SIMDATA (GRAFDATA) FILE FORMAT
Revised, December 18, 1997
Reconstructed from printed output, Jan. 31, 2008

G.N. Reeke

CONTENTS

I. Introduction

This document defines the format of a SIMDATA (formerly called GRAFDATA) file. The SIMDATA file is designed to contain one or more data items recorded as a function of time, usually during the course of a neural network simulation. The file is organized as a hierarchical tree, so that data may be multidimensional, and the indexing in each dimension may be the same or different for different data items. The file begins with a header containing a list of the names, types, sizes, and dimensions of the items present, so that a program reading the file may select any desired subset of the data for further processing. The name SIMDATA reflects one purpose of the file, which is to permit the user to generate graphs showing events in the course of a neural simulation, after the fact and using any desired graphing tools. Another common use is to provide a database for performing *ex post facto* statistical analyses of simulations.

Because extremely voluminous data may be generated in the course of a large simulation, a major design goal of the SIMDATA file has been to permit data to be stored in a compact form. Thus, byte and halfword variables can easily be accommodated. In addition, selection of the data to be used occurs at two stages: first, when the file is written, and then again when it is read. The user can select a subset of the total data that is likely to be of interest for writing to the SIMDATA file and can then further select from these data, after the run is completed, the items actually to be used for graphs and statistics. A SIMDATA file may contain multiple segments, written during different stages of a simulation and containing different data items.

Currently, SIMDATA files are produced only by the CNS simulator. Because MATLAB software is commonly used for *ex post facto* statistics and graphics, a MATLAB "mex" file ("getgd3") is provided to permit data from SIMDATA files to be read directly into MATLAB variables. A possible future use will be to permit cell state information from a completed CNS simulation to be used as input to corresponding cells in a later simulation, for example, to avoid the need to simulate a standard visual subsystem within a more complex simulation.

The version of the SIMDATA file described in this document (version 3) is a major revision of the version 2 standard with several new features described below. Version 2 files can still be read into MATLAB with the "getgd2" program, but CNS will no longer write such files. (Version 1 has long been obsolete and will not be discussed further.)

A. History

The SIMDATA file was originally designed by J.C. Pearson, G.N. Reeke, and L.H. Finkel for use in connection with the SM model of group formation in the somatosensory cortex. The file was read by a program called GRAPH which ran on the IBM 5080 graphics system in use at the time. This program allowed variables such as cell activity to be plotted as a function of time and cell number in various colors and styles. The file structure was later incorporated in the CNS simulator.

Although a set of FORTRAN routines for generating and parsing SIMDATA files was provided as part of SM, these were not convenient to use and were never translated into C. In the hope of making SIMDATA files more accessible for use with future simulation software and related tools, this document defines a set of interface routines that programmers may use to easily incorporate SIMDATA files into their own applications.

B. Summary of Revisions

The revisions in the version 3 SIMDATA file specification address several inadequacies in version 2:

1. Identifier, title and time stamp. A file type identifier (the word "SIMDATA"), a user title, and a time stamp have been added to facilitate identification of SIMDATA files.

2. Names for selectors. Previously, selectors at all levels could only be integers. The new specification permits names, so that, for example, a cell type can be referred to by a name such as "PYRAMIDAL_CELLS" rather than just by a number. This eliminates the need for the user to change all the cell type selectors in a statistical routine just because a cell type was added somewhere in the middle of a simulation. Numbers are still used, for example, to select individual cells within a cell type.

3. Meaningful level names. In the original SIMDATA specification, hierarchy levels could have only a single name, independent of selections at higher levels. For the most part, only meaningless names like 'LEV1' could be assigned under these conditions. Levels are now named differently in different tree branches, permitting meaningful names to be assigned. Selectors at each level can now be interpreted unambiguously without needing to check higher numbered levels to resolve ambiguity. (For simplicity in parsing, all level names are required to be unique.) For example, when object information is selected at level 1, this level is called 'OBJECT', and when a repertoire is selected at level 1, this level is called 'REPERTOIRE'.

4. Format descriptors. There was no provision for

unsigned fixed-point values. There was no provision for a single named variable to contain arrays or packed subvariables, as used, for example, to encode x,y coordinates of objects on the input array. This practice has now been formalized. Type and scale information has been placed in separate format description items.

5. Multiple data segments. Multiple data segments existed in the version 2 specification, but were not supported by the interface routines provided. This feature has been more cleanly defined so that, for example, a separate data segment at the start of the SIMDATA file can be used to store information that does not change during the course of a simulation, such as cell connectivity or axonal delays. Segment offset information has been provided to permit backwards scanning to locate earlier data segments.

6. Complexity restrictions. The FORTRAN implementation of SIMDATA specified a maximum of 7 data selection subscripts (hierarchy levels), 23 variables, and 2000 data items. Dynamic memory allocation permits these restrictions to be removed in C implementations of SIMDATA.

II. SIMDATA File Description

The data file is composed of a general header block defining the data structure of the application program, followed by one or more independent data segments, each comprising a segment header block containing data selectors and a data block containing the actual data for one or more time points. The data block of one segment is followed immediately by the segment header block of the next segment. The last segment is followed by the 4 byte integer -999999, which acts as an internal end-of-file marker.

It is a characteristic of applications using the SIMDATA file that the number of data segments and the number of time points in each segment may not be known at the time the header blocks are written because the user may be working interactively. Accordingly, the number of data segments is not stored in the general header block, and the number of time points is not stored in the segment header block. Programs that read SIMDATA files may be required to search for data segments by reading headers and time points sequentially until the desired header block is found. This process is simplified by placing the length of data for each time point in the segment header block, so that the headers of segments being skipped do not have to be parsed to obtain this information. The file offset of the previous segment is also included to facilitate backwards searching.

All information in the first three header records, and all but the first 4 bytes of all other header records, are ASCII strings or ASCII-encoded decimal numbers. (Thus, headers may be

read directly in file dumps.) All items in data records, as well as the first four bytes of header records after the first three, are big-endian binary numbers. (The binary number at the start of each header record encodes the length of the data portion of the record and is negative to distinguish it from the first four bytes of a data record). Floating-point data must be in IEEE binary format.

The following format descriptions for header records use FORTRAN FORMAT notation, in which, for example, "I4" represents a 4-byte integer and "A60" represents a 60-byte ASCII string. "int32" represents a 32-bit big-endian binary integer.

A. General Header

Record 1: File identification and version number

Format: "GRAFDATA V3A",2I4
Data: nlvls,nvars

GRAFDATA V3A = literal string representing the type of data file and the current version number. This field has the same length and position in other binary data files used by various applications of the present author.

nlvls = number of levels in the application's data hierarchy (this is the number of level name records contained in the general header. It is also the maximum number of levels that can occur in any SIMDATA file written by this application--the number of levels in the present file may be less than or equal to 'nlvls' and may be different in different data segments).

nvars = number of variables for which format descriptions are provided in the general header (not all of these variables are necessarily recorded in the present file and different variables may be recorded in different data segments).

Record 2: Title

Format: A60
Data: title

title = literal string containing user's description of the particular run recorded in this file. In applications that use the ROCKS user interface routines, this string is obtained from the TITLE card by a call to gettit().

Record 3: Time stamp

Format: 6A2
Data: year,month,day,hour,min,sec
year,month,day,hour,min,sec = year, month, etc. giving time at which this file was created.

Records 4 through 3 + nlvls: Level_names

There is one level_names record for each of the 'nlvls' levels in the data tree. Each gives one or more names that may be used on data selector records to refer to that level. The name chosen in each case unambiguously indicates to a program parsing the file that the correct node to match a given data request has been found (see "Uniqueness Rule" below).

Format: int32,A*(|len|)
Data: len,lnam1,lnam2,...
len = length of the following list of level names (not including 'len' itself) multiplied by -1 and encoded as a 32-bit binary integer.
lnam1,lnam2,... = one or more names, usually singular. Names for the n'th level are in record number (n+3). Each level must have at least one name. Each name may be no longer than 12 characters and must contain at least one nonnumeric character. All names must be unique across all levels.

Records 4+nlvls through 3+nlvsl + nvars: Variable descriptors

The rest of the general header consists of one variable descriptor record for each of the 'nvars' different named variables that might occur in the following data. (It is to be understood that zero, one, or more instances of each of these variables may occur in the following data segments.)

Format: int32,A15,4I4,I8
Data: len,vname,vtype,vlev,vscl,vlen,vdim
len = length of the data in this record (always 32) multiplied by -1 and encoded as a 32-bit binary integer.
vname = name of the variable (maximum 15 characters).
vtype = code indicating type of variable as follows:
0 = signed integer or fixed point
1 = unsigned integer or fixed point
2 = single-precision (32-bit) real
3 = double-precision (64-bit) real
4 = colored pixel (8-, 16-, or 24-bit)
(Higher 'vtype' values are reserved for later definition of additional types.)
vlev = level in data hierarchy of this variable. (This information is redundant, but may be used for checking segment header records for internal consistency.)

vscl = binary scale of fixed point data values. Integers have scale 0. A scale of 's' indicates that the data items have been multiplied by 2^s before being recorded as integers in the SIMDATA data segment.
vlen = length of each item of this type in bytes.
vdim = dimension of this item. If the item is a scalar, vdim is coded as zero.

B. Data Segment

There may be one or more data segments in a SIMDATA file. Each data segment begins with a segment header that describes the exact data contained in the segment. Each header record begins with a negative binary integer encoding the length of that record, whereas each data record begins with a positive binary integer encoding the time value for that record. This convention makes it possible for a program reading a SIMDATA file to skip to any desired segment and to any data within a segment without knowing in advance the number of segments or the number of time points per segment. To permit backwards scanning, each segment header also contains the file offset (seek offset) of the previous segment header.

For retrieval purposes, segments are numbered consecutively beginning with segment 1, except that if the first segment in the file has a single time point designated as time 0, then that segment is known as segment 0. Segment 0 is used for recording information that does not change during a simulation.

1. Segment Header

The segment header describes the following data as a multilevel tree structure. At each level, the tree may contain one or more branches. Each branch is defined by a data selector (which may be a name, a number, a range, or a list of these) which gives the identity and quantity of information in that branch. This information may consist of zero or more variables from the list given in the general header, and, recursively, zero or more subtrees at a lower level.

The order of the actual data in the data segment is determined by considering each data selector to define a "for" loop that ranges over the values of that selector. This loop first picks up data specified by variable names at the current branch point (these may be thought of as terminal branches of the data tree), then picks up data specified by any recursively contained lower level branches. The sequence of current-level data followed by lower-level data is iterated for each value of the selector at the current branch point.

The data selectors define the subset of all the data in the

simulation run that is contained in the SIMDATA file. Further data selectors may be used by the application reading the file to select sub-subsets of the data for analysis or graphing. While the method of specifying data selectors need not be the same in the creating application and the consuming application, it is convenient both for the user and for the writer of the parsing routines if both are indeed the same. For this reason, the selectors are encoded in the segment header in the form of user "control cards" in a standard format that might be used in both applications. However, it is also possible (and currently the case in CNS) that the data selectors in the segment header can be generated from information provided by the user in some other form or manner.

a. Uniqueness Rules

Because data selectors are composed of names or numbers that are not necessarily unique, two conditions, referred to as "uniqueness rules", must be met by programs writing SIMDATA files in order that programs reading such files be able unambiguously to locate any data requested by the user. The first rule is that the creating program is not allowed to write more than one instance of the same variable with identical selectors at all levels. This rule is in fact sufficient to assure unambiguous interpretation of the file, given a complex backtracking algorithm in the parsing program to reject tentative matches that turn out to be incorrect when examined at higher-numbered data levels. The second rule eliminates the need for backtracking. It states that whenever the possibility exists that identical selectors could be used at the same data hierarchy level for different variables, a unique level name must be assigned to the selectors for each variable.

The data selector syntax assures that the level number and name associated with each set of selectors are available to the parsing program, allowing it to apply the uniqueness rules. This information also permits the user to see at a glance what variables the selectors refer to. Of course, the parsing program is not required to make use of this information. Programs that are willing to use backtracking can locate requested data without requiring their users to enter names for each level of selection.

All of this will be clarified by a few examples which are given following the formal segment header record descriptions.

Data segment header record

```
Format:  int32,"SEGMENT",I5,I12
Data:  len,segno,oprvseg
     len = length of the data in this record (always 24) multi-
          plied by -1 and encoded as a 32-bit binary integer.
     segno = segment number of this segment.
```

oprvseg = offset from start of file of previous data segment header record, or 0 if this is the first data segment.

Data selectors cards

Format: int32,A*(|len|)
Data: len,data_selector
len = length of the following data selector multiplied by -1 and encoded as a 32-bit binary integer.
data_selector = data selector, encoded as described below.

There may be as many data selector records as required to define the tree structure of the data. Each selector may be encoded in a single record of whatever length is necessary (but less than 999999 characters), or, alternatively, it may be broken down into multiple records using the rules of continuation for ROCKS control cards (see document entitled "The ROCKS Routines: General User Interface (C Implementation) User's Guide". (This provision permits programs creating SIMDATA files to copy selector cards directly from the user input file to the SIMDATA file. Routines reading selector records should use the 'len' field to allocate sufficient memory to contain the following data, but should also be prepared to process continuation records if received.)

Each data selector is in the form of a control card containing 4 or 5 whitespace-delimited fields as follows:

id level lvlname selector [vlist]

id = a keyword that identifies the card as a data selector. This keyword is determined by the application program but must contain at least one nonnumeric, nonwhitespace character. It is usually the word "LEVEL". The id field is ignored by all programs that deal with generic SIMDATA files.
level = a number giving the level of the data structure to which this selector pertains, in the range 1 <= level <= nlvsl.
lvlnam = one of the level names given in the header for the level in the data structure specified by the preceding 'level' field. This name must unambiguously associate the selector with the particular set of data variables to which it pertains. A final 'S' may be added to make the plural so the card will read better when there are multiple selectors.
selector = a data item that identifies the data associated with this selector record in the following data segment block. (Selectors are matched up with corresponding user data requests in the program reading the SIMDATA file to determine the data subset to be extracted for processing.) The items on the selector appear in the

same order as and define the order of the corresponding data items in the file. Numeric selectors must appear in ascending order within a given data selector card.

A selector may be any of the following:

(1) A name. A name is any string of 15 or fewer characters, at least one of which is nonnumeric.

(2) A number. A numeric selector is a positive decimal integer.

(3) A range. A range is a construct of the form ns-ne[+ni] or ns[+ni]-ne, where ns, ne, and ni are are positive integers specifying the start, end, and increment of the range. If the increment is omitted, it is assumed to be 1. For example, the range 3+2-8 specifies data items 3, 5, and 7 at this level. (Earlier SIMDATA formats used colons to separate the fields of a range, but the order of ni,ne was different from the MATLAB convention and the colons could not be parsed by ROCKS scan. The current convention resolves these problems.)

(4) A list. A list is one or more names, or one or more numbers and ranges, separated by commas. Names and numbers cannot be intermixed. (Note: A list cannot contain embedded whitespace characters, as the first such character terminates the selector field and initiates the vlist field.)

vlist = a list of the names of the variables selected at this level, separated by commas and containing no embedded whitespace. The vlist may be omitted if it is empty, as indicated by the [] in the above format description. The variables listed must be a subset of those enumerated in the general header.

Data length record

Following the last data selector is a data length record containing the following information:

Format: int32,"LENGTH",I12," NITS",I9
Data: len,ldata,nits

len = length of the data in this record (always 32) multiplied by -1 and encoded as a 32-bit binary integer.

ldata = length in bytes of the data in each data record that follows in this segment, not including the initial word that identifies the time value of the record. (This information is redundant, but is included here so that a program may skip over the data in a segment without parsing the data selectors for that segment.)

nits = maximum number of iterations of the underlying simulation for each time step recorded in the SIMDATA file. There will be not more than 'nits' data records in the file for each time step, all identified with the same time value in the first word of the record.

2. Data Records

As mentioned above, the number of data records in each segment is unpredictable. However, all data records (in a given segment) have a constant length and all begin with a positive integer value, whereas header records begin with a negative integer value.

Data Record

Format: big-endian binary data
Data: itstep,data
itstep = time value for this data record. The time value must always be positive. As mentioned above, if the first data segment contains a single data record with itstep == 0, then that segment is considered to be segment 0, otherwise segments are numbered from 1. If 'nits' > 1, there can be more than one record with the same 'itstep' value.
data = simulation data as described by the data selectors in the segment header.

C. End-of-file Marker

Following the last data record in each data segment is either a new set of data header records describing an additional data segment, or else an internal end-of-file marker consisting of the integer -999999, which cannot be the length field of any data header record.

Format: int32
Data: -999999

D. Examples

[To be provided]

III. Software Tools

A. Preparation of SIMDATA files

[To be written]

B. Reading SIMDATA files--C programs

A package of routines for reading SIMDATA file headers is available in the NSI tools library. Routines are provided to open a SIMDATA file, to process each type of header record, and to locate the file to a particular data segment. Selectors are parsed, generating data structures that can be used by the application to locate data in the file. The package uses a

subset of the ROCKS library that is compatible with environements in which the normal C library formatting and memory management routines are not available, for example, in MATLAB mex functions. (The application programmer must write replacement routines only for abexit(), abexitm(), mallocv(), callocv(), reallocv(), and freev() if the normal versions do not work in the application's environment.)

Please consult the source file, gditools.c, for detailed documentation of the calling sequences for these routines. Examples of how they are used can be found in the source code for the MATLAB mex routine getgd3, whose function is described in the next section. Some of the gditools routines take arguments that are pointers to "callback" routines that may be used to print or perform other operations on the information being processed. The arguments to each callback routine are documented along with the routine that calls it. The functions available in the gditools module are:

gdiopen: Given a file name, allocates and opens an RFdef path to the file. This routine must be called before using the rest of the module.
gdickid: Reads the initial record of a SIMDATA file and checks the id and version. Reads and saves the number of levels and variables.
gdinlvls: Returns the number of levels in a SIMDATA file as read by the previous call to gdickid.
gdinvars: Returns the number of variables in a SIMDATA file as read by the previous call to gdickid.
gdititle: Reads the title record of a SIMDATA file and returns a pointer to it.
gditime: Reads the time stamp of a SIMDATA file and returns a pointer to it.
gdilvlnm: Reads and stores the level names from a SIMDATA header. Calls a user callback routine which may be used to print this information in an application-determined format.
gdivars: Reads and stores variable descriptor information from a SIMDATA file. Checks that type and length codes are acceptable to the package. Calls a user callback routine which may be used to print this information in an application-determined format.
gdiqlvl: Obtains the level id number corresponding to a given level name.
gdiqvar: Obtains the variable index corresponding to a given variable name.
gdiqseg: Reads the current header record and checks that it is a SEGMENT record. Returns the number of the segment or generates an error if a SEGMENT record is not found.
gdigoseg: Accesses a requested segment in a SIMDATA file. Generates an error if the read location is not at a

SEGMENT record. On return, the read location is just after the requested SEGMENT record.

gdiparse: Parses the data selectors in a segment header and builds a tree of GDNode structures that the application can use to locate the data. Reads, checks, and stores information from the LENGTH record.

gdinits: Returns the number of inner iterations in a trial cycle of the model as read by the previous call to gdiparse.

gdirecl: Returns the length of a data record in a SIMDATA file as read by the previous call to gdiparse.

gdiclear: If there is an existing tree of selector information, removes it from memory.

gdiclose: Closes an open SIMDATA file and releases all storage allocated in connection with processing that file.

C. Reading SIMDATA files--MATLAB

[This section is a revision of the documentation for getgd, the "mex" function written to read data from Version 2 SIMDATA files into MATLAB 4. Check with your local system manager for information on how to access this function at your installation.]

Function getgd3 is a user-written MATLAB(TM) function that permits a MATLAB user to read selected data for one variable as a function of time from a version 3 SIMDATA file into a MATLAB array or matrix, making the data available for any desired MATLAB operations. getgd3 works only with version 5.0 and later of MATLAB because it uses the cell array and multidimensional array features that first became available in MATLAB 5.0.

Synopsis: x = getgd3(file, seg, item, s1, s2, ..., time)

Arguments:

file is a character string giving the file name (or full path name) of the SIMDATA file to be processed.

seg is an integer giving the number of the data segment from which data should be retrieved. Segment 0, if present, contains unchanging data, such as connectivity matrices for a neural model, stored in a single record with time == 0. Segments 1 ff. contain simulation data.

item is a character string giving the name of the data item to be retrieved. This argument must match one of the variable names in the SIMDATA file header (case is ignored). Note that if the item is an array, all array elements are returned for each instance selected by the selectors s1, s2, ...

s1, s2, ... are "selectors" that define the particular data to be retrieved. There must be one selector for each level out to the requested data item and each selector must specify data that are present at its level in the

data tree in the requested segment (precise matching rules are given below). Each selector may be specified in full form or in short form. A full-form selector is a 1 x 2 MATLAB cell array in which the first element is a string giving the applicable level name (an added 'S' will be ignored) and the second element is a short-form selector (see next paragraph), for example, { 'ARMS', [1 2] } would select data from the first and second arms in a data set. Level names may be abbreviated to any unambiguous initial substring and case is ignored. The choice of full or short form is yours: the full form provides more complete documentation and more efficient parsing; the short form is easier to type. Full and short-form selectors may be mixed in the same call.

Each short-form selector specifies one or more selections. Selectors can be names, numbers, ranges, lists of names, or lists of numbers, encoded as follows: (1) Names are given by MATLAB strings. (2) Numbers are given by positive integers. (3) Ranges are given by matrices of two or three numbers in which the first number is a positive integer giving the beginning of the range, the second is a negative integer giving minus the end of the range, and the third is an optional negative integer giving minus the increment (stride) of the range. If the third number is not present, the stride is assumed to be 1. For example, the range [1 -10 -3] will be interpreted as the range from 1 to 10 with increment 3, i.e. elements 1,4,7,10 will be accessed. (Do not omit the blanks between elements: [1-10] will be interpreted as an arithmetic expression yielding [-9], which is not a valid selector. See below for another method of expressing ranges with increments not equal to 1). (4) A list of names is a cell array containing one or more names, for example, { 'V1', 'V2' }. (A single name used in a full-form selector must be coded as a list: { 'N1', 'N2' } is a list of two names, but { 'N1', { 'N2' }} refers to variable 'N2' at level 'N1'). (5) A list of numeric selectors is a vector containing one or more concatenated numbers or ranges, for example, [1 3 -8 10]. The first number in a list cannot be negative.

The data will be returned in the order the selectors are listed. Selectors need not be in increasing numerical order, but retrieval may be more efficient if they are. Names and numbers cannot be mixed in the same list.

Letting "rSel" represent a requested data selector and "fSel" a file data selector, rSel matches fSel if: (1) rSel is name and fSel is a name and the names are identical in length and case. (2) rSel is a number and fSel is a number and the numbers are the same. (3)

rSel is a number and fSel is the rSel'th name in a list of names. (4) rSel is a number and fSel is a range and the number is contained within the range. (5) rSel is a range and fSel is a range and the values in the rSel range are a subset of the values in the fSel range. (6) rSel is a range and fSel contains discrete elements and ranges that together include all of rSel.

The selectors at all levels must form a product set, that is, for each selector specified at any one level, there must be matches for all specified selectors at all other levels. The number of items selected is the product of the numbers of items selected at each level. Equivalently, the selectors must form a tree, i.e. no branches in the selector array may recoalesce at a lower selection level. For example, you can select data from two celltypes of the same repertoire at once, or from one celltype in each of two repertoires (selected by the same name or number), but not from a celltype in one repertoire and another celltype with a different name in another repertoire.

time is a row vector of integers defining the time points for which data are to be retrieved. Time values are encoded using the same rules as for numeric variable selector lists: positive values indicate single time points, while positive-negative pairs indicate ranges of time points. See examples below.

Value returned:

x is a multidimensional array containing the requested data. The dimensionality of this array is equal to the number of indices needed to assign, in left-to-right order, one index corresponding to the array index of the item if it is stored as an array in the SIMDATA file, one index for each selector that has more than one value in order from high level selectors to low level selectors, one index to select an iteration cycle if NITS in the segment header is greater than 1, and one index to select a time point. All of these dimensions are condensed so that each index runs from 1 to the number of items contained in that dimension. All data are scaled as specified in the SIMDATA file header.

Notes and disclaimers:

(1) The SIMDATA header does not indicate the number of segments or time points contained in the data. Therefore, all values of the 'seg' and 'time' argument are initially accepted, but an error may occur later if an attempt is made to read more data than are present in the file.

(2) The SIMDATA file headers are parsed anew each time getgd3 is called. This permits the user to recreate a SIMDATA

file in another window while MATLAB is executing and still sucessfully retrieve the new data the next time getgd3 is called.

(3) Data items are requested in terms of the index values specified in the SIMDATA header and not in terms of the positions of the items in the SIMDATA file. For example, if data for objects 2 and 3, but not object 1, are in the file, they must be requested as objects 2 and 3. An error is generated if object 1 is requested in this case. However, the requested data will be returned at index values 1 and 2 in the array returned to MATLAB. (In versions of CNS prior to V8A, index values for objects, object identifiers, joints, windows, and values were incorrectly specified in the SIMDATA header according to the positions of those objects in the SIMDATA file. For example, if windows 2 and 3 were stored, they were numbered as 1 and 2 in the SIMDATA file and would have had to be requested as such. This change in numbering may required changes in getgd calls when moving from getgd to getgd3.)

(4) The rules for using abbreviations and for matching case in entering level names, selector names, and variables names may be summarized as follows:

Type of Name	Case Must Match	Abbreviations Allowed
Level	No	Yes
Selector	Yes	No
Variable	No	No
Selector		

(5) Inasmuch as numeric selector lists are MATLAB vectors, ranges may be specified with MATLAB "colon" notation, e.g. [1:3:13] is equivalent to [1 4 7 10 13]. (This construct is expanded before being passed to getgd3 and hence is transparent to getgd3.) However, when a large range of adjacent integers is needed, the getgd3 notation using a positive, negative pair of numbers is more efficient because only the starting and ending values are stored and parsed, not the entire range.

(6) The number of iterations per time step actually recorded in the SIMDATA file may be equal to or less than the value NITS given in the segment header and may even differ for different value of the time step index. Accordingly, getgd3 allocates space for NITS cycles in the output matrix. Data that do not exist in the data file will be filled in with zero values.

(7) The program d3ml, written by Karl Friston, reads version 2 SIMDATA files and cannot be used with version 3 files.

(8) MATLAB arrays are stored in double-precision floating point format, whereas many SIMDATA items are stored in shorter formats, such as single bytes or 32-bit integers. To save disk storage, it is probably better to keep the original SIMDATA file rather than to store MATLAB data derived from it with the use of getgd3.

Errors:

The following errors are detected by getgd3 and result in a call to the MATLAB function mexErrMsgTxt:

(1) Named file cannot be opened.

(2) Requested item does not match an item name contained in the SIMDATA file.
(3) The number of selector arrays does not match the level of the requested item in the SIMDATA hierarchy.
(4) The numerical value of a selector is outside the range contained in the SIMDATA file.
(5) A time selector is negative or exceeds the number of time points stored in the SIMDATA file.
(6) A negative data or time selector does not follow a positive selector.
(7) A segment selector exceeds the number of segments stored in the SIMDATA file.
(8) Read error or premature end-of-file on input file.

In addition, an error may occur after getgd3 returns if an attempt is made to store the result in a variable of the wrong "shape".

Examples:

w = getgd3('run17.graf', 1, 'WINDOW', 2, [1 -100 201:210]) requests the location of window 2 for time steps 1 to 100 and 201 to 210 from the first data segment in file "run17.graf". Because window data consist of an array of two numbers (the x,y coordinates of the window), the array returned (call it 'w') has the format:

```
w(1,1) = x(window 2, time 1),
w(2,1) = y(window 2, time 1),
w(1,2) = x(window 2, time 2),
w(2,2) = y(window 2, time 2),
....
w(1,110) = x(window 2, time 210),
w(2,110) = y(window 2, time 210)
```

s = getgd3('run18.graf', 2, 'STATE', 'VIS', 'MT', [10 -20], [100 -120]), which could also be written s = getgd3('run18.graf', 2, 'STATE', { 'REPERTOIRE' 'VISION' }, { 'CELLTYPE' 'MT' }, { 'CELLS' [10 -20] }, [100 -120]), requests data for the cell state ("s(i)" in CNS parlance) of cells 10 through 20 in celltype "MT" of repertoire "VIS" for time points 100 through 120 in the second data segment of file "run18.graf". The data are returned in a two-dimensional array (call it 's') as follows:

```
s( 1,1) = s(rep "VIS", layer "MT", cell 10, time 100),
s( 2,1) = s(    "VIS",       "MT",      11,      100), ...
s(11,1) = s(    "VIS",       "MT",      20,      100), ...

s( 1,2) = s(    "VIS",       "MT",      10,      101), ...
s( 2,2) = s(    "VIS",       "MT",      11,      101), ...
s(11,2) = s(    "VIS",       "MT",      20,      101), ...
```

c = getgd3('run19.graf', 1, 'CIJ', 2, 8, [10 15 20], 3, [0 -99], [100 -120]) requests data for the strengths ("c(ij)" values) of connections 0 to 99 of the third connection type of cells 10, 15, and 20 of the eighth celltype of the second repertoire for time points 100 through 120. Because only one connection type is requested, there is no index assigned to this variable. Connection numbers are indexed from 1 to 100 in the first dimension, cell numbers are collapsed into indexes 1,2,3 in the second dimension, and time points from 1 to 21 in the third dimension, causing the data to be arranged as follows:

```
c(  1,1, 1) = cij(cell 10, conntype 3, conn  0, time 100),
c(  2,1, 1) = cij(cell 10, conntype 3, conn  1, time 100),
     ...
c(100,1, 1) = cij(cell 10, conntype 3, conn 99, time 100),
c(  1,2, 1) = cij(cell 15, conntype 3,       0, time 100),
     ...
c(  1,1, 2) = cij(     10,          3,       0,      101),
c(  2,1, 2) = cij(     10,          3,       1,      101),
     ...
c( 11,3,11) = cij(     20,          3       10,      110),
     ...
c(100,3,21) = cij(     20,          3,      99,      120)
```