            [NOSUB=0 | NOSUB=1]
```

E.g. `$ESTIMATION MAXEVAL=450 PRINT=5`

This record requests that the Estimation Step be implemented.

This record is optional.

With versions of NONMEM through VI, multiple $ESTIMATION records in the same problem were considered to continue a single $ESTIMATION record. With NONMEM 7, a sequence of two or more $ESTIMATION records within a given problem will result in the sequential execution of separate NONMEM Estimation Steps. (A given $ESTIMATION record may still be continued if '$ESTIMATION' is omitted from subsequent records in the block.) If $TABLE statements succeed multiple $EST statements within a run, the table results (as well as scatter plots if requested via $SCATTER) will pertain to the last analysis.

The following section describes the classical methods. All estimation methods obtain parameter estimates by minimizing an objective function whose arguments are the parameters of the model. The methods differ from each other because they use different types of objective functions (see Guide VII).

If the `METHOD` option is omitted, then the first-order estimation method is used. With single-subject data, and when a CONTR routine is not supplied by the user, this particular method is simply the extended least squares method. If the option is used, then the option value (kind) can be `ZERO`, in which case the first-order estimation method is used, or it can be `CONDITIONAL`, in which case a conditional estimation is used.

The `INTERACTION` option can be used when the statistical model includes ε variables (see end of chapter II for a discussion about where it does not) and where the variance of some observation, conditional on the values of the η variables, depends on these values. In this case the first-order conditional estimation method with interaction is used.

If the `LAPLACIAN` option is used, the Laplacian (conditional) estimation method is used. This option cannot also be used with the `INTERACTION` option.

Conditional estimates of individual-specific η values may be obtained and displayed. These estimates are empirical Bayesian estimates, conditional not only on the data, but, importantly, also on values for the population parameters. If the first-order estimation method is used, they may be obtained *after* the population parameter estimates have themselves been obtained. To obtain them, include the option `POSTHOC`. The term 'posthoc estimates' is commonly applied to these particular conditional estimates. When the first-order estimation method is used, and a mean-variance intraindividual model is used, the posthoc estimates are computed under the assumption that the variance model is that of the mean individual; see Guide VII. To obtain posthoc estimates without this assumption use the final estimates as initial estimates and the options `MAXEVALS=0`, `METHOD=CONDITIONAL`, `INTERACTION` (see below).

If a conditional estimation method is used, the conditional estimates are obtained *simultaneously* with the population parameter estimates. In this case the option `POSTHOC` is superfluous, but it may be used. The term 'conditional estimates' applies when empirical Bayesian estimates are obtained, whether or not a conditional estimation method is used, and no matter what values are used for population parameters. For example, the term can apply to the conditional estimates associated with using the first-order conditional estimation method and using the initial estimates of the population parameters (to see how to

obtain these; see below).

The number n_1 is the number of significant digits required in the final parameter estimate. If the SIGDIGITS option is omitted, n_1 defaults to 3. If the option is used, n_1 must be a positive integer less than 9.

The number n_2 is the maximum allowable number of evaluations of the objective function which can occur during the minimization search. If the MAXEVALS option is omitted, n_2 defaults to a generous number.

The number n_2 can be 0. In this case the Estimation Step is not implemented. However, a number of statistics can be obtained:

The value of the objective function is obtained using the initial parameter estimates, unless an $MSFI record is used, in which case the function is computed using the final parameter estimates from the earlier problem producing the Model Specification File. The options METHOD, INTERACTION and LAPLACIAN can be used to specify the objective function to be used (unless an $MSFI record is used, in which case these options are ignored, and the objective function specified by these options, *as they were used with the problem generating the Model Specification File*, is the one that is used). The rules governing the use of these options are as given above.

The output from the Covariance Step is obtained if requested. A $MSFI record must be used, and the computations are based on the final parameter estimates from the earlier problem. The options METHOD, INTERACTION and LAPLACIAN are ignored; the objective function specified by these options, *as they were used with the problem generating the Model Specification File*, is the one that is used. The SIGDIGITS option may be omitted, in which case the usual default value for n_1 (i.e. 3) is used in these computations. The value used with the current problem must be no greater than the number of digits actually obtained in the final estimate from the earlier problem. If the value 4, say, was used in the earlier problem, and 4.4 digits were actually obtained, then although the value 3 could be used in the current problem, it would be much better to again use 4. The SIGDIGITS option can be used, and n_1 can be any positive integer less than 9.

Conditional estimates may be obtained and may be displayed. These estimates are based on the initial population parameter estimates, unless a $MSFI record is used, in which case they are based on the final parameter estimates from the earlier problem. If posthoc estimates are desired, use the option POSTHOC. If conditional estimates associated with some particular conditional estimation methods are desired, use METHOD=CONDITIONAL along with INTERACTION or LAPLACIAN if necessary (unless a $MSFI record is used, in which case these options are ignored, and the effects of these options on conditional estimates, *as these options were used with the problem generating the Model Specification File*, are the ones that result). The rules governing the use of these options are as given above. Variables defined in PRED (in PK or ERROR if PREDPP is used) that depend on η's are displayed using the conditional estimates.

The number n_2 can be -1 if a $MSFI record is used. Then the maximum allowable number of objective function evaluations is the same as that used in the problem that produced the Model Specification File.

The number $n_3 - 1$ is the number of iterations skipped between iteration summaries. When $n_3 = 0$, no iteration summaries are printed. If the PRINT option is omitted, n_3 defaults to 9999 so that iteration summaries for only the 0th and last iterations are printed.

NOSUB is a new option with NONMEM 7.4 that controls a new feature, label substitution.

For information concerning the options ABORT and NOABORT, see section IV.G.

If the MSFO option is used, a Model Specification File is output. The name of the file is given. This name should not include embedded spaces, commas, semicolons, or parentheses. The file contains certain model information pertaining to the problem, and it contains other information which allows the minimization search in this problem, if terminated prematurely and unsuccessfully because the limit n_2 is

attained, to be smoothly continued in a succeeding run. It also contains information which allows the Covariance, Table, and Scatterplot Steps which follow the successful termination of the search in this run to be implemented in a succeeding run. A Model Specification File should not be output when a $SIMULATION record appears and the number of subproblems exceeds 1.

If the `NOREPEAT` option is used, the estimate obtained at the end of the minimization search is taken to be the final parameter estimate. If the `REPEAT` option is used, then upon successful termination of the search, the search is repeated after the UCP are first rescaled so that they are all 0.1. (See Guide I, section C.3.5.1, where the UCP are referred to as STP. For repeating and rescaling, see Guide II, section F, where the UCP are referred to as RCP). In this case n_2 is used to limit the number of objective function evaluations over both searches combined, and a Model Specification File, if output, contains the information holding at the termination of the second search.

The `LIKELIHOOD` and `-2LOGLIKELIHOOD`] options are used with odd-type data, discussed in Chapter II.D. `PREDICTION` is the default.

If the `OMITTED` option is used, the Estimation Step is not implemented, even though the $ESTIMATION record appears. When used, no other option should be used.

`ESTM` is an alias for `ESTIMATION`. The numbers `0` and `1` can be used with the `METHOD` option instead of `ZERO` and `CONDITIONAL`, respectively. `LAPLACEAN` (and hence its abbreviation `LAPLACE`) is an alias for `LAPLACIAN`.

Many additional methods and options are available. These are described in Guide VIII and On-line help. (See Guide Introduction_7 "$ESTIMATION Record (NM75)")

III.III.B.15. $COVARIANCE Record

```
$COVARIANCE [SPECIAL] [MATRIX=] [PRINT=[E][R][S]
            [COMPRESS]
            [SLOW|NOSLOW|FAST]
            [SIGL=n] [SIGLO=n]
            [NOFCOV]
            [PARAFILE=[filename|ON|OFF]
            [RESUME]
            [CONDITIONAL|UNCONDITIONAL]
            [OMITTED]
```

E.g.
```
$COVARIANCE
```

This record requests that the Covariance Step be implemented.

This record is optional.

The SPECIAL option should be used if the data are single-subject and a recursive PRED subroutine (such as PREDPP) is used. A recursive PRED subroutine is such that the PRED computation with a data record depends on the PRED computation(s) with previous data records. With PREDPP and single-subject data, this option is the default.

The character c is either R or S. Use of R or S means that the covariance matrix is taken to be the inverse of the R or S matrix, respectively. The R and S matrices are two matrices from statistical theory, the Hessian and Cross-Product Gradient matrices respectively. See Guide II, section D.2.5. If R is used, the SPECIAL option need not (and best not) be used. If the MATRIX option is omitted, the covariance matrix is taken to be $R^{-1}SR^{-1}$.

When the Covariance Step is implemented, standard error estimates are always printed, along with the covariance matrix (upon which the standard error estimates are based), the inverse covariance matrix, and the correlation form of the covariance matrix. When the PRINT option is used, other outputs are also available. The characters E, R, and S, represent the eigenvalues of the covariance matrix, the R matrix, and the S matrix, respectively. When the PRINT option is used, one or more of these characters should be chosen. The chosen characters need not be separated, but a comma or spaces can separate two characters: e.g. PRINT=ER, or =E R, or =E,R. If the character c is chosen to be R (S) with the MATRIX option, then this character need not be chosen with the PRINT option; in this case the covariance matrix is the R (S) matrix.

If the CONDITIONAL option is used, the Covariance Step is implemented only when either the Estimation Step terminates successfully, or a Model Specification File is input and the Estimation Step is not implemented. If the UNCONDITIONAL option is used, then the Covariance Step is implemented when either the Estimation Step is implemented or a Model Specification File is input.

If the OMITTED option is used, the Covariance Step is not implemented, even though the $COVARIANCE record appears. When used, no other option should be used.

Many additional methods and options are available. These are described in Guide VIII and On-line help.

III.III.B.16. $TABLE Record

$TABLE [list1] [*BY* list2]
[**PRINT**|*NOPRINT*] [*FILE*=filename]
[*NOHEADER*|*ONEHEADER*] [*ONEHEADERALL*]
[*NOTITLE*|*NOLABEL*]
[*FIRSTONLY*|*LASTONLY*|*FIRSTLASTONLY*]
[**NOFORWARD**|*FORWARD*]
[**APPEND**|*NOAPPEND*]
[*FORMAT*=s_1] [*LFORMAT*=s_1] [*RFORMAT*=s_1]
[*IDFORMAT*=s_1]
[*NOSUB=[0|1]*]
[*EXCLUDE_BY* list3]
[*PARAFILE*=[filename|*ON*|*OFF*]
[*ESAMPLE*=n_1][*WRESCHOL*]
[*SEED*=n_2][*CLOCKSEED=[0|1]*]
[*RANMETHOD=[n|S|m]*]
[*VARCALC=[0|1|2]*]
[*FIXEDETAS=(*list*)*]
[**UNCONDITIONAL**|*CONDITIONAL*] [*OMITTED*]

E.g. `$TABLE ID DOSE WT TIME`

This record requests that the Table Step be implemented.

This record is optional. There should be one $TABLE record per table, up to ten $TABLE records per problem (not counting continuation records). A $TABLE record cannot be continued by series of contiguous blocks; each contiguous block defines a different table.

A table is a two-dimensional array. There is a one-to-one correspondence between the rows of a table and the data records of the data set. The elements of a row, the row items , are items associated with the corresponding data record. With NONMEM 7.4, some data records may be excluded from the table; see `EXCLUDE_BY` below.

List1 is a list of up to eight labels (and synonyms), unless the `NOAPPEND` option is used (see below), in which case the list may be up to 12 labels, or the the `NOPRINT` option is used (see below), in which case the list can be as long as the value of constant PDT in resource/SIZES (default is 500). The list may include labels used in the $INPUT record for different data item types. It may include labels chosen from the list `ETA1`, `ETA2`, ..., `ETA9`, `ETA10`,..., `ETA99`,`ETA100`,..., `ETA999` which label conditional estimates of the η's (called η items). (There can be at most LVR η's in the model, where LVR is a constant in resource/SIZES)

The list may include labels for PRED-defined items; for a description of these labels see below and also section IV.F.

The row items of a given row are those with labels specified in list1, along with the the dependent variable (`DV`), prediction (`PRED`), residual (`RES`), and weighted residual (`WRES`) row items. These last four items always appear as the last four row items. If list1 is omitted, only these four types of items appear in the table. With NONMEM V, option `NOAPPEND` may be used to request that these four items not be appended to the list, in which case each of them may be listed individually anywhere in list1 and will appear where it is listed. All four may also be explicitly appended to the list using option `APPEND`, which is the default. With NONMEM 7, related special diagnostic items may also be listed; these are

described in the Guide Introduction to NONMEM 7 and Guide VIII and on-line help for $TABLE.† With NONMEM 7.3, MDVRES ("missing dependent variable MDV for residual RES") is a reserved variable in abbreviated code. When MDVRES is set to 1 in abbreviated code for a particular record, the values of RES, and WRES and related special diagnostic items are 0 in tables and scatterplots. See Chapter IV J.2.

If the BY option is omitted, the order of the labels in list1 determines the order in which the items with these labels appear in the rows.

Prediction items are always population predictions, i.e. they are computed at the mean value of η (0). Residual items are always based on these predictions, as are weighted residual items, and with the latter the weights are also computed at $\eta = 0$. With a mixture model, each individual is classified into one of the subpopulations of the mixture according to an empirical Bayesian computation, conditional on the individual's data and on the final estimates of the population parameters. For a data record from the individual record, the prediction, residual, weighted residual, and η items in the corresponding row are based on the submodel defining the subpopulation into which the individual is classified.

List2 is a list comprised of labels from list1. The BY option should be used only if the rows of the table are to be sorted on the items of particular types. The rows of the table are sorted on the items corresponding to the 1st label in list2, then secondarily sorted on the items corresponding to the 2nd label in list2, etc. The items corresponding to the labels in list2, appear in the first consecutive columns of the table, and the order of the labels in list2 determines the order of these columns. The items corresponding to the labels in list1 which are not in list2 appear in the next consecutive columns of the table, and the order of these labels in list1 determines the order of these columns.

If the PRINT option is used, the table is printed in the NONMEM report. This is the default. If the NOPRINT option is used, the table is not printed; it is written to a formatted file and can be used as NM-TRAN or NONMEM data sets, or can be used with other computer programs. If this option is used the FILE option (see below) must also be used. To obtain both printed copies and the formatted file, use the PRINT option along with the FILE option. If the NOPRINT option is used, the maximum number of labels in list1 is given by constant PDT in resource/SIZES.f90; default is 500. However, if the number of labels exceeds 8 with a particular table, the rows of that table cannot be sorted.

The FILE option gives the name of a formatted file to which tables may be written. The name may not include embedded spaces, commas, semicolons, or parentheses. The file is called a table file.

Options FORMAT, LFORMAT, RFORMAT, and IDFORMAT can be used to override the default format for the table file, which is s1PE11.4.
See Introduction to NONMEM 7 and Guide VIII and on-line help.

Options NOFORWARD and FORWARD affect the positioning of the file. With NOFORWARD, when the table file is opened during a given problem, it is positioned at the start of the file. This is the default. With FORWARD, when the table file is opened during a given problem, it is forwarded to the end of the file. This allows a table file to accumulate tables from multiple problems.
See Introduction to NONMEM 7 and Guide VIII and on-line help.

Tables are split into segments of 900 rows each. Of course, a table may need only one segment. Segments usually have headers comprised of two records. Text in the first record identifies the table and segment, and text in the second record gives the labels for the tabled items. If the NOHEADER option is used, the headers do not appear. This may be useful when a table is to be read by another computer program. If the ONEHEADER option is used, only the first segment of each table has a header. This may be

† The special diagnostic items are:
NPRED, NRES, NWRES, PREDI, RESI, WRESI, CPRED, CRES, CWRES, CPREDI, CRESI, CWRESI
CIPRED, CIRES,CIWRES, CIPREDI, CIRESI,CIWRESI, NIPRED, NIRES, NIWRES,
IPREDI, IRESI,IWRESI,IPRD, IRS,IWRS, EPRED, ERES, EWRES, ECWRES,
EIPRED, EIRES,EIWRES, NPDE, NPD, OBJI

useful in order to separate tables from each other.
ONEHEADERALL (nm74) is used only with the FILE option and FORWARD. Only the first line of the table file is a header line. May also be coded ONEHEADERPERFILE.

Similar options are NOTITLE and NOLABEL which suppress table titles and column labels respectively. With FIRSTONLY, only information corresponding to the first data record from each individual record appears in the table.
With LASTONLY and FIRSTLASTONLY (nm74) only information corresponding to the last data record from each individual record, or from the first and last records, appear in the table.

If the UNCONDITIONAL option is used, the Table Step is always implemented. If the CONDITIONAL option is used, the Table Step is implemented only when either the Estimation Step terminates successfully, or the Estimation Step is not implemented.

If the OMITTED option is used, the Table Step is not implemented, even though the $TABLE record appears. When used, no other option should be used.

Parentheses surounding a list are optional. However, they should be used when a label can be confused with an alias for an option, e.g. when a label COND is used. Two list items may be separated by a comma or spaces. Options cannot be coded among the labels of either list1 or list2.

Synonyms for the prediction, residual, and weighted residual item types can be defined with the $TABLE record in the same manner as synonyms for data item types can be defined with the $INPUT record. The reserved labels for these item types are PRED, RES, and WRES. If for example, the synonym PR is to be defined for the prediction item type, then PR=PRED should occur among the labels in list1. The synonym will be used in *all* tables and scatterplots, and control records following the $TABLE record defining the synonym can use the synonym. Synonyms may also be used for the special diagnostic items.

Options PRINT, NOPRINT, HEADER, NOHEADER, NOLABEL, FILE, FIRSTONLY, FORWARD, NOFORWARD, APPEND, NOAPPEND, FORMAT apply to the individual $TABLE record. They must be specified for each table to which they apply.

For η items, the label ETA(n) can be used instead of ETAn. However, the labels in the table will be ETA1 through ETA9 followed by ET10 through ET999.

With NONMEM 7.3, instead of requesting each ETA specifically a range of etas may be requested:
ETAS(k:n) is equivalent to ETAk, ..., ETAn (where n > k).

LAST can be used in place of n, and requests the last (highest numbered) η in the problem , e.g. ETAS(1:LAST). If :LAST is omitted, it is assumed, so that ETAS(1) is equivalent to ETAS(1:LAST). Note that ETA(1) requests a single η_1, but ETAS(1) requests all the η's.

With NONMEM 7.4, a more flexible syntax is available:
The word TO may be used in place of ":".
The BY expression may be used:
ETAS(1 TO 10 by 3) prints out etas 1,4,7,10
ETAS(LAST TO 1 by -3) prints out etas 10,7,4,1 (assuming LAST=10)

With NONMEM 7.4, a symbolic label specified in $ABBR REPLACE may be listed in $TABLE. For example:

```
$ABBR REPLACE ETA(CL)=ETA(1)
  ...
$TABLE ETA(CL)
```

A PRED-defined item is a value stored in some variable defined in PRED (in PK, ERROR, etc. if PREDPP is used); see section IV.F. It may be displayed provided the variable is listed in MODULE NMPRD4. If the variable is defined in abbreviated code, it is normally listed in NMPRD4 (for exceptions see sections III.B.7 and IV.H). The label used in the $TABLE record for the values of the variable

can be the variable name. However, if the name is longer than 9 characters, the label used in the table itself is comprised of the first nine characters only. Alternatively, a synonym for the label in the $TABLE record can be defined in the same manner as synonyms for labels of data item types can be defined with the $INPUT record. E.g., suppose there is a user-defined variable `CLEARANCE_M`. The $TABLE record may list `CLEARANCE_M=CLM`, in which case the label in all tables is `CLM` rather than truncated as `CLEARANCE`.

Pred-defined elements in a list may include elements of vectors used in abbreviated code: `VECTRA(1)`, `VECTRA(2)`, ... ,`VECTRA(9)`, or alternatively, labels `VA_1`, `VA_2`, ... ,`VA_9`, corresponding to VECTRA(1), VECTRA(2), ..., VECTRA(9). The labels in the output will be VA_1, VA_2, ... , VA_9. Similarly, for `VECTRB` and `VECTRC`. See Chapter IV, and Introduction to NONMEM 7 and Guide VIII and on-line help. Pred-defined elements in a list may include elements of G and H. E.g.,
`$TABLE G11 G21 G31 H11 H21.`
See Guide VIII $TABLE.ctl

The definition of a variable in an abbreviated code can generate additional definitions of other variables, called generated variables , appearing in the generated code, but not appearing in the abbreviated code. The names of generated variables are all six characters long. Certain generated variables symbolize the values of partial derivatives and are normally listed in NMPRD4 so that their values can be displayed like other PRED-defined items. The names of these variables, and the four character labels used in the tables for the values of these generated variables, are described in section IV.F. They can look strange and uninformative, e.g. `A00004` and `0004`. Either the six or four character label can be used in the $TABLE record. A synonym for the label used in the $TABLE record can be defined in the same manner as synonyms for labels of data item types can be defined with the $INPUT record. E.g. `DCL2=A00004`, in which case the label in the table is `DCL2` rather than `0004`. Use of a synonym in this case can be particularly helpful.

If a variable is defined in a user-supplied code, its values may be displayed provided the variable is listed in MODULE NMPRD4. Its name is not known to NM-TRAN. Instead, it is identified by its position in the MODULE. If it is the Ith variable listed in the MODULE, then the label used in the $TABLE record for its values can be either `COM(I)` or the four character label `:..I`, where the dots indicate leading `0` digits if needed. The latter label is the one used in the table. If I exceeds 999, then the label in the $TABLE record must be `COM(I)`, and the label used in the table is `:..K`, where K is I (mod 1000). A synonym for a label used in the table can be defined with the $TABLE record in the same manner as synonyms for labels of data item types can be defined with the $INPUT record. E.g. `SIZE=COM(20)`, in which case the label `SIZE` is used in the table rather than `:020`. Use of a synonym in this case too can be particularly helpful.

If no abbreviated code is present, then all the variables listed in NMPRD4 may be labeled in the $TABLE record and in the tables themselves in the manner just described. The `COMRES` option in the $ABBREVIATED record is set equal to the length of NMPRD4, and the option is unnecessary and, in fact, must not be used.

If any abbreviated code is also used, the `COMRES` option in the $ABBREVIATED record must be used to reserve n positions in NMPRD4 for the variables defined in user-supplied code; see section B.7. On the $TABLE record, labels for variables defined in abbreviated code and labels for variables defined in user-supplied code may both be used. However, labels of the form `COM(I)` for variables defined in user-supplied code must refer only to the reserved portion of the MODULE, i.e. it must be true that $I \leq n$.

A label for a PRED-defined item defined in an abbreviated code may be given in a $TABLE (or $SCATTERPLOT) record of a problem specification beyond the first problem specification.

The number of different types of PRED-defined items that may be displayed in all tables and scatterplots is given by constant PDT in resource/SIZES.f90; default is 500.

If a synonym is defined for an item type, a different synonym for the same item type cannot be defined on another $TABLE or $SCATTERPLOT record for the same problem.

With NONMEM 7.4, it is possible to exclude data records (i.e., rows) from a table file, using the EXCLUDE_BY option. List3 is comprised of one or more items that are permitted in list1, e.g., data item labels and labels of PRED-defined items in MODULE NMPRD4. Labels in list3 are called exclusion variables. If one or more of them have a non-zero value for a given data record, the row of the table corresponding to that data record will be excluded from the table file. Exclusion variables are not listed in the table file. They have no effect on the printed table and scatters in the NONMEM output, e.g., they do not cause any rows to be deleted from the printed table and are displayed in the printed table.

For example,

```
$PK
  ...
EXCL=0
IF(ID.GE.45.AND.ID.LE.53) EXCL=1
  ...
$TABLE ID TIME DV IPRED CL V1 Q V2 ETAS(1:LAST) EXLUDE_BY EXCL
 NOAPPEND FILE=exctable.par NOPRINT
```

The table file exctable.par will not contain records from subjects 45 to 53.

See Guide Introduction to NONMEM 7 and Guide VIII and on-line help for $TABLE.

NOSUB is an option of $TABLE that is similar to the same option of $ESTIMATION.

Options ESAMPLE, WRESCHOL, SEED, CLOCKSEED, RANMETHOD, PARAFILE, VARCALC, FIXEDETAS are beyond the scope of this guide. See Guide Introduction to NONMEM 7 and Guide VIII and on-line help for $TABLE

III.III.B.17. $SCATTERPLOT Record

```
$SCATTERPLOT list1 VS list2 [BY list3]
             [FROM n1] [TO n2] [UNIT]
             [ORD0|NOORD0] [ABS0|NOABS0] [FIRSTONLY] [OBSONLY]
             [NOSUB=0 | NOSUB=1]
             [UNCONDITIONAL|CONDITIONAL] [OMITTED]
```

E.g.

```
$SCATTERPLOT      (RES WRES) VS TIME BY ID
```

This record requests that the Scatterplot Step be implemented. Each such record defines families of scatterplots. At most 20 families can be defined by all $SCATTERPLOT records in a single problem specification.

This record is optional.

List1, list2, and list3 are each lists of item labels. The same labels that may be used in a $TABLE record (see section B.16) may be used in any list of the $SCATTERPLOT record. Synonyms may be defined in these lists. If a synonym is defined for an item type, a different synonym for the same item type cannot be defined on another $TABLE or $SCATTERPLOT record for the same problem.

A base plot is defined by each pair of labels, (A,B), where A is from list1 and B is from list2. Each data record has associated with it a pair of items (a,b), where a and b have the labels A and B, respectively. The items are also refered to as the values of A and B. The base plot consists of all such pairs of values, except that if A or B is DV, RES, or WRES (see section B.16), or a synonym of one of these labels, and if the MDV data item of the data record is 1, then the pair is excluded from the plot. Unless the base plot is partitioned (see below), the values a are plotted on the ordinate axis of the base scatterplot (the long axis on the printed output), and the values b are plotted on the abscissa axis of the base scatterplot (the short axis on the printed output). The values that are plotted are the same ones that would appear in a table were the values tabled; see section B.16.

List3 is a list of at most two item labels. If list3 is empty, i.e. if the BY option is not used, as many scatterplot families are defined by a $SCATTERPLOT record as are the number of base plots; each family consists of a single base plot. If list3 contains one label, C, this label is used as a separator with each of the base plots. The base plot is not actually output; rather, each base plot is separated into a one-way partitioned family of scatterplots, each of which is output. A different scatterplot of the family is defined for each distinct value c of C. It consists of all pairs (a,b) of the base plot such that the value of C for the data record associated with (a,b) equals c. For example, if C is ID, the base plot A vs B is partitioned into different scatterplots, where all the points of a given scatterplot are all those associated with a particular individual. If there are many individuals in the data set, it may not be wise to output a one-way partitioned family using ID as a separator. If list3 contains two labels, these labels are used as a pair of separators with each of the base plots; each base plot is separated into a two-way partitioned family of scatterplots.

If the UNIT option is used, a 45 degree line (i.e. a line of unit slope) is superimposed on the families of scatterplots. If the ORD0 option is used, a horizontal line through the zero value on the ordinate axis is superimposed on the families of scatterplots. If the ABS0 option is used, a vertical line through the zero value on the abscissa axis is superimposed on the families of scatterplots. If the FIRSTONLY option is used, only the first data record from each individual record may contribute a point to the scatterplot. If the OBSONLY option is used, the scatterplot will only use data records with MDV=0. This option applies independently of FIRSTONLY. It is not necessary when either DV, RES, or WRES is plotted.

All scatterplots resulting from a $SCATTERPLOT record usually use only the data records *from* N_1 *to* N_2, where $N_1 = 1$ and $N_2 =$ may be as large as the total number of data records.† That is, only points corresponding to these data records are included in a scatterplot. The number of points in the scatterplot may be fewer than $N_2 - N_1 + 1$ if any point is excluded as described above, or if the scatterplot is partitioned. Points from records beyond N_2, if any, may be displayed by using options `FROM` and `TO`.

If the `FROM` option is used, N_1 is set to n_1. If the `TO` option is used, N_2 is set to n_2. Of course, n_2 must not be less than n_1.

The `NOSUB` option is the same as the `NOSUB` option of the $ESTIMATION and $TABLE records.

If the `UNCONDITIONAL` option is used, the Scatterplot Step is always implemented. If the `CONDITIONAL` option is used, the Scatterplot Step is implemented only when either the Estimation Step terminates successfully, or the Estimation Step is not implemented.

If the `OMITTED` option is used, the Scatterplot Step is not implemented, even though the $SCATTERPLOT record appears. When used, no other option should be used.

Parentheses surounding a list are optional. However, they should be used when a label can be confused with an alias for an option, e.g. when a label `COND` is used. Two list items may be separated by a comma or spaces. Options cannot be coded among the labels of either list1, list2, or list3.

Synonyms for the prediction, residual, and weighted residual item types can be defined with the $SCATTERPLOT record in the same manner as synonyms for data item types can be defined with the $INPUT record. The reserved labels for these data item types are `PRED`, `RES`, and `WRES`. If for example, the synonym `PR` is to be defined for the prediction item type, then `PR=PRED` should occur among the labels in list1 or list2. The synonym will be used in *all* tables and scatterplots, and control records following the $SCATTERPLOT record defining the synonym can use the synonym.

Each of the options `UNCONDITIONAL`, `CONDITIONAL`, and `OMITTED` applies to the Scatterplot Step as a whole, and if one of these is used, an option contradicting it cannot be used in another $SCATTERPLOT record for the same problem. If a synonym is defined for an item type, a different synonym for the same item type cannot be defined on another $SCATTERPLOT or $TABLE record for the same problem.

List1 and list2 may be separated by an asterisk rather than by `VS`. If either list is enclosed by parentheses, or if both lists contain only one item, then the asterisk or `VS` may be omitted, e.g.

```
$SCAT CP TIME
```

Options cannot be coded between list2 and the `BY` option, but they can precede list1 or follow the last list.

The number of different types of PRED-defined items that may be displayed in all tables and scatterplots is 20.

`ORDZERO` is an alias for `ORD0`. `FIRSTRECORDONLY` and `FIRSTRECONLY` are aliases for `FIRSTONLY`. The `UNIT` and `FROM` option names cannot be abbreviated.

† To restore the NONMEM VI behavior, use TO n1+899.

III.III.B.18. $SUPERPROBLEM Record (nmv)

```
$SUPER [SCOPE=n1] [ITERATIONS=n2]
       [NOPRINT|PRINT]
```

E.g.
```
$SUPER     SCOPE=2 ITERATIONS=10
```

A superproblem is specified by including the $SUERPROBLEM record before the first $PROBLEM record of the superproblem.

A superproblem is a sequence of problems within a NONMEM run that can be iterated. With every iteration each problem is run exactly as was specified for the first iteration. Differences in the results from a given problem between iterations are due to differences to input files that have been made by later problems within the superproblem or, when the problem involves simulation, to different random numbers being used. There can be a one-level nesting of superproblems, one superproblem within another.

III.III.B.19. $WARNING Record (nmv)

```
$WARNINGS [NONE] [n] [RESET | NORESET ]
          [WARNINGMAXIMUM= [ NONE | n | (list) ] ]
          [DATAMAXIMUM= [ NONE | n | (list) ] ]
          [ERRORMAXIMUM=n ]
```

E.g.
```
$WARNING     WMAX=1,EMAX=9999
```

Limits warning messages from NM-TRAN.

NM-TRAN generates various informational and warning messages. These messages are not fatal (NONMEM may still be run) and may be ignored. Warning messages can be suppressed or their numbers limited for a given run by use of the NM-TRAN control record $WARNINGS.

III.III.B.20. $INCLUDE Record (nmvi)

```
$INCLUDE filename [ n ]
```

E.g.
```
$INCLUDE ctlfile2 11
```

The INCLUDE statement was added to NONMEM V. (See Chapter IV K.4). The $INCLUDE record was added to NONMEM VI.

This record causes NM-TRAN to read control stream records from a different file. The character "$" is optional. (This is the only control record for which $ is optional.) The filename must be the first option on the record

n is optional. It gives the number of copies of the file to be read. Default is 1.

III.III.B.21. $NONPARAMETRIC Record (nmvi)

```
$NONPARAMETRIC [MARGINALS|ETAS] [MSFO=filename] [RECOMPUTE]
```

```
                [EXPAND] [NPSUPP=n | NPSUPPE=n]
                [BOOTSTRAP [STRAT=label] [STRATF=label]]
                [PARAFILE=[filename|ON|OFF]
                [UNCONDITIONAL|CONDITIONAL] [OMITTED]

$NONPARAMETRIC     ETAS
```

Requests that the NONMEM Nonparametric Step be implemented.

NONMEM obtains either marginal cumulatives or conditional (non-parametric) estimates of etas. When present, the $ESTIMATION record must also be present and must specify METHOD=1 or POSTHOC.

III.III.B.22. $OMIT Record (nmvi)

```
 $OMIT item1 item2 item3 ...
```

E.g.

```
$OMIT   TIME
```

```
Defines data item types to be excluded from template matching when raw
data averages are computed.  Raw data averages are computed when the
NONMEM RAW_ data item is used.
```

III.III.B.23. $PRIOR Record (nmvi)

```
$PRIOR subroutine [(conditional clause1 ), (conditional clause2) ... ]
       [DISPLAY[=ALL|CNT]]    [ICMAX=n]
       [argument1 , argument2 ...]
```

```
E.g.
$PRIOR TNPRI (PROBLEM 2) PLEV=.9999 ISS=0 IVAR=1
```

Starting with NONMEM VI, a frequency-prior may be specified for data-analytic purposes, or to simulate parameter values (in a Simulation Step). Two NONMEM utility routines help in this regard. NWPRI allows the THETA vector to be constrained by a "normal shaped prior", and/or the OMEGA matrix to be constrained by an "inverse-Wishart shaped prior", where both priors are specified by the user. TNPRI allows all parameters to be constrained by a prior that arises automatically from a NONMEM analysis of a prior data set. (See the PRIOR and NWPRI and TNPRI Help Items and the NWPRI and TNPRI examples.) Implementing PRIOR routines has been made easier by control record, $PRIOR, which may be used instead of a user-written PRIOR routine.

The $PRIOR record provides instructions for a generated PRIOR subroutine in FSUBS. There may be at most 10 $PRIOR records per problem. $PRIOR may not be present if a user-written PRIOR routine is listed on the $SUBROUTINES record. $PRIOR must follow the $SUBROUTINES or $PRED record. $PRIOR is a control record, not a block of abbreviated code. Therefore, only options of the $PRIOR record may be used. E.g., verbatim code may not be used. NONMEM Users Guide VIII and on-line help for $PRIOR describe all the options.

When NWPRI is used, $THETA, $OMEGA, and $SIGMA records are used to provide prior information, in addition to the usual $THETA, $OMEGA, and $SIGMA records. These additional records must be in a specific order, and the record names describe the structure of the information rather than the kind of information. This is described in the on-line help entry for NWPRI. With NONMEM 7.3, there is an easier way to provide this information, using informative record names $THETAP, etc., as described in

the following section.

III.III.B.24. $THETAP, $THETAPV Record (nm73)

III.III.B.25. $OMEGAP, $OMEGAPD Record (nm73)

III.III.B.26. $SIGMAP, $SIGMAPD Record (nm73)

$THETAP value$_1$ [value$_2$] [value$_3$] ...
$THETAPV value$_1$ [value$_2$] [value$_3$] ...
$OMEGAP value$_1$ [value$_2$] [value$_3$] ...
$OMEGAPD value$_1$ [value$_2$] [value$_3$] ...
$SIGMAP value$_1$ [value$_2$] [value$_3$] ...
$SIGMAPD value$_1$ [value$_2$] [value$_3$] ...

E.g.

```
; Prior information of THETAS (NTHP=4 of them)
$THETAP (2.0 FIX) (2.0 FIX) (2.0 FIX) (2.0 FIX)
; Variance to prior information of THETAS (NTHPxNTHP=4x4 of them).
$THETAPV BLOCK(4)
10000 FIX
0.00 10000
0.00  0.00 10000
0.00  0.00 0.0 10000
```

The records may be in any order, and the options of $PRIOR need not be specified. The name of the record describes the kind of information it gives to NWPRI, rather than the structure of the information, as follows:

$THETAP for THETA priors
$THETAPV for variance-covariance matrix for THETA's
$OMEGAP for OMEGA prior
$OMEGAPD for degrees of freedom (or dispersion factor) for OMEGA prior
$SIGMAP for SIGMA prior
$SIGMAPD for degrees of freedom (or dispersion factor) for SIGMA prior

III.III.B.27. $CHAIN Record (nm72)

```
$CHAIN [FILE=filename]
       [FORMAT=s1] [ORDER=xxxf]
       [NOTITLE=[0|1]] [NOLABEL=[0|1]]
       [ISAMPLE=n][NSAMPLE=n]
       [SEED=n] [SELECT=n]
       [RANMETHOD=[n|S|m] ]
       [CTYPE=[0|1|2|3|4]]
       [DF=n] [DFS=n] [IACCEPT=n]
```

E.g.

```
$CHAIN FILE=example1_previous.txt NSAMPLE=0 ISAMPLE=-1000000000
```

Supplies initial estimates for an entire problem. Option METHOD=CHAIN of the $ESTIMATION record will only set thetas, omegas, and sigmas for initial values of the estimation process. Its scope is

therefore limited in that it will not impact the parameters used in simulating data for the Simulation step. To introduce initial thetas, omegas, and sigmas that will cover the entire scope of a given problem, use the $CHAIN record.

III.III.B.28. $SIZES Record (nm72)

`$SIZES` [constant=value] [constant=value] ...

E.g.
```
 $SIZES LIM1=30000 MAXFCN=2000000 NO=500
```

$SIZES is optional. If present, it must precede the first $PROBLEM or $SUPER record. by the $ANNEAL record.

Certain constants are used in NM-TRAN, NONMEM and PREDPP. These are in file resource/sizes.f90. The user may override many of the constants with the $SIZES record. Any non-zero value specified on the $SIZES record overrides both the default and the value that NM-TRAN would have specified. (A value of 0 is ignored.) As of NONMEM 7.3, as an alternative to modifying sizes.f90 to very large maximum sizes, you can tell NMTRAN the maximum size that may be needed by specifying a $SIZES constant as a negative value. Thus, a user can give NMTRAN permission to deal with all problems that have data input files that have up to 1000 data items, and up to 150 etas and epsilons, and up to 200 thetas, by the following:

```
 $SIZES PD=-1000 LVR=-150 LTH=-200
```

but the values of these constants when the NONMEM executable is constructed will be only what is needed for the particular problem.

III.III.B.29. $ANNEAL Record (nm73)

`$ANNEAL` number-list1:value1 number-list2:value2 ...

E.g.
```
$ANNEAL 1-3,5:0.3 6,7:1.0
```
Sets starting diagonal Omega values for purposes of simulated annealing by NONMEM subroutine CONSTRAINT. This facilitates EM (Expectation Maximization) search methods.

In the above example, initial values of OMEGA(1,1), OMEGA(2,2), OMEGA(3,3), and OMEGA(5,5) are set to 0.3, while initial OMEGA(6,6) and OMEGA(7,7) are set to 1.0.

III.III.B.30. $ETAS Record (nm73)
III.III.B.31. $PHIS Record (nm73)

`$ETAS` [[$value_1$ [$value_2$] [$value_3$] ... [$value_n$]]
[`FILE`=filename `FORMAT`=s_1 [`TBLN`=n]]
`$PHIS` [[$value_1$ [$value_2$] [$value_3$] ... [$value_n$]]
[`FILE`=filename `FORMAT`=s_1 [`TBLN`=n]]

E.g.

```
$ETAS 0.4 3.0 3.0 5.0
$PHIS 0.4 3.0 3.0 5.0
$ETAS FILE=myprevious.phi FORMAT=s1pE15.8 TBLN=3
$PHIS FILE=myprevious.phi FORMAT=s1pE15.8 TBLN=3
```

Specifies Initial Values for Etas or Phis

By default, the initial value used for ETA's in the Estimation Step search is 0. The $ETAS and $PHIS records provide different initial estimates. When the record is $PHIS, values are entered as phi values, convenient for EM methods. The eta values will then be evaluated as eta(i) = phi(i) – mu(i) for each eta, where mu(i) = mu_i is evaluated according to their definitions in the abbreviated code.

III.III.B.32. $LEVEL Record (nm73)

`$LEVEL item=(`$n_1[m_1]$, $n_2[m_2]$ ...) ...`

E.g.

```
$LEVEL SID=(4[1],5[2],6[3])
```

Specifies nested random levels above subject ID. Item is the name of a data item listed on $INPUT. It defines an additional nested random level and is referred to as a "super ID" data item. The notation $n_k[m_k]$ states that ETA(n_k) is associated with this super ID item, and ETA(m_k) is nested within ETA(n_k).

III.III.B.33. $DEFAULT Record (nm74)

```
$DEFAULT [NOSUB=[-1|0|1]]
```

E.g.

```
$DEFAULT NOSUB=1
```

Specifies certain defaults for NONMEM. If present, it must appear following $PROBLEM record.

The `NOSUB` option is the same as the `NOSUB` option of the $ESTIMATION and $TABLE and $SCATTER records.

IV. Abbreviated Code

IV.IV.A. Introduction

It can be seen from the example in chapter I that an abbreviated code is much like a FORTRAN code. Indeed, nearly every statement of an abbreviated code is syntactically a FORTRAN statement. The reader is advised to become familiar with very basic FORTRAN coding, in particular the use of assignment statements, conditional statements, arithmetic expressions, and logical expressions. However, not all FORTRAN constructs can be used in an abbreviated code, an abbreviated code does not constitute a complete FORTRAN coded subroutine (this is not a disadvantage), and certain symbols used in an abbreviated code have (semantical) meaning different from standard FORTRAN meaning. For the purposes of developing user-supplied type subroutines for NONMEM, restrictions imposed by abbreviated code (on the use of certain FORTRAN constructs) are not many; usually, use of abbreviated code is quite adequate. The advantages to using an abbreviated code are considerable and outweigh some inconvenience which the restrictions impose.

Nonetheless, one may still want to use FORTRAN statements not allowed in abbreviated code. An escape mechanism is available. Verbatim code can be inserted into abbreviated code. This is FORTRAN code that is itself inserted without change, i.e. verbatim, into the FORTRAN subroutine generated by NM-TRAN. Because this code is essentially not processed by NM-TRAN, other than to copy it into the generated routine, symbolic differentiation is not used with this code. For this and other reasons, verbatim code should only be used by those who understand well how generated subroutines are structured.

The purpose of an abbreviated code is to specify the computation of special quantities called the left-hand quantities . These quantities are symbolized by *reserved* variables or arrays elements. There are mandatory left-hand quantities and optional left-hand quantities. For each mandatory (optional) left-hand quantity, there must (may) exist some assignment or conditional assignment statement in the abbreviated code that defines the symbol used for the quantity. In the case of an abbreviated code for PRED there is one mandatory left-hand quantity, symbolized by the reserved variable `Y`, and described below. There are other optional left-hand quantities symbolized by reserved array elements `COM(n)`, which play a minor role; see section E.3.

There are other special quantities called right-hand quantities , which can be used in the computation, and these are also symbolized by reserved variables or array elements. The symbols are used in abbreviated code as though they are already defined. These quantities are: the data items of a data record, symbolized by the variables given by the labels and synonyms specified in the $INPUT record; values of the θ parameters, symbolized by the array elements `THETA(1)`, `THETA(2)`, etc.; values of η and ε variables, symbolized by the array elements `ETA(1)`, `ETA(2)`, ..., and `EPS(1)`, `EPS(2)`, ..., ; others where noted.

Generated PRED subroutines contain the declaration
`USE NMPRD_REAL,ONLY: ETA,EPS`
so that ETA and EPS values may be obtained from NONMEM's GETETA, SIMETA, and SIMEPS subroutines, as appropriate (for some discussion of SIMETA, see section III.B.13).

The symbols can be used in the right-hands of assignment statements and, with some restrictions for `ETA`'s, `EPS`'s, and all other random variables (see below), they can also be used in the right-hands of conditional assignment statements and in the conditional expressions of conditional statements. Except where noted, a symbol for a left-hand quantity, once it has been has been defined in abbreviated code, can also be used similarly. For example, `Y` may be used on the right, e.g. `LOGY=LOG(Y)`. Also, variables other than left-hand quantities can be defined by abbreviated code and used similarly. On the other hand, *a variable defined by abbreviated code cannot be used as a label in a $INPUT record.* This means that quantities cannot be stored into the data record using abbreviated code (see though, section I).

The array elements ETA(1), ETA(2), ..., and EPS(1), EPS(2), ..., can also be regarded as symbolizing the η and ε *random variables*, not simply values of these variables. For practical purposes, this means that if another variable A is defined in abbreviated code in terms of these elements, then A can be viewed as a function of the η and ε variables, and NM-TRAN generates code to compute various derivatives of A with respect to the η and ε variables. These derivatives are called the η- and ε-derivatives . The variable A itself can be regarded as symbolizing a random variable, so that if it is used to define yet another variable B, the η- and ε-derivatives of B are also computed, and so on. In general, in addition to the η and ε random variables, any variable or array element whose value depends on the value of an η or ε variable is itself regarded as a random variable.† In the example of chapter I, all variables defined with the string of assignment statements, i.e. the variables KE to Y, are random variables.

The variable Y is (typically) an example of a random variable. If PRED is called with a data record containing an actual observation, Y symbolizes the value of the observation under the statistical model. If PRED is called with a data record with a missing dependent variable data item equal to 1, Y symbolizes a prediction. This prediction, however, is obtainable from the same code used to give the model-based value for the observation, and one need not (although one may) give different code depending on the value of the MDV data item. With odd-type data, Y is a conditional likelihood for the observation. See J.1. "Indicator Variables, Random Variables and Recursion Code" for an example of odd-type (categorical) data.

A random variable may be used in the right or left-hand of a conditional assignment statement, i.e., it may be defined conditionally. A random variable may be used in the conditional expression of a conditional statement, but in this case care should be taken that either the statement is executed only during the Simulation Step (see section D), or that as a result of the expression being true, the EXIT statement is executed (see section G.2). A random variable may *not* be defined in a nested conditional assignment statement. An alternative way to code this type of computation is illustrated in section IV.K.1 Random Variables and Recursion code below.

An abbreviated code is part of an NM-TRAN control record. For example, an abbreviated code for PRED is a part of a $PRED record. The statements comprising the code are contained in one or more continuation records of the control record. However, only one statement can be contained in any one record. A $PRED record along with all its continuation records is called a $PRED block . NM-TRAN comments can be included in an abbreviated code in the usual way, e.g.

```
Y=THETA(1)*WT+THETA(2)*AGE+ETA(1)   ;linear regression model
```

Using symbols η's and ε's, and their counterparts in abbreviated code, the ETA's and EPS's, can be confusing. If the data are population data, intraindividual effects are represented by ε variables, but with single-subject data, they are represented by η variables; see section II.C.4. NM-TRAN abbreviated code offers a way to alleviate this confusion. With single-subject data the array element ERR(n) may be used instead of ETA(n); using one has the same effect as using the other. With population data the array element ERR(n) may be used instead of EPS(n). Therefore, in either case one can be safe in always using the symbols ERR(n) to represent the random intraindividual effects.

IV.IV.B. General Restrictions

Each statement may be placed anywhere within its containing record, one statement per record.
FORTRAN 95 continuation lines are permitted. The character & should be placed at the end of the line to be continued.

†Any variable A that can be regarded as a random variable, is called a true-value variable in the first edition of this guide, in certain other NONMEM Users Guides, and in NM-TRAN error messages. This is because the variable can also be regarded as symbolizing the true, albeit unknown, value of the random variable that gives rise to the data at hand.

For example, the following two code fragments are identical:

```
CL=EXP(THETA(1)*WERT+EPS(1))
```

```
CL=EXP(THETA(1)*WERT &
   +EPS(1))
```

Comments may be included on any line after the semicolon character ";". An exception is the FORTRAN continuation character &, which may not be followed by a comment.

No statement types other than assignment, IF, THEN, ELSE, ELSEIF, ENDIF, DO WHILE, ENDDO, CALL, WRITE, PRINT, RETURN, OPEN, CLOSE, REWIND. e.g., no GOTO, READ, FORMAT.

For use of WRITE, PRINT, OPEN, CLOSE, and REWIND statements in abbreviated code, see Guide VIII and on-line help.

A special statement type, EXIT, is permitted. Both lower and upper case may be used for all user-defined and reserved words. FORTRAN statement numbers may not be used.

All variables or array elements are treated as having type DOUBLE PRECISION i.e., as double precision floating-point numbers, except where noted. Any valid FORTRAN numerical constant can be used, but if it is an integer constant, it is treated as a DOUBLE PRECISION constant.

All variable names consist of 1-20 letters (A-Z), numerals (0-9), and the character '_', beginning with a letter. (The length 20 is specified by constant SD in SIZES).

Names of array elements which are left- or right-hand quantities can appear. Variable names of exactly six characters, starting with `A`, `B`, or `C` and followed with various combinations of five of the digits `0-9` (e.g. `B00003`), and the variable name `BBBBBB`, are reserved for internal use and may not be used in an abbreviated code; see section F.

Use of the exponential operator ** is allowed. However, as with FORTRAN, it is more efficient to use the multiplication operator * when the exponent is a small (≤ 4) integer. Although when B=0 and P>0 the computation R=B**P is mathematically defined and is 0, some floating point processors fail with a floating point error. Powers are computed as follows (NONMEM V):

```
Q=0
IF (B.EQ.0) Q=1
R=(1-Q)*(B+Q)**P
```

FORTRAN functions LOG (natural log), LOG10, EXP, SQRT, SIN, COS, TAN, ASIN, ACOS, ATAN, ABS, INT, MIN, MAX, MOD may be used. The NONMEM functions PHI and GAMLN may be used.

Functions LOG, LOG10, EXP, SQRT, SIN, COS, TAN, ASIN, ACOS, ATAN, ABS, GAMLN may have random variable arguments and the partial derivatives are computed. Note that the partial derivative of ABS(X) with respect to η is mathematically undefined at X=0. We are arbitrarily defining it to be $\frac{\partial X}{\partial \eta}$. If the value of X affects the value of the objective function, X must always be either positive or negative. If the argument of GAMLN is a random variable, it must always be positive. Function PHI may have a random argument but no partial derivates are computed.

The INT, MOD, MIN, and MAX functions produce discontinuous results. No partial derivatives are computed. If they are used outside of a simulation block and the function value affects the value of the

objective function, then an error in the NONMEM Estimation Step will probably occur.

With NONMEM 7.4, routines are available that protect against domain violations, divide by zero, and floating point overflows. Each of these routines start with the letter P, followed by the name of the mathematical operation they are to perform. For example, PLOG is the protective code routine that performs the LOG operation. The protect functions and the $ABBREVIATED PROTECT record are discussed below in section IV.J.6. PROTECT functions.

User-written functions FUNCA, FUNCB and FUNCC (called abbreviated functions) may be used as right-hand quantities in abbreviated codes. Such a function may have a single argument. However, it may be a vector. A function may depend on arguments which may be random variables, in which case the function too becomes a random variable.

Left-hand quantities VECTRA(n), VECTRB(n), VECTRC(n) may be used, and these become elements of reserved vectors VECTRA, VECTRB, VECTRC. A subscript n must be a positive integer constant. Any reserved vector may be used with any FUNC, e.g. X=FUNCA(VECTRB).

The code for the functions must be written by the user in FORTRAN (See Guide VIII ABBREVIATED FUNCTION Help Item.) The code must be in a file included using the $SUBROUTINE OTHER option, as discussed in Chapter III.B.6.

With NONMEM 7.3, reserved function names are FUNCA through FUNCI.

With NONMEM 7.4, reserved function names are FUNCA through FUNCZ. Reserved vector names are VECTRA through VECTRZ.

With NONMEM 7.4, the $ABBREVIATED FUNCTION and $ABBREVIATED VECTOR records may be used to declare user-defined functions and vectors. See section IV.J.7, below

With NONMEM 7, the $ABBREVIATED DECLARE record may be used to declare user-defined arrays and vectors, which may be used for random-quantities. INTEGER variables and DOWHILE variables (which are also integer) may be defined and may be used for looping. Integer variables and integer expressions may be used as as subscripts for THETA's and user-defined arrays. Integer variables and integer expressions may be used as subscripts for other arrays, e.g., random variables ETA(I), but only in a WRITE or PRINT statement. Declared variables are global, i.e., are defined in all blocks of abbreviated code except $MIX. Declared variables are automatically initialized to 0.
Here is an example.

```
$ABBR DECLARE INTEGER I, REAL X(10)
$PRED
   I=1
   X(I)=THETA(I)
```

For more examples, see IV.IV.J.3. The DOWHILE Statement

Previous restrictions on nested parentheses and conditional definition of random variables are removed. Specifically:

Parentheses may be nested, e.g.,

```
A=(THETA(1)+(THETA(2)+C)*2)/E
```

Parentheses may be used within logical expressions, e.g.,

```
IF (Q.EQ.(R+C)/D) A=3
```

Parentheses may be used within an expression for an argument of a FORTRAN library function, e.g.,

```
A=EXP(-(THETA(1)+C)*TIME)
```

Conditional statements may be nested if random variables are not defined. The default is 10 levels of nested IF's. This may be increased. E.g.,

```
A=THETA(3)
IF (Q.EQ.1) THEN
   A=THETA(1)
   IF (R.NE.B) A=A+THETA(2)
ENDIF
```

An ELSE IF clause may be used.

Random variables may appear in the right and left-hands of conditional assignment statements, e.g.,

```
IF (Q.EQ.1) A=THETA(1)*(1+ETA(1))
IF (Q.EQ.0) A=THETA(2)*(1+ETA(1))
```

A random variable may be redefined, e.g.,

```
A=TVV+TVV*ETA(2)
A=A+THETA(4)
```

A variable on the left-hand of an assignment statement may also be used on the right-hand even if the assignment redefines the variable to be a random variable, e.g.,

```
A=THETA(1)
A=A*EXP(ETA(1))
```

The logical operators .NOT., .AND., .OR., may not be used within parentheses.
Here is an example of how to write code to avoid this restriction. (See Guide VIII and on-line help for ignore/accept example.)
Suppose ACC is a variable that is to have values 0/1 depending on the value of A and B. The desired code is:

```
ACC=0
IF ((A == 1.OR.A == 2).AND.B<100) ACC=1
```

There are two workarounds. One is to clear the parentheses:

```
ACC=0
IF (A==1.AND.B<100.OR.A==2.AND.B<100) ACC=1
```

(this is always possible, no matter how complicted the conditional expression).
The second is to use several statements. There are may ways to do this. In the following, the .AND. of multiple conditions is false if any of the conditions is false.

```
ACC=1
IF (A.NE.1.AND.A.NE.2) ACC=0
IF (B.GE.100) ACC=0
```

An IF-ENDIF block must be completely within a contiguous block of
the NM-TRAN record containing the abbreviated code.

invalid

```
$PRED
   IF (A.EQ.B) THEN
      Y=W1
$PRED
   ELSE
      Y=W2
   ENDIF
```

valid

```
$PRED
   IF (A.EQ.B) THEN
      Y=W1
   ELSE
      Y=W2
   ENDIF
```

IV.IV.C. Restrictions Specific to an Abbreviated Code for PRED

There are further restrictions specific to each type of abbreviated code. For an abbreviated code for PRED, certain variables which occur as arguments to the PRED subroutine may not be used. These are `DATREC`, `INDXS`, `G`, and `H`. The variable `F` can be used; it has no special meaning in the code.

IV.IV.D. Extensions Specific to an Abbreviated Code for PRED

Extensions specific to each type of abbreviated code can exist. In general, any feature described in this section for $PRED may be used with some or all other blocks, such as PREDPP blocks $PK and $ERROR. Exceptions are noted when appropriate. For an abbreviated code for PRED, certain reserved variables may be used to symbolize some special right-hand quantities. These variables, `ICALL` and `NEWIND`, occur as arguments to the generated PRED subroutine. They are now described.

The variable `ICALL` has the value 0 if the call to PRED is the first call to PRED in the run (the run initialization call). It has the value 1 if the call to PRED is the first call to PRED in the problem or superproblem (a problem initialization call). It has the value 3 if the call to PRED is the last call to PRED in the problem (a problem finalization call). It has the value 2 if the call to PRED is a regular call during data analysis, and the value 4 if the call is a regular call during data simulation. It has the value 5 if the call to PRED occurs when expectations are being computed (the marginal data item MRG_ has a non-zero value for some records). It has the value 6 if the call to PRED occurs when raw data averages are being computed (the raw-data-average data item RAW_ has a non-zero value for some records).

Abbreviated code may test the value of ICALL. Such code defines specific kinds of blocks. For example:

```
IF (ICALL==0) THEN
 Run Initialization block
ENDIF
IF (ICALL==1) THEN
 Problem Initialization block
ENDIF
IF (ICALL==2) THEN
 Data Analysis block
ENDIF
IF (ICALL==3) THEN
 Problem and Run Finalization block
ENDIF
```

```
IF (ICALL==4) THEN
 Simulation block
ENDIF
IF (ICALL==5) THEN
 Expectation block
ENDIF
IF (ICALL==6) THEN
 Data Average block
ENDIF
```

The specific rules for each type of block are described in the help NONMEM Users Guide VIII and online help. In particular, for ICALL 0, 1, 3 blocks, see help item for Initialization-Finalization block.

At initialization and finalization calls the data items occuring as right-hand quantities are those of the first data record. At the run initialization call the THETA's are 0. At a problem initialization (finalization) call, the THETA's are the initial (final) estimates. During an initialization (finalization) call, the ETA's are 0 (0, or conditional estimates for the first individual if conditional estimates have been requested).

The variable NEWIND has value 0 if the data record is the first record of the data set. It has the value 1 for the first record of the data set if the THETA value does not differ from value at last call with this record, and PRED is non-recursive (see disucussion in III.III.B.15), or, more usually, if the data record is the first data record of the second or subsequent individual record. It has the value 2 if the data record is the second or subsequent data record of an individual record. With single-subject data individual records are defined in such a way that data records are contained in a number of different individual records; see section II.C.4.1. Therefore, except when the data record is the first data record of the data set and the value of NEWIND=0, the value of NEWIND can be 1 or 2.

IV.IV.E. Some Special right-hand Quantities

Certain reserved variables listed in NONMEM MODULEs (and available to any subroutine for which an abbreviated code may be given) may be used in abbreviated code to symbolize some special right-hand quantitites. Some of these are now described. They are intended for use in advanced NONMEM applications. This section may be ignored by beginning NONMEM users. Some of these variables are associated with a data record or individual record. They change values with calls to NONMEM PASS routine. This is discussed in section IV.J.2, below. See help item for variables in modules.

IV.IV.E.1. MIXNUM and MIXEST and MIXP

These are right-hand variables that can be used when a mixture model is used. With a mixture model there are one or more submodels that can be used to describe an individual's data. PRED (PK, ERROR, etc. if PREDPP is used) must be able to compute its outputs under each of these submodels. However, with each call to the routine, the outputs are computed only under one submodel. With data from a given individual's record, and when ICALL=2 or 4, MIXNUM is the number of the submodel of the mixture that should be used to obtain the outputs. It is an input to PRED (or PREDPP) set by NONMEM. The submodels are enumerated 1, 2, ..., m, where m is the number of possible submodels that can be used to describe the individual's data, a number returned in an argument of the routine MIX; see Guide VI, section III.L.2. The number m can vary between individual records. When ICALL=1 or 3, MIXNUM is 1.

The variable MIXEST can be used when a mixture model is used. With data from a given individual's record, and when ICALL=3, MIXEST is the number of the submodel of the mixture that "best" describes the individual's data. It is an output (result) or consequence from the estimation set by NONMEM and changes value with calls to PASS. The best submodel is selected according to a Bayesian computation, conditional on the individual's data and on the final estimates of the population parameters. When ICALL=1, 2, or 4, MIXEST is 1.

MIXP is an array variable. They are the mixture probabilities P(i) computed by subroutine MIX or by the $MIX block of abbreviated code. (In the $MIX block, MIXP is simply the P array.)

See the $MIX block, below.

IV.IV.E.2. COMACT

The variable COMACT can be used to identify those calls to the routine with which PRED-defined items (see section F) are obtained by NONMEM for the purpose of displaying these items in tables and scatterplots. COMACT is 1 or 2 or 3 with such calls; otherwise it is 0.

When COMACT is non-0, NONMEM is making a copying pass. The data records are being passed to PRED for the purpose of computing values of variables which will be copied from NMPRD4 for tables and scatterplots. NONMEM only makes a copying pass when PRED-defined items are listed in $TABLE or $SCATTER records.

Abbreviated code that tests COMACT defines a copying block, e.g.,

```
IF (COMACT>0) THEN
 Copying block
ENDIF
```

There are (up to) three sets of calls with the data records of an individual record. With ETA's set to 0, there is a first set of calls with COMACT=1. If conditional estimates are requested, then with ETA's set to these estimates, there is a second set of calls with COMACT=2. If conditional (nonparametric) estimates of etas are requested, there is a third set of calls with COMACT=3.

If a mixture model is used, with each value of MIXNUM (see section E.1) there are more than two sets of calls. With ETA's set to 0, there is a set of calls *for each distinct value* of MIXNUM with COMACT=1. If conditional estimates are requested, then with ETA's set to these estimates, there is a set of calls *for each distinct value* of MIXNUM with COMACT=2. Note that nonparametric estimates cannot be obtained with a mixture model.

PRED-defined items are stored in variables defined in PRED (PK or ERROR if PREDPP is used) (see section F). Normally, these items may change from call to call. That is, at one call an item is computed and stored in a variable V; then it is available in V at the start of the subsequent call; at that call, though, another item may be computed and stored in V. Therefore, if an item is computed at a call C_1 with a particular data record (of the individual record) when COMACT=1, it may no longer be available at a call C_2 with the same data record when COMACT=2, due to there being (in general) multiple calls to PRED (PK and ERROR) with different data records of the individual record between C_1 and C_2. There are, however, situations where it is desired that the item be available at C_2. This problem is solved by making a special use of NONMEM MODULE NMPRD4.

Items may change from call to call whether they are stored in a locally defined variable or in a globally defined variable listed in NMPRD4 (see sections III.B.16-17). However, an initial section of NMPRD4 can be identified to NONMEM as the save region . All items stored in this region at a call to PRED (PK or ERROR) with a particular data record when COMACT=1 are available at a call with the same data record when COMACT=2. In fact, with mixture models in mind, if at any call to PRED (PK or ERROR) with a particular data record (when COMACT=1 or 2 and MIXNUM has any value), an item is stored in a variable listed in the save region, then the item is available at any subsequent call with the same data record (when COMACT=1 or 2 and MIXNUM has any value).

The save region of NMPRD4 is comprised of the first n_2 positions, where n_2 is the integer given with the COMSAV option on the $ABBREVIATED record (see section III.B.7). If the option is omitted, or if n_2 is 0, there is no save region.

PRED-defined variables that are defined in a copying block are referred to as implicit SAVE variables. Another way to store an item in a variable listed in the save region is described in the next section.

IV.IV.E.3. COM(n)

There are additional left- and right-hand quantities symbolized by the array elements COM(n) . The nth element refers to the nth item stored in MODULE NMPRD4. These quantities are useful when a user-supplied routine has stored items in this MODULE, and these items are to be used by abbreviated code. Abbreviated code may also store items in reserved COM elements. Typically, the first n_1 positions of NMPRD4 are reserved to allow user-supplied routines to store PRED-defined items in these positions so that they may be displayed in tables and scatterplots (see section III.B.16). The integer n must not exceed the integer n_1, where n_1 is the integer specified with the COMRES option on the $ABBREVIATED record (see section III.B.7). Even if the computation of COM(n) depends on η's or ε's, COM(n) is not regarded as a random variable. That is, η- and ε-derivatives of COM(n) are always 0, but never actually computed.

PRED-defined items defined by abbreviated code can also be stored in the first n_1 positions, since the array element COM(n) can function as an optional left-hand quantity. This can facilitate the communication in both directions between a user-supplied routine and a routine specified by abbreviated code. It can even be used to allow two-way communication between two abbreviated codes. (Regular variables used on the left in one abbreviated code cannot be used on the left in another abbreviated code, as long as variables are listed in NMPRD4, but see sections H and III.B.7.) Another use of these left-hand quantities is to allow abbreviated code to specify that an item be stored in the save region of NMPRD4 (see section E.2). COM(i) variables that are defined in a copying block are referred to as explicit SAVE variables. Implicit and explicit SAVE variables cannot both appear in abbreviated code.

The rule given above still holds: COM(n) *is not regarded as a random variable*. This means that while COM(n) may have a value that depends on η's and epsilons's, the η- and ε-derivatives of COM(n) are always 0. This in turn implies that abbreviated code cannot define a random variable to be listed in the save region of NMPRD4.

IV.IV.E.4. NONMEM Counter Variables

Counter variables are right-hand variables that NONMEM sets so that user code can determine where NONMEM is in a problem.

NIREC, NDREC (nmvi)
NPROB, IPROB (nmv)
S1NUM, S2NUM, S1NIT, S2NIT, S1IT, S2IT (nmv)
NREP, IREP (nmv)
LIREC, NINDR , INDR1, INDR2 (nmvi)

Counters include (in the order above):
record counters;
problem iteration counters;
super-problem iteration counters;
simulation repetition counters;
number of data records in the individual record; number of individual records in the data set containing an observation record, and the indices of the first and last such individual records.

Some of these variables may be used as right-hand quantities in abbreviated code for certain blocks of code, e.g., only in initialization/finalization blocks. They change values if appropriate during a pass thru the data set, e.g., during initialization/finalization.

IV.IV.E.5. Other Reserved Variables

The following lists some of the other reserved variables. Variables that can be used on the left-hand provide information to NONMEM. Variables that can be used on the right-hand are set by NONMEM.

MSEC, **IFIRSTEM** (right-hand) (nmiv, nm72)

These are the "partial derivative indicator" variables set by NONMEM. NONMEM does not always require that first or second η- and ε-derivatives be computed. MSEC=1 when NONMEM is expecting second-partial eta-derivatives with the current call to PRED; 0 otherwise. IFIRSTEM=1 when NONMEM is expecting first-partial eta-derivatives with the current call to PRED; 0 otherwise. These derivatives are almost always needed by classical NONMEM methods with population data. However, NONMEM does not always use these derivatives for the newer Bayesian methods. Generated code in FSUBS copies IFIRSTEM to a local variable, FIRSTEM. This allows the abbreviated code to set FIRSTEM=1 so that first derivatives are computed in unusual circumstances.
(See Turning on First Derivative Assessments for EM/Bayes Analysis(NM72) in NONMEM 7 Guide.)

Note that NM-TRAN rearranges the order of statements in FSUBS. Statements that compute second-partial eta-derivatives are collected together into blocks of second derivative code to be executed only with MSEC is 1. Starting with NONMEM 7.2, statements that compute first-partial eta-derivatives are collected together into blocks of first derivative code to be executed only with FIRSTEM is 1. Option NOFASTDER of the $ABBREVIATED record prevents NM-TRAN from doing this and restores the order of statements in FSUBS to what it was in previous versions.

NEWL2 (right-hand) (nmv)

NEWL2 = 1 if the data record is the first of an L2 record;
NEWL2 = 2 otherwise.
NEWL2 may be used as a right-hand quantity in $PRED and $ERROR blocks, and in an $INFN block in conjunction with PASS. It changes with calls to PASS.

OBJECT (right-hand) (nmv)

The final value of the objective function. This value should only be used at ICALL = 3 (finalization block) in $INFN or $PRED.

IERE and **IERC** (right-hand) (nmv) The return codes from the Estimation Step and Covariance Steps, respectively. Values of 0 indicate normal termination. They should only be used at ICALL = 3 (finalization block) in $INFN or $PRED.

RPTI,RPTO,RPTON,PRDFL (left-hand) (nmvi)

The "Repetiton Variables" may be used to mark a data record as a repetition base when NONMEM's repetition feature is used, as an alternative to use of the RPT_ data item.

With Version VI, variables RPTI, RPTO, RPTON and PRDFL can be set and/or tested, thus allowing a subsequence of data records to be repeatedly passed to PRED multiple times before the next data record following the last record of the subsequence is passed. Subsequences can be nested. This "repetition feature" allows e.g. kinetics to be computed by a convolution integral. The user can exercise control over which pass through a sequence it is, during which the output from PRED with a given data record will be used by NONMEM. This "PRED control" allows e.g. PRED output with a given record to involve computations over subsequent, as well as prior, records. (See the NMPR10 Help Item and the Repetition 1 and Repetition 2 examples.)

YLO, **YUP** (left-hand) (nmvi)

With a given data record, either of the limits YLO or YUP may be set so that during the analysis an interval is defined in which (or outside of which) an observation is conditioned to exist. May be set in $PRED and $ERROR.

PR_Y (right-hand) (nmvi)

PR_Y is the estimated probability that an observation will fall within (or outside) the interval. PR_Y may be used as a right-hand quantity in abbreviated code for $PRED, $PK, $ERROR, and $INFN blocks.

CTLO, **CTUP** (left-hand) (nmvi)

An observation may be the event that the value of a normally distrib- uted variable falls in a given interval. CTLO is the lower endpoint of the interval. CTUP is the upper endpoint of the interval. May be set in $PRED or $ERROR.

PR_CT (right-hand) (nmvi)

PR_CT is the estimated probability that an observation will be of the category in question. PR_CT may be used as a right-hand quantity in abbreviated code for $PRED, $PK, $ERROR, and $INFN blocks.

F_FLAG (left-hand) (nmvi)

The data may include both population data and odd-type data. PRED may compute both predictions and likelihoods. F_FLAG=0 indicates that Y or F is being set to a "prediction" of the observation. F_FLAG=1 (or 2) indicates that Y or F is being set to a likelihood (or -2 log likelihood) value for this particular observation. May be set in $PRED and $ERROR. See J.1. "Indicator Variables, Random Variables and Recursion Code" for an example.

IIDX, **CNTID** (right-hand) (nmvi)

These array variables contain values of the ID data item and individual contributions to the objective function. The values are in data-set order. With NONMEM 7, the additional output file root.phi contains the same information.

SKIP_ (left-hand) (nmvi)

SKIP_ is used to control premature termination of a problem (with subproblems), superproblem or superproblem iteration. In a finalization block of abbreviated code one may set SKIP_ and/or use the following phrases:

```
END PROBLEM                                (same as SKIP_=1)
END SECOND-LEVEL SUPERPROBLEM              (same as SKIP_=3)
END FIRST-LEVEL SUPERPROBLEM               (same as SKIP_=5)
END SECOND-LEVEL SUPERPROBLEM ITERATION    (same as SKIP_=2)
END FIRST-LEVEL SUPERPROBLEM ITERATION     (same as SKIP_=4)
```

TEMPLT (right-hand) (nmvi)

This right-hand array is used in two different contexts.

At ICALL=6, the template data record is stored in TEMPLT and may be used on the right in a data average block.

When MIX is called, the first data record of the individual record is stored in TEMPLT and may be used in $MIX.

Items of the template record may be referred to by position or by label, e.g., TEMPLT(1) or TEMPLT(ID).

OMEGA(n,m), **SIGMA(n,m)** (right-hand) (nmvi)

Current values of OMEGA and SIGMA in $PRED, $PK, $ERROR at ICALL 2. In $INFN and $PRED at the run initialization call, they are 0's. At a problem initialization (finalization) call, they are the initial (final) estimates. May specify individual elements or the entire array. In the case of OMEGA and SIGMA, may also specify BLOCK or DIAG.

SETHET(n), **SEOMEG(n,m)**, **SESIGM(n,m)**, **SETHETR(n)** (right-hand) (nmvi)

The standard errors of the estimates of THETA, OMEGA, and SIGMA. In $INFN and $PRED, at ICALL 3. May specify individual elements or the entire array. If $THETAR is used, SETHET contains standard error of internal values, and SETHETR contains standard errors of reported values. If $THETAR record not used, SETHET=SETHETR. In the case of OMEGA and SIGMA, may also specify BLOCK or DIAG.

THSIMP, **THSIMPR OMSIMP**, **SGSIMP** (right-hand) (nmvi)

These are the values of THETA, THETAR, OMEGA and SIGMA that are produced during a Simulation Step using the user-supplied routine PRIOR. (THETA=THETAR if $THETAR record not used.) May be used in $PK, $ERROR, $INFN and $PRED blocks. After being set during the Simulation Step, they remain available during problem finalization (i.e., ICALL=3).

DEN_, **CDEN_(i)** (right-hand) (nmvi)

The nonparametric density and the marginal cumulative value for the ith. eta. Values are computed by NONMEM when the Nonparametric step is performed and marginal cumulatives are requested ($NONPARAMETRIC MARGINALS). Values are available during the pass with COMACT=2 and (if PASS is called) at ICALL=3. These variables may be used as right-hand quantities in abbreviated code blocks $PK, $ERROR, $INFN and $PRED.

PRED_, **RES_**, **WRES_** (right-hand) (nmvi)

These values may be used when ICALL is 3 ($PRED and $INFN) They have the same values as PRED, RES, WRES in NONMEM outputs. They change with calls to PASS. With NONMEM 7, all the special diagnostic items from Chapter III may be used in this manner.

CORRL2 (left-hand) (nm7)

CORRL2 is a reserved variable used for modelling correlation across residual variables (i.e., within L2 records.) May be set in $ERROR or $PRED, and used in $PK and $INFN. See the example in J.3. below.

MDVRES (left-hand) (nm73)

Set MDVRES to 1 in the $PRED or $ERROR block if you do not want to include a particular observation in the computation of residual and weighted residuals.

ETASXI (left-hand) (nm73)

ETASXI stands for eta shrinkage exclude/include.
Set ETASXI(i) to 2 to include ETA(i) in average eta shrinkage assessment.
Set ETASXI(i) to 1 to exclude ETA(i).
This variable may be used only in $PK and $ERROR and $PRED blocks. See also $ESTIMATION record ETASTYPE option.

NPDE_MODE (right-hand) (nm73)

NONMEM sets NPDE_MODE=1 when it is computing NPDE and NPD (Monte-Carlo generated normalized probability distribution error), when these are listed in $TABLE or $SCATTER. Otherwise, NPDE_MODE=0.

DV_LOQ (left-hand) (nm73)

"LOQ" stands for "limit of quantification". If the user's ERROR/PRED sets DV_LOQ when NONMEM sets NPDE_MODE=1, then the NPDE is being evaluated during this call, and this censored value is to be treated as if it is a non-censored datum with value of DV_LOQ.

IV.IV.E.6. PRED_IGNORE_DATA_TEST and PRED_IGNORE_DATA

Code such as the following may be used in $PRED, $PK, or $INFN:

```
$INFN
     IF(PRED_IGNORE_DATA_TEST==1) THEN
      PRED_IGNORE_DATA=0
      IF(AGE>35.0) PRED_IGNORE_DATA=1
      IF( ID>10.AND.ID<18.OR.ID>60.AND.ID<70 ) PRED_IGNORE_DATA=1
      RETURN ;Assures no additional computation code in INFN is executed
      ENDIF
```

In a user-written subroutine, the declarations are

```
       USE NMPRD_INT, ONLY: PRED_IGNORE_DATA,PRED_IGNORE_DATA_TEST
```

The IGNORE=(list) and ACCEPT=(list) options of $DATA provide a limited means of filtering the input data set, which is performed by NMTRAN. To provide more elaborate filtering for excluding data, PRED can request that NONMEM filter out additional data records at the begining of the run or problem. This is done by setting the reserved variable PRED_IGNORE_DATA to a non-zero value within $INFN, $PK, or $PRED, for each record to be ignored.

For details, see Guide VIII PRED_IGNORE_DATA BLOCK
See **INTRODUCTION TO NONMEM 7, PRED_IGNORE_DATA Feature (NM75)**

IV.IV.F. PRED-Defined Items

The values of a variable defined in a user-supplied PRED or in an abbreviated code for PRED is called a PRED-defined item. For the purpose of this definition, PREDPP is not considered to be either type of PRED specification. However, for documentation purposes, the values of a variable defined in a user-supplied routine used by PREDPP, or in an abbreviated code for such a routine is also called a PRED-defined item. If, for example, the routine is PK, such a value is also called a PK-defined item . PRED-defined items can be displayed in tables and scatterplots; see sections III.B.16-17.

The definition of a variable in an abbreviated code can generate additional definitions of other variables, called generated variables , appearing in the generated code but not appearing in the abbreviated code. The names of generated variables are all seven characters long. Certain generated variables symbolize the values of partial derivatives and are normally listed in NMPRD4 so that their values can be displayed like other PRED-defined items. For the PRED subroutine, the names of these variables are now described. For the subroutines of PREDPP, they are described in sections V.C.5,6,7,9.

Variables which symbolize (first-, second-, mixed-) partial η-derivatives of random variables defined in abbreviated code for PRED (first- and second-partial η-derivatives of random variables defined in abbreviated code for PK if PREDPP is used) are generated and displayable. They have names `A.....`, where the dots stand for various combinations of six digits `0-9`.

Variables which symbolize first-partial ε-derivatives of random variables defined in abbreviated code for PRED (ERROR if PREDPP is used) are displayable. They have names `C.....`, where the dots stand for various combinations of six digits `0-9`.

It is not possible to know what variable symbolizes a given partial derivative without first obtaining and inspecting the generated subroutine. Comment lines in the code describe the correspondence. The name for the variable can be used in a $TABLE or $SCATTERPLOT record of a subsequent run (provided the abbreviated code is the same in that run). With versions of NONMEM prior to NONMEM 7.4.1, the labels for variables which symbolize partial derivatives were converted to 4 characters, as described in earlier versions of the guide. The labels were not always unique or meaningful, although the values displayed were always correct. A work-around for earlier versions is to use an alias that is meaningful to the user. For example, with CONTROL4 and NONMEM 7.4.0, NM-TRAN generates a variable named A000039 for the "DERIVATIVE OF D W.R.T. ETA(001)" If this is to be displayed in a table and a

meaningful column header for the table is desired, an alias such as the following could be used:

```
$TABLE .... A000039=dDdETA1
```

IV.IV.G. PRED Error-Recovery

PRED may exit with a nonzero PRED error return code. This section describes how to implement such an exit with abbreviated code. It describes the ABORT and NOABORT options used in the $THETA and $ESTIMATION records.

IV.IV.G.1. Background

When PRED exits with a nonzero PRED error return code, either the NONMEM run is immediately aborted or an error-recovery procedure is implemented. An error-recovery procedure entails continued calls to PRED, but with values for the THETA's or ETA's different from those with previous calls which resulted in nonzero return codes. There are two error-recovery procedures: one with which different values for ETA's are tried, the ETA-recovery , a second with which different values for THETA's (and possibly ETA's) are tried, the THETA-recovery . Whenever it is possible to implement the ETA-recovery, this is done. If this procedure fails, or if it is not possible to implement the ETA-recovery, and the error return code is obtained during either the search in the Estimation Step or the search in the Initial Estimation Step, then a choice exists between an abort and implementation of the THETA-recovery. If the THETA-recovery fails, or if it is not actually possible to implement the THETA-recovery, the NONMEM run is aborted.

A PRED error return code can have values 0, 1, or 2. The value 0 means that the return is a normal return. The value 1 means that if the choice exists between an abort or implementation of the THETA-recovery, then this choice is to be made using control stream information. The value 2 means that if the choice exists between an abort or implementation of the THETA-recovery, then the abort should be chosen.

When an abort occurs, an error message will appear in the NONMEM output, in the intermediate output file (if such a file exists), and in the PRED Error file. When the THETA-recovery is implemented, an error message appears in the intermediate output file (if such a file exists), in the PRED Error file, and if recovery is not possible, in the NONMEM output. The error message is called a PRED error message .

When the PRED error return code is 1, and a choice exists between implementation of the THETA-recovery and an abort, the THETA-recovery is implemented if the NOABORT option is used on the $ESTIMATION ($THETA) record. If the NOABORT option is not used, then the run is aborted. Often, the most appropriate response to an abort occuring during the Initial Estimate Step, or during the Estimation Step after the 0th iteration summary has been output, is to rerun the problem requesting that the THETA-recovery procedure be implemented. *Warning*: If the NOABORT option is used before an actual abort has occured, be sure to check the PRED Error file for possibly useful diagnostic information that is otherwise available in NONMEM output when an abort occurs.

The NOABORTFIRST option of $THETA is the same as NOABORT, but also applies to the first value of the theta vector that is tried. With NONMEM 7 and the NOABORT option of $ESTIMATION, NONMEM will force most non-positive Hessian matrices to be positive definite, allowing NONMEM to continue. With NONMEM 72 and option NOHABORT of $ESTIMATION, positive definite corrections are made at all levels of the estimation. This can hide a serious ill-posed problem, so use with care.

IV.IV.G.2. Implementation

An quick exit from PRED with a nonzero return code can be implemented in abbreviated code for PRED (or in abbreviated code for PK, or ERROR, in which case the exit is from PREDPP) with this statement:

```
EXIT n k
```

where n is the return code (1 or 2) and k is a user code (1-999). If the user code is used, it can be any value the user wishes in the indicated range. In this case a part (shown here) of the PRED error message gives the user code:

PRED SUBROUTINE: USER ERROR CODE = k

The user code can be omitted. If it omitted, then the return code too can be omitted, and then the return code is 1 by default.

IV.IV.G.3. Rejecting Simulated Results and Simulation Error Forgiveness

The "EXIT n k" statement may be used for Rejecting Simulated Results and Simulation Error Forgiveness. If it is desired that the simulation be immediately terminated, then use an EXIT 2 code:

```
IF(ICALL==4.and.IPRED<0.1 .and. TIME>20.0) EXIT 2
```

With versions of NONMEM prior to 7.2, the "EXIT 1" statement in the Simulation step also caused NONMEM to abort. As of NONMEM 7.2, if an error occurs in PREDPP during simulation such as

```
PK PARAMETER FOR KA IS NON-POSITIVE
```

or a user-implemented EXIT 1 is issued during simulation, then PRED will be called with a new ETA and EPS. This feature is referred to as Simulation Error Forgiveness. NONMEM describes this as PRED SIMULATION REDO in the NONMEM report file. It writes to the NONMEM report file a description of the data record and THETA and ETA values, for example

```
 PRED SIMULATION REDO
 PRED EXIT CODE = 1
 INDIVIDUAL NO.       1   ID= 1.00000000000000E+00   (WITHIN-INDIVIDUAL) DATA REC NO.   1
 THETA=
  3.00E+00   8.00E-02   4.00E-02
 ETA=
  4.66E-01   2.91E-03   9.95E-01
 MESSAGE ISSUED FROM SIMULATION STEP
```

If ten such errors occur in the same subject, then it is supposed that the cause of the simulation error is not due to an occasional bad random sample, but is caused by a systematic error in the control stream file. The simulation step is terminated with the message

```
PRED ERROR OCCURRED TOO OFTEN ON SIMULATION
```

instead of a message

```
SIMULATION STEP PERFORMED
```

and (for example)

```
 SOURCE  1:
    SEED1:    1763452741   SEED2:              0
```

See **INTRODUCTION TO NONMEM 7, Simulation Error Forgiveness (NM72)**
See **INTRODUCTION TO NONMEM 7, Rejecting Simulated Results (NM75)**
With NONMEM 7.5, the REDO and "PRED ERROR OCCURRED TOO OFTEN" messages are written to PRDERR rather than the NONMEM report file, which says only

```
THERE ARE SIMULATION ERROR MESSSAGES IN PRDERR
```

As with earlier versions, the message

```
SIMULATION STEP PERFORMED
```

appears in the NONMEM report file only if the "PRED ERROR OCCURRED TOO OFTEN" condition did not happen.

With NONMEM 7.5, the PRED EXIT CODE k may be in the range 1000-9999. For example,
`IF(ICALL==4.and.IPRED<0.01 .and. TIME>20.0) EXIT 1 2300`
This can only occur with user's EXIT code; PREDPP will not generate this kind of EXIT. NONMEM will try PRED SIMULATION REDO up to 10000 times. The message "PRED SIMULATION REDO" itself is written to PRDERR up to 30 times. After that, the following message is written to PRDERR:

```
 SUBSEQUENT PRED SIMULATION REDO ERROR MESSSAGES SUPPRESSED
```

NONMEM continues trying new ETA and EPS. Be careful that the condition does not occur too often (causing wasteful computation). After 10000 tries, the simulation is terminated as a protection against an infinite loop. The following message is written to PRDERR:

```
 TOO MANY CONSECUTIVE PRED ERRORS (>10000) OCCURRED ON SIMULATION
```

IV.IV.H. Pseudo-Statements

A pseudo-statement is a statement of abbreviated code of the form of an (unconditional) assignment statement, i.e. A=B, where A is a specific reserved variable, and B is an integer constant. The variable A symbolizes a type of left-hand quantity, but, unlike other such quantities, it cannot be used as a right-hand quantity. The variable characterizes the type of the pseudo-statement. There are different types of pseudo-statements specific to each type of abbreviated code. There is usually a restriction on the permitted values for the integer B. Each type of pseudo-statement can appear only once in an abbreviated code for a particular routine. It must appear before other statements of abbreviated code ocurring in that routine.

For an abbreviated code for PRED, there is only one allowable type of pseudo-statement. The variable is `COMRES`, and the only permitted value for the integer constant is -1. So, the statement must look like

```
COMRES=-1
```

If this statement appears, then no variables defined in the abbreviated code are listed in NMPRD4. In this regard, see also the discussion of the option `COMRES=1` in section III.B.7. With an abbreviated code for PRED (where no additional abbreviated code for another routine can also be used), the effect of the above pseudo-statement is identical to use of that option.

IV.IV.I.1. Verbatim Code

As mentioned in section IV.A, verbatim code should be used only by those who understand well how generated codes are structured. Verbatim code is FORTRAN code which may be inserted into abbreviated code. This code is in turn inserted unchanged, i.e. verbatim, into the generated subroutine. NM-TRAN does not generate code for the computation of η- and ε-derivatives based on verbatim code, so particular care in using verbatim code is needed when variables which are interpreted as random variables in abbreviated code are used in verbatim code.† NM-TRAN does not check whether verbatim code uses correct FORTRAN syntax; this check will be made by the FORTRAN compiler. A portion of abbreviated code which includes verbatim code might look like:

```
IF (ICALL.EQ.4.AND.NEWIND.NE.2) THEN
" 5 IF (ABS(ETA(2)).GT.5) THEN
"       CALL SIMETA (ETA)
"       GO TO 5
"   ENDIF
ENDIF
```

Neither the use of the absolute value function, a subroutine call, the use of a statement number, nor a GO

†Do not use verbatim code to circumvent the occurence of NM-TRAN error messages concerned with the use of random variables in abbreviated code. If so used, computations will most likely be incorrect.

TO are FORTRAN constructs which were originally allowed in abbreviated code. Starting with NONMEM VI, abbreviated code may use the ABS function, "CALL SIMETA", "CALL SIMEPS", "CALL RANDOM", and the DOWHILE/ENDDO statements for loops. All the verbatim examples in this section of the guide can be implemented with abbreviated code. See section IV.J below. However, there are still times that verbatim code is necessary, and the following discussion is relevant to any verbatim code.

Essentially, any line whose first nonblank character is a double quote is recognized as containing verbatim code. The double quote is dropped, and the remaining characters of the line are copied to the generated code. For the user's convenience, verbatim code following a statement number is adjusted so that it conforms to FORTRAN 77 conventions regarding where FORTRAN code is placed in a line; see more discussion of this below. This means that in the example the text after the 5 is moved so that it begins in position 7 of the generated line; see the generated code below. There is a leading blank before the 5. As with all FORTRAN statements with statement numbers, a statement number can be placed anywhere in positions 1-5. A single line of verbatim code, other than a FORTRAN comment statement (see below), may be copied to more than one line of generated code if NM-TRAN determines that the characters of the generated line would otherwise extend beyond position 100; FORTRAN continuation lines are created as necessary to limit each generated line to 100 characters. Very long strings (e.g., long variable names and constants) are copied to lines of generated code of at most 160 characters, with FORTRAN continuation lines as needed. NM-TRAN does not check the number of characters in a variable name or constant. Different compilers have different limits, and errors may occur when generated FSUBS is compiled if these limits are exceeded.

In the above example, by default the verbatim code is placed in the generated subroutine after some initial executable code required for routine initialization purposes; see below. This initial code contains a call to SIMETA to obtain simulated values of the `ETA`'s. The effect of the verbatim code is to replace the value of `ETA(2)` with a value less than 5 in absolute value if necessary. For this code to have the desired effect, the option `NEW` must be used in the $SIMULATION record.

In actuality, NM-TRAN does modify verbatim code. It has already been seen that a placement adjustment takes place. However, perhaps more surprisingly, certain variables occuring in verbatim code are replaced by certain array elements. To see how this happens, recall that a variable that is a label for data items of a particular type can be used freely in abbreviated code. The mechanism by which this is allowed does not by itself generally imply that this same variable can be used freely in verbatim code. For example, the abbreviated code

```
A=THETA(2)*WT*EXP(ETA(2))
```

can generate the FORTRAN code

```
      WT=DATREC(005)
      B00001=DEXP(ETA(002))
      A=THETA(002)*WT*B00001
```

which defines `WT` and then uses this variable. The definition is given in terms of a reference to the `DATREC` array where the weight data item is to be found. This array is defined as an argument to the generated routine. The analogous verbatim code

```
" A=THETA(2)*WT*EXP(ETA(2))
```

copied without change into generated code, contains an undefined variable (`WT`) if the variable `WT` is not used in abbreviated code. To avoid this difficulty, any instance of a variable that is a label for data items and that is used in verbatim code is replaced with a reference to the DATREC array (EVTREC array if

PREDD is used). Therefore, the above verbatim code is actually translated into

```
A=THETA(2)*DATREC(005)*EXP(ETA(2))
```

This rule is called the replacement rule . It applies as well to instances of the variable occuring on the left of an assignment statement. This enables data items to be transgenerated during a Simulation Step (e.g. see section III.B.13, and Guide VI, Figure 2 along with the accompanying discussion in section III.L.1). The rule applies to essentially *all* instances wherever they appear. Because of the replacement rule, a more accurate name for verbatim code might be quasi-verbatim code.

Implementation of the replacement rule is most often helpful to the user. However, if truly verbatim code, i.e. nonmodified code, is desired, there is a way to obtain it. This involves using the character '@' as an "escape" character. If the escape character is used immediately before an instance of the variable, the variable is not replaced. For example, if the verbatim code is

```
" A=THETA(2)*@WT*EXP(ETA(2))
```

then the generated code is

```
A=THETA(2)*WT*EXP(ETA(2))
```

This same generated code can be obtained with the verbatim code

```
"@ A=THETA(2)*WT*EXP(ETA(2))
```

If the escape character is used immediately after the double quote, the entire line is copied without change, except that the double quote and escape characters are deleted.

NM-TRAN comment lines are not copied to the generated code. A FORTRAN comment statement can be inserted into the generated routine. Here is an example:

```
"C this line is used at debug time
```

If the character immediately following the double quote is either 'C', 'c', '*', '"', or '!', the line is recognized to be a comment statement. C or * in column 1 was a FORTRAN 77 convention and is still permitted by NM-TRAN for upwards compatiblity of old control streams. The use of '"' was special to NONMEM, and is also permitted. However, ! is the FORTRAN 95 convention, and should be used in new control streams. The characters following the double quote are copied starting at position 1 of into the generated line, The character at position 1 is always set to be !.

Notice that lower case can be used in verbatim code. However, lower case is converted to upper case before the replacement rule is applied.

As mentioned above, and as seen from some examples, verbatim code is adjusted so that it conforms to FORTRAN 77 conventions regarding where FORTRAN code is placed in a line. Alphabetic text that "starts" a line of verbatim code or that follows a statement number is adjusted so that it begins in position 7 of the line of generated code, unless the line is recognized to be a FORTRAN comment line.†

†NM-TRAN copies any nonalphanumeric character occuring in verbatim code (other than '@') into generated code. With UNIX systems, one can take advantage of this and use tab characters if the compiler permits such characters. A tab character is a nonblank, nonalphanumeric, special character other than '@' or '!'. It can be an actual tab character. Then alphabetic text that "starts" a line of verbatim code or that follows (i) a statement number or (ii) a tab character or (iii) a statement number followed by a tab character, is adjusted so that it begins in position 7 of the line of generated code.

FORTRAN continuation lines can be expressed with verbatim code using Fortran 95 syntax. The character & should be placed at the end of the line to be continued. The presence of a character in position 6 is no longer of significance, so the first example of verbatim code can be spaced differently without causing a problem.

It is not obvious where exactly verbatim code is placed in the generated code since the user is not excercising strict control over the latter. However, some control over where verbatim code is placed is available. A generated subroutine has four sections.

First section: nonexecutable declaration statements

Second section: executable code required for initialization purposes

Third section: code implementing abbreviated code

Fourth section: code that stores subroutine outputs

RETURN statement.

Verbatim code can be specifically placed immediately after the first section, throughout the third section, or immediately after the fourth section (prior to the `RETURN` statement). Verbatim header statements are used for this purpose.

Verbatim code that is to go immediately after the first section must precede all abbreviated code and must be immediately preceded by the header statement

```
"FIRST
```

Spaces before or after the word `FIRST` are permitted. No abbreviated code may precede `"FIRST` except for pseudo-statements. NM-TRAN comment lines may precede `"FIRST`. This block of verbatim code is called the FIRST block. FORTRAN requires that all declarations precede all executable statements, so declarations must be placed in this block.

Verbatim code that is to be placed throughout the third section can be preceded by the header statement

```
"MAIN
```

Consider this variation of the first example of verbatim code from above.

```
"MAIN
IF (ICALL.EQ.4.AND.NEWIND.NE.2) THEN
" 5 IF (ABS(ETA(2)).GT.5) THEN
"       CALL SIMETA (ETA)
"       GO TO 5
"   ENDIF
ENDIF
```

Spaces before or after the word `MAIN` are permitted. No abbreviated code may precede `"MAIN` except for pseudo-statements. NM-TRAN comment lines may precede `"MAIN`. The lines of verbatim code that go into the third section need not be contiguously placed within the abbreviated code as they are in this example. Verbatim code may be intermingled with abbreviated code. A line of verbatim code that follows a line L of abbreviated code is copied to the generated code following all generated code implementing L. There may be abbreviated code preceding the first verbatim code that goes into the third section, as in this example. Then `"MAIN` may be omitted; by default this first verbatim code goes into the third section.

Suppose the first verbatim code that is to go into the third section precedes all lines of abbreviated code. If there is a FIRST block, then "MAIN is necessary. In this case "MAIN acts a delimiter, ending the FIRST block. Without it, this verbatim code would appear to belong to the FIRST block.

Consider this extension of the previous example

```
"FIRST
" DOUBLE PRECISION R
"MAIN
IF (ICALL.EQ.4.AND.NEWIND.NE.2) THEN
" 5 IF (ABS(ETA(2)).GT.5) THEN
"      CALL SIMETA (ETA)
"      GO TO 5
"   ENDIF
"   ETAS1=ETA(1)
"   ETAS2=ETA(2)
"   CALL RANDOM (2,R)
"   WT=70+7*R
"   @WT=WT
"   WGHT=WT
ENDIF
"   IF (ICALL.EQ.4) @WT=WGHT
```

Prior to NONMEM 7, the number returned in R by the NONMEM utility routine RANDOM was always a single-precision number (see section III.B.13), and since the FORTRAN declaration statement IMPLICIT DOUBLE PRECISION appears in a generated code the declaration REAL R had to be added. Since declarations must precede executable code, the declaration is included in the FIRST block. With NONMEM 7, R need not be declared. It is declared explicitly to be DOUBLE PRECISION for the sake for the example. As with the previous example, the statement "MAIN may be omitted. However, with this variation

```
"FIRST
" DOUBLE PRECISION R
"MAIN
" IF (ICALL.EQ.4.AND.NEWIND.NE.2) THEN
" 5 IF (ABS(ETA(2)).GT.5) THEN
"      CALL SIMETA (ETA)
"      GO TO 5
"   ENDIF
"   ETAS1=ETA(1)
"   ETAS2=ETA(2)
"   CALL RANDOM (2,R)
"   WT=70+7*R
"   @WT=WT
"   WGHT=WT
" ENDIF
"   IF (ICALL.EQ.4) @WT=WGHT
```

"MAIN serves as a necessary delimiter of the FIRST block. The verbatim code just shown is discussed in greater detail at the end of this section.

Verbatim code that is to go immediately after the fourth section should be immediately preceded by the header statement

```
"LAST
```

Spaces before or after the word LAST are permitted. If this statement is used, it, and the *contiguous lines of* verbatim code that follow it, must be placed at the very end of the abbreviated code (one can ignore the presence of NM-TRAN comment statements). These lines of verbatim code, the LAST block, are placed altogether, immediately before the RETURN statement of the generated routine.

Generally, a line of verbatim code that is not in a FIRST block, and not in a LAST block, goes into the third section of generated code by default, whether or not the statement "MAIN actually appears.

For certain subroutines (e.g. DES), the ability to use certain verbatim header statements does not exist, or if a header statement can be used, it has no effect. Descriptions of these particular cases are included with the descriptions of each of the different abbreviated codes.

The presence of verbatim code anywhere in an abbreviated code has two useful side effects.

> The generated code then includes needed declarations for all reserved variables that can be used in the abbreviated code. (When verbatim code is not present, only declarations for those reserved variables used in the abbreviated code are included.) Hence any reserved variable may be used in verbatim code without the need to also include declarations for it in verbatim code.
>
> Lines of abbreviated code that follow some verbatim code may then include variables that are not defined in the abbreviated code preceding the verbatim code. Normally, the presence of an undefined variable is considered to be invalid, and this raises an error. When verbatim code is present, this restriction is relaxed since the variable may have been defined in the verbatim code.

A variable defined by verbatim code is implicitly a double precision variable. (The actual declaration is

```
IMPLICIT REAL(KIND=DPSIZE) (A-Z)
```

This implicit declaration can be overridden with an explicit type declaration in the FIRST block.

In the remainder of this section some verbatim code shown above is examined in greater detail.

```
"FIRST
" DOUBLE PRECISION R
"MAIN
" IF (ICALL.EQ.4.AND.NEWIND.NE.2) THEN
" 5 IF (ABS(ETA(2)).GT.5) THEN
"       CALL SIMETA (ETA)
"       GO TO 5
"   ENDIF
"   ETAS1=ETA(1)
"   ETAS2=ETA(2)
"   CALL RANDOM (2,R)
"   WT=70+7*R
"   @WT=WT
"   WGHT=WT
" ENDIF
"   IF (ICALL.EQ.4) @WT=WGHT
```

In this example ETAS1 and ETAS2 are the labels for data items, and by the replacement rule, these items become the simulated values of η_1 and η_2. (During the Simulation Step, transgeneration of data items is allowed. Transgenerated items are stored in NONMEM's internal copy of the data set.) Weights are also simulated (normally distributed values with mean 70Kg and standard deviation 7Kg if the option NORMAL is used in the definition of the second random source). WT also is a data item label, and so weights are stored as transgenerated data items. Since weight is used in subsequent computations in abbreviated code, the variable WT is defined in the second section of generated code, *but as being equal to the weight in the data record as that record appears when the second section of code is executed*. Since the subsequent computations should involve transgenerated weight, WT is redefined after the transgeneration, *using the escape character*. Notice that weight is simulated only with the first data record of the individual record, and that it is stored as a transgenerated item only with this first data record. Its value, though, may be needed in computations during the Simulation Step with each of the data records of the individual record. WGHT is a PRED-defined item, not a data item label. Its value in all these data records remains unchanged from the value to which it is set with the first data record. So, WGHT is given the simulated value, and with each data record, WT is redefined to be this value, again using the escape character. Were WGHT not used, the value of WT with any data record other than the first record would be the data item in that record.

This strategy works if only simulation is implemented (indeed, if the ONLYSIMULATION option is used in the $SIMULATION record). However, if data analysis is also implemented, then unless further steps are taken, a problem arises when the routine is called with a data record other than the first data record of an individual record. This is because the simulated weight value is only stored in the first data record. Here is another variation of the same verbatim code, achieving the identical effect, but also addressing this problem.

```
"FIRST
" DOUBLE PRECISION R
" IF (ICALL.EQ.4.AND.NEWIND.NE.2) THEN
"    CALL RANDOM (2,R)
"    WT=70+7*R
"    WGHT=WT
" ENDIF
" IF (ICALL.EQ.4) WT=WGHT
"MAIN
" IF (ICALL.EQ.4.AND.NEWIND.NE.2) THEN
" 5 IF (ABS(ETA(2)).GT.5) THEN
"      CALL SIMETA (ETA)
"      GO TO 5
"   ENDIF
"   ETAS1=ETA(1)
"   ETAS2=ETA(2)
" ENDIF
```

Here weight is simulated immediately after the declarations, before the second section, and because of the replacement rule, it is stored in the data record. Therefore, when WT is redefined in the second section, it is given this value. This is true with the first data record of the individual record, and by virtue of the use of WGHT, it is also true with subsequent data records. However, in this variation the escape character does not need to precede WT with the statement WT=WGHT, since it is not the variable itself that needs to be redefined; that happens subsequently in the second section. Rather, the data item needs to be transgenerated, so that when the variable is redefined, the variable is defined to be the transgenerated value. As a result, with this variation, the transgeneration takes place with all data records, whereas with

the previous variation the transgeneration only takes place with the first data record of the individual record.

Here is yet a simpler variation.

```
"FIRST
" DOUBLE PRECISION R
" IF (ICALL.EQ.4.AND.NEWIND.NE.2) THEN
"    CALL RANDOM (2,R)
" ENDIF
" IF (ICALL.EQ.4) WT=70+7*R
"MAIN
" IF (ICALL.EQ.4.AND.NEWIND.NE.2) THEN
" 5 IF (ABS(ETA(2)).GT.5) THEN
"      CALL SIMETA (ETA)
"      GO TO 5
"   ENDIF
"   ETAS1=ETA(1)
"   ETAS2=ETA(2)
" ENDIF
```

IV.IV.I.2. Verbatim Code with NONMEM 7

(1) With NONMEM 7, the use of executable statements in the "FIRST section of verbatim code is deprecated. Chapter II section II.II.D describes how certain reserved variables (such as elements of the COM array) are declared as pointer variables, and then assigned to targets by executable statements. The FIRST block is inserted between the declarations and the executable statements, and should only have declarations. As an example, suppose the $PRED code is:

```
$PRED
"FIRST
"! first code here
" COM(1)=1
   Y=THETA(1)+ETA(1)
```

The generated FSUBS contains:

```
 ...
      REAL(KIND=DPSIZE), POINTER :: DEN_,CDEN_(:)
! first code here
      COM(1)=1
 ...
     IF (ICALL <= 1) CALL ASSOCNMPRD4
```

Subroutine ASSOCNMPRD4 contains

```
COM=>VRBL
```

There is a segmentation violation with NONMEM 7, because COM(1) is used before it has been assigned to a target by subroutine ASSOCNMPRD4. This will happen in all blocks of abbreviated code, not just $PRED.

All the examples of verbatim code in the above section work correctly with NONMEM 7.4, but future changes to NONMEM could cause problems.

A better version of the last ("simpler") example follows, in which there is no executable code in the FIRST block.

```
"FIRST
" DOUBLE PRECISION R
"MAIN
" IF (ICALL.EQ.4.AND.NEWIND.NE.2) THEN
"    CALL RANDOM (2,R)
" ENDIF
" IF (ICALL.EQ.4) WT=70+7*R
" @WT=WT
" IF (ICALL.EQ.4.AND.NEWIND.NE.2) THEN
" 5 IF (ABS(ETA(2)).GT.5) THEN
"      CALL SIMETA (ETA)
"      GO TO 5
"   ENDIF
"   ETAS1=ETA(1)
"   ETAS2=ETA(2)
" ENDIF
```

Because of the replacement rule, only WT in the data file is simulated. The local variable WT was assigned before the MAIN block, and must be reassigned using

```
@WT=WT
```

(2) All the verbatim examples can be implemented using abbreviated code. For example,

```
 IF (ICALL.EQ.4.AND.NEWIND.NE.2) THEN
    CALL RANDOM (2,R)
 ENDIF
 IF (ICALL.EQ.4) WT=70+7*R

 IF (ICALL.EQ.4.AND.NEWIND.NE.2) THEN
 DOWHILE(ABS(ETA(2)).GT.5)
 CALL SIMETA (ETA)
 ENDDO
 ETAS1=ETA(1)
 ETAS2=ETA(2)
 ENDIF
```

Each call to SIMETA replaces all the etas, so code similar the following is needed if bounds are put on two different etas:

```
 IF (ICALL.EQ.4.AND.NEWIND.NE.2) THEN
 DOWHILE(ABS(ETA(2)).GT.5.OR.ABS(ETA(1)).GT.0.52)
 CALL SIMETA (ETA)
 ENDDO
 ETAS1=ETA(1)
```

```
 ETAS2=ETA(2)
 ENDIF
```

IV.IV.J. Advanced Coding Techniques

This section desribes some advanced techinques that can be used in abbreviated code.

IV.IV.J.1. Indicator Variables, Random Variables and Recursion code

In abbreviated code, an indicator variable is a variable whose value is 0 or 1. It may be identified with an input data item, or it may be a variable defined in the abbreviated code. Indicator variables are used to make a choice in a computation.

Random variables may be assigned using conditional statements. Suppose, for example, ICU is a variable which is either 0 or 1. The following code can be used:

```
(a)
 IF (ICU.EQ.0) THEN
 CLM=TVCLM+ETA(1)
 ELSE
 CLM=TVCLM+ETA(2)
 ENDIF
```

This can be coded unconditionally using ICU as an indicator variable:

```
(b)
CLM=TVCLM+(1-ICU)*ETA(1)+ICU*ETA(2)
```

NM-TRAN implements (a) using PRED-defined indicator variables `Q00...` so that the assignment of CLM is in fact unconditional.

Indicator variables may be used explicitly to avoid the restriction that a random variable may not be defined in a nested IF. For example, example10.ctl (bayes10.exa) contains this code for simultaneous analysis of PK and categorical data, where TYPE is listed in $INPUT:

```
IF (TYPE.EQ.0) THEN
 ; PK Data
     F_FLAG=0
     Y=F+F*ERR(1) ; a prediction
  ELSE
 ; Categorical data
     F_FLAG=1
     A=DEXP(EXPP)
     B=1+A
     Y=DV*A/B+(1-DV)/B      ; a likelihood
  ENDIF
ENDIF
```

The value of DV is 0 or 1 and is used as an indicator variable to avoid assigning Y in a nested IF, which is not permitted:

```
IF (TYPE.EQ.0) THEN
  ...
ELSE
  ...
```

```
IF (DV==1) Y=A/B
IF (DV==0) Y=1/B
ENDIF
```

Random variables may be redefined, and may appear on the right-hand side of conditional assignment statements. They may be used recursively in their own redefinition. If a random variable is defined by "incomplete" conditionals (i.e., conditionals which do not include both true and false cases), and all tests fail, the random variable has the value zero. This is a major difference between random and non-random variables. Consider these examples:

```
IF (WT.GT.0) K=WT*THETA(1)*EXP(ETA(1))
IF (WT.GT.0) TVV=WT*THETA(1)
```

If WT is <= 0, K is zero. NM-TRAN prints a warning message when it detects such code.

If WT is <= 0, TVV retains its prior value, which may have been computed with a previous event record. When a COM(i) variable is defined by incomplete conditional statements and all tests fail, the variable retains its value (as do all non-random variables).

In $PK, $ERROR, and $PRED records, recursion code may be used in an explicit manner, as in this example:

```
IF (WT.GT.0) THEN
  K=THETA(1)*WT*EXP(ETA(1))
ELSE
  K=K
ENDIF
```

If the condition is false, K retains its value set with the previous data record.

Recursion code can be used in $PRED, $PK, and $ERROR records for other purposes as well, e.g., to implement recursive kinetics in $PRED, and to compute the sum of a random variable in a DOWHILE block. The following two fragments of code illustrate how one can use abbreviated code to implement recursive kinetics in $PRED. The first example works with a single bolus dose and the second example works with single or multiple bolus doses. Similar code can be used in $PK and $ERROR.

```
K=THETA(1)*EXP(ETA(1))
IF (TIME.EQ.0) THEN
  OLDA=AMT
  T=TIME
ENDIF
A=OLDA*EXP(-K*(TIME-T))
OLDA=A
T=TIME

K=THETA(1)*EXP(ETA(1))
IF (TIME.EQ.0) THEN
  A=AMT
  T=TIME
ELSE
  A=A*EXP(-K*(TIME-T))+AMT
```

```
   ENDIF
   T=TIME
```

The above forms of recursion work for recursion from one data record to the next ("inter-record" recursion). It is also possible to use recursion in a do-while loop ("intra-record", or "do-while" recursion).

Example of a do-while recursive loop using a random variable:

```
 TERM=THETA(1)*EXP(ETA(1))
 SUM=0
 DO WHILE(condition)
 SUM=SUM+TERM
 ...
 ENDDO
```

A product loop such as

```
  PROD=PROD*TERM
```

is also possible, as are other ways the dowhile recursive variable can be used, so long as the variable appears on both sides of the equal sign within the DOWHILE loop: V= ... V ...

IV.IV.J.2. Use of NONMEM's PASS Utility

The NONMEM utility routine PASS can be used to read and/or to modify ("transgenerate") data records. This can be done at ICALL 0 (run initialization), 1 (problem initialization) and/or 3 (problem and run finalization). Repeated calls ("CALL PASS") are used to pass through the data set. Data record items are referred to by the name they are given on the $INPUT record. NONMEM data items ID and MDV may not be transgenerated. Any other data item (including DV) may be transgenerated. Additional data items can be generated at both the beginning and ending of a problem. Since the finalization call actually occurs before the Table and Scatterplot Steps, new data items generated at this call can be tabled and scatterplotted. This is described in NONMEM Users's Guide Part II, D.2.2. See also Guide VI, Chapter VI. See Section J.3, below, for abbreviated code.

Each call to PASS obtains the next data record, so the values of data items change with each call to PASS. Only those PRED-defined items that are displayed in tables or scatters will have values appropriate to the current record; a PRED-defined item that is listed in NMPRD4 but is *not* displayed will always have the value from the first data record. NEWIND and ETA values change to those appropriate to the current record. Some of the reserved right-hand variables described above also change value, as noted.

IV.IV.J.3. The DOWHILE Statement

The DOWHILE statement may be used for loops in abbreviated code.
An example is given above that uses DOWHILE during simulation. During data anaysis (with ICALL=2), code such as this can be used:

```
$ABBR DECLARE DOWHILE ILOOP
$PRED
 ILOOP=1
 DOWHILE (condition)
  .. statements ..
 ILOOP=ILOOP+1
 ENDDO
```

A random variable may be computed recursively in such a loop.

Transgeneration examples

Section D.2 describes the use of NONMEM's PASS Utility to modify (transgenerate) the NONMEM data set at initialization / finalization calls to PRED. Here is an example of abbreviated code that can be used with NONMEM V:

```
IF (ICALL.EQ.0) THEN
  MODE=0
  CALL PASS (MODE)
  MODE=2
  CALL PASS (MODE)
  DO WHILE (MODE.EQ.2)
  ... transgeneration statements ...
  CALL PASS (MODE)
  ENDDO
ENDIF
```

With NONMEM VI, the DOWHILE (DATA) abbreviated code statement facilitates this feature by supplying all the needed code to pass through the data set.

```
IF (ICALL==0) THEN
  DOWHILE (DATA)
  ... transgeneration statements ...
  ENDDO
ENDIF
```

Here is fragment of code for modelling auto-correlation. NO is a reserved variable giving the maximum number of observations per individual record; it can be used only in the context of the $ABBREVIATED DECLARE record. CORRL2 is a reserved variable for modelling correlation across residual variables.

```
$ABBR DECLARE T(NO)
$ABBR DECLARE DOWHILE J
$ABBR DECLARE INTEGER I
 ...
$PRED (or $ERROR)
 IF(NEWIND.NE.2) I=0
 IF(MDV.EQ.0)THEN
    I=I+1
    T(I)=TIME
    J=1
    DO WHILE (J<=I)
    CORRL2(J,1)=EXP(-THETA(4)*(TIME-T(J)))
    J=J+1
    ENDDO
 ENDIF
```

IV.IV.J.4. MU Modelling (nm7)

Here is an example of MU modelling:

```
MU_2=THETA(4)
CL=MU_2+ETA(2)
```

Variables MU_i are reserved. The new NONMEM 7 EM (Expectation Maximization) methods and Gibbs sampling methods are most efficiently implemented if the user supplies information on how the

THETA parameters are associated arithmetically with the ETAs and individual parameters, wherever such a relationship holds. Calling the individual parameters phi, the relationship should be
phi_i=mu_i(theta)+eta(i)
for each parameter i that has an eta associated with it, and mu_i is a function of THETA. The association of one or more THETA's with ETA(1) must be identified by a variable called MU_1. Similarly, the association with ETA(2) is MU_2, that of ETA(5) is MU_5, etcetera. This is called MU Referencing or MU Modelling.

MU_i should be assigned unconditionally, using indicator variables if necessary. E.g., suppose GENDR is 0 for males, 1 for females, and LCLM=log transformed clearance, male, etc.

A model for the mean of phi_1 could be:

```
MU_1=(1.0-GNDR)*(LCLM+LAGE*CLAM) + GNDR*(LCLF+LAGE*CLAF)
```

(See Bayes Example 2.)

Another example is with a mixture model, where Q=1 for MIXNUM=1 and Q=0 for MIXNUM=2 (See Bayes Example 5).

```
MU_2=Q*THETA(2)+(1.0-Q)*THETA(3)
```

Option `CHECKMU` of the $ABBREVIATED requests that MU model statements be checked, and is the default. Option `NOCHECKMU` can be used to prevent NM-TRAN from attempting to check the MU model statements.

IV.IV.J.5. INCLUDE statement (nmv)

Identical to the $INCLUDE record (Chapter III B.20). May be used in a block of abbreviated code to read records from a different file.

An include file nonmem_reserved_general may be found in the util directory of the NONMEM installation (nm73). It contains declarations for variables in MODULEs that can be used on the right or left in abbreviated code. It also has definitions of functions that may be useful. See also useful_variables.pdf in the guides directory for a listing of such variables. This file is used in examples/example8.ctl, in which NONMEM variables BAYES_EXTRA and ITER_REPORT are used. Some of the other variables defined in nonmem_reserved_general include:

MDVI1, MDVI2, MDVI3 (modifies NONMEM's treatment of "non-impact" records with MDV>=100).
PI

IV.IV.J.6. PROTECT functions (nm74)

With NONMEM 7.4, a versions of the functions are available that protect against domain violations, divide by zero, and floating point overflows. The protect functions are as follows.

For all routines, if X=not a number, X is converted to machine precision value, which is about 1.0E-15, before performing an operation on it. If X>INFNTY (where INFNTY is approximately 1.0E+154), then X is converted to INFNTY before an operation is performed on it.

PLOG(x)
: Returns LOG of x. If x<SMALLZ, where SMALLZ is approximately 2.8E-103, then LOG(SMALLZ) is returned.

PLOG10(x)
: Returns LOG10 of x. If x<SMALLZ, where SMALLZ is approximately 2.8E-103, then

LOG10(SMALLZ) is returned.

PSQRT(x)
: Returns SQRT of x. If x=0.0d+00, then 0 is returned.

PEXP(x)
: Returns EXP of x. If x>40.0, then PEXP(100.0) is returned (avoids overflow).

PDZ(x)
: Returns 1/x . Protects against divide by zero. If abs(x)<SMALLZ, then 1/SMALLZ is returned.

PZR(x)
: Returns x . protects against zero. If abs(x)<SMALLZ, then SMALLZ is returned.

PNP(x)
: Returns x. Protects against non-positive. If X<SMALLZ, then SMALLZ is returned.

PHE(x)
: Returns x. Protects against high exponent. If X>100, then 100 is returned. Thus PEXP(x)=EXP(PHE(x)).

PNG(x)
: Returns x. Protects against negative. If X<0.0, then 0.0 is returned.

PTAN(x)
: Returns tan(x). Protects against returning infinity on inputs near pi/2.

PATAN(x)
: Returns atan(x). Protects against large intputs.

PACOS, PASIN
: Returns acos(x), asin(x), respectively. If |X| is between 1.0 and 1+10**(-08), then x is submitted as 1 or -1. So, "dirty ones" are cleaned up, but values clearly beyond 1 are allowed to trip up the function, so the user is aware of the logical error in the code, and fix the issue.

In addition, there are first derivative (such as PLOGD1), and second derivative (such as PLOGD2) companion routines available which NONMEM uses for analytical derivatives.

If the record

```
$ABBREVIATED PROTECT
```

is present, NM-TRAN will automatically replace all LOG (or DLOG) with PLOG, EXP (or DEXP) with PEXP, SQRT (or DSQRT) with PSQRT, / operations with *PDZ(), and B**E operations with PEXP(E*PLOG(B)). When you use $ABBR PROTECT, you will find a considerable improvement in estimation stability, regardless of estimation method used. Alternately, P versions can be coded explicitly in abbreviated code.

The source code of these routines are available in ..\source\PROTECT.f90. If you wish to modify their behavior, then copy PROTECT.f90 to your run directory, rname and modify it, such as PROTECTB.f90, then refer to this modified code with

```
$SUBROUTINES OTHER=PROTECTB.f90
```

IV.IV.J.7. $ABBR FUNCTION and $ABBR VECTOR (nm74)

With NONMEM 7.4, A user-defined function may be declared as follows:

```
$ABBR FUNCTION function_name(input_vector_name,dimension,usage)
```

function_name
: is the name of the function. As with reserved functions FUNCA etc., the code for the function must be written by the user in FORTRAN.

(See Guide VIII ABBREVIATED FUNCTION Help Item.)
The code must be in a file included using the $SUBROUTINE OTHER option, as discussed in Chapter III.B.6.

input_vector_name
is the name of an input vector that may be used to pass arguments to the function.

dimension
specifies how many input arguments function_name will use, and defines input_vector_name as a vector with this length. "Dimension" is a property of both the function and of the input vector.

usage
is the maximum number of times the function may appear in the abbreviated code, that is, the maximum number of occurances of function_name. It is not an error if there are fewer occurances. If usage is omitted, NMTRAN will supply the exact number. If usage is coded, NMTRAN will generate an error message if function_name appears in abbreviated code more than "usage" number of times.

Note that a given function may be used with other input vectors, and a given vector may be used with other functions.

A user-defined vector also may be declared as follows:

```
$ABBR VECTOR input_vector_name (dimension)
```

input_vector_name
is the name of an input vector.

dimension
Specifies the length of the vector. The dimension of a vector should be no less than the dimension of all the functions which which it is used.

A vector and its length may be declared independently of a function, and vice-versa. The asterisk may be used as a place holder, e.g.,

```
$ABBR FUNCTION BIVARIATE(*,5) ; when BIVARIATE is called, NDIM will be 5
```

Here is an example.

```
$ABBR FUNCTION BIVARIATE(VBI,5,3)
```

A vector VBI is defined of length 5. There is a function called BIVARIATE. When BIVARIATE is used, the value 5 is passed to it as argument NDIM. BIVARIATE uses 5 elements from the input vector. Function BIVARIATE may appear in abbreviated code at most 3 times.

IV.IV.L. $MIX Record (nmv)

`$MIX`
the abbreviated code

Optional. This record gives an abbreviated code for the MIX subroutine. If it appears, it must be with the first problem specification, and only with this problem specification. The use of $MIX is independent of the choice of $PRED vs. PREDPP. When MIX is called, the first data record of the individual record is stored in TEMPLT. General rules for abbreviated code are described above. Specific rules for $MIX are described in NONMEM Users Guide VIII and on-line help. Reserved variables MIXP, MIXNUM, MIXEST are described above.

Required left-hand variables are NSPOP, the number of sub-populations, and P(i) where for each i (i=1, ... , NSPOP), P(i) is the modeled fraction of the population in the ith subpopulation.

Elements of the data record can be accessed; see $CONTR and the DATA array. Variables defined in $MIX are not listed in MODULE NMPRD4 and may not be displayed in $TABLE and $SCATTER. Variables from PRINFN (from PREDPP $INFN block) and DEFINEDVARIABLES (from $ABBR DECLARE control record) are not defined in $MIX abbreviated code.

IV.IV.M. $THETAI Record (nm73)

`$THETAI`
the abbreviated code

Optional. This record gives an abbreviated code for the THETAISUB subroutine, which transforms the initial values in the $THETA and $THETAP records. If the initial estimate for an element of theta is transformed, so is the upper and lower bounds for that theta, if any. The record name may also be coded as $THI. If it appears, it must be with the first problem specification, and only with this problem specification. The use of $THETAI is independent of the choice of $PRED vs. PREDPP. Specific rules for $THETAI are described in Introduction to NONMEM 7 and Guide VIII and on-line help.

IV.IV.N. $THETAR Record (nm73)

`$THETAR`
the abbreviated code

Optional. This record gives an abbreviated code for the THETARSUB subroutine, which transforms the final theta values for the NONMEM report and additional output files. The record name may also be coded as $THR. The use of $THETAR is independent of the choice of $PRED vs. PREDPP. Specific rules for $THETAR are described in Introduction to NONMEM 7 and Guide VIII and on-line help.

V. NM-TRAN with PREDPP

V.V.A. Introduction

NM-TRAN is designed to facilitate the use of PREDPP. This chapter addresses special considerations regarding the use of NM-TRAN with PREDPP.

Here follows an example of an NM-TRAN control stream; it is meant to be used with PREDPP along with the NM-TRAN data set shown in Appendix VI. This NM-TRAN control stream is recorded on the NONMEM distribution medium; see Guide III. The control stream and data set are constructed so to accomplish the same things as do the control stream and data set considered in chapter I, but they are constructed on the assumption that PREDPP is to be used. NM-TRAN will translate the data set and control stream to a NONMEM data set, a NONMEM control stream, and completely coded PK and ERROR subroutines. These NM-TRAN outputs are given in Appendix VII. The effect of using them as inputs to a NONMEM run will be to produce essentially the same output obtained from using the NONMEM control stream and PK and ERROR subroutines shown in Appendix II of Guide VI.

```
$PROB  THEOPHYLLINE   POPULATION DATA
$INPUT      ID DOSE=AMT TIME CP=DV WT
$DATA       THEOPP

$SUBROUTINES  ADVAN2

$PK
;THETA(1)=MEAN ABSORPTION RATE CONSTANT (1/HR)
;THETA(2)=MEAN ELIMINATION RATE CONSTANT (1/HR)
;THETA(3)=SLOPE OF CLEARANCE VS WEIGHT RELATIONSHIP (LITERS/HR/KG)
;SCALING PARAMETER=VOLUME/WT SINCE DOSE IS WEIGHT-ADJUSTED
   CALLFL=1
   KA=THETA(1)+ETA(1)
   K=THETA(2)+ETA(2)
   CL=THETA(3)*WT+ETA(3)
   SC=CL/K/WT

$THETA  (.1,3,5) (.008,.08,.5) (.004,.04,.9)
$OMEGA BLOCK(3)  6 .005 .0002 .3 .006 .4

$ERROR
   Y=F+EPS(1)

$SIGMA  .4

$EST     MAXEVAL=450  PRINT=5
$COV
$TABLE          ID DOSE WT TIME
$SCAT           (RES WRES) VS TIME BY ID
```

V.V.B. Data Set Translation with PREDPP

When PREDPP is used, EVID and MDV data items are required in the NONMEM data set. When PREDPP is used, MDV data items need not be included in the NM-TRAN data set; NM-TRAN can automatically include them in the NONMEM data set. Such MDV data items are called generated MDV

data items . Under certain circumstances, NM-TRAN can also automatically include EVID data items in the NONMEM data set. Such EVID data items are called generated EVID data items .

If EVID data items are not included in the NM-TRAN data set, they are generated as follows: If each of the AMT and RATE data items in an event record is either 0, or is represented by a NM-TRAN null data item, or is missing, then the generated EVID data item is 0 (meaning that the event is an observation event). Otherwise, the generated EVID data item is 1 (meaning that the event is a dose event). If there are other-type, or reset, or reset-dose events in the data set, then the user himself should supply EVID data items.

If MDV data items are not included in the NM-TRAN data set, they are generated as follows: If the data record is the last record of an event record, then the generated MDV data item is 0 or 1 according as the EVID data item (generated or user-supplied) is 0 or not 0. If the data record is not the last record of an event record, then the generated MDV data item is 1.

With PREDPP, interdose interval (II) data items may appear. NM-TRAN can translate II data items expressed in the form hr:min to II data items expressed in the form hr.fr, where min (minutes) can be any two-digit integer from 00 to 59, and fr is a decimal fraction of the hour to two digits. Here hr (hours) can be any nonnegative integer. Examples of II data item translation are these: 1:30 $\rightarrow$ 1.50; 1.30 $\rightarrow$ 1.30. Note that if the data item has a colon, it is translated; otherwise, it is left unchanged. It is the user's responsibility to make sure that the units of the TIME data items and the II data items are consistent. For example, suppose that TIME data items in the NM-TRAN data set are expressed as relative times in minutes; they are then left unchanged by NM-TRAN. II data items in the NM-TRAN data set should then also be expressed in minutes, and they should not contain colons so that NM-TRAN leaves them unchanged.

With NONMEM 7.3, values may also have the form hh:mm:ss (i.e., hours:minutes:seconds). As described in Chapter II, the TRANSLATE option of the $DATA record was new to NONMEM V and has been expanded with NONMEM 7.3. Any values may be given for dividing TIME and II values, and any precisions may be requested. An example is:

```
$DATA TRANSLATE (II/0.01/6)
```

which divides II values by 0.01, and writes 6 digits to the right of the decimal for the II data item. See Help guide for more details.

V.V.C. Control Records with PREDPP

There are a few control records which are specific to the use of PREDPP. These are described in the following subsections, along with details concerning the nonspecific control records which are pertinent to the use of PREDPP only. A listing of NM-TRAN control records and options which should be used with PREDPP is given in Appendix V.

V.V.C.1. $INPUT Record

The $INPUT record is described in detail in section III.B.2.

NM-TRAN recognizes these PREDPP-specific reserved labels: `EVID`, `TIME`, `AMT`, `RATE`, `SS`, `II`, `ADDL`, `CMT`, `PCMT`, `CALL`. By using any one of these labels in an item of the $INPUT record, the user defines the PREDPP data item type whose name corresponds to the label.
With NONMEM 7.2, the following labels are also recognized: `XVID1`, `XVID2`, `XVID3`, `XVID4`, `XVID5`. These stand for "extra" EVIDs. They are used with the "Repeated Observation Records" feature for specialized methodologies such as stochastic differential equations ("sde"). See Introduction to NONMEM 7.3 and Guide VIII and on-line help.

When PREDPP is not used, any of these labels can be used, but with the exception of `TIME`, they have no special significance.

Ignoring items with the `DROP` or `SKIP` labels, the total number of items in a NM-TRAN data record cannot exceed PD in resource/SIZES.f90 (default is 50; See Chapt III),
including generated EVID data items, generated MDV data items, and, if the data set is single-subject, generated ID data items (all of which actually do not appear in the NM-TRAN data set). Generated EVID data items are assigned the label `EVID`, and generated MDV data items are assigned the label `MDV`. These labels can be used in subsequent NM-TRAN control records of the problem specification.

V.V.C.2. $BIND Record

`$BIND` $value_1$ $value_2$ ... $value_n$

E.g.

```
$INPUT       ID     DOSE=AMT     TIME     CP=DV     WT    PREP
$BIND        -      -            -        -         NEXT  DOSE
```

This record is used in conjunction with the $INPUT record to further define the meaning of certain data items occuring as right-hand quantities in an abbreviated code for PK when PK is called at nonevent dose times (see below and Guide VI, section III.B.2). (Normally, PK is not called at these times.) At such calls, PK has access to several different data items of the same type, from several different event records. The definitions in the $BIND record precisely determine which of these data items are represented by the label for the data item type used in the abbreviated code.

This record is optional. If it appears, it must be with the first problem specification, and only with this problem specification. It can only be used when an abbreviated code for PK is also used.

There appear n values, where n should not exceed the total number of labels listed on the $INPUT record, including all `DROP` and `SKIP` labels. The ith value corresponds to the ith label.

A nonevent dose time is a time that a lagged dose or additional dose enters, or starts to enter, the system. It is some time after the event time occuring on the dose record. When PK is called at such a time, information from three event records is available. These are the dose record itself, the last event record, and the next event record. The last event record is the last event record with event time occuring before the nonevent dose time. The next event record is the first event record with event time occuring after the nonevent dose time. This event record is also called the argument record (see Guide VI, section III.B.2). PK is called at nonevent dose times only if a certain PK calling-protocol is requested (see section C.5).

Some data item types are NONMEM data item types: `ID`, `L2`, `MDV`, `DV`, and these are not affected by the $BIND definitions. At a nonevent dose time, these labels always represent the data items from the dose record; this is the mandatory representation. Some data items types are PREDPP data item types: `AMT`, `RATE`, `SS`, `ADD`, `II`, `MT`, `CMT`, `CALL`, `EVID`, and these are not affected by the $BIND definitions. These labels also always represent the data items from the dose record; this is the mandatory representation. The `TIME` data item type is a PREDPP data item type, and it is not affected by the $BIND definition; the label always represents the time data item from the next event record. The other data item types (labeled `WT` and `PREP` in the above example) are only recognized and responded to by the user's PK routine. With the $BIND record the user may define the label of such a data item type to represent the data item from the dose record, the last event record, or the next event record. If a label definition for a data item type is not given explicitly, then by default the label represents the data item from the next event record.

If the value is (explicitly) `DOSE`, `LAST`, or `NEXT`, the label represents the item from the dose record, the last event record, or the next event record, respectively. If one of these values is used, it should not contradict the mandatory representation for the label (of a NONMEM or PREDPP data item). The value '-' serves as a place holder; a value is assumed that corresponds to the mandatory or default representation. If the number n is less than the total number of labels listed on the $INPUT record, the values corresponding to the remaining labels are taken to correspond to the respective mandatory and default representations. If the $BIND record is not used, the values corresponding to all labels are taken to correspond to the respective mandatory and default representations.

In the above example, in abbreviated PK code the variable `PREP` denotes the preparation indicator on the dose record describing the dose entering the system at a nonevent dose time. If the $BIND record is not used, the variable denotes the preparation indicator on the next event record, which may not even be a dose record. The variable `WT` denotes the weight on the next event record, which is the default

representation.

If the label in the $INPUT record is `DROP` or `SKIP`, the issue of label representation is moot and is ignored. In this case the value can be '-', or it can be `DROP` or `SKIP`.

$INPUT and $BIND records may be interleaved to help maintain a perspicuous visual relationship in the control stream. The above example might have been written thusly:

```
$INPUT      ID    DOSE=AMT    TIME    CP=DV
$BIND       -     -           -       -

$INPUT      WT   PREP
$BIND       NEXT DOSE
```

As with changes to the $INPUT record, changes to the $BIND record may cause changes to the generated codes. In this case care should be taken in using the previous load module.

Even when a value is `LAST` or `NEXT`, it is still possible for an abbreviated code for PK to access the data item in the dose record; see section C.5.

$BIND has no effect when PK is called at a model event time (MTIME).

V.V.C.3. $SUBROUTINES Record

```
$SUBROUTINES [subname1 = name1] [subname2 = name2] ...
             [SUBROUTINES=kind]
             [TOL=n1] [ATOL=n2] [SSTOL=n3] [SSATOL=n4]
             [DES=COMPACT|DES=FULL]
```

E.g.

```
$SUBROUTINES    ADVAN=ADVAN8 TOL=4
```

NM-TRAN recognizes these additional subnames: ADVAN, SS, TRANS, PK, ERROR, DES, AES, INFN, TOL, MODEL

The first two subnames, ADVAN and SS, are *not* generic names of subroutines which can be user-supplied, in contrast to all other possible subnames which can appear in the $SUBROUTINES record. Rather, they are generic names of subroutines from the PREDPP Library. With PREDPP the subname ADVAN must appear and be set equal to one of the names: ADVAN1, ADVAN2, ..., whichever specific ADVAN routine is chosen from the Library.†

The subname SS need appear only if steady-state data items are included in the data set. Even then, if subroutine ADVANk is chosen, it will be assumed that subroutine SSk is also chosen, and the subname-name pair SS=SSk need not appear. (If ADVAN8 or ADVAN10 is chosen, it will be assumed that subroutine SS6 is also chosen.)

The third subname, TRANS, is a generic name of a subroutine which can come from the PREDPP Library or be user-supplied. If the subroutine comes from the Library, the name must be either TRANS1, or TRANS2, etc. If the subroutine is user-supplied, the name must be different from any name of a TRANS routine in the Library. A subname-name pair for the TRANS routine need not be given; then TRANS=TRANS1 is "understood".

If the PK subroutine is not user-supplied, so that PK is not used as a subname, then an abbreviated code must be given for PK. If the ERROR subroutine is not user-supplied so that ERROR is not used as a subname, then an abbreviated code must be given for ERROR. Abbreviated codes may be given for neither routine, or for only one of them, or for both of them.

Whether the DES routine is used depends on the ADVAN routine used. Similarly, whether the AES routine is used depends on the ADVAN routine used. If the DES (AES) routine is required by the ADVAN routine, and if it is not user-supplied, so that DES (AES) is not used as a subname, then an abbreviated code must be given for DES (AES). Abbreviated code for the AES routine is actually given by two abbreviated codes; see sections C.8 and C.9.

The INFN routine is always called by PREDPP. If it is not user-supplied, the INFN routine from the PREDPP library is used. It is a stub that does nothing. INFN may be user supplied or defined by an $INFN routine (see section C.11).

Whether the TOL routine is used depends on the ADVAN routine used. Similarly, whether the MODEL routine is used depends on the ADVAN routine used.

If the TOL routine is required by the ADVAN routine, and if it is neither user-supplied nor defined by a $TOL record (see section C.10), so that TOL is not used as a subname, then using the TOL option, NM-

† ADVAN stands for ADVANCE because it is the task of each ADVAN routine to advance the state vector of compartment amounts (and partial derivatives of the compartment amounts with respect to random variables, when appropriate) from each point in time to the next. Another possible name for ADVAN would have been SOLVER, because most ADVAN routines solve a set of differential equations, either analytically (ADVAN1-5, ADVAN7, ADVAN11, ADVAN12) or by integration (ADVAN6, ADVAN8, ADVAN9, ADVAN13, ADVAN14, ADVAN15).

TRAN will generate a complete FORTRAN coded TOL subroutine. The number n_1 with the `TOL` option is the number of accurate digits (NRD; "Number of required digits") required in the computation of all compartmental drug amounts. As a rule of thumb, one should begin by taking n_1 to be n + 1 or n + 2, or with double precision, perhaps to n + 2 or n + 3, where n is the option value with the `SIGDIGITS` option on the $ESTIMATION record. If one succeeds with this setting, one might try increasing n_1 slightly. With ADVAN9 the number of accurate digits can be specified on a compartment-specific basis. However, to do this, either a TOL routine must be user-supplied or the $TOL record must be used.

With ADVAN13, the TOL option of the $SUBROUTINE record (or the $TOL record) is a relative tolerance. It should specify larger NRD values than for other ADVANs (e.g., ADVAN6). ATOL (Absolute tolerance) may be specified on the $ESTIMATION or $COVARIANCE record.

With NONMEM 7.4, one or more additional options `ATOL=`n_2, `SSTOL=`n_3, `SSATOL=`n_4 are also permitted on the $SUBROUTINES record.

ATOL specifies the absolute tolerance for ADVAN9, ADVAN13, ADVAN14, and ADVAN15. Optional. Default is 1.0E-12.
SSTOL specifies the relative tolerance for Steady State evaluation. Optional. Default is TOL.
SSATOL specifies the absolute tolerance for Steady State evaluation. Optional. Default is ATOL.

The generated TOL routine will set values as follows for the options that are coded:

`NRD(1)=`n_1 `(The TOL option)`
`ANRD(1)=`n_2 `(The ATOL option)`
`NRD(0)=`n_3 `(The SSTOL option)`
`ANRD(0)=`n_4 `(The SSATOL option)`

It is also possible to code TOL=name to specify the name of a user-supplied TOL routine, or to include $TOL abbreviated code, either of which allows all these values to be assigned by compartment. A user-supplied TOL routine also allows values to be assigned for each NONMEM step and problem. See also the ATOL option of the $ESTIMATION record, and the TOL and ATOL options of the $COVARIANCE record.

If the MODEL routine is required by the ADVAN routine, the MODEL routine may be user-supplied. Then the PK routine must also be be user-supplied If a DES (AES) routine is needed, then it too must be user-supplied. If the MODEL routine is not user-supplied, so that `MODEL` is not used as a subname, then using information supplied in the $MODEL record (see next section C.4), NM-TRAN will generate a complete FORTRAN coded MODEL subroutine. In this case an abbreviated code for PK may be given, as may abbreviated codes for DES and AES.

Each of the subnames `ADVAN`, `SS`, and `TRANS`, along with the equal sign that follows it, may be omitted when it is followed by a specific name of a PREDPP Library routine; see, for example, the control stream in section A. However, when the subname is not omitted, the specific name of a PREDPP Library routine can be given by the associated number only, e.g. `ADVAN=1` instead of `ADVAN=ADVAN1`.

The $SUBROUTINES record may be used with PREDPP to supply "OTHER" routines, as described in chapter III. An example is given in NONMEM 7 guide, "Stochastic Differential Equation Plug-In(NM72)". The code in sde9.ctl is
`SUBROUTINE ADVAN6 TOL=9 DP OTHER=SDE.f90`
The file SDE.f90 is found in the examples directory.

V.V.C.4. $MODEL Record

```
$MODEL [NCOMPARTMENTS=n1] [NEQUILIBRIUM=n2] [NPARAMETERS=n3]
       [COMPARTMENT=([name] [attribute1] [attribute2] ...)] ...
       [LINK compnamea [TO|AND] compnameb BY k [l]] ...
       [I_SS=n]
```

```
E.g.
$MODEL   NPARAMETERS=3 NCM=2†
         COMP=(DEPOT DEFDOSE INITIALOFF) COMP=(CENTRAL DEFOBS NOOFF)
         LINK DEPOT CENTRAL BY 3
         LINK CENTRAL OUTPUT BY 1
```

This record gives information from which NM-TRAN can generate the complete FORTRAN coded MODEL routine.

This record is required when the ADVAN routine requires a MODEL routine, and this routine is not user-supplied. It is only required for the first problem specification. It applies for all problem specifications in the control stream, and it must not appear with a problem specification other than the first.

The number n_1 is the total number of compartments other than the output compartment. The maximum vale of n is given by constant PC in resource/SIZES.f90; With NONMEM 7, the default is 30. (The value may be over-ridden by user via $SIZES record up to a maximum of 999). The NCOMPARTMENTS option may be coded as NCM or NCOMPS.

If the NCOMPARTMENTS option is omitted, this number is taken to be the number of COMPARTMENT clauses that appear in the record (and its continuation records).

The number n_2 is the number of equilibrium compartments; this number must not exceed n_1. If the NEQUILIBRIUM option is omitted, this number is computed from the attributes of the COMPARTMENT clauses. If the option is used, however, then the last n_2 compartments are understood to be equilibrium compartments, whether or not any of these compartments are defined with COMPARTMENT clauses which include the EQUILIBRIUM attribute.

The number n_3 is the number of explicit (and implicit; see sections C.7-9) basic PK parameters. When an abbreviated code for the PK routine is used, the NPARAMETERS option may be omitted. (When implicit basic PK parameters are defined, this is a convenient practice.) In this case, and in the case of a general linear model, the number of basic PK parameters is the total number of K-type parameters recognized in the abbreviated code. In the case of a general nonlinear model and TRANS1, the number of basic parameters is the larger of (i) the largest subscript used with the P array in PK code, and (ii) the number of variables defined in PK abbreviated code and used in DES and/or AES abbreviated codes when such abbreviated codes are used. See next section C.5.

Each COMPARTMENT clause defines a single compartment. The compartments are numbered in the order in which their defining clauses appear in the record (and its continuation records). The name is the name given to the compartment, 1-8 characters from the FORTRAN character set. If spaces or nonalphanumeric characters are used, enclose the name in double or single quotes. With NONMEM 7.4, the maximum number of characters is given by SD in resource/SIZES.f90 (default is 30).
The name may not be one of the compartment attributes below, unless it is enclosed in single or double quotes. E.g., COMP=(DEFOBS) is not permitted but COMP=("DEFOBS",DEFOBS) is permitted.
If omitted, the name COMP n is given to the compartment, where n is the compartment number. The compartment name is used in PREDPP problem summary pages and in LINK clauses in the $MODEL record. The name of the output compartment and the attributes of this compartment are set by PREDPP.

Option NCM=2 is needed with versions of NONMEM prior to NONMEM 7.4.1 to avoid a spurious error message from NM-TRAN.

For the purposes of NM-TRAN, the name of the output compartment is OUTPUT, and its number is 0 or $n_1 + 1$, either will do.

With NONMEM 7.5, compartment names defined in $MODEL are automatically available for substitution without requiring $ABBR REPLACE records. This is called "implicit" compartment name replacement. For example:

```
$MODEL
   COMP=(DEPOT)
```

allows substitutions to be made for A(DEPOT), DADT(DEPOT) etc, in abbreviated code:

```
$DES
DADT(DEPOT)=-KA*A(DEPOT)
```

The compartment number 1 is present in the generated subroutines.

Each attribute is one of: INITIALOFF, NOOFF, NODOSE, DEFOBSERVATION, DEFDOSE, EQUILIBRIUM, EXCLUDE. When an attribute is used, it specifies the opposite of the default attribute. The default attributes are: The compartment is initially on, may be turned on and off, may receive a dose, is not the default observation compartment, is not the default dose compartment, is not an equilibrium compartment, is included in the computation for the output compartment. If a user-defined comparment has attributes INITIALOFF NODOSE then it is initally off, may be turned on and off, and may not receive a dose. Such a compartment is called an output-type compartment. There may be more than one.†

Attributes EQUILIBRIUM and EXCLUDE are used only with ADVAN9 and ADVAN15, and the attribute NODOSE may be omitted when EQUILIBRIUM is also used, for it is then the default. Definitions for equilibrium compartments must follow the definitions for the nonequilibrium compartments.

If no attributes are used, the parentheses may be omitted. If neither compartment name nor attributes are used, i.e. the clause is simply COMPARTMENT, a compartment is defined where all defaults apply. In this case if the clause does not end the $MODEL record, it must be followed by a comma.

If no compartment has the attribute DEFOBSERVATION, the first compartment defined with the name CENTRAL is given the attribute.

If there is no such compartment, the first compartment is given the attribute DEFOBSERVATION. With versions of NONMEM prior to 7.4.1 a compartment that is initially off could thus be assigned as DEFOBSERVATION. This results in an error message if the PCMT data item is not used to specify explicitly which compartment should be used for the prediction associated with a dose or other-type event. With 7.4.1, the first compartment that is not INITIALOFF is given the attribute DEFOBSERVATION and PCMT is not needed.

If no compartment has the attribute DEFDOSE, the first compartment defined with the name DEPOT and which may receive doses is given the attribute. If there is no such compartment, the first compartment which may receive doses is given the attribute.

The LINK clauses need only be used with general linear models and only when the PK routine is user-supplied. When an abbreviated code for PK is used, this code accomplishes what the LINK clauses otherwise accomplish, and LINK clauses should not be used. A LINK clause defines a route of first-order drug distribution between the compartment A with name compname_a and the compartment B with name compname_b. These names are established in the COMPARTMENT clauses. A compartment number, rather than a compartment name, can be used. If distribution occurs from A to B, the TO symbol is used. The LINK clause also specifies the internal number given to the rate constant quantifying the first-order distribution. This is the number k following the BY symbol. (It is the number of the row of the GG array where, as a result of basic parameter translation by the TRANS routine, the typical/subject-specific value

† Output-type compartments have been part of PREDPP since the first version, but were not discussed in detail. For such compartments, the value of CMT may be negative on an obervation record to obtain an observation and turn the compartment off, just as it may with the default output compartment.

and η derivatives for the rate constant can be located; see Guide VI, section III.M.) If distribution occurs in both directions, the AND symbol is used. In this case the number k is the internal number of the rate constant quantifying first-order distribution from A to B, and l is the internal number of the rate constant quantifying distribution from B to A. Both k and l must not exceed n_3.

K is an alias for LINK. Also, the symbol BY may be coded IS, or =, or omitted. This allows a unidirectional link to be tersely coded as: Kmn=k (which is equivalent to LINK m TO n BY k). When the number of compartments exceeds 10, the LINK clause syntax Kmn=k may be ambiguous. The letter T may be used to separate the two compartment numbers KmTn.

Attributes DEFOBSERVATION and DEFDOSE may be abbreviated by initial substrings of length 4 or more.

I_SS= n requests the Initial Steady-State feature of PREDPP (NONMEM VI). It may be used with the general non-linear models (ADVAN6, ADVAN8, ADVAN9, ADVAN13, ADVAN14, ADVAN15). Values of n are

0 No initial state state (the default)
1 Initial steady state
2 Initial steady state, adds to current compartment amounts.
3 Initial steady state, use current compartment amounts as initial estimates.

The results are identical to those that would be computed by a steady-state dose event record with SS=I_SS and AMT=0 and RATE=0. If endogenous drug is specified in the differential equations, non-zero initial conditions will be computed.

The example of a $MODEL record given at the beginning of this subsection serves to produce a MODEL subroutine which, along with ADVAN5 or ADVAN7 implementing a general linear model, and the (code generated from the) abbreviated code for PK given in the example of section A, has the same effect as using ADVAN2 and the abbreviated code for PK. The same $MODEL record, without the LINK clauses, may be used along with ADVAN5 or ADVAN7, and a suitable abbreviated PK code, to achieve the same effect; see the example in Appendix VIII. The same $MODEL record, without the LINK clauses, may be used along with ADVAN6, ADVAN8, ADVAN9, ADVAN13, ADVAN14, ADVAN15, and suitable abbreviated PK and DES codes, to achieve the same effect; see the example in Appendix VIII.

V.V.C.5. $PK Record

`$PK`
abbreviated code

This record gives an abbreviated code for the PK routine. It, along with all its continuation records is called a $PK block .

This record is optional. If it appears, it must be with the first problem specification, and only with this problem specification. It must precede any $ERROR records in the problem specification. It cannot be used with a user-supplied MODEL subroutine (and should not be used with a user-supplied TRANS routine; see Guide VI, section III.M).

The basic PK parameters comprise the set of mandatory left-hand quantities. These depend on the ADVAN and TRANS subroutines used.

For any ADVAN among ADVAN1 through ADVAN4, and ADVAN10, ADVAN11, ADVAN12 (the analytic ADVAN's), and for any TRANS which may be used with this ADVAN, the basic PK parameters are given in a list with that TRANS in Guide VI, section VII.C. The reserved variables symbolizing these parameters are those whose names are identical to the names used in the list.

For either ADVAN5 or ADVAN7 (the general linear models), and for TRANS1 (which is the only non-user supplied TRANS which may be used with these ADVAN's), the basic PK parameters are the rate constants. The reserved variable symbolizing the rate constant that quantifies the first-order distribution of drug from compartment number m to compartment number n is `K`mn. (See section C.4 for a description of compartment numbering.) The occurence of this variable on the left of an assignment statement or conditional assignment statement establishes the possiblity that this distribution can take place. The variable `K`m`0` may be used instead of `K`mn, where n is the number of the output compartment.
When the number of compartments exceeds 10, the syntax `K`mn`=`k may be ambiguous. The letter `T` may be used to separate the two compartment numbers `K`m`T`n.
The rate constants are numbered (these numbers are used internally by the ADVAN) according to the order in which they appear in the abbreviated code. See the first example in Appendix VIII.

For ADVAN6, ADVAN8, ADVAN9, ADVAN13, ADVAN14, ADVAN15 (the general nonlinear models), and for TRANS1, the reserved array elements symbolizing explicit basic PK parameters are `P(1)`, `P(2)`, etc. The value of the nth element of the P vector passed to DES and AES (see sections C.7-9 and Guide VI, sections VI.C, VI.E) is the value stored in `P(n)`. These values, like other PK-defined items may be displayed; the appropriate labels are described below. Implicit basic PK parameters are discussed in sections C.7-9.

The additional PK parameters comprise a set of optional left-hand quantities. The reserved variables symbolizing these parameters are as follows:

Name	Parameter
`S`n	Scaling parameter for compartment number n
`F`n	Bioavailability fraction for compartment n
`R`n	Rate parameter for compartment n
`D`n	Duration parameter for compartment n
`ALAG`n	Absorption lag parameter for compartment n
`F0`	Output fraction
`TSCALE`	Time scale parameter
`MTIME(`i`)`	Model event times parameters (nmvi)

Scaling parameters, bioavailablity fractions, the ouput fraction, and the time scale parameter default to the value 1 if they do not appear in the abbreviated code. Absorption lag parameters default to the value 0 if they do not appear in the abbreviated code.

See section C.4 for a description of compartment numbering. The variable `FO` or `Fn`, where n is the number of the output compartment, may be used instead of `F0`. The variable `S0` may be used instead of `Sn`, where n is the number of the output compartment. The variable `SC` may be used to mean the scaling parameter with the central compartment. The variable `XSCALE` may be used instead of `TSCALE`. Whichever of the alternate variable names is used first for a basic or additional PK parameter, this name must be used consistently throughout the $PK block.

A model event time parameter `MTIME(i)` defines a time to which the system is advanced. When the time is reached, indicator variables are set and a call to PK is made. At this call (and/or subsequent to this call) PK or DES or AES or ERROR can use the indicator variables to change some aspect of the system, e.g., a term in a differential equation, or the rate of an infusion. `MTIME(i)` parameters are not associated with any specific compartment or dose. They are ignored if they have the value 0. `MTIME` parameters have no effect on steady-state doses; even if PK computes `MTIME(i)` < II, this produces future changes in the system, and does not apply retroactively to the preceding implied doses.

`MTDIFF` is an optional left-hand variable. It is of interest when model time parameters MTIME are used. The value of `MTDIFF` is 0 when PK is called. If PK sets `MTDIFF` to a value other than 0, e.g., `MTDIFF=1`, then PREDPP will understand that with that call to PK, the values of one or more of the MTIME(i) have possibly been reset. `MTDIFF`=0 (the default) can save considerable run time when there are many model time parameters. Note that the results are unpredictable if the times are in fact changed when `MTDIFF`=0.

Array elements `A_0(1)`, ..., `A_0(n)` may be used on the left-hand to assign initial values to compartments. A reserved right-hand variable, `A_0FLAG`, is set to 1 by PREDPP when PK may initialize compartments to specific amounts. For example,

```
IF (A_0FLG.EQ.1) THEN
   compartment initialization block
ENDIF
```

A compartment initialization block includes statements such as

```
   A_0(n)=....
```

This specifies the initial amount for compartment n. `A_INITIAL(n)` is a synonym for `A_0(n)`.
The above code fragment is an explicit compartment initialization block. `A_0(n)` may be assigned a value with an unconditional statement. This defines an implicit compartment initialization block. NMTRAN inserts "IF (A_0FLG.EQ.1) ..." before the statement and "ENDIF" after it. Indicator variables may be used to avoid conditional assignment statements. See the help items for Compartment Initialization.

A reserved left-hand variable `I_SS` may be set to the same values as the I_SS option of $MODEL record. This allows initial steady-state to be set conditionally, e.g., if some subjects are at steady-state and others are not.

There is a type of pseudo-statement specific to PK abbreviated code. It has the form `CALLFL`=n. The different permissible values for n imply different PK calling-protocols. A calling protocol phrase can be used instead of the CALLFL pseudostatement. The phrase must be enclosed in parentheses. Examples of phrases follow each value of CALLFL.

value	calling-protocol
-2	PK called with every event record and at every nonevent dose time

$PK (NON-EVENT)
$PK (ADDITIONAL OR LAGGED)

-1 PK called with every event record (default)
$PK (EVERY EVENT)
$PK (EVERY)

0 PK called with first event record of the individual record
and with every subsequent event record where the time data item
differs from the time data item of the previous event record
$PK (NEW TIME)
$PK (NEW EVENT TIME)

1 PK called only with first event record of the individual record
$PK (ONCE PER INDIVIDUAL RECORD)
$PK (ONCE/IND.REC.)

The value must be -2 when the $BIND record is used.

If the pseudo-statement does not appear, the value -1 is assumed. This allows PK to properly function in most situations.

Values 0 and 1 correspond to call-limiting protocols. Limiting calls to PK, when this causes no undesireable effect, can result in a substantial reduction in CPU time. The CALL data item can be used with a call-limiting protocol to override the protocol and force calls with specific event records; see Guide VI, section V.J. When n is 0 or 1, and the data are single-subject data, then PK is called with the first event record of the data set, instead of the first event record of the individual record.

As usual, the label of a data item type can be used as a variable in the abbreviated code. When PK is called at a nonevent dose time t, the data item referenced by a given label may refer to the data item on either the dose record or the next event record following t. The defaults are described in section C.2, and a method (using the $BIND record) is given for changing them.

There are several reserved variables and array elements symbolizing special right-hand quantities:

The variable `ICALL` symbolizes a special right-hand quantity. The values are identical to those for `ICALL` in PRED, as described in Chapter IV. It has the value 2 if the call is a regular call during data analysis, and the value 4 if the call is a regular call during data simulation. It has the value 5 if the call to PK occurs when expectations are being computed (the marginal data item MRG_ has a non-zero value for some records).

If there is abbreviated code in the $PK block that tests for ICALL=0, ICALL=1, or ICALL=3, this code is moved by NM-TRAN to the INFN routine as if it had been coded explicitly as part of an $INFN block. Such code is called $PK-INFN code. With verbatim code in the FIRST block (see section IV.I), `ICALL` will never have the value 0 or 3. ICALL has the value 1 if the call to PK is the first call to PK in the problem. At this call, the `THETA`'s are the initial estimates; the `ETA`'s are undefined. Verbatim code is not moved to the INFN routine.

The variable `NEWIND` symbolizes a special right-hand quantity. It has the value 0 when PK is called with the first event record of the data set. It has the value 1 when PK is called with the first event record of the second or subsequent individual record. It has the value 2 when PK is called with the second or subsequent event record of an individual record. With single-subject data individual records are defined in such a way that event records are contained in a number of different individual records; see section II.C.4.1. Therefore, except when the event record is the first data record of the data set and the value of `NEWIND`=0, the value of `NEWIND` can be 1 or 2.

The variable `DOSTIM` symbolizes a special right-hand quantity. It has a nonzero value only when `CALLFL=-2` and PK is being called at a nonevent dose time (see section C.2), in which case the value is

this time. DOSTIM should be regarded a random variable when *any* ALAGn variable is a random variable. In this regard, if any ALAGn is defined as a random variable, it must be defined as such before any occurence of the variable DOSTIM in an assignment statement.

The array element DOSREC(n) symbolizes a special right-hand quantity. It has a nonzero value only when CALLFL=−2 and PK is being called at a nonevent dose time, in which case it is the value of the nth data item in the (last data record of the) dose event record describing the dose. A label for a data item in the event record may be used instead of the integer n, e.g. one can use DOSEC(PREP) to symbolize the value of PREP on the dose event record. The $BIND record can also be used to insure that a variable such as PREP symbolizes the value on the dose event record. However, another example of the use of DOSREC, which is sometimes useful and with which the $BIND record cannot help as readily, is the use of DOSREC(TIME) to symbolize the value of TIME on the dose event record, i.e. the time the dose was actually administered.

PREDPP sets right-side variables MNOW and indicator variables MNEXT(i) and MPAST(i) when Model Event (MTIME(i)) parameters are defined. MNOW=i if MNEXT(i)=1 for some i. MNOW=0 otherwise. MNEXT(i)=1 during the advance from the previous time to MTIME(i). Otherwise, MNEXT(i)=0. The previous time may be an event time, a non-event time, or a model event time. MPAST(i)=0 until the call to PK subsequent to the one for which MNEXT(i)=1. At that call MPAST(i) becomes 1.

The array elements A(1), ..., A(n) symbolize special right-hand quantities, the amounts in compartments 1 through n. They are the latest computed compartment amounts when PK is called. This is the called the state-vector of compartment amounts. A right-hand variable TSTATE is the state-time, i.e., the time at which they were computed. In a population study, where η variables affect the drug amounts through their affect on PK parameters, these amounts are random variables. $OMEGA records referring to ηs explicitly used in $PK code should precede the $PK record, or if an $MSFI record is used, it should precede the $PK record and include the option NPOP=m. If an element A(n) appears, then the variable A cannot also be used.

Chapter IV describes the use of NONMEM MODULE NMPRD4. If COMRES=1 is *not* present in $ABBREVIATED or $PK records, then PK-defined variables are listed in NMPRD4 (see sections III.B.7 and IV.H) and may be used in other routines and blocks of abbreviated code. The symbol COM(n) may be used on the left or the right if n refers to a reserved position in MODULE NMPRD4 (see section III.E.3). Variables defined in $INFN and listed in MODULE PRINFN are also global and may be used in $PK and other blocks of abbreviated code, on the left and on the right. Variables defined by $ABBREVIATED DECLARE records are also global and may be used in $PK and other blocks of abbreviated code, on the left and on the right.

The abbreviated code may not use certain variables which occur as arguments to the PK subroutine. These variables are: IDEF, IREV, EVTREC, NVNT, INDXS, IRGG, GG, and NETAS.

Also, the array elements EPS(1), EPS(2), etc. may not be used in the abbreviated code for PK.

Variables which symbolize (first-, and second-) partial η-derivatives of random variables defined in abbreviated code for PK are generated and displayable. The appropriate labels are the same as those used for the same kinds of derivatives computed in a generated code for PRED; see section IV.F.

Values of the variable P(n) are displayable. For this purpose, they are stored in a variable in MODULE NMPRD4 with name Pm, where m is an integer with 5 digits and equal to n, with leading 0's if needed (invariably). The values of variables P00..., in particular, are labeled 6... in tables and scatterplots. E.g. The values of variable P00003 are labeled 6003 and are the values of P(3).

If the data are population data, calls to GETETA and SIMETA occur in PK, as they occur in PRED when the $PRED record is used (for some discussion of SIMETA, see section III.B.13), to obtain values for all the η variables. Both PK and ERROR contain the declaration

```
USE NMPRD_REAL,ONLY: ETA,EPS
```

so that ETA has the same value in both routines. (Generated PRED also declares ETA and EPS this

way.)

PRED-error recovery (see section IV.G) is supported by PREDPP and NM-TRAN. This means that the EXIT statement described in section IV.G.2, when used in an abbreviated code for PK (or ERROR) generates an quick return to NONMEM with a PRED return code.

V.V.C.6. $ERROR Record

`$ERROR`
abbreviated code

This record gives an abbreviated code for the ERROR routine. It, along with all its continuation records is called a $ERROR block .

This record is optional. If it appears, it must be with the first problem specification, and only this problem specification. It must succeed all $PK records in the problem specification.

There is only one mandatory left-hand quantity: the quantity symbolized by `Y` and described in section IV.A.
`Y` may be also used on the right, e.g. `LOGY=LOG(Y)`. (nmv)

There is a type of pseudo-statement specific to ERROR abbreviated code. It has the form `CALLFL=n`. The different permissible values for n imply different ERROR calling-protocols. A calling protocol phrase can be used instead of the CALLFL pseudostatement. The phrase must be enclosed in parentheses. Examples of phrases follow each value of CALLFL.

value	calling-protocol
-1	ERROR called with every event record (default) $ERROR (EVERY EVENT) $ERROR (EVERY)
0	ERROR called only with observation event records $ERROR (OBSERVATION EVENT) $ERROR (OBS)
1	ERROR called only with first event record of the individual record $ERROR (ONCE PER INDIVIDUAL RECORD) $ERROR (ONCE/IND.REC.)

If the pseudo-statement does not appear, the value -1 is assumed.

Values 0 and 1 correspond to call-limiting protocols. Limiting calls to ERROR, when this causes no undesireable effect, can result in a substantial gain in CPU time. The CALL data item can be used with a call-limiting protocol to override the protocol and force calls with specific event records; see Guide VI, section V.J. When n is 1, and the data are single-subject data, ERROR is called only with the first event record of the data set, instead of the first event record of the individual record.

Another call-limiting protocol is implementable, but not with the use of a pesudo-statement in the abbreviated code for ERROR. With this protocol, calls to ERROR are limited to one per problem. This protocol is implemented whenever the abbreviated code consists of *only* one of the following statements:

```
Y=F+ERR(1)
Y=F*(1+ERR(1))
Y=F+F*ERR(1)
Y=F*EXP(ERR(1))
```

or with `ETA` or `EPS` occuring instead of `ERR`. This protocol is not implemented if verbatim code or a pseudo-statement is used in the $ERROR block. (In the last three cases the complete code instructs PREDPP that HH(1) contains $\frac{\partial \log\ y}{\partial \varepsilon}$, and so in each case it is sufficient to set HH(1) to 1, and do this

once only at an initial problem call to ERROR.)

With any call-limiting protocol, during the Simulation Step, ERROR is called with every event record.

As usual, the label of a data item type can be used as a variable in the abbreviated code.

The variable `F` symbolizes a special right-hand quantity: the value of the scaled drug amount in the observation or other designated compartment (see Guide VI, section V.H) at the event time. In a population study, where η variables affect the scaled drug amount through their affect on PK parameters, the scaled drug amount is a random variable.

The array elements `A(1)`, ..., `A(n)` symbolize special right-hand quantities, the amounts in compartments 1 through n. This is the state-vector of compartment amounts. In a population study, where η variables affect the drug amounts through their affect on PK parameters, these amounts are random variables. If an element `A(n)` appears, then the variable `A` cannot also be used.

Variables defined in the $PK block can be used in the $ERROR block (unless variables are not listed in MODULE NMPRD4 in either PK or ERROR; see sections III.B.7 and IV.H). They may not be used on the left if they are random variables. The symbol `COM(n)` may be used on the left or the right if n refers to a reserved position in MODULE NMPRD4 (see section III.E.3). Variables defined in $INFN and listed in MODULE PRINFN are also global and may be used in $ERROR, on the left and on the right. Variables defined by $ABBREVIATED DECLARE records are also global and may be used in $ERROR, on the left and on the right.

The variable `ICALL` symbolizes a special right-hand quantity. It has the value 2 if the call to ERROR is a regular call during data analysis, and the value 4 if the call is a regular call during data simulation. It has the value 5 if the call to ERROR occurs when expectations are being computed (the marginal data item MRG_ has a non-zero value for some records). It has the value 6 if the call to ERROR occurs when raw data averages are being computed (the raw-data-average data item RAW_ has a non-zero value for some records).

If there is abbreviated code in the $ERROR block that tests for ICALL=0, ICALL=1, or ICALL=3, this code is moved by NM-TRAN to the INFN routine as if it had been coded explicitly as part of an $INFN block. Such code is called $ERROR-INFN code. With verbatim code in the FIRST block (see section IV.I), `ICALL` will never have the value 0 or 3. ICALL has the value 1 if the call to ERROR is the first call to ERROR in the problem. At this call, the `THETA`'s are the initial estimates; the `ETA`'s are undefined. Verbatim code is not moved to the INFN routine.

The variable `NEWIND` symbolizes another special right-hand quantity. It has the value 0 when ERROR is called with the first event record of the data set. It has the value 1 when ERROR is called with the first event record of the second or subsequent individual record. It has the value 2 when ERROR is called with the second or subsequent event record of an individual record. With single-subject data individual records are defined in such a way that event records are contained in a number of different individual records; see section II.C.4.1. Therefore, except when the event record is the first data record of the data set and the value of `NEWIND`=0, the value of `NEWIND` can be 1 or 2.

The abbreviated code may not use certain variables which occur as arguments to the ERROR subroutine. These variables are: `IDEF`, `IREV`, `EVTREC`, `NVNT`, `INDXS`, `G`, and `HH`.

Variables which symbolize (first-, second-, mixed-) partial η-derivatives of random variables defined in abbreviated code for ERROR are displayable. They have names `D.....`, where the dots stand for various combinations of 5 digits `0-9`. The values of variables `D00...`, in particular, are labeled `3...` in tables and scatterplots. E.g. The values of variable `D00123` are labeled `3123`. The label for the values of a variable `D01...`, or higher, is the first four characters of the variable name. E.g. The values of `D05677` are labeled `D056`.

PRED-error recovery (see section IV.G) is supported by PREDPP and NM-TRAN. This means that the EXIT statement described in section IV.G.2, when used in an abbreviated code for PK or ERROR

generates an quick return to NONMEM with a PRED return code.

The following discussion, relating to the $ERROR block, supplements that found in section II.C.4.2.

If (i) `EPS`'s are used in the $ERROR block, or (ii) a $SIGMA record is used, the data are inferred to be population data. If (iii) a $PK record is not used and a $OMEGA record precedes the $ERROR record, the data are inferred to be population data. This is true even if i and ii do not hold, and if ii does not hold, it is assumed that a record

```
$SIGMA  DIAGONAL(n)
```

where n is the largest index used with an `EPS` or `ERR` in the $ERROR block, might have equivalently been used. If (iv) the `NPOPETAS` option is used on an $MSFI record with a positive option value, the data are inferred to be poulation data. If neither i, ii, iii, nor iv hold, the data are inferred to be single-subject. See section II.C.4

Suppose a $PK record is not used, a $ERROR record is used, and only `ETA`'s or only `ERR`'s occur in the $ERROR block. As a corollary of iii (and comments in sections III.B.10 and IV.A), the following hold:

> An $OMEGA record can precede or follow the $ERROR record, in which case the data are taken to be population or single subject, respectively. If the $OMEGA record precedes the $ERROR record, then `ERR`'s must be used.
>
> If an $OMEGA record precedes the $ERROR record, but is continued following the $ERROR record, the data are taken to be population data, and again `ERR`'s must be used.
>
> If an $OMEGA record is not used, the data are taken to be single-subject, and it is assumed that a record
> `$OMEGA  DIAGONAL(n)`
> where n is the largest index used with an `ETA` or `ERR` in the $ERROR block, might have equivalently been used.

If option LIKELIHOOD or -2LL is used on the $ESTIMATION record, NM-TRAN recognizes the data as odd-type data. The data are categorical, rather than continuous. Then η variables still represent random interindividual effects, and random intraindividual variablity exists, but it is expressed without the use of ε variables or $SIGMA records. If the data are population and there is no $PK record, the $OMEGA block must precede the $ERROR block.

V.V.C.7. $DES Record

`$DES`
abbreviated code

This record gives an abbreviated code for the DES routine. It, along with all its continuation records, is called a $DES block .

This record is optional. If it appears, it must be with the first problem specification, and only this problem specfication. It may not appear when the MODEL routine is user-supplied. Implicit basic PK parameters may be used in the $DES block (see below) only when the $PK block precedes the $DES block.

The mandatory left-hand quantities are the first-order derivatives of the differential equations, symbolized by `DADT(n)`, for the derivative of the amount in compartment n (excluding the output compartment) with respect to time. At least one such array element must be defined.

Symbolic differentiation is used to obtain code used in the generated subroutine to compute the elements of the DA, DP, and DT arrays.

The allowable right-hand quantities include: current compartment amounts, symbolized by `A(n)`, for the amount in the nth compartment (including both equilibrium compartments when they exist, but excluding the output compartment); PK parameters (obtained from TRANS1), symbolized by `P(m)`, for the mth parameter; and time, symbolized by `T`. In a population study, where η variables affect the PK parameters, and, therefore, also affect the compartment amounts, these right-hand quantities should be regarded as random variables. However, η-derivatives are not computed in the generated routine itself. For technical reasons, these right-hand quantities should be regarded as random variables even when the data are single-subject. This means, for example, that these variables may be used in conditional assignment statements subject to the usual restriction on random variables. (Drug input information is not available for computations; PREDPP itself incorporates this type of information appropriately.)

Also, a PK-defined item may be used as a right-hand quantity. However, when so used, it becomes a special quantity called an implicit basic PK parameter , and a few special considerations apply:

> Implicit basic PK parameters may be used in the $DES block only when the $PK block precedes the $DES block.
>
> One should think of elements of the `P` array as including values for explicit basic parameters , i.e. parameters defined in the $PK block in array elements `P(n)`, followed by values for the implicit basic parameters. Therefore, the option value for the `NPARAMETERS` option in the $MODEL record should be large enough to include all these elements of this extended `P` array. Note: Explicit basic parameter values are displayable; see section C.5.
>
> Even additional PK parameters become implicit basic PK parameters.

The symbol `COM(n)` may be used on the left or the right if n refers to a reserved position in MODULE NMPRD4 (see section III.E.3). Variables defined in $INFN and listed in MODULE PRINFN may be used in $DES, on the left and on the right. Variables defined by $ABBREVIATED DECLARE records are also global and may be used in $DES, on the left and on the right.

The abbreviated code may not use certain variables which occur as arguments to the DES subroutine. These variables are: `IR`, `DA`, `DP`, and `DT`.

Variables which symbolize partial derivatives in the DA array are displayable. They have names `E.....`, where the dots stand for various combinations of 5 or 6 digits `0-9`. (The number of digits depends on the version of NONMEM.)

Variables which symbolize partial derivatives in the DP array are displayable. They have names `F.....`, where the dots stand for various combinations of 5 or 6 digits `0-9`.

Variables which symbolize partial derivatives of the DT array are displayable. They have names `E.....`, where the dots stand for various combinations of 5 or 6 digits `0-9`.

With versions of NONMEM prior to NONMEM 7.4.1, the labels for variables which symbolize partial derivatives were converted to 4 characters, as described in earlier versions of the guide. The labels were not always unique or meaningful, although the values displayed were always correct. A work-around for earlier versions is to use an alias that is meaningful to the user. For example, with CONTROL7 and NONMEM 7.4.0, NM-TRAN generates a variable named E000004 for the "DERIVATIVE OF DADT(1) W.R.T. A(01)". If this is to be displayed in a table and a meaningful column header for the table is desired, an alias such as the following could be used:

```
$TABLE .... E000004=DADT1A1
```

DES-defined items may be displayed in tables or scatterplots and are computed at the event time in the data record.

Verbatim header statements `"FIRST` and `"MAIN` can be used.

An example of a $DES record, using implicit basic parameters, is given in Appendix VIII.

The following features were added starting with NONMEM V.

The allowable right-hand side quantities include the data items of a data record, symbolized by the labels and synonyms specified in the $INPUT record, and values of THETA, symbolized by THETA(1), THETA(2), etc. If quantities depending only on data items and THETA parameters are computed directly in $DES rather than $PK, improvement in run time is possible. A DES routine may test DOSTIM and DOSREC in a logical expression. It may use them on the right-hand side of an assignment statement. If DOSTIM is a random variable, DOSTIM must not be used in $DES. However, DOSTIM may always be used in a $PK block or PK routine to define a random variable which may be used in the DES routine.

A reserved right-hand variable, `ISFINL` (nmvi), is set to 1 by PREDPP when DES is called after the final advance to an event or non-event time, during Simulation or Copying pass (COMACT>0). One use of ISFINL is for DES to calculate quantities for display in tables or via WRITE statements.

ICALL cannot be used in DES abbreviated code. The generated DES routine tests for ICALL=1, performs PREDPP-required initialization code, and then executes a RETURN statement. Values of `DADT` are not evaluated at ICALL=1, as may happen with a user-written DES routine, so there is no need for a test of ICALL in the abbreviated code.†

With ADVAN9 and ADVAN15, a $DES block may not appear when there are only equilibrium compartments. With ADVAN10 and steady-state doses with constant infusion doses, basic PK parameters KM and VM may be used in the $DES abbreviated code.

NM-TRAN generates the appropriate code in FSUBS. Details are in Guide VI, Chapter VI.

The $ABBREVIATED record has an option that affects the formats of arrays in the generated DES subroutine:

```
$ABBR DES=FULL vs DES=COMPACT
```

† NONMEM V Supplemental Guide Section 82 ($DES Record) states incorrectly that ICALL blocks testing for ICALL values 4 and 5 are permitted.

Normally, there is no reason for the user to supply this option. NM-TRAN will choose the appropriate format. Details are in Guide VI, Appendix IV.

V.V.C.8. $AESINITIAL Record

`$AESINITIAL`
abbreviated code

The complete AES routine can be divided into two parts: code which computes the amounts in the equilibrium compartments at the beginning of an integration interval (these amounts depend on the amounts in the nonequilibrium compartments) and code which computes values for the right sides of the algebraic equations. An abbreviated code for AES is actually divided into two abbreviated codes which correspond to the two parts of the complete AES routine. This record gives an abbreviated code for the first part of the AES routine. The $AES record (see the next section C.9) gives an abbreviated code for the second part. The $AESINITIAL record, along with all its continuation records, is called a $AESINITIAL block .

This record is optional. If it appears, it must be with the first problem specification, and only this problem specfication. If there are no equilibrium compartments, but ADVAN9 or ADVAN15 is used, the record need not appear. It may not appear when the MODEL routine is user-supplied. Implicit basic PK parameters may be used in the $AESINITIAL block (see below) only when the $PK block precedes the $AESINITIAL block.

In order to compute the amounts in the equilibrium compartments at the beginning of an integration interval, the algebraic system must be solved. However, only an approximate solution is needed. If requested, this will be used only as an initial solution, and ADVAN9 will numerically obtain a more precise solution.

The amounts in the equilibrium compartments at the beginning of the integration interval comprise a set of mandatory left-hand quantities. They are symbolized by `A(n)`, where n is a number of an equilibrium compartment. At least one such quantity must be defined.

The variable `INIT` symbolizes an optional left-hand quantity. If it is set to 0, the amounts `A(n)` are regarded as only approximations to the exact amounts. In this case ADVAN9 solves numerically for more precise amounts satisfying the algebraic equations, using the approximations as an initial solution. If the computation of the amounts in the $AESINITIAL block determines these amounts accurately to at least the number of digits given by the option value of the `TOL` option in the $SUBROUTINES record, or by `NRD` or `NRD(1)` as given with the $TOL record, then `INIT` should either not be set, or set to 1.

The allowable right-hand quantities include: nonequilibrium compartment amounts at the beginning of the integration interval, symbolized by `A(n)`, for the amount in the nth *nonequilibrium* compartment; PK parameters (obtained from TRANS1), symbolized by `P(m)`, for the mth parameter; and the time at the beginning of the integration interval, symbolized by `T`. In a population study, where η variables affect the PK parameters, and, therefore, also affect the compartment amounts and (if some η affects an absorption lag parameter) possibly `T` as well, these right-hand quantities should be regarded as random variables. However, η-derivatives are not computed in the generated routine itself. For technical reasons, these right-hand quantities should be regarded as random variables even when the data are single-subject. This means, for example, that these variables may be used in conditional assignment statements subject to the usual restriction on random variables.

Also, a PK-defined item may be used as a right-hand quantity. However, when so used, it becomes a special quantity called an implicit basic PK parameter , and a few special considerations apply:

> Implicit basic PK parameters may be used in the $AESINITIAL block only when the $PK block precedes the $AESINITIAL block.
>
> One should think of elements of the `P` array as including values for explicit basic parameters , i.e. parameters defined in the $PK block in array elements `P(n)`, followed by values for the implicit basic parameters. Therefore, the option value for the `NPARAMETERS` option in the $MODEL

record should be large enough to include all these elements of this extended P array. Note: Explicit basic parameter values are displayable; see section C.5.

Even additional PK parameters become implicit basic PK parameters.

There is a type of pseudo-statement specific to AESINIT abbreviated code. It has the form CALLFL=n. The different permissible values for n imply different calling-protocols for ADVAN9 and ADVAN15, rather than the AES subroutine. CALLFL may be used only when there are only equilibrium compartments and there is no TIME data item. ADVAN9 and ADVAN15 is called by default with every event record. A calling protocol phrase can be used instead of the CALLFL pseudostatement. The phrase must be enclosed in parentheses. Examples of phrases follow each value of CALLFL.

value	calling-protocol
-1	ADVAN9 or 15 called with every event record (default) $AES (EVERY EVENT) $AES (EVERY)
1	ADVAN9 or 15 called only with first event record of the individual record $AES (ONCE PER INDIVIDUAL RECORD) $AES (ONCE/IND.REC.)

If the pseudo-statement does not appear, the value -1 is assumed. This allows ADVAN9 and ADVAN15 to properly function in most situations.

The symbol COM(n) may be used on the left or the right if n refers to a reserved position in MODULE NMPRD4 (see section III.E.3). Variables defined in $INFN and listed in MODULE PRINFN may be used in $AES, on the left and on the right. Variables defined by $ABBREVIATED DECLARE records are also global and may be used in $AES, on the left and on the right.

The abbreviated code may not use certain variables which occur as arguments to the AES subroutine. These names are: IR, DA, DP, and DT.

The verbatim header statement "LAST cannot be used. There is no fourth section of generated code.

Any (non-random) variables defined in the $AESINITIAL block may be used in a $AES block (see next section C.9). A random variable defined in terms of PK parameters, but not amounts, may also be used in a $AES block.

The following features were added starting with NONMEM V.

The allowable right-hand side quantities include the data items of a data record, symbolized by the labels and synonyms specified in the $INPUT record, and values of THETA, symbolized by THETA(1), THETA(2), etc. If quantities depending only on data items and THETA parameters are computed directly in $AES rather than $PK, improvement in run time is possible. A AES routine may test DOSTIM and DOSREC in a logical expression. It may use them on the right-hand side of an assignment statement. If DOSTIM is a random variable, DOSTIM must not be used in $AES. However, DOSTIM may always be used in a $PK block or PK routine to define a random variable which may be used in the AES routine.

A reserved right-hand variable, ISFINL (nmvi), is set to 1 by PREDPP when AES is called after the final advance to an event or non-event time, during Simulation or Copying pass (COMACT>0). One use of ISFINL is for AES to calculate quantities for display in tables or via WRITE statements.

ICALL cannot be used in AES abbreviated code. The generated AES routine tests for ICALL=1, performs PREDPP-required initialization code, and then executes a RETURN statement. Values of A(n) are not evaluated at ICALL=1, as may happen with a user-written AES routine, so there is no need for a

test of ICALL in the abbreviated code.†

† NONMEM V Supplemental Guide Section 84 ($AES Record) states incorrectly that ICALL blocks testing for ICALL values 4 and 5 are permitted.

V.V.C.9. $AES Record

`$AES`
abbreviated code

The complete AES routine can be divided into two parts: code which computes the amounts in the equilibrium compartments at the beginning of the integration interval (these amounts depend on the amounts in the nonequilibrium compartments) and code which computes values for the right sides of the algebraic equations. An abbreviated code for AES is actually divided into two abbreviated codes which correspond to the two parts of the complete AES routine. This record gives an abbreviated code for the second part of the AES routine. The $AESINITIAL record (see section C.8) gives an abbreviated code for the first part. The $AES record, along with all its continuation records, is called a $AES block . This record is optional. If it appears, it must be with the first problem specification, and only this problem specfication. If there are no equilibrium compartments, but ADVAN9 or ADVAN15 is used, the record need not appear. It may not appear when the MODEL routine is user-supplied. Implicit basic PK parameters may be used in the $AES block (see below) only when the $PK block precedes the $AES block.

The mandatory left-hand quantities are values of expressions which when set to 0, constitute the system of algebraic equations describing the equilibrium kinetics. They are symbolized by `E(n)`. The number n must be an equilibrium compartment number. However, there need not be any particular relationship between the subscripts and the equilibrium compartments. At least one such quantity must be computed.

Symbolic differentiation is used to obtain code used in the generated subroutine to compute the elements of the DA, DP, and DT arrays.

The allowable right-hand quantities include: current compartment amounts, symbolized by `A(n)`, for the amount in the nth compartment (including all compartments except the output compartment); PK parameters (obtained from TRANS1), symbolized by `P(m)`, for the mth parameter; and time, symbolized by `T`. In a population study, where η variables affect the PK parameters, and, therefore, also affect the compartment amounts, these right-hand quantities should be regarded as random variables. However, η-derivatives are not computed in the generated routine itself. For technical reasons, these right-hand quantities should be regarded as random variables even when the data are single-subject. a random variable. This means, for example, that these variables may be used in conditional assignment statements subject to the usual restriction on random variables.

Also, a PK-defined item may be used as a right-hand quantity. However, when so used, it becomes a special quantity called an implicit basic PK parameter , and a few special considerations apply:

> Implicit basic PK parameters may be used in the $AES block only when the $PK block precedes the $AES block.
>
> One should think of elements of the `P` array as including values for explicit basic parameters , i.e. parameters defined in the $PK block in array elements `P(n)`, followed by values for the implicit basic parameters. Therefore, the option value for the `NPARAMETERS` option in the $MODEL record should be large enough to include all these elements of this extended `P` array. Note: Explicit basic parameter values are displayable; see section C.5.
>
> Even additional PK parameters become implicit basic PK parameters.

Non-random variables defined in the $AESINITIAL block may be used. A random variable defined in a $AESINITIAL block in terms of PK parameters, but not amounts, may also be used in a $AES block.

The symbol `COM(n)` may be used on the left or the right if n refers to a reserved position in MODULE NMPRD4 (see section III.E.3). Variables defined in $INFN and listed in MODULE PRINFN may be used in $AES, on the left and on the right. Variables defined by $ABBREVIATED DECLARE records are also global and may be used in $AES, on the left and on the right.

The abbreviated code may not use certain variables which occur as arguments to the AES subroutine. These names are: IR, DA, DP, and DT.

Variables which symbolize partial derivatives in the DA array are displayable. They have names E....., where the dots stand for various combinations of 5 or 6 digits 0-9.

Variables which symbolize partial derivatives in the DP array are displayable. They have names F....., where the dots stand for various combinations of 5 or 6 digits 0-9. Variables which symbolize partial derivatives in the DT array are displayable. They have names E....., where the dots stand for various combinations of 5 or 6 digits 0-9.

The verbatim header statements "FIRST and "MAIN cannot be used.

V.V.C.10. $TOL Record

`$TOL`
abbreviated code

This record can be used to specify tolerances (i.e. accuracies) to which compartment amounts are computed by an ADVAN or SS routine that uses numerical techniques (see Guide VI, section VII). Different tolerances can be specified for the compartment amounts in different compartments. Whereas this record may be needed with ADVAN9, ADVAN13, ADVAN14, ADVAN15, which support compartment-specific tolerances, it is not needed with all other ADVAN and SS routines, where only a single tolerance applying to all compartments can be specified. In these other cases use of the `TOL`, `ATOL`, `SSTOL`, `SSATOL` options on the $SUBROUTINES record suffices.

This record is optional. If it appears, it must be with the first problem specification, and only this problem specfication. It cannot be used when the `TOL` option is used in the $SUBROUTINES record.

The abbreviated code is very limited and brief; it is not like any other. The only statements that are permitted have one of the two forms

`NRD`=m
`NRD` (i) =m

With NONMEM 7.4, additional statements are permitted:

`ANRD` (i) =m
`NRDC` (i) =m
`ANRDC` (i) =m

where
i is a compartment number (excluding the output compartment number),
and m is the number of accurate digits required in the computation of
the drug amounts in compartment i.
Form 1 is a shorter way of writing form 2 with i = 1.

Once the tolerance for a compartment is explicitly specified, it cannot be respecified. A tolerance for a given compartment need not be specified; see below. The order of the statements is immaterial.

If $i > 1$ is the least compartment number such that the tolerance for compartment i is not specified, then for all $j \geq i$, the tolerance for compartment j is the tolerance explicitly specified for compartment $i - 1$. This implies that if the only statement used is `NRD`=m, the number of accurate digits required in the computation of all compartmental drug amounts is m. In this case, as a rule of thumb, one should begin by taking m to be $n + 1$ or $n + 2$, or with double precision, perhaps to $n + 2$ or $n + 3$, where n is the option value with the `SIGDIGITS` option on the $ESTIMATION record. If one succeeds with this setting, one might try increasing m slightly.

If ADVAN9 and SS9 are used together, and if different tolerances are specified for different compartments, only that tolerance specified as SSTOL (or compartment 1 if SSTOL is not specified) is used for the computation of steady-state compartmental amounts. If any ADVAN other than ADVAN9 is used, or if SS6 is used, the tolerances specified for compartments with numbers larger than 1 are ignored, and again only that tolerance specified as SSTOL (or compartment 1, if SSTOL is not specified) is used for the computation of all compartmental amounts.

A user-supplied TOL routine also allows values to be assigned differently for each compartment. With NONMEM 7.4, values may be assigned differently for each NONMEM step and for each problem. See

Guide VI PREDPP, Chapter VI, Section D.B. See also the TOL option of the $ESTIMATION and $COVARIANCE records.

V.V.C.11. $INFN Record (nmvi)

`$INFN`
abbreviated code

This record gives an abbreviated code for the INFN routine. It, along with all its continuation records is called an $INFN block .

$INFN abbreviated code allows the user to carry out any of his own programmed computations at both the beginning of a problem and again, at the ending of the problem. This record is optional. If it appears, it must be with the first problem specification, and only this problem specification.

If there is abbreviated code in the $PK or $ERROR blocks that test for ICALL=0, ICALL=1, or ICALL=3, this code is moved by NM-TRAN to the INFN routine as if it had been coded explicitly as part of an $INFN block. Such code is called $PK-INFN and $ERROR-INFN, respectively. The order of code in the generated INFN routine is:

1) "FIRST code from $INFN
2) "MAIN code from $INFN
3) abbreviated code from $INFN
4) $PK-INFN blocks
5) $ERROR-INFN blocks
6) "LAST code from $INFN

The following remarks apply to both explicitly coded $INFN blocks and to $PK-INFN and $ERROR-INFN blocks.

A $INFN block can be used to perform all the functions described in a $PRED block at ICALL values 0, 1, 3, as described in Chapter IV Section D. This functionality is also described in Guide VI Chapter VI.A See the help item for Initialization-Finalization block

Here is a brief summary.

There are no mandatory left-hand quantities.

No variable that has a reserved meaning in another block of abbreviated code may be defined on the left in $INFN.

Optional left-hand variables:

INFN-defined variables (i.e., variables defined first on the left in $INFN or in $PK-INFN or $ERROR-INFN blocks) are permited. These are listed in MODULE PRINFN rather than NMPRD4. MODULE PRINFN is declared in all generated routines PK, ERROR, etc., and the variables can be used on the right or left in these routines. INFN-defined variables are not initialized or modified by NONMEM or PREDPP. Hence the variables in the MODULE may be initialized or modified at ICALL values 0 or 1 or 3, and will retain whatever values they are given. These variables cannot be displayed in tables or scatterplots. If they are to be displayed, WRITE statements can be used, or their values can be assigned to variables that are listed in MODULE NMPRD4. The implementation of MODULE PRINFN is discussed in Chapter II Section II.B and II.E.

Right side variables:

INFN-defined variables that appeared earlier as left-hand side quantities.

Previously defined PK- or ERROR-defined items that are not INFN-defined. They are listed in NMPRD4 as usual. This MODULE is declared in INFN. Typically, such variables are used at ICALL=3 during

calls to PASS or a pass through the dataset using DOWHILE(DATA). They may be used only on the right side and may not be recomputed. They should also be displayed in tables or scatters because items in NMPRD4 that are *not* displayed have the value 0 at ICALL=3.

Data item labels specified on the $INPUT Record may be used on the left and on the right. This means that the NONMEM data set may be changed. Data transgeneration may take place at ICALL values 0, 1, 3. At each call, the user has read-write access to his data via use of the NONMEM utility routine PASS described in Guide II. This may be done more easily with the DOWHILE(DATA) statement of abbreviated coe. It is discussed in Chapter IV.J.3. Thus data can be transgenerated, and additional data items can be produced at both the beginning and ending of a problem. Since the finalization call actually occurs before the Table and Scatterplot Steps, new data items generated by INFN at this call can be tabled and scatterplotted.

There are several reserved variables and array elements symbolizing special right-hand quantities. They include special diagnostic items and counter variables. They are listed in Chapter IV.E.

Appendix I. NM-TRAN Control Records

$SIZES

$SUPERPROBLEM

$PROBLEM text

$INPUT item$_1$ item$_2$ item$_3$ ···

$INDEX [label$_1$|value$_1$] [label$_2$|value$_2$] [label$_3$|value$_3$] ...

$CONTR DATA= ([label$_1$|0] [label$_2$|0] [label$_3$|0])

$DATA [filename|*] [(format)] [IGNORE=c$_1$] [NULL=c$_2$] [**NOWIDE**|WIDE] [CHECKOUT]
[RECORDS=n$_1$] [LRECL=n$_2$] [**NOREWIND**|REWIND]

$SUBROUTINES [subname$_1$ = name$_1$] [subname$_2$ = name$_2$] ...
[SUBROUTINES=kind]

$ABBREVIATED [COMRES=n$_1$] [COMSAV=n$_2$] [DERIV2=NO] [DERIV2=NOCOMMON]

$PRED
the abbreviated code

$THETA value$_1$ [value$_2$] [value$_3$] ...
[NUMBERPOINTS=n] [**ABORT**|NOABORT]

$OMEGA [**DIAGONAL(n)**|BLOCK(n)|BLOCK(n) SAME|BLOCK SAME]
[[value$_1$] [value$_2$] [value$_3$] ... [FIXED]]

$SIGMA [**DIAGONAL(n)**|BLOCK(n)|BLOCK(n) SAME|BLOCK SAME]
[[value$_1$] [value$_2$] [value$_3$] ... [FIXED]]

$MSFI filename [**NORESCALE**|RESCALE] [NPOPETAS=n]

$SIMULATION (seed1 [seed2] [**NORMAL**|UNIFORM] [NEW]) ...
[SUBPROBLEMS=n] [ONLYSIMULATION] [OMITTED]

$ESTIMATION [METHOD=kind] [**NOINTERACTION**|INTERACTION] [**NOLAPLACIAN**|LAPLACIAN]
[**NOPOSTHOC**|POSTHOC] [SIGDIGITS=n$_1$] [MAXEVALS=n$_2$] [PRINT=n$_3$]
[**ABORT**|NOABORT] [MSFO=filename] [**NOREPEAT**|REPEAT] [OMITTED]

$COVARIANCE [SPECIAL] [MATRIX=c] [PRINT=[E][R][S]
[**CONDITIONAL**|UNCONDITIONAL] [OMITTED]

$TABLE [list1] [BY list2] [**PRINT**|NOPRINT] [FILE=filename] [NOHEADER|ONEHEADER]
[**UNCONDITIONAL**|CONDITIONAL] [OMITTED]

$SCATTERPLOT list1 VS list2 [BY list3]

[UNIT] [ORD0] [FROM n_1] [TO n_2]
[**UNCONDITIONAL**|CONDITIONAL] [OMITTED]

$INCLUDE filename [n]

Appendix II. NM-TRAN Data Set -- Example

Here is a NM-TRAN data set which NM-TRAN translates into a NONMEM data set shown in Appendix III. The NONMEM data set in Appendix III is identical to that shown in Figure 75 of NONMEM Users Guide, Part I, except for spacing between fields. This NM-TRAN data set is recorded on the NONMEM distribution medium as file THEO; see Guide III.

```
1      4.02      0.         .74     79.6
1       .        0.25      2.84      .
1       .        0.57      6.57      .
1       .        1.12     10.5       .
1       .        2.02      9.66      .
1       .        3.82      8.58      .
1       .        5.1       8.36      .
1       .        7.03      7.47      .
1       .        9.05      6.89      .
1       .       12.12      5.94      .
1       .       24.37      3.28      .
2      4.4       0.        0.       72.4
2       .         .27      1.72      .
2       .         .52      7.91      .
2       .        1.        8.31      .
2       .        1.92      8.33      .
2       .        3.5       6.85      .
2       .        5.02      6.08      .
2       .        7.03      5.4       .
2       .        9.        4.55      .
2       .       12.        3.01      .
2       .       24.3        .90      .
3      4.53      0.        0.       70.5
3       .         .27      4.4       .
3       .         .58      6.9       .
3       .        1.02      8.2       .
3       .        2.02      7.8       .
3       .        3.62      7.5       .
3       .        5.08      6.2       .
3       .        7.07      5.3       .
3       .        9.        4.9       .
3       .       12.15      3.7       .
3       .       24.17      1.05      .
4      4.4       0.        0.       72.7
4       .         .35      1.89      .
4       .         .6       4.6       .
4       .        1.07      8.6       .
4       .        2.13      8.38      .
4       .        3.5       7.54      .
4       .        5.02      6.88      .
4       .        7.02      5.78      .
4       .        9.02      5.33      .
4       .       11.98      4.19      .
```

```
4       .       24.65      1.15       .
5      5.86      0.        0.        54.6
5       .         .3       2.02       .
5       .         .52      5.63       .
5       .        1.       11.4        .
5       .        2.02      9.33       .
5       .        3.5       8.74       .
5       .        5.02      7.56       .
5       .        7.02      7.09       .
5       .        9.1       5.9        .
5       .       12.        4.37       .
5       .       24.35      1.57       .
6      4.        0.        0.        80.
6       .         .27      1.29       .
6       .         .58      3.08       .
6       .        1.15      6.44       .
6       .        2.03      6.32       .
6       .        3.57      5.53       .
6       .        5.        4.94       .
6       .        7.        4.02       .
6       .        9.22      3.46       .
6       .       12.1       2.78       .
6       .       23.85       .92       .
7      4.95      0.         .15      64.6
7       .         .25       .85       .
7       .         .5       2.35       .
7       .        1.02      5.02       .
7       .        2.02      6.58       .
7       .        3.48      7.09       .
7       .        5.        6.66       .
7       .        6.98      5.25       .
7       .        9.        4.39       .
7       .       12.05      3.53       .
7       .       24.22      1.15       .
8      4.53      0.        0.        70.5
8       .         .25      3.05       .
8       .        0.52      3.05       .
8       .         .98      7.31       .
8       .        2.02      7.56       .
8       .        3.53      6.59       .
8       .        5.05      5.88       .
8       .        7.15      4.73       .
8       .        9.07      4.57       .
8       .       12.1       3.         .
8       .       24.12      1.25       .
9      3.1         .0        .0       86.4
9       .         .3       7.37       .
9       .         .63      9.03       .
9       .        1.05      7.14       .
9       .        2.02      6.33       .
```

```
 9       .        3.53      5.66       .
 9       .        5.02      5.67       .
 9       .        7.17      4.24       .
 9       .        8.8       4.11       .
 9       .       11.6       3.16       .
 9       .       24.43      1.12       .
10      5.5       0.         .24     58.2
10       .         .37      2.89       .
10       .         .77      5.22       .
10       .        1.02      6.41       .
10       .        2.05      7.83       .
10       .        3.55     10.21       .
10       .        5.05      9.18       .
10       .        7.08      8.02       .
10       .        9.38      7.14       .
10       .       12.1       5.68       .
10       .       23.7       2.42       .
11      4.92      0.         0.        65.
11       .         .25      4.86       .
11       .         .5       7.24       .
11       .         .98      8.         .
11       .        1.98      6.81       .
11       .        3.6       5.87       .
11       .        5.02      5.22       .
11       .        7.03      4.45       .
11       .        9.03      3.62       .
11       .       12.12      2.69       .
11       .       24.08       .86       .
12      5.3       0.         0.        60.5
12       .         .25      1.25       .
12       .         .5       3.96       .
12       .        1.        7.82       .
12       .        2.        9.72       .
12       .        3.52      9.75       .
12       .        5.07      8.57       .
12       .        7.07      6.59       .
12       .        9.03      6.11       .
12       .       12.05      4.57       .
12       .       24.15      1.17       .
```

Appendix III. NM-TRAN Outputs -- Example

The NM-TRAN outputs obtained from using the NM-TRAN control stream listed in chapter I, along with the NM-TRAN data set listed in Appendix II, are given on the following pages of this appendix. Some code that is extraneous to the control stream was removed from SUBROUTINE PRED for the purposes of simplicity.

Data Set - FDATA

```
1  4.02    0.   .74  79.6
1        0.25  2.84
1        0.57  6.57
1        1.12  10.5
1        2.02  9.66
1        3.82  8.58
1         5.1  8.36
1        7.03  7.47
1        9.05  6.89
1       12.12  5.94
1       24.37  3.28
2   4.4    0.    0.  72.4
2         .27  1.72
2         .52  7.91
2          1.  8.31
2        1.92  8.33
2         3.5  6.85
2        5.02  6.08
2        7.03   5.4
2          9.  4.55
2         12.  3.01
2        24.3   .90
3  4.53    0.    0.  70.5
3         .27   4.4
3         .58   6.9
3        1.02   8.2
3        2.02   7.8
3        3.62   7.5
3        5.08   6.2
3        7.07   5.3
3          9.   4.9
3       12.15   3.7
3       24.17  1.05
4   4.4    0.    0.  72.7
4         .35  1.89
4          .6   4.6
4        1.07   8.6
4        2.13  8.38
4         3.5  7.54
4        5.02  6.88
4        7.02  5.78
4        9.02  5.33
4       11.98  4.19
4       24.65  1.15
5  5.86    0.    0.  54.6
5         .3  2.02
5         .52  5.63
5          1.  11.4
```

```
5         2.02  9.33
5          3.5  8.74
5         5.02  7.56
5         7.02  7.09
5          9.1   5.9
5          12.  4.37
5        24.35  1.57
6    4.     0.    0.   80.
6          .27  1.29
6          .58  3.08
6         1.15  6.44
6         2.03  6.32
6         3.57  5.53
6           5.  4.94
6           7.  4.02
6         9.22  3.46
6         12.1  2.78
6        23.85   .92
7  4.95    0.    .15  64.6
7          .25   .85
7           .5  2.35
7         1.02  5.02
7         2.02  6.58
7         3.48  7.09
7           5.  6.66
7         6.98  5.25
7           9.  4.39
7        12.05  3.53
7        24.22  1.15
8  4.53    0.    0.  70.5
8          .25  3.05
8         0.52  3.05
8          .98  7.31
8         2.02  7.56
8         3.53  6.59
8         5.05  5.88
8         7.15  4.73
8         9.07  4.57
8         12.1    3.
8        24.12  1.25
9   3.1    .0    .0  86.4
9          .3  7.37
9          .63  9.03
9         1.05  7.14
9         2.02  6.33
9         3.53  5.66
9         5.02  5.67
9         7.17  4.24
9          8.8  4.11
9         11.6  3.16
```

```
 9       24.43  1.12
10   5.5    0.   .24  58.2
10          .37  2.89
10          .77  5.22
10         1.02  6.41
10         2.05  7.83
10         3.55 10.21
10         5.05  9.18
10         7.08  8.02
10         9.38  7.14
10         12.1  5.68
10         23.7  2.42
11  4.92    0.    0.   65.
11          .25  4.86
11           .5  7.24
11          .98    8.
11         1.98  6.81
11          3.6  5.87
11         5.02  5.22
11         7.03  4.45
11         9.03  3.62
11       12.12  2.69
11       24.08   .86
12   5.3    0.    0.  60.5
12          .25  1.25
12           .5  3.96
12           1.  7.82
12           2.  9.72
12         3.52  9.75
12         5.07  8.57
12         7.07  6.59
12         9.03  6.11
12       12.05  4.57
12       24.15  1.17
```

Control Stream - FCON

```
FILE    FSTREAM
PROB    THEOPHYLLINE   POPULATION DATA
DATA       1   0 132   5   0
ITEM       1   4   0   0   1   0   0   0   0   0   0   0   0   0   0
LABL                   ID,                DOSE,                TIME
                       CP,                  WT
FORM
(5E6.0)
STRC       3   3   1   0   0   0   1   1   0
STRC       1   3
THCN       1   0   0   0
THTA     3.000000000000000E+00, 8.000000000000000E-02, 4.000000000000000E-02
LOWR     1.000000000000000E-01, 8.000000000000000E-03, 4.000000000000000E-03
UPPR     5.000000000000000E+00, 5.000000000000000E-01, 9.000000000000000E-01
BLST     6.000000000000000E+00, 5.000000000000000E-03, 3.000000000000000E-01
         2.000000000000000E-04, 6.000000000000000E-03, 4.000000000000000E-01
DIAG     4.000000000000000E-01
ESTM       0 450   3   5   0   0   0   0   0   0   0   0   0   0   0   0   0   0
           0   0   0   0  -1  -1
COVR       0   0   0   0   0   0   1   0   0
COVT      -1  -1  -1  -1   0   0
CPAR
TABL       1   1
TABL       4     1 0     2 0     5 0     3 0
SCAT       1   2
SCAT           3       7       1       1       0       0       0       0       0
               0       0       0
SCAT           3       8       1       1       0       0       0       0       0
               0       0       0
```

File Stream - FSTREAM

```
DATA    FDATA
****
```

Report - FREPORT

```
NM-TRAN VERSION II LEVEL 1.0
GENERATED DP SUBROUTINES:
PRED
NONMEM SUBROUTINES: ALL
```

Generated and User-Supplied Subroutines - FSUBS

```
      SUBROUTINE PRED (ICALL,NEWIND,THETA,DATREC,INDXS,F,G,H)
      USE SIZES,     ONLY: DPSIZE,ISIZE
      USE PRDIMS,    ONLY: GPRD,HPRD
      USE NMPRD_REAL,ONLY: ETA,EPS
      IMPLICIT REAL(KIND=DPSIZE) (A-Z)
      REAL(KIND=DPSIZE) :: DATREC
      SAVE
      INTEGER(KIND=ISIZE) :: ICALL,NEWIND,INDXS
      REAL(KIND=DPSIZE) :: G(GPRD,*),H(HPRD,*)
      DIMENSION :: THETA(*),DATREC(*),INDXS(*)
      DOSE=DATREC(002)
      TIME=DATREC(003)
      WT=DATREC(005)
      IF(DOSE /= 0.D0)THEN
      DS=DOSE*WT
      W=WT
      ENDIF
      KA=THETA(001)+ETA(001)
      KE=THETA(002)+ETA(002)
      CL=THETA(003)*W+ETA(003)
      B00005=-KE*TIME
      B00006=-KA*TIME
      B00007=DEXP(B00005)
      B00008=DEXP(B00006)
      D=B00007-B00008
!                     A00034 = DERIVATIVE OF B00005 W.R.T. ETA(002)
      A00034=-TIME
!                     A00035 = DERIVATIVE OF B00006 W.R.T. ETA(001)
      A00035=-TIME
!                     A00036 = DERIVATIVE OF B00007 W.R.T. ETA(002)
      A00036=B00007*A00034
!                     A00037 = DERIVATIVE OF B00008 W.R.T. ETA(001)
      A00037=B00008*A00035
!                     A00039 = DERIVATIVE OF D W.R.T. ETA(001)
      A00039=-A00037
      B00009=KA-KE
      E=CL*B00009
!                     A00041 = DERIVATIVE OF B00009 W.R.T. ETA(002)
      A00041=-1.D0
!                     A00043 = DERIVATIVE OF E W.R.T. ETA(002)
      A00043=CL*A00041
      F=DS*KE*KA/E*D
      B00010=DS*KA/E*D
      B00011=DS*KE/E*D
      B00012=-DS*KE*KA/E/E*D
!                     A00047 = DERIVATIVE OF F W.R.T. ETA(001)
      A00047=B00012*CL+B00011
!                     A00048 = DERIVATIVE OF F W.R.T. ETA(002)
```

```
      A00048=B00012*A00043+B00010
!                     A00049 = DERIVATIVE OF F W.R.T. ETA(003)
      A00049=B00012*B00009
      B00013=DS*KE*KA/E
!                     A00050 = DERIVATIVE OF F W.R.T. ETA(001)
      A00050=B00013*A00039+A00047
!                     A00051 = DERIVATIVE OF F W.R.T. ETA(002)
      A00051=B00013*A00036+A00048
      Y=F+EPS(001)
!                     A00052 = DERIVATIVE OF Y W.R.T. ETA(002)
      A00052=A00051
!                     A00053 = DERIVATIVE OF Y W.R.T. ETA(001)
      A00053=A00050
!                     A00054 = DERIVATIVE OF Y W.R.T. ETA(003)
      A00054=A00049
!                     C00031 = DERIVATIVE OF Y W.R.T. EPS(001)
      C00031=1.D0
      G(001,1)=A00053
      G(002,1)=A00052
      G(003,1)=A00054
      H(001,1)=C00031
      F=Y
      RETURN
      END
```

Appendix IV. Another Example

Figures 1 and 2 of NONMEM Users Guide, Part I, show a PRED routine and a NONMEM control stream (with embedded data), respectively. The NONMEM outputs resulting from Figures 1 and 2 are shown in Figures 3-18 of Part I. NM-TRAN inputs and outputs are given on the following pages of this appendix. The inputs correspond to Figures 1 and 2. The NONMEM outputs resulting from the NM-TRAN outputs and corresponding to those shown in Figures 4-18 are exactly the same as those shown in Figures 4-18. The NONMEM output corresponding to that shown in Figures 3a-b, the problem summary, is a little different (in part because NM-TRAN generates ID data items).

NM-TRAN Data Set

```
      320        .27      1.71
      320        .52      7.91
      320       1.        8.31
      320       1.92      8.33
      320       3.5       6.85
      320       5.02      6.08
      320       7.03      5.4
      320       9.        4.55
      320      12.        3.01
      320      24.3        .90
```

NM-TRAN Control Stream

```
$PROB    SIMPLE NONLINEAR REGRESSION OF CP VS TIME DATA FROM ONE SUBJECT
$INPUT      DOSE TIME CP=DV
$DATA       DATA

$PRED
;THETA(1)=ABSORPTION RATE CONSTANT (1/HR)
;THETA(2)=ELIMINATION RATE CONSTANT (1/HR)
;THETA(3)=VOLUME OF DISTRIBUTION (LITERS)
   D=EXP(-THETA(2)*TIME)-EXP(-THETA(1)*TIME)
   E=THETA(3)*(THETA(1)-THETA(2))
   F=DOSE*THETA(1)/E*D
   Y=F+ETA(1)

$THETA  (.4,1.7,7.) (.025,.102,.4) (10,29,80)

$EST     MAXEVAL=240 SIGDIGITS=4 PRINT=2
$COV
$TABLE          TIME
$SCAT           (CP PRED RES) VS TIME
$SCAT           PRED VS CP  UNIT
```

NONMEM Data Set - FDATA

```
  320  .27 1.71 1
  320  .52 7.91 2
  320   1. 8.31 1
  320 1.92 8.33 2
  320  3.5 6.85 1
  320 5.02 6.08 2
  320 7.03  5.4 1
  320   9. 4.55 2
  320  12. 3.01 1
  320 24.3  .90 2
```

NONMEM Control Stream - FCON

```
FILE    FSTREAM
PROB    SIMPLE NONLINEAR REGRESSION OF CP VS TIME DATA FROM ONE SUBJECT
DATA       1   0  10   4   0
ITEM       4   3   0   0   1   0   0   0   0   0   0
LABL    DOSE    TIME      CP    .ID.
FORM
(3E5.0,1F2.0)
STRC       3   1   0   0   0   1   0   0   0
THCN       1   0   0
THTA         1.7    .102      29
LOWR          .4    .025      10
UPPR          7.      .4      80
DIAG   2
ESTM       0 240   4   2   0   0   0   0   0   0   0
COVR       0   0   0   0   0
TABL       1   1   0   0
TABL       1   2   0
SCAT       1   4
SCAT       2   3   0   0   0   0       0       0   0
SCAT       2   5   0   0   0   0       0       0   0
SCAT       2   6   0   0   0   0       0       0   0
SCAT       3   5   0   0   0   1       0       0   0
```

Generated and User-Supplied Subroutines - FSUBS

The NONMEM VI versions are shown. With NONMEM 7 and higher, MODULES are used rather than COMMONS.

```
      SUBROUTINE PRED (ICALL,NEWIND,THETA,DATREC,INDXS,F,G,H)
      IMPLICIT DOUBLE PRECISION (A-Z)
      REAL DATREC
      SAVE
      INTEGER ICALL,NEWIND,INDXS
      DIMENSION THETA(*),DATREC(*),INDXS(*),G(10,*),H(10,*)
      DIMENSION ETA(10)
      COMMON/ROCM12/MSEC
      INTEGER MSEC
      COMMON/NMPRD4/D,E,Y,A00011,BBBBBB(0996)
      IF (ICALL.EQ.4) THEN
       CALL SIMETA(ETA)
      ELSE
      IF (NEWIND.NE.2) THEN
       ETA(01)=0.D0
      ENDIF
      ENDIF
      DOSE=DATREC(01)
      TIME=DATREC(02)
      B00001=-THETA(02)*TIME
      B00002=-THETA(01)*TIME
      B00003=DEXP(B00001)
      B00004=DEXP(B00002)
      D=B00003-B00004
      B00005=THETA(01)-THETA(02)
      E=THETA(03)*B00005
      F=DOSE*THETA(01)/E*D
      Y=F+ETA(01)
C                     A00011 = DERIVATIVE OF Y W.R.T. ETA(01)
      A00011=1.D0
      G(01,1)=A00011
      F=Y
      RETURN
      END
```

NONMEM File Stream - FSTREAM

```
DATA    FDATA
****
```

NM-TRAN Report - FREPORT

```
 NM-TRAN VERSION 7.3.0
 GENERATED DP SUBROUTINES:
 PRED
 NONMEM SUBROUTINES: ALL
```

Appendix V. NM-TRAN Control Records with PREDPP

```
$PROBLEM  text

$INPUT item1 item2 item3 ···

$BIND value1 value2 ... valuen

$INDEX [label1|value1] [label2|value2] [label3|value3] ...

$CONTR    DATA= ([label1|0] [label2|0] [label3|0])

$DATA [filename|*] [(format)] [IGNORE=c1] [NULL=c2] [NOWIDE|WIDE] [CHECKOUT]
      [RECORDS=n1] [LRECL=n2] [NOREWIND|REWIND]

$SUBROUTINES [subname1 = name1] [subname2 = name2] ...
             [SUBROUTINES=kind] [TOL=n1]

$MODEL [NCOMPARTMENTS=n1] [NEQUILIBRIUM=n2] [NPARAMETERS=n3]
       [COMPARTMENT=([name] [attribute1] [attribute2] ...)] ...
       [LINK compnamea [TO|AND] compnameb BY k [l]] ...

$ABBREVIATED  [COMRES=n1] [COMSAV=n2] [DERIV2=NO] [DERIV2=NOCOMMON]

$PK
      abbreviated code

$ERROR
      abbreviated code

$DES
      abbreviated code

$AESINITIAL
      abbreviated code

$AES
      abbreviated code

$TOL
      abbreviated code

$THETA  value1  [value2]  [value3] ...
        [NUMBERPOINTS=n] [ABORT|NOABORT]

$OMEGA  [DIAGONAL(n)|BLOCK(n)|BLOCK(n) SAME|BLOCK SAME]
        [[value1]  [value2]  [value3] ... [FIXED]]

$SIGMA  [DIAGONAL(n)|BLOCK(n)|BLOCK(n) SAME|BLOCK SAME]
        [[value1]  [value2]  [value3] ... [FIXED]]
```

```
$MSFI   filename [NORESCALE|RESCALE] [NPOPETAS=n]

$SIMULATION  (seed1 [seed2] [NORMAL|UNIFORM] [NEW]) ...
             [SUBPROBLEMS=n] [ONLYSIMULATION] [OMITTED]

$ESTIMATION  [METHOD=kind] [NOINTERACTION|INTERACTION] [NOLAPLACIAN|LAPLACIAN]
             [NOPOSTHOC|POSTHOC] [SIGDIGITS=n1] [MAXEVALS=n2] [PRINT=n3]
             [ABORT|NOABORT] [MSFO=filename] [NOREPEAT|REPEAT] [OMITTED]

$COVARIANCE  [SPECIAL] [MATRIX=c] [PRINT=[E][R][S]
             [CONDITIONAL|UNCONDITIONAL] [OMITTED]

$TABLE  [list1] [BY list2] [PRINT|NOPRINT] [FILE=filename] [NOHEADER|ONEHEADER]
        [UNCONDITIONAL|CONDITIONAL] [OMITTED]

$SCATTERPLOT  list1 VS list2 [BY list3]
              [UNIT] [ORD0] [FROM n1] [TO n2]
              [UNCONDITIONAL|CONDITIONAL] [OMITTED]
```

Appendix VI. NM-TRAN Data Set with PREDPP -- Example

Here is a NM-TRAN data set which NM-TRAN translates into a NONMEM data set shown in Appendix VII. The NONMEM data set in Appendix VII is identical to that shown in Appendix II of NONMEM Users Guide, Part VI, except for spacing between columns. This NM-TRAN data set is recorded on the NONMEM distribution medium; see Guide III.

```
1      4.02      0.          .        79.6
1       .        0.         .74        .
1       .        0.25      2.84        .
1       .        0.57      6.57        .
1       .        1.12     10.5         .
1       .        2.02      9.66        .
1       .        3.82      8.58        .
1       .        5.1       8.36        .
1       .        7.03      7.47        .
1       .        9.05      6.89        .
1       .       12.12      5.94        .
1       .       24.37      3.28        .
2      4.4       0.          .        72.4
2       .        0.        0.          .
2       .         .27      1.72        .
2       .         .52      7.91        .
2       .        1.        8.31        .
2       .        1.92      8.33        .
2       .        3.5       6.85        .
2       .        5.02      6.08        .
2       .        7.03      5.4         .
2       .        9.        4.55        .
2       .       12.        3.01        .
2       .       24.3        .90        .
3      4.53      0.          .        70.5
3       .        0.        0.          .
3       .         .27      4.4         .
3       .         .58      6.9         .
3       .        1.02      8.2         .
3       .        2.02      7.8         .
3       .        3.62      7.5         .
3       .        5.08      6.2         .
3       .        7.07      5.3         .
3       .        9.        4.9         .
3       .       12.15      3.7         .
3       .       24.17      1.05        .
4      4.4       0.          .        72.7
4       .        0.        0.          .
4       .         .35      1.89        .
4       .         .6       4.6         .
4       .        1.07      8.6         .
4       .        2.13      8.38        .
4       .        3.5       7.54        .
```

```
4       .        5.02      6.88       .
4       .        7.02      5.78       .
4       .        9.02      5.33       .
4       .       11.98      4.19       .
4       .       24.65      1.15       .
5      5.86      0.         .        54.6
5       .        0.        0.         .
5       .         .3       2.02       .
5       .         .52      5.63       .
5       .        1.       11.4        .
5       .        2.02      9.33       .
5       .        3.5       8.74       .
5       .        5.02      7.56       .
5       .        7.02      7.09       .
5       .        9.1       5.9        .
5       .       12.        4.37       .
5       .       24.35      1.57       .
6      4.        0.         .        80.
6       .        0.        0.         .
6       .         .27      1.29       .
6       .         .58      3.08       .
6       .        1.15      6.44       .
6       .        2.03      6.32       .
6       .        3.57      5.53       .
6       .        5.        4.94       .
6       .        7.        4.02       .
6       .        9.22      3.46       .
6       .       12.1       2.78       .
6       .       23.85       .92       .
7      4.95      0.         .        64.6
7       .        0.        .15        .
7       .         .25       .85       .
7       .         .5       2.35       .
7       .        1.02      5.02       .
7       .        2.02      6.58       .
7       .        3.48      7.09       .
7       .        5.        6.66       .
7       .        6.98      5.25       .
7       .        9.        4.39       .
7       .       12.05      3.53       .
7       .       24.22      1.15       .
8      4.53      0.         .        70.5
8       .        0.        0.         .
8       .         .25      3.05       .
8       .        0.52      3.05       .
8       .         .98      7.31       .
8       .        2.02      7.56       .
8       .        3.53      6.59       .
8       .        5.05      5.88       .
8       .        7.15      4.73       .
```

```
 8       .        9.07      4.57       .
 8       .       12.1       3.         .
 8       .       24.12      1.25       .
 9      3.1        .0        .        86.4
 9       .         .0       0.         .
 9       .         .3       7.37       .
 9       .         .63      9.03       .
 9       .        1.05      7.14       .
 9       .        2.02      6.33       .
 9       .        3.53      5.66       .
 9       .        5.02      5.67       .
 9       .        7.17      4.24       .
 9       .        8.8       4.11       .
 9       .       11.6       3.16       .
 9       .       24.43      1.12       .
10      5.5       0.         .        58.2
10       .        0.         .24       .
10       .         .37      2.89       .
10       .         .77      5.22       .
10       .        1.02      6.41       .
10       .        2.05      7.83       .
10       .        3.55     10.21       .
10       .        5.05      9.18       .
10       .        7.08      8.02       .
10       .        9.38      7.14       .
10       .       12.1       5.68       .
10       .       23.7       2.42       .
11      4.92      0.         .        65.
11       .        0.        0.         .
11       .         .25      4.86       .
11       .         .5       7.24       .
11       .         .98      8.         .
11       .        1.98      6.81       .
11       .        3.6       5.87       .
11       .        5.02      5.22       .
11       .        7.03      4.45       .
11       .        9.03      3.62       .
11       .       12.12      2.69       .
11       .       24.08       .86       .
12      5.3       0.         .        60.5
12       .        0.        0.         .
12       .         .25      1.25       .
12       .         .5       3.96       .
12       .        1.        7.82       .
12       .        2.        9.72       .
12       .        3.52      9.75       .
12       .        5.07      8.57       .
12       .        7.07      6.59       .
12       .        9.03      6.11       .
12       .       12.05      4.57       .
```

```
        12        .        24.15      1.17        .
```

Appendix VII. NM-TRAN Outputs with PREDPP -- Example

The NM-TRAN outputs obtained from using the NM-TRAN control stream listed in chapter V, along with the NM-TRAN data set listed in Appendix VI, are given on the following pages of this appendix.

Data Set - FDATA

```
1  4.02    0.         79.6 1 1
1          0.   .74       0 0
1        0.25  2.84       0 0
1        0.57  6.57       0 0
1        1.12  10.5       0 0
1        2.02  9.66       0 0
1        3.82  8.58       0 0
1         5.1  8.36       0 0
1        7.03  7.47       0 0
1        9.05  6.89       0 0
1       12.12  5.94       0 0
1       24.37  3.28       0 0
2   4.4    0.         72.4 1 1
2          0.    0.       0 0
2         .27  1.72       0 0
2         .52  7.91       0 0
2          1.  8.31       0 0
2        1.92  8.33       0 0
2         3.5  6.85       0 0
2        5.02  6.08       0 0
2        7.03   5.4       0 0
2          9.  4.55       0 0
2         12.  3.01       0 0
2        24.3   .90       0 0
3  4.53    0.         70.5 1 1
3          0.    0.       0 0
3         .27   4.4       0 0
3         .58   6.9       0 0
3        1.02   8.2       0 0
3        2.02   7.8       0 0
3        3.62   7.5       0 0
3        5.08   6.2       0 0
3        7.07   5.3       0 0
3          9.   4.9       0 0
3       12.15   3.7       0 0
3       24.17  1.05       0 0
4   4.4    0.         72.7 1 1
4          0.    0.       0 0
4         .35  1.89       0 0
4          .6   4.6       0 0
4        1.07   8.6       0 0
4        2.13  8.38       0 0
4         3.5  7.54       0 0
4        5.02  6.88       0 0
4        7.02  5.78       0 0
4        9.02  5.33       0 0
4       11.98  4.19       0 0
4       24.65  1.15       0 0
```

```
5  5.86    0.         54.6 1 1
5          0.    0.        0 0
5           .3  2.02        0 0
5          .52  5.63        0 0
5           1.  11.4        0 0
5         2.02  9.33        0 0
5          3.5  8.74        0 0
5         5.02  7.56        0 0
5         7.02  7.09        0 0
5          9.1   5.9        0 0
5          12.  4.37        0 0
5        24.35  1.57        0 0
6    4.    0.          80. 1 1
6          0.    0.        0 0
6          .27  1.29        0 0
6          .58  3.08        0 0
6         1.15  6.44        0 0
6         2.03  6.32        0 0
6         3.57  5.53        0 0
6           5.  4.94        0 0
6           7.  4.02        0 0
6         9.22  3.46        0 0
6         12.1  2.78        0 0
6        23.85   .92        0 0
7  4.95    0.         64.6 1 1
7          0.   .15        0 0
7          .25   .85        0 0
7           .5  2.35        0 0
7         1.02  5.02        0 0
7         2.02  6.58        0 0
7         3.48  7.09        0 0
7           5.  6.66        0 0
7         6.98  5.25        0 0
7           9.  4.39        0 0
7        12.05  3.53        0 0
7        24.22  1.15        0 0
8  4.53    0.         70.5 1 1
8          0.    0.        0 0
8          .25  3.05        0 0
8         0.52  3.05        0 0
8          .98  7.31        0 0
8         2.02  7.56        0 0
8         3.53  6.59        0 0
8         5.05  5.88        0 0
8         7.15  4.73        0 0
8         9.07  4.57        0 0
8         12.1    3.        0 0
8        24.12  1.25        0 0
9   3.1    .0         86.4 1 1
9          .0    0.        0 0
```

```
 9          .3  7.37       0 0
 9         .63  9.03       0 0
 9        1.05  7.14       0 0
 9        2.02  6.33       0 0
 9        3.53  5.66       0 0
 9        5.02  5.67       0 0
 9        7.17  4.24       0 0
 9         8.8  4.11       0 0
 9        11.6  3.16       0 0
 9       24.43  1.12       0 0
10   5.5    0.        58.2 1 1
10          0.   .24       0 0
10         .37  2.89       0 0
10         .77  5.22       0 0
10        1.02  6.41       0 0
10        2.05  7.83       0 0
10        3.55 10.21       0 0
10        5.05  9.18       0 0
10        7.08  8.02       0 0
10        9.38  7.14       0 0
10        12.1  5.68       0 0
10        23.7  2.42       0 0
11  4.92    0.         65. 1 1
11          0.    0.       0 0
11         .25  4.86       0 0
11          .5  7.24       0 0
11         .98    8.       0 0
11        1.98  6.81       0 0
11         3.6  5.87       0 0
11        5.02  5.22       0 0
11        7.03  4.45       0 0
11        9.03  3.62       0 0
11       12.12  2.69       0 0
11       24.08   .86       0 0
12   5.3    0.        60.5 1 1
12          0.    0.       0 0
12         .25  1.25       0 0
12          .5  3.96       0 0
12          1.  7.82       0 0
12          2.  9.72       0 0
12        3.52  9.75       0 0
12        5.07  8.57       0 0
12        7.07  6.59       0 0
12        9.03  6.11       0 0
12       12.05  4.57       0 0
12       24.15  1.17       0 0
```

Control Stream - FCON

```
FILE    FSTREAM
PROB    THEOPHYLLINE   POPULATION DATA
DATA       1   0 144   7   0
ITEM       1   4   7  11   1   0   0   0   0   0   0
INDX       6   3   2   0   0   0   0   0   0   0   0
LABL      ID    DOSE    TIME      CP      WT    EVID     MDV
FORM
(5E6.0,2F2.0)
STRC       3   3   1   0   0   0   1   1   0
STRC       1   3
THCN       1   0   0
THTA           3     .08     .04
LOWR          .1    .008    .004
UPPR           5      .5      .9
BLST           6    .005      .3   .0002    .006      .4
DIAG          .4
ESTM       0 450   3   5   0   0   0   0   0   0   0
COVR       0   0   0   0   0
TABL       1   1   0   0
TABL       4   1   0   2   0   5   0   3   0
SCAT       1   2
SCAT       3   9   1   1   0   0       0       0   0
SCAT       3  10   1   1   0   0       0       0   0
```

File Stream - FSTREAM

```
DATA    FDATA
****
```

Report - FREPORT

```
NM-TRAN VERSION II LEVEL 1.0
SUBROUTINES FROM THE PREDPP LIBRARY:
PRED   PREDI  CHECK  SADVAN  ADVAN2  SSS0   TRANS1  INFN
GENERATED DP SUBROUTINES:
PK     ERROR
NONMEM SUBROUTINES: ALL
```

Generated and User-Supplied Subroutines - FSUBS

The NONMEM VI versions are shown. With NONMEM 7 and higher, MODULES are used rather than COMMONS.

```
      SUBROUTINE PK(ICALL,IDEF,THETA,IREV,EVTREC,N,INDXS,IRGG,GG,NETAS)
      IMPLICIT DOUBLE PRECISION (A-Z)
      REAL EVTREC
      SAVE
      INTEGER ICALL,IDEF,IREV,N,INDXS,IRGG,NETAS
      DIMENSION IDEF(7,*),THETA(*),EVTREC(IREV,*),INDXS(*),GG(IRGG,11,*)
      COMMON/PRRAND/ETA(10),EPS(10)
      COMMON/PROCM1/NEWIND
      INTEGER NEWIND
      COMMON/ROCM12/MSEC
      INTEGER MSEC
      COMMON/NMPRD4/KA,K,CL,SC,Y,A00011,A00012,A00015,A00020,A00021
      COMMON/NMPRD4/A00014,BBBBBB(0989)
      IF (ICALL.LE.1) THEN
      IDEF(1,01)=-9
      IDEF(1,02)= 1
      IDEF(3,02)= 4
      CALL GETETA(ETA)
      RETURN
      ENDIF
      IF (NEWIND.NE.2) THEN
       IF (ICALL.EQ.4) THEN
        CALL SIMETA(ETA)
       ELSE
        CALL GETETA(ETA)
       ENDIF
      ENDIF
      WT=EVTREC(N,05)
      KA=THETA(01)+ETA(01)
C                     A00011 = DERIVATIVE OF KA W.R.T. ETA(01)
      A00011=1.D0
      K=THETA(02)+ETA(02)
C                     A00012 = DERIVATIVE OF K W.R.T. ETA(02)
      A00012=1.D0
      CL=THETA(03)*WT+ETA(03)
      SC=CL/K/WT
      B00001=1.D0/K/WT
C                     A00014 = DERIVATIVE OF SC W.R.T. ETA(03)
      A00014=B00001
      B00002=-CL/K/K/WT
C                     A00015 = DERIVATIVE OF SC W.R.T. ETA(02)
      A00015=B00002*A00012
      GG(01,1,1)=K
      GG(01,03,1)=A00012
      GG(03,1,1)=KA
      GG(03,02,1)=A00011
```

```
      GG(04,1,1)=SC
      GG(04,03,1)=A00015
      GG(04,04,1)=A00014
      IF (MSEC.EQ.1) THEN
      B00003=-1.D0/K/K/WT
C                     A00016 = DERIVATIVE OF B00001 W.R.T. ETA(02)
      A00016=B00003*A00012
      B00004=-1.D0/K/K/WT
      B00005=CL/K/K/K/WT
C                     A00018 = DERIVATIVE OF B00002 W.R.T. ETA(02)
      A00018=B00005*A00012
      B00006=CL/K/K/K/WT
C                     A00019 = DERIVATIVE OF B00002 W.R.T. ETA(02)
      A00019=B00006*A00012+A00018
C                     A00020 = DERIVATIVE OF A00015 W.R.T. ETA(02)
      A00020=A00012*A00019
C                     A00021 = DERIVATIVE OF A00015 W.R.T. ETA(03)
      A00021=A00012*B00004
      GG(04,03,03)=A00020
      GG(04,04,03)=A00021
      ENDIF
      RETURN
      END
      SUBROUTINE ERROR (ICALL,IDEF,THETA,IREV,EVTREC,N,INDXS,F,G,HH)
      IMPLICIT DOUBLE PRECISION (A-Z)
      REAL EVTREC
      SAVE
      INTEGER ICALL,IDEF,IREV,N,INDXS
      DIMENSION IDEF(*),THETA(*),EVTREC(IREV,*),INDXS(*),G(10,*)
      DIMENSION HH(10,*)
      COMMON/PRRAND/ETA(10),EPS(10)
      COMMON/ROCM12/MSEC
      INTEGER MSEC
      COMMON/NMPRD4/KA,K,CL,SC,Y,A00011,A00012,A00015,A00020,A00021
      COMMON/NMPRD4/A00014,BBBBBB(0989)
      IF (ICALL.LE.1) THEN
      IDEF(2)=2
      HH(1,1)=1.0D0
      RETURN
      ENDIF
      IF (ICALL.EQ.4) THEN
      CALL SIMEPS(EPS)
      ENDIF
      Y=F+EPS(01)
      F=Y
      RETURN
      END
```

Appendix VIII. Additional NM-TRAN Control Streams

The NM-TRAN control stream listed in chapter V specifies that ADVAN2 be used. The results from NONMEM-PREDPP, using this control stream along with the NM-TRAN data set listed in Appendix VI, can be essentially duplicated, using NM-TRAN control streams which specify different ADVAN's, but which name the the same data set. Two such control streams are given here, one specifying ADVAN7 and the other specifying ADVAN6. See Guide III for some relative performance statistics. The two control streams are recorded on the NONMEM distribution medium; see Guide III.

NM-TRAN Control Stream Specifying ADVAN7

```
$PROB  THEOPHYLLINE   POPULATION DATA
$INPUT      ID DOSE=AMT TIME CP=DV WT
$DATA       THEOPP

$SUBROUTINES  ADVAN7
$MODEL COMP=(DEPOT,INITIALOFF,DEFDOSE) COMP=(CENTRAL,DEFOBS,NOOFF)

$PK
;THETA(1)=MEAN ABSORPTION RATE CONSTANT (1/HR)
;THETA(2)=MEAN ELIMINATION RATE CONSTANT (1/HR)
;THETA(3)=SLOPE OF CLEARANCE VS WEIGHT RELATIONSHIP (LITERS/HR/KG)
;SCALING PARAMETER=VOLUME/WT SINCE DOSE IS WEIGHT-ADJUSTED
   CALLFL=1
   K12=THETA(1)+ETA(1)
   K20=THETA(2)+ETA(2)
   CL=THETA(3)*WT+ETA(3)
   S2=CL/K20/WT

$THETA  (.1,3,5) (.008,.08,.5) (.004,.04,.9)
$OMEGA BLOCK(3)  6 .005 .0002 .3 .006 .4

$ERROR
   Y=F+EPS(1)

$SIGMA  .4

$EST     MAXEVAL=450  PRINT=5
$COV
$TABLE          ID DOSE WT TIME
$SCAT           (RES WRES) VS TIME BY ID
```

NM-TRAN Control Stream Specifying ADVAN6

```
$PROB  THEOPHYLLINE   POPULATION DATA
$INPUT      ID DOSE=AMT TIME CP=DV WT
$DATA       THEOPP

$SUBROUTINES  ADVAN6  TOL=5
$MODEL COMP=(DEPOT,INITIALOFF,DEFDOSE) COMP=(CENTRAL,DEFOBS,NOOFF)

$PK
;THETA(1)=MEAN ABSORPTION RATE CONSTANT (1/HR)
;THETA(2)=MEAN ELIMINATION RATE CONSTANT (1/HR)
;THETA(3)=SLOPE OF CLEARANCE VS WEIGHT RELATIONSHIP (LITERS/HR/KG)
;SCALING PARAMETER=VOLUME/WT SINCE DOSE IS WEIGHT-ADJUSTED
   CALLFL=1
   KA=THETA(1)+ETA(1)
   KE=THETA(2)+ETA(2)
   CL=THETA(3)*WT+ETA(3)
   S2=CL/KE/WT

$THETA  (.1,3,5) (.008,.08,.5) (.004,.04,.9)
$OMEGA BLOCK(3)  6 .005 .0002 .3 .006 .4

$DES
   DADT(1)=-KA*A(1)
   DADT(2)= KA*A(1)-KE*A(2)

$ERROR
   Y=F+EPS(1)

$SIGMA  .4

$EST     MAXEVAL=450  PRINT=5
$COV
$TABLE          ID DOSE WT TIME
$SCAT           (RES WRES) VS TIME BY ID
```

Appendix IX. Another Example with PREDPP

Figures 7, 11, and 25 of NONMEM Users Guide, Part VI, show a PK routine, an ERROR routine and a NONMEM control stream (with embedded data), respectively. Problem summary pages from NONMEM and PREDPP, resulting from Figures 7, 11, and 25 are shown in Figures 27-28 of Part VI. NM-TRAN inputs and outputs are given on the following pages of this appendix. The inputs correspond to Figures 7, 11, and 25. The resulting problem summary page from NONMEM is a just a little different from that shown in Figure 27 of Part VI due to the fact that NM-TRAN generates EVID, MDV and ID data items. The NM-TRAN data set and control stream are recorded on the NONMEM distribution medium; see Guide III.

NM-TRAN Data Set

```
      320        .0        .
       .         .27      1.71
       .         .52      7.91
       .        1.        8.31
       .        1.92      8.33
       .        3.5       6.85
       .        5.02      6.08
       .        7.03      5.4
       .        9.        4.55
       .       12.        3.01
       .       24.3        .90
```

NM-TRAN Control Stream

```
$PROBLEM  THEOPHYLLINE   SINGLE SUBJECT DATA
$INPUT  DOSE=AMT TIME CP=DV
$DATA  DATA3
$SUBROUTINES  ADVAN2

$PK
CALLFL=1
KA=THETA(1)
K=THETA(2)
SC=THETA(3)

$ERROR
Y=F+ERR(1)

$THETA  (0,1.7)  (0,.102)  (0,29)

$ESTIMATION  MAXEVAL=240  PRINT=2
$COVR
$TABLE TIME
$SCAT    CP VS TIME
$SCAT    PRED VS TIME
$SCAT    RES VS TIME
$SCAT    PRED VS CP  UNIT
```

NONMEM Data Set - FDATA

```
  320   .0      1 1 1
       .27 1.71 0 0 1
       .52 7.91 0 0 2
        1. 8.31 0 0 1
      1.92 8.33 0 0 2
       3.5 6.85 0 0 1
      5.02 6.08 0 0 2
      7.03  5.4 0 0 1
        9. 4.55 0 0 2
       12. 3.01 0 0 1
      24.3  .90 0 0 2
```

NONMEM Control Stream - FCON

```
FILE    FSTREAM
PROB    THEOPHYLLINE   SINGLE SUBJECT DATA
DATA       1   0  11    6   0
ITEM       6   3   5  11   1   0   0   0   0   0   0
INDX       4   2   1   0   0   0   0   0   0   0   0
LABL    DOSE    TIME       CP    EVID     MDV    .ID.
FORM
(3E5.0,3F2.0)
STRC       3   1   0   0   0   1   0   0   0
THCN       1   0   0
THTA         1.7    .102      29
LOWR           0       0       0
UPPR     1000000 1000000 1000000
DIAG   2
ESTM       0 240   3   2   0   0   0   0   0   0   0
COVR       0   0   0   0   1
TABL       1   1   0   0
TABL       1   2   0
SCAT       1   4
SCAT       2   3   0   0   0   0       0       0   0
SCAT       2   7   0   0   0   0       0       0   0
SCAT       2   8   0   0   0   0       0       0   0
SCAT       3   7   0   0   0   1       0       0   0
```

Generated and User-Supplied Subroutines - FSUBS

The NONMEM VI versions are shown. With NONMEM 7 and higher, MODULES are used rather than COMMONS.

```
      SUBROUTINE PK(ICALL,IDEF,THETA,IREV,EVTREC,N,INDXS,IRGG,GG,NETAS)
      IMPLICIT DOUBLE PRECISION (A-Z)
      REAL EVTREC
      SAVE
      INTEGER ICALL,IDEF,IREV,N,INDXS,IRGG,NETAS
      DIMENSION IDEF(7,*),THETA(*),EVTREC(IREV,*),INDXS(*),GG(IRGG,11,*)
      COMMON/PRRAND/ETA(10),EPS(10)
      COMMON/ROCM12/MSEC
      INTEGER MSEC
      COMMON/NMPRD4/KA,K,SC,Y,BBBBBB(0996)
      IF (ICALL.LE.1) THEN
      IDEF(1,01)=-9
      IDEF(1,02)= 1
      IDEF(3,02)= 4
      RETURN
      ENDIF
      KA=THETA(01)
      K=THETA(02)
      SC=THETA(03)
      GG(01,1,1)=K
      GG(03,1,1)=KA
      GG(04,1,1)=SC
      RETURN
      END
      SUBROUTINE ERROR (ICALL,IDEF,THETA,IREV,EVTREC,N,INDXS,F,G,HH)
      IMPLICIT DOUBLE PRECISION (A-Z)
      REAL EVTREC
      SAVE
      INTEGER ICALL,IDEF,IREV,N,INDXS
      DIMENSION IDEF(*),THETA(*),EVTREC(IREV,*),INDXS(*),G(10,*)
      DIMENSION HH(10,*)
      COMMON/PRRAND/ETA(10),EPS(10)
      COMMON/ROCM12/MSEC
      INTEGER MSEC
      COMMON/NMPRD4/KA,K,SC,Y,BBBBBB(0996)
      IF (ICALL.LE.1) THEN
      IDEF(2)=2
      HH(1,1)=1.0D0
      RETURN
      ENDIF
      IF (ICALL.EQ.4) THEN
      CALL SIMETA(ETA)
      ENDIF
      Y=F+ETA(01)
      F=Y
      RETURN
```

```
      END
```

NONMEM File Stream - FSTREAM

```
DATA     FDATA
****
```

NM-TRAN Report - FREPORT

```
 NM-TRAN VERSION II LEVEL 1.0
 SUBROUTINES FROM THE PREDPP LIBRARY:
 PRED   PREDI   CHECK   SADVAN   ADVAN2   SSS0    TRANS1   INFN
 GENERATED DP SUBROUTINES:
 PK     ERROR
 NONMEM SUBROUTINES: ALL
```