LIFELINES REFERENCE GUIDE

LifeLines Version 3.0.2
Thomas T. Wetmore IV
2 Barton Street
Newburyport, MA 01950 USA
29 April 1995
ttw@beltway.att.com

INTRODUCTION

LifeLines is a genealogy program that runs on UNIX systems. It maintains genealogical records (persons, families, sources, events and others) in a database, and generates reports from those records. There are no practical limits on the number of records that can be stored in a LifeLines database, nor on the amounts or kinds of data that can be kept in the records. LifeLines does not contain built-in reports. Instead it provides a programming subsystem that you use to program your own reports and charts. The programming subsystem also lets you query your databases and process your data in any way. LifeLines uses the terminal independent features of UNIX to provide a screen and menu based user interface.

LifeLines is a non-commercial, experimental system that is use at your own risk software. I developed LifeLines for personal use and shared it with friends. Enough of a demand arose through word of mouth and internet, that I have made the LifeLines source code and other information freely available at two ftp sites, *ftp.cac.psu.edu* and *hoth.stsci.edu*. Though LifeLines is free in all its forms, I retain all rights.

INSTALLATION

You may be installing LifeLines from a source distribution package or as an executable program already prepared for your UNIX system. The source distribution comes with the *readme*, build script and *make* files necessary to build LifeLines. Follow the instructions in the *readme* file. The executable program will be named *lines302*; I recommend changing it to *lines*. *Lines* is the LifeLines program; put it in a directory in your execution path. If you get the program in executable form, follow whatever instructions came with it.

The source distribution package also contains documentation and some LifeLines programs.

STARTING LIFELINES AND CREATING DATABASES

You normally start LifeLines with the command:

```
lines database
```

where *database* is the name of a LifeLines database. If LifeLines finds the database, LifeLines opens the database and takes you to the program's main menu. If the database doesn't exist, LifeLines asks whether it should create it, and if you answer yes, does so. You may create any number of databases, but only one can be accessed by LifeLines at a time. If you built the LifeLines executable from the source package, the executable may be named *lines302* rather than *lines*. You may either change its name or invoke LifeLines by using the command *lines302* instead.

The full command line interface to LifeLines is:

```
lines [-rwfkam] [database]
```

The options mean the following:

`–r`	open database with read only access
`–w`	open database with writeable access
`–f`	force open the database – use only in emergency
`–a`	log dynamic memory operation (for debugging)

The *-r* option opens the database with read only access. When in this mode LifeLines will not let you modify the database; no other operations are affected. The *-w* option opens the database with writeable access. If the database cannot be opened with the requested mode LifeLines quits immediately. When you open a database with neither the *-r* or *-w* options, LifeLines first tries to open the database with writeable access; if not possible LifeLines then tries to open the database with read-only access; and if this is not possible LifeLines quits. A LifeLines database may be open simultaneously by any number of programs with read-only access; however, if a database is open by a program with writeable access, then it cannot be opened by any other other program.

In rare situations the read/write mode mechanism can fail; when this happens a database may appear unopenable. If this happens use the *-f* option to force open the database; this will open the database and reset the mode mechanism. This is a *dangerous* feature; you can use it to open the same database with writeable access more than once; the results are unpredictable and generally disasterous.

If you don't give the name of a database on the command line, LifeLines will prompt you for it. If you do not use the *LLDATABASES* shell variable (described later), the name you enter must be the name of a LifeLines database directory expressed using normal UNIX absolute or relative path naming. If you do use the *LLDATABASES* variable, LifeLines will search for the database in the directories named in the variable; this can be very convenient.

INTRODUCTION TO GEDCOM

LifeLines records are stored in GEDCOM format; you organize, edit and maintain your data in this format. GEDCOM is a standard that defines a file format for moving genealogical data between computer systems. LifeLines has adopted this format for structuring the records in its databases. This approach provides a structured yet flexible method for storing all the data you wish to record. There are few restrictions on the format, amount or type of information you may store in a LifeLines database.

GEDCOM is defined at two levels. At the *syntactic* level GEDCOM is a simple set of rules for organizing and structuring data into records, with no concern about the types of records, types or formats of information in the records, or the relationships between records. At the *semantic* level GEDCOM adds an additional set of rules that specify what record types are allowed, how records must be structured, how data within the records must be identified and formatted, and what specific relationships between the record types are allowed. In principle there can be multiple semantic versions of GEDCOM, though in practice there is only one, *lineage-linked* GEDCOM. Unfortunately this semantic version of GEDCOM is poorly defined, and every genealogical system has interpreted it in different ways.

LifeLines uses GEDCOM primarily at the syntactic level, though it does impose a few of the generally accepted lineage-linked semantic restrictions. This means some important things. It means that you can store all your genealogical data in your LifeLines database, and that you have wide freedom in how you choose your own conventions for structuring and formatting your data. But it also means that the way you store data in your databases can be different from the way that someone else stores their data. This can be a problem if you share data with others or share report programs with other LifeLines users. My recommendation is to use GEDCOM lineage-linking conventions wherever possible.

LifeLines does not use forms or screens to guide you through entering or changing data. Instead you use a screen editor and directly edit the data records. This requires you to understand the GEDCOM format, and be able to edit data in GEDCOM format, before you can use LifeLines. The GEDCOM format is quite simple; this introduction will provide all you need to know about GEDCOM in order to use LifeLines.

Here is an example GEDCOM person record:

```
0 @I25@ INDI
1 NAME Thomas Trask /Wetmore/ Sr
1 SEX M
1 BIRT
   2 DATE 13 March 1866
   2 PLAC St. Mary's Bay, Digby, Nova Scotia
   2 SOUR Social Security application
1 NATU
  2 NAME Thomas T. Wetmore
  2 DATE 26 October 1888
  2 PLAC Norwich, New London, Connecticut
  2 AGE 22 years
  2 COUR New London County Court of Common Pleas
  2 SOUR court record from National Archives
1 OCCU Antiques Dealer
1 DEAT
  2 NAME Thomas Trask Wetmore
  2 DATE 17 February 1947
  2 PLAC New London, New London, Connecticut
  2 AGE 80 years, 11 months, 4 days
  2 CAUS Heart Attack
  2 SOUR New London Death Records
1 FAMC @F11@
1 FAMS @F6@
1 FAMS @F12@
```

A GEDCOM *record* is made up of *lines*. Each line has a *level number* and a *tag*, and most lines have a *value* following the tag. The first line in every record has a *cross-reference* index between the level number and the tag.

Level numbers allow data to be structured to any degree of detail; lines with higher level numbers expand on lines with lower numbers. Each record begins at level 0, and each deeper level increments the level by one. LifeLines does not restrict the structuring depth. Tags are uppercase (by convention) code words that specify the kind of information on the line or on the higher numbered lines that follow. The information after the tag, if any, is the value of the line.

The first line in a record indicates its type. There are four fixed record types in LifeLines databases: person, family, source and event. The first, 0 level line in these records have tags *INDI*, *FAM*, *SOUR* and *EVEN*, respectively. Besides these record types, you may create your own record types by using any other tag on the 0 level line of a record. The lines that begin records are the only level 0 lines used in LifeLines. Each level 0 line has a cross-reference index between the level number and the tag. This index is the record's internal reference key; other records may refer to this record by using this index. Cross-reference indexes are bracketed by @ characters.

The first line in the example record has the *INDI* tag, identifying it as a person. The cross-reference index value, *I25*, can be used by other records to refer to this record.

The second line in the example has the person's name. Each person record in a LifeLines database must have at least one *1 NAME* line, and its value must be in GEDCOM name format. This format allows names to be as long as needed, but the surname, which may be placed anywhere in the name, must be separated from the rest of the name by one or two slashes. For example:

```
1 NAME John/Smith
1 NAME John /Smith/
1 NAME John/Smith/Jr.
```

The second slash is required only if name elements follow the surname. White space is optional before the first slash and after the second. If you don't know a person's surname, or the person doesn't have a surname, you may use / or // or no slashes at all. For example:

```
1 NAME Mary//
1 NAME Mary/
1 NAME Mary
```

are all ways to enter a person named Mary with no known surname. A person may have any number, including zero, given names and/or initials. A LifeLines person record may have any number of *1 NAME* lines, though the person will be displayed with the first name value only. Persons are indexed under all their names, however, so you will be able to search for persons by any of their names.

The next line in the example gives the person's sex. LifeLines doesn't require a *1 SEX* line, but you should include it. The value of the line should be *M* or *F* if the sex is known; it can be left blank or set to *U* or *?*, say, if not known. A person must have a *1 SEX* line with a value of either *M* or *F* before he or she can be made a spouse or parent in a family.

The example record also contains three *events*: birth, naturalization, and death. An event begins with a level 1 line whose tag indicates the event type. For example, *BIRT* is the tag for a birth event.

Events usually have at least a *2 DATE* and a *2 PLAC* line and often a *2 SOUR* line. The *DATE* and *PLAC* lines give the date and place of the event. The value of a LifeLines *DATE* line is free format, though LifeLines will try to parse it for specific day, month and year information. The value of a *PLAC* line is usually a comma-separated list of geopolitical units, starting with the most specific, ending with the most general. The *SOUR* line indicates the source of information about the event. The *SOUR* line can be the root of a full description of the source, or the value of the *SOUR* line can be a cross-reference key that refers to the source record that describes the source.

The naturalization event (with tag *NATU*) shows a few other lines. The *2 NAME* line gives the person's name as recorded in the source (only *1 NAME* lines must follow GEDCOM format). The *2 AGE* line gives the person's age at the time of the event. The *2 COUR* line indicates the court where naturalization occurred.

The final event is a death event (tag *DEAT*). The *2 CAUS* line gives the cause of death.

At the end of the record are three lines that refer to family records. A *1 FAMC* line refers to a family record that the person belongs to as a child; its value is the cross-reference index value of that family. A *1 FAMS* line refers to a family record that the person belongs to as a spouse or parent.

When using LifeLines to edit a person, you will not be able to edit the cross reference values on the *0 INDI*, *1 FAMC* or *1 FAMS* lines; these are maintained by LifeLines.

Here is an example family record:

```
0 @F6@ FAM
1 HUSB @I25@
1 WIFE @I26@
1 MARR
  2 DATE 31 March 1891
  2 PLAC New London, New London, Connecticut
  2 SOUR New London Vital Records
1 CHIL @I27@
1 CHIL @I17@
```

The *0 FAM* line assigns the family the cross-reference index of *F6*. The record contains *1 HUSB* and *1 WIFE* lines that refer to the two spouses/parents. The record also holds a marriage event (tag *MARR*) and two *1 CHIL* lines that refer to two children in the family. When editing family records, you cannot edit the *0 FAM*, *1 HUSB*, *1 WIFE*, or *1 CHIL* lines; these are maintained by LifeLines.

When you create new records for your database, you are free to invent tags and structure your data in any way you see fit. However, it is good practice to use standard GEDCOM tags and value formats. LifeLines does enforce a small set of conventions that you must obey. Within person records, LifeLines requires that you use *1 NAME* and *1 SEX* lines with their special meanings and value formats. Though not required, LifeLines assumes that you will use *1 BIRT*, *1 DEAT*, *1 CHR*, and *1 BURI* lines for birth, death, baptism and burial events, respectively. In family records, LifeLines assumes you will use the *1 MARR* event for marriage events. Within person records, you are not allowed to use *0 INDI*, *1 FAMC* or *1 FAMS* lines, since these are used to maintain linkage information. Within family records, you are not allowed to use *0 FAM*, *1 HUSB*, *1 WIFE* or *1 CHIL* lines.

The indentation shown in the examples is not part of GEDCOM format. When LifeLines prepares records for you to edit, however, it always indents the records, making them easier to read and understand. You do not need to follow this indentation scheme when you edit the records. Indentation is removed from the data before it is stored in the database.

MAIN MENU

After LifeLines opens an existing database, or creates a new one, it presents you with the main menu:

```
Please choose an operation:
   b  Browse the persons in the database
   a  Add information to the database
   d  Delete information from the database
   r  Generate reports from the database
   t  Modify character translation tables
   u  Miscellaneous utilities
   x  Handle source, event and other records
   q  Quit and return to UNIX
```

Select an operation by striking the proper selection letter.

The *browse* operation lets you browse the database and perform many operations on the data. The *add* operation lets you add new information, and the *delete* operation removes information. The *generate reports* operation reads report programs and generates output reports. The *modify* character translation tables operation changes the translation tables. The *miscellaneous utilities* operation provides such things as backup and restore. The *handle* source, event and other records operation gives you access to these three record types. The *quit* operation closes the database and returns to UNIX.

The *browse* operation deserves special mention, because it provides a rich environment for searching, viewing, adding, modifying, merging and deleting information in the database. You will find that you operate from the browsing modes most of the time. The operations are all described in later sections.

ENTERING THE FIRST PERSON

Normally you add persons to the database from the browsing modes, but when entering the first person there is no one in the database to browse to. To add the first person to a LifeLines database, first select the *add* operation from the main menu. You will be prompted with the add menu (described later). Strike *p* to add a person. LifeLines creates a template of a GEDCOM person record, and puts you in a screen editor to edit the template. The default template is:

```
0 INDI
1 NAME Fname /Surname/
1 SEX MF
1 BIRT
  2 DATE
  2 PLAC
  2 SOUR
1 DEAT
  2 DATE
  2 PLAC
  2 SOUR
```

Edit the template to create the new person's record. Change the name to the person's name. Assign the person's sex by deleting either *M* or *F*. Fill out the birth and death events as best you can. If the person is alive, remove the death event. Remove any *DATE* and *PLAC* lines you do not have the information for.

The default template provides lines for one birth and one death event. You can expand the record with other events (even more birth or death events) and lines. Indentation makes it easier to read and edit the record, but isn't necessary. You may change the default edit template by defining the user option *INDIREC* (described later).

Here is how I might edit the template when creating a record about myself:

```
0 INDI
1 NAME Thomas Trask /Wetmore/ IV
1 SEX M
1 BIRT
  2 DATE 18 December 1949
  2 PLAC New London, New London, Connecticut
  2 SOUR Birth Certificate
1 OCCU Software Engineer
1 RESI
  2 DATE 1982 to 1995
  2 PLAC Newburyport, Essex, Massachusetts
  2 ADDR 2 Barton Street, Newburyport, MA 01950
... lots of other events and facts
```

When you edit a person record, don't add or modify *INDI*, *FAMC* or *FAMS* lines. LifeLines creates and maintains these lines through specific user commands.

When you finish editing and leave the editor, you automatically return to LifeLines. If you made an error (*eg*, didn't use proper level numbers or didn't follow the proper name convention), LifeLines displays an error message, and asks if you want to re-edit the record. If you don't, LifeLines doesn't add the person to the database.

When the record is in proper format, LifeLines asks if you are sure you want to add the person to the database. If you answer yes, the person is added; if you answer no, the person isn't. In both cases LifeLines returns to the main menu.

SCREEN EDITORS AND ENVIRONMENT VARIABLES

With LifeLines you maintain the database records using a screen editor. This is different than other genealogical programs where screens or forms are used to gather the data.The default screen editor for LifeLines is *vi*. This can be overridden by the *ED*, *EDITOR* or *LLEDITOR* environment variables. For example, if you prefer the *emacs* screen editor, you may add the line:

```
ED=emacs
```

to your login profile file, and LifeLines will use *emacs* for editing.

There are four other, LifeLines specific environment variables. They are *LLDATABASES*, *LLARCHIVES*, *LLPROGRAMS* and *LLREPORTS*. *LLDATABASES* and *LLPROGRAMS* are UNIX path list variables.

LLDATABASES can be set to a list of directories that hold LifeLines databases. When you execute the LifeLines programs these directories will be searched in turn for the database mentioned on the command line. For example,

```
LLDATABASES=.:/usr/ttw4/LifeLines/Databases
```

indicates that databases should be searched for in the current directory first, and if not found there, then searched for in:

```
/usr/ttw4/LifeLines/Databases
```

Each LifeLines database is implemented as a directory with specific contents. The *LLDATABASES* variable should be set to a list of directories that contain these database directories, *not* to a list of database directories themselves.

The environment variable *LLPROGRAMS* is used in the same way, but to specify the search path for LifeLines report generating and other programs (described later).

LLARCHIVES and *LLREPORTS* can each be set to specify a single directory. *LLARCHIVES* is used to select a directory where all database backup files will be stored, and *LLREPORTS* is used to select a directory where all generated reports and program outputs will be placed.

You are not required to use these environment variables; when a variable is not defined, LifeLines uses the current directory as its default value. If you do use the variables, you can override their use by specifying files and directories as either absolute or relative paths.

LifeLines uses the *curses* library for terminal independent I/O. This requires you to specify your terminal type with the *TERM* environment variable.

BASICS OF BROWSING

You will use the browsing modes of LifeLines most of the time. When in these modes you can quickly search for or browse through the persons and families in the database. When you find a person or family you are interested in, you can then edit their records.

The browsing modes also allow you to add new persons and families to the database, add spouses to families, add children to families, swap the order of spouses and children, merge persons and merge families, and perform other operations. The browsing modes also lets you remove spouses from families and remove children from families.

There are six browsing modes. The *person* and *family* modes concentrate on a single person and family, respectively. The *list* mode allows you to browse through a list of persons. The *pedigree* mode shows a four generation pedigree, the *person tandem* mode shows two persons at once, and the *family tandem* mode shows two families at once.

IDENTIFYING A PERSON OR LIST OF PERSONS TO BROWSE

To enter the browsing modes from the main menu strike *b*. LifeLines asks you to identify a person or list of persons to browse to:

```
Please identify person or persons to browse to.
Enter name, key, refn or list:
```

Enter either a name or partial name, or an internal key value, or a user-defined reference key (described later) or the name of a previously defined list of persons (described later), and strike return.

LifeLines allows wide flexibility in how to enter names. You may enter a name in upper or lower case or any combination. You may leave out all but the first given name, and, for given names, you may leave out any letters except the first. You may leave vowels out of the surname, and after four or five consonants have been typed, you may leave them out too. You must separate the given names from the surname by a slash, and if you enter given names after the surname (as in Chinese names), or any modifiers (such as Jr, Sr, IV, etc.), they must be separated from the surname by another slash. Here are a few of the ways I can enter my name:

```
Thomas Trask /Wetmore/ IV
thomas/wetmore/iv
t t/wetmr/i
th tr/Wetmore
t/wtmr/iv
```

You may browse to the list of all persons with the same surname by using the * character as the first initial. For example:

```
*/wetmore
```

matches all persons with surname Wetmore. This is the only wildcard feature supported by LifeLines.

After you enter a name, LifeLines searches for all persons who match. There are three possibilities: no one matches; one person matches; or more than one person matches. In the first case LifeLines writes:

```
There is no one in the database with that name.
```

and leaves you in the main menu.

If one person matches, LifeLines enters the person browse mode displaying the matched person. If more than one person matches, LifeLines enters the list browsing mode with the list of matching persons.

You may also identify a person by entering his or her internal, cross-reference key value. The internal key values of all person records are an *I* followed by digits. In the current version, when you enter a key value you must omit the *I*. If LifeLines finds a person with the key value you provide, LifeLines enters the person browsing mode displaying that person.

The browse command *b* is also available from most browsing modes. The command works the same way from those modes as it does from the main menu.

ZIP IDENTIFYING A NEW PERSON

Some LifeLines operations need you to identify a person, not for the purpose of browsing, but for the purpose of completing an operation you have requested. For example, when you add a child to a family, LifeLines may ask you to identify the child. When this happens a panel pops up that asks you to identify a person. You respond by typing a name or key exactly as you would for the *b* command. If no one matches, LifeLines returns to the previous browsing mode. If the name matches persons in the database LifeLines displays something like:

```
Please choose from among these persons.
  >Thomas Trask Wetmore, b. 1826, N.B. (42)
   Thomas Trask Wetmore IV, b. 1949, Conn. (1)
   Thomas Trask Wetmore III, b. 1925, Conn. (6)
   Thomas Trask Wetmore Jr, b. 1896, Conn. (11)
   Thomas Trask Wetmore Sr, b. 1866, N.S. (23)
   Thomas Trask Wetmore V, b. 1982, Mass. (5)
---------------------------------------------------------
Commands: j Move down    k Move up   i Select    q Quit
```

Use the *j* and *k* commands to move the selection cursor (>) to the correct person, and then use the *i* command to select that person. There may be more persons in the list than you can see at once. If this is so then you can use the *j* and *k* commands to scroll through the full list. If you don't find the proper person, use the *q* command and LifeLines asks whether you want to enter another name.

When LifeLines creates a list of names for you to select from, it tries to add extra information to the name; this helps determine which name to choose, and is important in databases where many persons have the same name. LifeLines also places the person's key value at the end of each menu line; this may be helpful in large databases.

Some browse modes provide the *z* command, which allows you to browse to a new person using the zip style of identification rather than the *b* style.

BROWSE DISPLAY BASICS

The screen display for each browsing mode is made up of panels. At the bottom of each display is a message panel used for one line messages. Each display contains one or two data panels showing information from the database. And each display has a panel with the operation menu available for that mode.

PERSON BROWSE MODE

After you identify a person to browse to, LifeLines enters the person browse mode. The top panel in the display gives basic information about the person. The middle panel provides a menu of commands. For example:

```
person: Thomas Trask WETMORE Sr (25)
  born: 13 March 1866, St. Mary's Bay, Digby, Nova Scotia
  died: 17 February 1947, New London, New London, Connecticut
  father: Daniel Lorenzo WETMORE, b. 1821, N.S., d. 1903, Conn. (48)
  mother: Mary Ann DOTY, b. 1824, N.S., d. 1897, Conn. (59)
  spouse: Margaret Ellen KANEEN, b. 1855, Eng., d. 1900, Conn. (26)
    child: Portia Louise WETMORE, b. 1892, Conn., d. 1921, Conn. (27)
    child: Thomas Trask WETMORE, b. 1896, Conn., d. 1970, Conn. (17)
  spouse: Arleen M KEENEY, m. 1914, Conn. (75)


Please choose an operation:
  e  Edit the person       g  Browse to family     p  Show pedigree
  f  Browse to father      u  Browse to parents    n  Create new person
  m  Browse to mother      b  Browse to persons    a  Create new family
  s  Browse to spouse/s    h  Add as spouse        x  Swap two families
  c  Browse to children    i  Add as child         t  Enter tandem mode
  o  Browse to older sib   r  Remove as spouse     z  Browse to person
  y  Browse to younger sib d  Remove as child      q  Return to main menu

LifeLines -- Person Browse Screen
```

The commands perform a wide variety of functions.

e **Edit the person**. Edit the person's database record. LifeLines puts the record in a file, and then runs a screen editor so you can edit the record. When you return from the editor, LifeLines asks you to confirm any changes; the person is changed only if you answer yes.

f **Browse to father**.
m **Browse to mother**. Browse to the person’s father or mother. If the parent isn't in the database LifeLines doesn't change the display. If there are more than one father or mother, LifeLines asks you to select one.

s **Browse to spouse/s**. Browse to the person’s spouse. If the person has more than one spouse, LifeLines asks you to select one. If the person has no spouse, the display does not change.

c **Browse to children**. Browse to one of the person’s children. If there is more than one child, LifeLines asks you to select one. If the person has no children, the display does not change.

o **Browse to older sib**.
y **Browse to younger sib**. Browse to the person’s next older sibling or next younger sibling. If the person has no such sibling, the display does not change. Only siblings from the same family are browsed by this command.

g **Browse to family**. Browse to the family the person is a spouse or parent in, and switch to the family browse mode. If the person is in more than one family, LifeLines asks you to identify which one. If the person is not a spouse or parent in any family, the display does not change.

u **Browse to parents**. Browse to the family the person is a child in, and switch to the family browse mode. If the person is not a child in a family, the display does not change. If the person is a child in more than one family, LifeLines asks you to identify which one.

b **Browse to persons**. Browse to a new person or list of persons. LifeLines asks you to identify a person or persons by name, key or list name, and depending on how many persons are identified, switches either to the list browse mode, or remains in the person browse mode.

h **Add as spouse**. Add the person as a spouse/parent to an existing family. LifeLines asks you to identify the family, and then asks you to confirm the request.

i **Add as child**. Add the person as a child to an existing family. The person may already be a child in another family. LifeLines asks you to identify the family, and then asks you to confirm the request.

r **Remove as spouse**. Remove the person as a spouse or parent from an existing family. If the person is a spouse or parent in more than one family, LifeLines asks you to identify the family.

d **Remove as child**. Remove the person as a child in an existing family.

p **Show pedigree**. Change to pedigree browse mode. The person becomes the root person in the pedigree display.

n **Create new person**. Create and add a new person to the database. LifeLines creates a record template and puts you into the screen editor to edit the record. When you return from the editor, LifeLines asks you to confirm the operation. If you do, the new person is added and becomes the current person. If not, the new person is not added, and LifeLines returns to the original display.

a **Create new family**. Create and add a new family to the database. The new family may have the current person as either a spouse/parent or as a child; LifeLines asks which. If you choose to create a family with the person as a spouse/parent, LifeLines asks you to identify the other spouse if he or she is known. In either case LifeLines creates a family template, and places you in the screen editor. When you return from the editor, LifeLines asks you to confirm the operation. If you do, LifeLines adds the family and shifts into family browse mode. If the command you ran just before the *a* command were the *n* command, and you choose to create a family with the person as a spouse/parent, LifeLines guesses that the other spouse in the family will be the person displayed just before the new person was created. LifeLines asks you if this is the case, and if so, automatically make that person the other spouse in the new family. If this is not the case, LifeLines asks you to identify the other spouse.

x **Swap two families**. Swap (change chronological order) any two families that the person belongs to as a spouse or parent. LifeLines asks you to identify the two families and then swaps them.

t **Enter tandem mode**. Change to the tandem person browse mode. LifeLines first asks you to identify the second person.

z **Browse to person**. Zip browse to a new person. LifeLines asks you to identify a person by name or key value, and if you do so, browses to that person.

q **Return to main menu**. Return to the LifeLines main menu.

LIST BROWSE MODE

This browse mode handles lists of persons. The top panel shows information about one person in the list. The left panel shows a list of up to 12 persons. The person shown in the top panel is identified by the **>** character. The right panel is the menu of available commands.

```
person: Thomas Trask WETMORE Sr (25)
  born: 13 March 1866, St. Mary's Bay, Digby, Nova Scotia
  died: 17 February 1947, New London, New London, Connecticut
  father: Daniel Lorenzo WETMORE, b. 1821, N.S., d. 1903, Conn. (48)
  mother: Mary Ann DOTY, b. 1824, N.S., d. 1897, Conn. (59)
  spouse: Margaret Ellen KANEEN, b. 1855, Eng., d. 1900, Conn. (26)
---------------------------------------------------------------------
  Thomas Trask WETMORE (42)              | Choose an operation:
  Thomas Trask WETMORE III (6)           |  j  Move down list
  Thomas Trask WETMORE IV (1)            |  k  Move up list
  Thomas Trask WETMORE (11)              |  e  Edit this person
 >Thomas Trask WETMORE Sr (23)           |  i  Browse this person
  Thomas Trask WETMORE (5)               |  m  Mark this person
                                         |  d  Delete from list
                                         |  t  Enter tandem mode
                                         |  n  Name this list
                                         |  b  Browse new persons
                                         |  a  Add to this list
                                         |  x  Swap mark/current
                                         |  q  Return to main menu
---------------------------------------------------------------------
LifeLines -- List Browse Screen
```

j **Move down list**.
k **Move up list**. Move down or up the list one person, respectively. The list panel is only large enough to show 12 persons. However, the list may contain many more persons. Use the *j* and *k* commands to scroll to these other persons.

e **Edit this person**. Edit the displayed person's database record. LifeLines runs the editor on the person’s record. When you return from the editor, LifeLines asks you to confirm any changes, and then leaves you in the list browse mode.

i **Browse this person**. Change to the person browse mode with the current person.

m **Mark this person**. Mark the current person if he/she is not marked; unmark the person is he/she is. The marked person is shown with an *x* by his/her name. Marked persons are used by the *t* and *x* commands. Only one person may be marked at a time.

d **Delete from list**. Remove the current person from the browse list (*not* from the database).

t **Enter tandem mode**. Change to the tandem person mode with the current person and the marked person as the two persons. If no person is marked there is no change.

n **Name this list**. Lists of persons may be named, allowing you to quickly browse back to them by giving a list name in response to the *b* command from different modes. LifeLines will prompt you for the name. List names are most convenient when short.

b **Browse new persons**. Browse to a new person or list of persons. You can identify a person or list of persons by name, internal or user key or by list name.

a **Add to this list**. Add more persons to the current browse list. LifeLines asks you to identify a new person or list of persons by name, key or list name, and they are added to and name-sorted into the current list.

x **Swap mark/current**. Swap the current person with the marked person in the list.

q **Return to main menu**. Return to the LifeLines main menu.

FAMILY BROWSE MODE

This browse mode displays information about a family. The top panel shows basic information about the family. The bottom panel shows the menu of available commands.

```
 father: Thomas Trask WETMORE IV (1)
   born: 18 December 1949, New London, New London, Connecticut
   died:
 mother: Luann Frances GRENDA (2)
   born: 10 July 1949, Pittsburgh, Allegheny, Pennsylvania
   died:
married: 1 August 1970, Governors Island, New York, New York
  child: Anna Vivian Wetmore, b. 1974, Alaska (3)
  child: Marie Margaret WETMORE, b. 1979, Conn. (4)
  child: Thomas Trask WETMORE V, b. 1982, Mass. (5)

Please enter the next family browse operation
  e  Edit the family      s  Add spouse to family   t  Enter family tandem
  f  Browse to father     a  Add child to family    b  Browse to new persons
  m  Browse to mother     r  Remove spouse from     z  Browse to new person
  c  Browse to children   d  Remove child from      q  Return to main menu
  n  Create new person    x  Swap two children

LifeLines - Family Browse Mode
```

e **Edit the family**. Edit the family's record. LifeLines writes the record to a file and puts you into an editor to edit the file. When you return from the editor, LifeLines asks you to confirm the update; the family is changed only if you do so.

f **Browse to father**.
m **Browse to mother**. Browse to the father/husband or mother/wife of the family, switching to person browse mode. If the parent is not there, there is no change.

c **Browse to children**. Browse to a child in the family, switching to the person browse mode. If the family has more than one child, LifeLines asks you to identify a specific child.

n **Create new person**. Create and add a new person to the database. LifeLines creates a record template and puts you into the screen editor to edit the record. When you return from the editor, LifeLines asks you to confirm the operation. If you do, the new person is added to the database. If not, the new person is not added. In both cases the display does not change.

s **Add spouse to family**. Add a spouse to the family. LifeLines asks you to identify the new spouse. If the command you ran just before the *s* command were the *n* command, LifeLines guesses that the new spouse will be the person just created. LifeLines asks if this is the case, and if so, makes that person the second spouse in the family. If not, LifeLines asks you to identify the other spouse.

a **Add child to family**. Add a child to the family. LifeLines asks you to identify the new child. If the command you ran just before the *a* command were the *n* command, LifeLines guesses that the new child will be the person just created. LifeLines asks if this is the case, and if so, adds that child to the family. If not, LifeLines asks you to identify the child. If the family already has children, LifeLines also asks where to place the new child in the family.

r **Remove spouse from**. Remove a parent/spouse from the family. LifeLines asks you to identify the person, and if you do, removes him or her. The person is not removed from the database.

d Remove child from. Remove a child from the family. LifeLines asks you to identify the child should, and if you do, removes the child from the family. The person is not removed from the database.

x Swap two children. Swap (change the chronological order) of any two children in the family. LifeLines asks you to identify the two children and then swaps them.

t Enter family tandem. This command takes you to the tandem family browse mode. LifeLines asks you to identify a second family, and then takes you to the tandem family mode, displaying both the two families.

b Browse to persons. Browse to a new person or list of persons. You can identify a person or list by name, by key, or by list name. If you successfully identify a new person or persons you will switch into the person or list browse modes.

z Browse to person. Zip browse to a new person. LifeLines asks you to identify a person by name or key value, and if you do, browses to that person.

q Return to main menu. Return to the LifeLines main menu.

TANDEM PERSON BROWSE MODE

The tandem person browse mode displays information about two persons. Its main use it to support the person merging operation. The top two panels show two persons in the format used in the person and list mode displays. The bottom panel gives the menu of available commands. For example:

```
person: Thomas Trask WETMORE Sr (25)
  born: 13 March 1866, St. Mary's Bay, Digby, Nova Scotia
  died: 17 February 1947, New London, New London, Connecticut
  father: Daniel Lorenzo WETMORE, b. 1821, N.S., d. 1903, Conn. (48)
  mother: Mary Ann DOTY, b. 1824, N.S., d. 1897, Conn. (59)
  spouse: Margaret Ellen KANEEN, b. 1855, Eng., d. 1900, Conn. (26)
-------------------------------------------------------------------
person: Thomas Trask WETMORE IV (1)
  born: 18 December 1949, New London, New London, Connecticut
  died:
  father: Thomas Trask WETMORE III, b. 1925, Conn. (6)
  mother: Joan Marie HANCOCK, b. 1928, Conn. (7)
  spouse: Luann Frances GRENDA, m. 1970, N.Y. (2)
-------------------------------------------------------------------
Please choose an operation:
 e  Edit top person    s  Browse top spouse/s  a  Add family
 t  Browse to top      c  Browse top children  j  Merge bottom to top
 f  Browse top father  b  Browse to persons    x  Switch top/bottom
 m  Browse top mother  d  Copy top to bottom   q  Return to main menu

-------------------------------------------------------------------
LifeLines - Two Person Browse Mode
```

e Edit top person. Edit the top person's record. LifeLines writes the record to a file, and puts you in the screen editor to edit the file. When you return from the editor, LifeLines asks you to confirm the update; the person is changed only if you do so.

t Browse to top. Switch to the person display with the top person as current person.

f Browse top father.
m Browse top mother. Replace the top person with his/her father or mother.

s Browse top spouse/s. Replace the the top person with his/her spouse. If the person has more than one spouse, LifeLines asks you to identify one.

c **Browse top children**. Replace the top person with one of his/her children. If the person has more than one child, LifeLines asks you to identify the one.

b **Browse to persons**. Browse to a new person or list of persons. LifeLines asks you to identify a new person or persons by name, key or list name, and then does as described in the section on identifying a person.

d **Copy top to bottom**. Copy the top person into the bottom person. A new person is not created; the same person is displayed twice.

a **Add family**. Create a new family record; LifeLines assumes the two displayed persons are to become the spouses/parents in the new family; they must be of opposite sex.

j **Merge bottom to top**. Merge the bottom person into the top person. LifeLines combines the two person records and places you in the screen editor to edit the combined record. When you are done, if you confirm the operation, LifeLines removes the bottom person from the database, and the top person is given the combined record. See the section on merging.

x **Switch top/bottom**. Swap the two persons in the display.

q **Return to main menu**. Return to the LifeLines main menu.

TANDEM FAMILY BROWSE MODE

The tandem family browse mode displays information about two families. Its main use it to support the family merging operation.The top two panels provide information about the two families you are browsing, and the bottom panel holds the menu of available commands. For example:

```
father: Thomas Trask WETMORE IV (1)
  born: 18 December 1949, New London, New London, Connecticut
mother: Luann Frances GRENDA (2)
  born: 10 July 1949, Pittsburgh, Allegheny, Pennsylvania
married: 1 August 1970, Governors Island, New York, New York
   child: Anna Vivian WETMORE, b. 1974, Alaska (3)
--------------------------------------------------------------
father: Thomas Trask WETMORE III (6)
  born: 26 October 1925, New London, New London, Connecticut
wife: Joan Marie Hancock (7)
  born: 6 June 1928, New London, New London, Connecticut
married: 5 February 1949, New London, New London, Connecticut
   child: Thomas Trask WETMORE IV, b. 1949, Conn. (1)
--------------------------------------------------------------
Please choose an operation:
 e  Edit top family       f  Browse to fathers       j  Merge bottom to top
 t  Browse to top         m  Browse to mothers       q  Return to main menu
 b  Browse to bottom      x  Switch top/bottom
--------------------------------------------------------------
LifeLines - Two Family Browse Mode
```

e **Edit top family**. This command lets you edit the top family's record. LifeLines writes the record into a file, and then puts you into an editor to edit that information. When you return from the editor, LifeLines asks you whether you are sure you want to update the family in the database. The family is changed only if you answer yes.

t **Browse to top**. Change to the family browse mode with the top family the current family.

b **Browse to bottom**. Change to the single family browse mode with the bottom family the current family.

f Browse to fathers.
m Browse to mothers. Change to the tandem person mode with the fathers or mothers of the two families as the two persons.

x Switch top/bottom. Swap the two families in the display.

j Merge bottom to top. Merge the bottom family into the top family. LifeLines combines the two family records and places you in the screen editor to edit the combined record. When you are done, if you confirm the operation, LifeLines deletes the bottom family from the database, and the top family is given the combined record. See the section on merging.

q Return to main menu. Return to the LifeLines main menu.

PEDIGREE BROWSE MODE

The pedigree browse mode displays a four-generation pedigree for the current person. The top panel holds the pedigree, and the bottom panel holds the menu of available commands. For example:

```
                  John WETMORE [1755-1848] (32)
            Daniel Van Cott WETMORE [1791-1881] (41)
                  Anna VAN COTT [1757-1802] (33)
      Daniel Lorenzo WETMORE [1821-1903] (48)
                  Thomas TRASK [-1836] (81)
            Hannah TRASK [1797-1829] (46)
                  Susannah PORTER [1754-] (82)
Thomas Trask WETMORE Sr [1866-1947]  (25)
                  Samuel DOTY [1759-] (501)
            Samuel DOTY [1787-] (74)
                  Hephzibah PORTER [1764-1853] (502)
      Mary Ann DOTY [1827-1897] (59)
                  Nathan SAVERY [1748-1826] (510)
            Lydia SAVERY [1806-] (75)
                  Deidamia SABEAN [1765-1845] (511)

Please choose an operation:
 e  Edit the person       m  Browse to mother      g  Browse to family
 i  Browse to person      s  Browse to spouse/s    b  Browse to persons
 f  Browse to father      c  Browse to children    q  Return to main menu

LifeLines - Pedigree Browse Mode
```

e Edit the person. Edit the current person.

i Browse to person. Change to the person display mode with the current person.

f Browse to father.
m Browse to mother. Browse to the father or mother of the current person, shifting the pedigree one generation back. If the parent is not in the database, there is no change.

s Browse to spouse/s. Browse to a spouse of the current person, shifting the display to the pedigree of that person. If the current person has more than one spouse, LifeLines asks you to identify the spouse; if the person has no spouse there is no change.

c Browse to children. Browse to a child of the current person, shifting the pedigree one generation forward. If the current person has more than one child, LifeLines asks you to identify the child; if the person has no children there is no change.

g Browse to family. Change to the family display; the family will be the one that the current person belongs to as spouse or parent. If there are more than one, LifeLines asks you to identify the proper one.

b **Browse to persons.** Browse to another person or list of persons; if you identify a single person the display remains in the pedigree display; if you identify more than one person the display changes to the list browse mode.

q **Return to main menu**. Leave the pedigree browsing mode and return to the main menu.

ADD OPERATION

If you choose the add operation from the main menu, LifeLines displays the add menu:

```
What do you want to add?
  p  Person – add new person to the database
  f  Family – create family record from one or two spouses
  c  Child – add a child to an existing family
  s  Spouse – add a spouse to an existing family
  q  Quit – return to the previous menu
```

These operations work in a straightforward way. LifeLines asks you the necessary questions, and lets you cancel at any time. The operations provided by this menu are also available from the browsing modes, and are often easier to perform there.

DELETE OPERATION

If you choose the delete operation at the main menu, LifeLines displays the delete menu:

```
What do you want to delete?
  c  Child – remove a child from his/her family
  s  Spouse – remove a spouse from a family
  p  Person – remove a person completely
  q  Quit – return to the previous menu
```

These operations also work in a straightforward way. LifeLines asks you the necessary questions and lets you cancel at any time.

You may also remove a child from his/her family, or remove a spouse/parent from his/her family, from the person browsing mode. In both cases, only a relationship is removed, not a person. On the other hand, the delete menu *must* be used if you want to completely remove a person from the database; this cannot be done from the browsing mode.

There is no special operation for removing a family record. LifeLines silently removes any family record that has no parent or child associated with it.

CHARACTER TRANSLATION

If you choose the modify character translation tables operation from the main menu, LifeLines displays the character translation menu:

```
Which character mapping do you want to edit?
   e  Editor to Internal mapping
   m  Internal to Editor mapping
   i  GEDCOM to Internal mapping
   x  Internal to GEDCOM mapping
   d  Internal to Display mapping
   r  Internal to Report mapping
   q  Return to main menu
```

LifeLines has little built-in knowledge of character codes. If you use 7-bit ASCII characters you will not encounter problems. However, many European and other languages require additional characters, and there are many 8-bit and other schemes for encoding those characters. LifeLines knows about none of them. However, LifeLines provides a number of character translation features you can use to manage character translation.

LifeLines provides facilities for mapping between characters whenever a data record changes form. LifeLines supports four forms:

internal	for records in the database
editor	for records being edited
display	for records being displayed
report	for records written to output file

When converting text from one form to another LifeLines normally does not convert characters codes. You may, however, override this default behavior by creating translation tables that LifeLines will use when converting between forms. There are six translation tables you may define. The following table shows the six tables and describes when they are applied:

internal to editor	when converting from internal, database form to editor form
editor to internal	when converting from editor form back to internal, database form
GEDCOM to internal	when reading GEDCOM input records and writing them to database
internal to GEDCOM	when writing internal database records to external GEDCOM file
internal to display	when displaying a record in a browsing mode display screen
internal to report	when writing internal database records to external report file

After you select a translation table you are placed in the editor to edit the table. Translation tables are made up of lines that look like:

```
pattern pattern
```

where a tab separates the patterns. Each pattern is an arbitrary sequence of verbatim ASCII characters and escape sequences. Translation occurs by finding all occurrences that match left patterns and replacing them with the corresponding right patterns.

There are four escape mechanisms used in patterns:

`#`*nnn*	*nnn* is a decimal character value
`$`*hh*	*hh* is a hexadecimal character value
`\#`	represents the # character
`\$`	represents the $ character
`\\`	represents the \ character

The character translation feature is not fully tested, and not all translations are currently implemented.

MISCELLANEOUS UTILITIES

If you choose the miscellaneous utilities operation, LifeLines displays the utilities menu:

```
What utility do you want to perform?
  s  Save the database in a GEDCOM file
  r  Read in data from a GEDCOM file
  k  Find a person's key value
  i  Identify a person from key value
  d  Show database statistics
  m  Show memory statistics
  e  Edit the place abbreviation file
  o  Edit the user options file
  q  Return to the main menu
```

s **Save the database in a GEDCOM file**. This command saves the complete LifeLines database in a GEDCOM file. All person, family, event, source and user-defined records are stored. This command may be used to periodically back up your database. When you use this command, LifeLines asks you for the name of the file. If you have defined the *LLARCHIVE* shell variable, LifeLines will store the file in the directory named in the variable.

r **Read in data from a GEDCOM file**. This command allows you restore a complete database from a GEDCOM file. When you select this command, LifeLines asks you for the name of the GEDCOM file. This command can also be used to import data from a GEDCOM file to an existing database. When LifeLines performs this command, it first reads the entire GEDCOM file and checks it for validity. If there are problems in the file, LifeLines describes them, writing them to the file *err.log*, and does not add any records to the database. If there are no problems, LifeLines adds all the records found in the file to the database (only header and trailer records are not stored in the database).

k **Find a person's key value**. This command finds the internal key value of a person.

i **Identify a person from key value**. This command identifies the person that has a particular internal key value.

d **Show database statistics**. This command summarizes the contents of the current database. It displays the number of person, family, source, event and other records in the database.

m **Show memory statistics**. This command is used by the author for debugging.

e **Edit the place abbreviation file**. This command allows you to edit the place abbreviations file. This file defines the abbreviations that are used by LifeLines when it creates lists of persons for you to select from. Each line in the file has the format:

```
word:abbr
```

where *word* is a word to be abbreviated, and *abbr* is its abbreviation. The word and its abbreviation are separated by a colon. For example:

```
Connecticut:Conn.
Massachusetts:Mass.
Nova Scotia:N.S.
```

When LifeLines constructs lists of persons for you to select from, it looks up the last component of certain *PLAC* lines in this file, and if it finds that component, replaces it with its abbreviation.

o **Edit the user options file**. This command allows you to edit the user options file. The user options file is a record kept in the database that holds user selectable options. Each option has a name and a string value. Each line in the options file has the format:

```
option=value
```

where *option* is the name of an option and *value* is the option's string value. If the value is more than one line long, then the last character in each non-final line must be a backslash. In version 3.0.2 there are four options:

INDIREC	Person record edit template
EVENREC	Event record edit template
SOURREC	Source record edit template
OTHRREC	Other record edit template

For example if you would like to replace the default person record template with the following:

```
0 INDI
1 NAME //
1 SEX
```

you would edit the user option file to contain:

```
INDIREC=0 INDI\
1 NAME //\
1 SEX
```

q **Return to main menu**. This command returns you to the main menu.

HANDLING SOURCE, EVENT AND USER-DEFINED RECORDS

LifeLines supports source, event and other, user-defined record types. You access these features through the *x* operation from the main menu. When you select this operation LifeLines displays the following menu:

```
What activity do you want to perform?
  1  Add a source record to the database
  2  Edit source record from the database
  3  Add an event record to the database
  4  Edit event record from the database
  5  Add an other record to the database
  6  Edit other record from the database
  q  Return to the main menu
```

The implementation of source, event and user-defined records is relatively new in LifeLines, and is still primitive. For example, sources cannot be searched by title or by author, you cannot browse to sources or events, as you can to persons and families, and so forth. In addition, there is no way to delete these new records. These shortcomings may be addressed in future releases.

1 **Add a source record to the database.** This operation is used to add a new source record to the database. LifeLines creates a template source and puts you in the screen editor to edit the template. The default template is:

```
0 SOUR
1 REFN
1 TITL Title
1 AUTH Author
```

Do not change the *0 SOUR* line. Otherwise you may edit this record any way you like. The *1 REFN* line is a special line you can use to give the source a symbolic name that can be used in other records to refer to the source record. See the section on using *REFN* values. Because many sources have a title and an author, the default template has these lines. Here is how I recorded one of the sources in my database:

```
0 SOUR
1 REFN jcw
1 TITL The Wetmore Family of America, and its Collateral Branches: with
  2 CONT Genealogical, Biographical, and Historical Notices
1 AUTH James Carnahan Wetmore
1 PUBL
  2 DATE 1861
  2 PLAC Albany, New York
  2 INST Munsell and Rowland
  2 ADDR 78 State Street
```

2 Edit source record from the database. Use this operation to edit an existing source record already in the database. When you select this operation LifeLines asks you to identify a source:

```
Which source record do you want to edit?
enter key or refn:
```

Identify a source by entering its key value, with or without the leading *S*, or by entering its *REFN* value. LifeLines retrieves the record and puts you in the editor with the record.

3 Add an event record to the database. This operation adds a new event record to the database. LifeLines creates a template event and puts you in the screen editor to edit the template. The default template is:

```
0 EVEN
1 REFN
1 DATE
1 PLAC
1 INDI
  2 NAME
  2 ROLE
1 SOUR
```

Do not change the *0 EVEN* line. Otherwise you may edit this record any way you like. The *1 REFN* line allows you to give this event a symbolic name you can use when you want to refer to this event from other records. See the section on using *REFN* values. The default template suggests that an event has a date, a place, and refers to persons in roles with respect to the event. There is far less experience with event-based GEDCOM than there is with simple person and family GEDCOM. You may even be wondering why you would need event records when you can simply tuck events away in person and family records. This is a topic may get covered in an appendix.

4 Edit event record from the database. Use this operation to edit an existing event record from the database. When you select this operation LifeLines asks you to identify an event:

```
Which event record do you want to edit?
enter key or refn:
```

You identify a event by entering its key value, with or without the leading *E*, or by entering its *REFN* value. LifeLines retrieves the record and places you in the screen editor with the record.

5 Add an other record to the database. This operation adds a new user-defined record to the database. LifeLines creates a template and puts you in the screen editor to edit the template. The default template is:

```
0 XXXX
1 REFN
```

Replace *XXXX* with the tag string you select for the type of the new record. You are free to choose any tag value except *INDI*, *FAM*, *SOUR* and *EVEN*. For example, if you keep record information about the ships that your North American immigrant ancestors arrived on, you would keep records about those ships in your database; the tag *SHIP* suggests itself for such records. The *1 REFN* line allows you to give this record a symbolic name you can use when you want to refer to it from other records. See the section on using *REFN* values.

6 **Edit other record from the database.** Use this operation to edit an existing user-defined record from the database. When you select this operation LifeLines asks you to identify the record:

```
What record do you want to edit?
enter key or refn:
```

You identify a record by entering its key value, with or without the leading *X*, or by entering its *REFN* value. LifeLines retrieves the record and places you in the screen editor with the record.

FAMILY STRUCTURE AND MERGING PERSONS AND FAMILIES

LifeLines 3.0.2 has relaxed most of restrictions on family structure that were imposed by earlier versions. For example, a family record may have more than one parent/spouse of the same sex; a person may be a child in more than family. This is a controversial issue. Some users insist that family relationships should imply biological relatedness, and that all other relationships should be handled by different means. Others insist that non-traditional families (any number of parents/spouses of any sex) should be allowed, and that children can be members of more than one family (*eg*, natural family and adoptive family). LifeLines no longer takes a position on this matter; you are free to set up families any way you like; the operations that add spouses and children to families no longer check for non-traditional arrangements. It is possible that a future release will include a user option to either disallow or to ask for confirmation about non-traditional relationships.

LifeLines provides features for merging persons together and for merging families together. The person merging feature is accessed from the tandem person browse mode, and the family merging feature is accessed from the tandem family browse mode. You browse to the two persons or families you want to merge and then use the *j* command. Merging is necessary when you discover that two or more person records, or two or more family records, represent the same person or family, respectively.

Versions of LifeLines prior to 3.0.2 required that persons and families meet certain criteria before they could be merged. The criteria ensured that the merged persons and families would still meet traditional family structuring rules. With the relaxation of the structuring rules, restrictions on merging have also been removed. It is now possible to create non-traditional relationships by merging traditional persons and/or families. For example, if you merge two persons that happen to be children in two different families, the merged person will be a child in *both* families. If you want to maintain only traditional relationships in your database you may have to makes further to changes to relationships after you complete a merge operation.

LINKING RECORDS TOGETHER AND USING THE REFN FEATURE

Records in a LifeLines database may refer to other records via cross-reference links. The lineage-linked references are maintained directly by LifeLines through operations found in the browsing mode menus. These references are the links from a person to families (*1 FAMC* and *1 FAMS*), and the links from a family to persons (*1 HUSB*, *1 WIFE* and *1 CHIL*). Because LifeLines maintains these links you are not allowed to change these lines when you are editing records. There are a couple of seeming exceptions to this rule. For example, you may change the order of *1 CHIL* lines in a family record in order to change the order of children in a family, and you may change the order of *1 FAMS* lines in a person record to change the order of families the person was a spouse or parent in. These operations are allowed because they don't affect which person records refer to which family records and vice versa.

Besides the lineage-links that are maintained by LifeLines, you may place your own links in records.

Probably the most common example of this is referring events within a person record to the record of the information source for the event. For example:

```
0 @I23@ INDI
1 NAME Thomas/Whitmore/
1 BIRT
  2 DATE about 1615
  2 PLAC England
  2 SOUR @S3@
...

0 @S3@ SOUR
1 REFN cat
1 TITL New England Marriages Prior to 1700
1 AUTH Clarence Almon Torrey
...
```

The *2 SOUR @S3@* line in the person record refers to the source record. LifeLines allows any specific structure within a record (in this case a birth event) to refer to another record. It is not possible to refer to a specific location within another record, though this may be supported eventually.

This example implies that when linking one record to another you must know the key of the target record (*S3* in the example). This is not desirable because internal record keys may change when the records are exported from one database or imported to another.

Because internal key values are not permanent, LifeLines allows you to assign a permanent user-defined key to any record in the database using the *1 REFN* line. The value of this line is a string that you choose as your permanent key value for the record. When adding a link to a record that has a user *REFN* key value, you may use that value instead of the internal key value. For example, when adding the person in the previous example you could edit the new record as follows:

```
0 INDI
1 NAME Thomas/Whitmore/
1 BIRT
  2 DATE about 1615
  2 PLAC England
  2 SOUR <cat>
```

Instead of using the actual key value of the source, *S3*, the *REFN* value *cat* was used. The *REFN* value must be enclosed by angle brackets when used this way. LifeLines automatically replaces the *REFN* link with the proper internal key value when the record is stored in the database.

The *REFN* value may also be used when searching for person, source, event and user-defined records. You should not add more than one *REFN* line to a record, and every *REFN* value should be unique.

THE LIFELINES PROGRAMMING SUBSYSTEM AND REPORT GENERATOR

The LifeLines programming subsystem lets you produce reports in any style or layout. You may generate files in troff, Postscript, TeX, SGML or any other ASCII-based format, for further text processing and printing. You access the report generator by choosing the *r* command from the main menu. You may also use the programming subsystem to create query and other processing programs that write their results directly upon the screen. For example, there is a LifeLines program that computes the relationship between any two persons in a database.

Each LifeLines program is written in the LifeLines programming language, and the programs are stored in normal files. When you direct LifeLines to run a program, it asks you for the name of the program file, asks you where you want the program's output written, and then runs the program.

For example, say you want LifeLines to generate an *ahnentafel*. Such a report might look like:

```
1. Thomas Trask WETMORE IV
     b. 18 December 1949, New London, Connecticut
2. Thomas Trask WETMORE III
     b. 15 October 1925, New London, Connecticut
3. Joan Marie HANCOCK
     b. 6 June 1928, New London, Connecticut
4. Thomas Trask WETMORE Jr
     b. 5 May 1896, New London, Connecticut
     d. 8 November 1970, New London, Connecticut
5. Vivian Genevieve BROWN
     b. 5 April 1896, Mondovi, Wisconsin
6. Richard James HANCOCK
     b. 18 August 1904, New London, Connecticut
     d. 24 December 1976, Waterford, Connecticut
7. Muriel Armstrong SMITH
     b. 28 October 1905, New Haven, Connecticut
8. Thomas Trask WETMORE Sr
     b. 13 March 1866, St. Mary's Bay, Nova Scotia
     d. 17 February 1947, New London, Connecticut
9. Margaret Ellen KANEEN
     b. 27 October 1859, Liverpool, England
     d. 10 May 1900, New London, Connecticut
... lots more
```

Here is a LifeLines program that generates this report:

```
proc main ()
{
      getindi(indi)
      list(ilist)
      list(alist)
      enqueue(ilist, indi)
      enqueue(alist, 1)
      while(indi, dequeue(ilist)) {
            set(ahnen, dequeue(alist))
            d(ahnen) ". " name(indi) nl()
            if (e, birth(indi)) { "    b. " long(e) nl() }
            if (e, death(indi)) { "    d. " long(e) nl() }
            if (par, father(indi)) {
                  enqueue(ilist, par)
                  enqueue(alist, mul(2,ahnen))
            }
            if (par,mother(indi)) {
                  enqueue(ilist, par)
                  enqueue(alist, add(1,mul(2,ahnen)))
            }
      }
}
```

Say this program is in the file *ahnen*. When you choose the *r* option from the main menu, LifeLines asks:

```
What is the name of the report program?
enter string:
```

You enter *ahnen*. Since the program generates a report, LifeLines asks where to write that report:

```
What is the name of the output file?
enter file name:
```

You enter a file name, say *my.ahnen*. LifeLines reads the program *ahnen*, executes the program, and writes the report output to *my.ahnen*. LifeLines reports any syntax or run-time errors found while trying to run the program.

A LifeLines program is made up of procedures and functions; every program must contain at least one procedure named *main*. The main procedure runs first; it may call other procedures, functions and built-in functions. In the ahnentafel example there is only one procedure.

A procedure body is a sequence of statements. In the example program the first five statements are:

```
getindi(indi)
list(ilist)
list(alist)
enqueue(ilist, indi)
enqueue(alist, 1)
```

The first statement calls the *getindi* (get individual) built-in function, which causes LifeLines to ask you to identify a person using the zip browse style of identification:

```
Identify person for interpreted report
enter name:
```

After you identify a person, he or she is assigned to the variable *indi*. The next two statements declare two list variables, *ilist* and *alist*. Lists hold sequences of things; there are operations for placing things on lists, taking things off, and iterating through the list elements. In the example, *ilist* holds a list of ancestors, in ahnentafel order, who have not yet been reported on, and *alist* holds their respective ahnentafel numbers.

The next two statements call the *enqueue* function, adding the first members to both lists. The person identified by the *getindi* function is made the first member of *ilist*, and the number one, this person's ahnentafel number, is made the first member of *alist*.

The rest of the program is:

```
while(indi, dequeue(ilist)) {
      set(ahnen, dequeue(alist))
      d(ahnen) ". " name(indi) nl()
      if (e, birth(indi)) { "    b. " long(e) nl() }
      if (e, death(indi)) { "    d. " long(e) nl() }
      if (par, father(indi)) {
            enqueue(ilist, par)
            enqueue(alist, mul(2,ahnen))
      }
      if (par, mother(indi)) {
            enqueue(ilist, par)
            enqueue(alist, add(1,mul(2,ahnen)))
      }
}
```

This is a loop that iteratively removes persons and their ahnentafel numbers from the two lists, and then prints their names and birth and death information. If the persons have parents in the database, their parents and their parents' ahnentafel numbers are then put at the ends of the lists. The loop iterates until the list is empty.

The loop is a *while* loop statement. The line:

```
while(indi, dequeue(ilist)) {
```

removes (via *dequeue*) a person from *ilist*, and assigns the person to variable *indi*. As long as there are persons on *ilist*, another iteration of the loop follows.

The statement:

```
set(ahnen, dequeue(alist))
```

is an *assignment* statement. The second argument is evaluated; its value is assigned to the first argument, which must be a variable. Here the next number in *alist* is removed and assigned to variable *ahnen*. This is the ahnentafel number of the person just removed from *ilist*.

The line:

```
d(ahnen) ". " name(indi) nl()
```

contains four expression statements; when expressions are used as statements, their values, if any, are treated as strings and written directly to the report output file. The *d* function converts its integer argument to a numeric string. The *". "* is a literal (constant) string value. The *name* function returns the default form of a person's name. The *nl* function returns a string containing the newline character.

The next two lines:

```
if(e, birth(indi)) { "    b. " long(e) nl() }
if(e, death(indi)) { "    d. " long(e) nl() }
```

write out basic birth and death information about a person. These lines are *if* statements. The second argument in the conditional is evaluated and assigned to the first argument, which must be a variable. The first *if* statement calls the *birth* function, returning the first birth event in a person's record. If the event exists it is assigned to variable *e*, and the body (the items between the curly brackets) of the *if* statement is executed. The body consists of three expression statements: a literal, and calls to the *long* and *nl* functions. *Long* takes an event and returns the values of the first *DATE* and *PLAC* lines in the event.

Finally in the program is:

```
if (par, father(indi)) {
      enqueue(ilist,par)
      enqueue(alist,mul(2,ahnen))
}
if (par,mother(indi)) {
      enqueue(ilist,par)
      enqueue(alist,add(1,mul(2,ahnen)))
}
```

These lines add the father and mother of the current person, if either or both are in the database, to *ilist*. They also compute and add the parents' ahnentafel numbers to *alist*. A father's ahnentafel number is twice that of his child. A mother's ahnentafel number is twice that of her child plus one. These values are computed with the *mul* and *add* functions.

LIFELINES PROGRAMMING REFERENCE

LifeLines programs are stored in files you edit with a screen editor. Programs are not edited from within the LifeLines program; edit them as you would any text file. The programs may be stored in any directories; they do not have to be kept in or associated with LifeLines databases. You may set the *LLPROGRAMS* shell variable to hold a list of directories that LifeLines will use to automatically search for programs when you request program execution.

Procedures and Functions

A LifeLines program is made up of one or more procedures and functions. A procedure has format:

```
proc name ( params ) { statements }
```

Name is the name of the procedure, *params* is an optional list of parameters separated by commas, and *statements* is a list of statements that make up the procedure body. Report generation begins with the first statement in the procedure named *main*. Procedures may call other procedures and functions. Procedures are called with the *call* statement described below.When a procedure is called, the statements making up its body are executed.

A function has format:

```
func name ( params ) { statements }
```

Name, *params* and *statements* are defined as in procedures. Functions may call other procedures and functions. When a function is called the statements that make it up are executed. A function differs from a procedure by returning a value to the procedure or function that calls it. Values are returned by the *return* statement, described below. Recursive functions are allowed. A function is called by invoking it in an expression.

Comments

You may comment your LifeLines programs using the following notation:

```
/*...comment text including any characters except */... */
```

Comments begin with a /* and end with a */. Comments may appear on lines of their own or on lines that have program constructs. Comments may span many lines. Comments may not be nested.

Statements

There are a number of statement types. The simplest is an *expression* statement, an expression that is not part of any other statement or expression. Expressions are defined more fully below. An expression statement is evaluated, and if its value is non-null (non-zero), it is assumed to be a string, and written to the program output file. If its value is null, nothing is written to the output file. For example, the expression *name(indi)*, where *indi* is a person, returns the person's name and writes it to the output file. On the other hand, the expression *set(n, nspouses(indi))* assigns the variable *n* the number of spouses that person *indi* has, but since *set* returns null, nothing is written to the output file.

The programming language includes *if* statements, *while* statements and procedure call statements, with the following formats:

```
if ([varb,] expr) { statements }
   [ elsif ([varb], expr) { statements } ]*
   [ else { statements } ]
while ([varb,] expr ) { statements }
call name ( args )
```

Square brackets indicate optional parts of the statement syntax. An *if* statement is executed by first evaluating the conditional expression in the *if* clause. If non-zero, the statements in the *if* clause are evaluated, and the rest of the *if* statement, if any, is ignored. If the value is zero, and there is an *elsif* clause following, the conditional in the *elsif* clause is evaluated, and if non-zero, the statements in that clause are executed. Conditionals are evaluated until one of them is non-zero, or until there are no more. If no conditional is non-zero, and if the *if* statement ends with an *else* clause, the statements in the *else* clause are executed. There are two forms of conditional expressions. If the conditional is a single expression, it is simply evaluated. If the conditional is a variable followed by an expression, the expression is evaluated and its value is assigned to the variable.

The *while* statement provides a looping mechanism. The conditional is evaluated, and if non-zero, the body of the loop is executed. After each iteration the expression is reevaluated; as long as it remains non-zero, the loop is repeated.

The call statement provides procedure calls. *Name* must match one of the procedures defined in the report program. *Args* is a list of argument expressions separated by commas. Recursion is allowed. When a call is executed, the values of its arguments are evaluated and used to initialize the procedure's parameters. The procedure is then executed. When the procedure completes, execution resumes with the first item after the call.

The report language also includes the following statement types:

```
include(string)
global(varb)
set(varb, expr)
continue()
break()
return([expr])
```

The *include* statement includes the contents of another file into the current file; its string expression is the name of another LifeLines program file. It is described in more detail below. The *global* statement must be used *outside* the scope of any procedure or function; it declares a variable to have global scope. The *set* statement is the *assignment* statement; the expression is evaluated, and its value is assigned to the variable. The *continue* statement jumps to the bottom of the current loop, but does not leave the loop. The *break* statement breaks out of the most closely nested loop. The *return* statement returns from the current procedure or function. Procedures have *return* statements without expressions; functions have *return* statements with expressions. None of these statements return a value, so none has a direct effect on program output.

In addition to these conventional statements, the report generator provides other iterator statements for looping through genealogical and other types of data. For example, the *children* statement iterates through the children of a family, the *spouses* statement iterates through the spouses of a person, and the *families* statement iterates through the families that a person is a spouse or parent in. These iterators and others are described in more detail later under the appropriate data types.

Expressions

There are four types of expressions: *literals*, *integers*, *variables* and built-in or user defined *function calls*.

A *literal* is any string enclosed in double quotes; its value is itself. An *integer* is any integer constant; its value is itself. A *variable* is a named location that can be assigned different values during program execution. The value of a variable is the last value assigned to it. Variables do not have fixed type; at different times in a program, the same variable may be assigned data of completely different types. An identifier followed by comma-separated list of expressions enclosed in parentheses, is either a call to a built-in function or a call to a user-defined *function*.

Include Feature

The LifeLines programming language provides an *include* feature. Using this feature one LifeLines program can refer to other LifeLines programs. This feature is provided by the include statement:

```
include(string)
```

where *string* is a quoted string that is the name of another LifeLines program file. When an include statement is encountered, the program that it refers to is read at that point, exactly as if the contents of included file had been in the body of the original file at that point. This allows you to create LifeLines program library files that can be used by many programs. Included files may in turn contain include statements, and so on to any depth. LifeLines will use the *LLPROGRAMS* shell variable, if set, to search for the include files.

Built-in Functions

There is a long list of built-in functions, and this list will continue to grow for some time. The first subsection below describes the value types used in LifeLines programs; these are the types of variables, function parameters and function return values. In the remaining sections the built-in functions are separated into logical categories and described.

Value Types

`ANY`	union of all types
`INT`	integer (on most systems a 32-bit signed value)
`BOOL`	boolean (0 represents *false*; anything else represents *true*)
`STRING`	text string
`LIST`	arbitrary length list of any values
`TABLE`	keyed look-up table
`INDI`	person; reference to a GEDCOM *INDI* record
`FAM`	family; reference to a GEDCOM *FAM* record
`SET`	arbitrary length set of persons
`NODE`	GEDCOM node; reference to a line in a GEDCOM tree/record
`EVENT`	event; reference to substructure of nodes in a GEDCOM record
`VOID`	type with no values

In the summaries of built-in functions below, each function is shown with its argument types and its return type. The types are from the preceding list. Sometimes an argument to a built-in function must be a variable; when this is so its type is given as *XXX_V*, where *XXX* is one of the types above. The built-ins do not check the types of their arguments. Variables can hold values of any type, though at any one time they will hold values of only one type. Note that *EVENT* is a subtype of *NODE*, and *BOOL* is a subtype of *INT*. Built-ins with type *VOID* actually return null (zero) values.

Arithmetic and Logic Functions

`INT add(INT, INT [,INT]*)`
`INT sub(INT, INT)`
`INT mul(INT, INT [,INT]*)`
`INT div(INT, INT)`
`INT mod(INT, INT)`
`INT exp(INT, INT)`
`INT neg(INT)`
	addition - two to 32 arguments
	subtraction
	multiplication - two to 32 arguments
	division
	modulus (remainder)
	exponentiation
	integer negation

`VOID incr(INT_V)` `VOID decr(INT_V)`	increment variable by one decrement variable by one

`BOOL and(BOOL, BOOL [,BOOL]*)`
`BOOL or(BOOL, BOOL [,BOOL]*)`
`BOOL not(BOOL)`
	logical and - two to 32 arguments
	logical or - two to 32 arguments
	logical not

`BOOL eq(ANY, ANY)`
`BOOL ne(ANY, ANY)`
`BOOL lt(ANY, ANY)`
`BOOL gt(ANY, ANY)`
`BOOL le(ANY, ANY)`
`BOOL ge(ANY, ANY)`
	equality (not strings)
	non-equality
	less than
	greater than
	less than or equal
	greater than or equal

Add, *sub*, *mul* and *div* do integer arithmetic. Functions *add* and *mul* can have two to 32 arguments; the sum or product of the full set of arguments is computed. Functions *sub* and *div* have two arguments each; *sub* subtracts its second argument from its first, and *div* divides its first argument by its second. The *mod* function returns the remainder after dividing the first parameter by the second. If the second argument to *div* or *mod* is zero, these functions return 0 and generate a run time error. *Exp* performs integer exponentiation. *Neg* negates its argument.

Incr and *decr* increment by one and decrement by one, respectively, the value of a variable. The argument to both functions must be a variable.

And and *or* do logical operations. Both functions take two to 32 arguments. All arguments are and'ed or or'ed together, respectively. The arguments are evaluated from left to right, but only up to the point where the final value of the function becomes known. *Not* does the logical not operation.

Eq, *ne*, *lt*, *le*, *gt* and *ge* evaluate the six ordering relationships between two integers.

Person Functions

`STRING name(INDI [,BOOL])`
`STRING fullname(INDI, BOOL, BOOL, INT)`
`STRING surname(INDI)`
`STRING givens(INDI)`
`STRING trimname(INDI,INT)`
	default name of
	many name forms of
	surname of
	given names of
	trimmed name of

`EVENT birth(INDI)`
`EVENT death(INDI)`
`EVENT baptism(INDI)`
`EVENT burial(INDI)`
	first birth event of
	first death event of
	first baptism event of
	first burial event of

`INDI father(INDI)`
`INDI mother(INDI)`
`INDI nextsib(INDI)`
`INDI prevsib(INDI)`
	first father of
	first mother of
	next (younger) sibling of
	previous (older) sibling of

`STRING sex(INDI)`
`BOOL male(INDI)`
`BOOL female(INDI)`
`STRING pn(INDI, INT)`
	sex of
	male predicate
	female predicate
	pronoun referring to

`INT nspouses(INDI)`
`INT nfamilies(INDI)`
`FAM parents(INDI)`
	number of spouses of
	number of families (as spouse/parent) of
	first parents' family of

`STRING title(INDI)`
`STRING key(INDI|FAM [,BOOL])`
`STRING soundex(INDI)`
`NODE inode(INDI)`
`NODE root(INDI)`
	first title of
	internal key of (work for families also)
	SOUNDEX code of
	root GEDCOM node of
	root GEDCOM node of

`INDI indi(STRING)`	find person with key value

`INDI firstindi()`
`INDI nextindi(INDI)`
`INDI previndi(INDI)`
	first person in database in key order
	next person in database in key order
	previous person in database in key order

`spouses (INDI, INDI, FAM, INT) { }`
`families (INDI, FAM, INDI, INT) { }`
`forindi (INDI, INT) { }`
	loop through all spouses of
	loop through all families (as spouse) of
	loop through all persons in database

These functions take a person as a parameter and return information about him or her.

Name returns the default name of a person; this is the name found on the first *1 NAME* line in the person's record; the slashes are removed and the surname is made all capitals; *name* can take an optional second parameter – if it is *true* the function acts as described above; if *false*, the surname is kept exactly as it is in the record.

Fullname returns the name of a person in a variety of formats. If the second parameter is *true* the surname is shown in upper case; otherwise the surname is as in the record. If the third parameter is *true* the parts of the name are shown in the order as found in the record; otherwise the surname is given first, followed by a comma, followed by the other name parts. The fourth parameter specifies the maximum length field that can be used to show the name; various conversions occur if it is necessary to shorten the name to fit this length.

Surname returns the surname of the person, as found in the first *1 NAME* line; the slashes are removed. *Givens* returns the given names of the person in the same order and format as found in the first *1 NAME* line of the record. *Trimname* returns the default name of the person trimmed to the maximum character length given in the second variable.

Birth, *death*, *baptism* and *burial* return the first birth, death, baptism and burial event in the person's record, respectively. An event is a level 1 GEDCOM node. If there is no matching event these functions return *null*.

Father, *mother*, *nextsib* and *prevsib* return the father, mother, next younger sibling and next older sibling of the person, respectively. If the person has more than one father (mother) the *father* (*mother*) function returns the first one. These functions return *null* if there is no person in the role.

Sex returns the person's sex as the string *M* if the person is male, *F* if the person is female, or *U* if the sex of the person is not known. *Male* and *female* return *true* if the person is male or female, respectively, or *false* if not.

Pn generates pronouns, useful when generating English text; the second parameter selects the type of pronoun:

0	He/She
1	he/she
2	His/Her
3	his/her
4	him/her

Nspouses returns the number of spouses the person has in the database, and *nfamilies* returns the number of families the person is a parent/spouse in; these two values are not necessarily the same. *Parents* returns the first family that the person is a child in.

Title returns the value of the first *1 TITL* line in the record. *Key* returns the key value of a person or family; it there is a second parameter and it is non-null, the leading *I* or *F* will be stripped. *Soundex* returns the Soundex code of the person.

Root and *Inode* return the root node of the person's GEDCOM node tree. Note that an *INDI* value is not a *NODE* value. If you want to process the nodes within a person node tree, you must first use the *root* or *inode* function to get the root of the person node tree. *Root* and *inode* are synonyms.

Indi returns the person who's key is passed as an argument; if no person has the key *indi* returns *null*.

Firstindi, *nextindi* and *previndi* allow you to iterate through all persons in the database. *Firstindi* returns the first person in the database in key order. *Nextindi* returns the next person after the argument person in key order. *Previndi* returns the previous person before the argument person in key order.

Spouses is an iterator that loops through each spouse a person has. The first argument is a person. The second argument is a person variable that iterates through the first person's spouses. The third argument is a family variable that iterates through the families the person and each spouse are in. The fourth argument is an integer variable that counts the iterations.

Families is an iterator that loops through the families a person was a spouse/parent in. The first argument is a person. The second argument is a family variable that iterates through the families the first person was a spouse/parent in. The third argument iterates through the spouses from the families; if there is no spouse in a particular family, the variable is set to *null* for that iteration. The fourth argument is an integer variable that counts the iterations.

Forindi is an iterator that loops through every person in the database in ascending key order. Its first parameter is a variable that iterates through the persons; its second parameter is an integer counter variable that counts the persons starting at one.

Family Functions

`EVENT marriage(FAM)`
`INDI husband(FAM)`
`INDI wife(FAM)`
`INT nchildren(FAM)`
`INDI firstchild(FAM)`
`INDI lastchild(FAM)`
`STRING key(FAM|INDI [,BOOL])`
`NODE fnode(FAM)`
`NODE root(FAM)`
	first marriage event of
	first husband/father of
	first wife/mother of
	number of children in
	first child of
	last child of
	internal key of (works for persons also)
	root GEDCOM node of
	root GEDCOM node of

`FAM fam(STRING)`	find family from key

`FAM firstfam()`
`FAM nextfam(FAM)`
`FAM prevfam(FAM)`
	first family in database in key order
	next family in database in key order
	previous family in database in key order

`children (FAM, INDI_V, INT_V) { }` `forfam (FAM_V, INT_V) { }`	loop through children of family loop through all families in database

These functions take a family as an argument and return information about it.

Marriage returns the first marriage event found in the family record, if any; it returns *null* if there is no marriage event.

Husband returns the first husband/father of the family, if any; and *wife* returns the first wife/mother of the family, if any. Each returns *null* if the requested person is not in the family.

Nchildren returns the number of children in the family.

Firstchild and *lastchild* return the first child and last child in a family, respectively.

Key was described in the section on person functions.

Root and *fnode* return the root node of a family GEDCOM node tree. Note that a *FAM* value is not a *NODE* value. If you want to process the nodes within a family node tree, you must first use *root* or *fnode* function to get the root of the family node tree. *Root* and *fnode* are synonyms.

Fam returns the family who's key is passed as an argument; if no family has the key *fam* returns *null*.

Firstfam, *nextfam* and *prevfam* allow you to iterate through all families in the database. *Firstfam* returns the first family in the database in key order. *Nextfam* returns the next family after the argument family in key order. *Prevfam* returns the previous family before the argument family in key order.

Children is an iterator that loops through the children in a family. Its first parameter is a family expression; its second parameter is a variable that iterates through each child; its third parameter is an integer counter variable that counts the children starting at one. These two variables may be used within the loop body.

Forfam is an iterator that loops through every family in the database in ascending key order. Its first parameter is a variable that iterates through the families; its second parameter is an integer counter variable that counts the families starting at one.

List Functions

`VOID list(LIST_V)`
`BOOL empty(LIST)`
`INT length(LIST)`
	declare a list
	check if list is empty
	length of list

`VOID enqueue(LIST, ANY)`
`ANY dequeue(LIST)`
`VOID requeue(LIST, ANY)`
	enqueue element on list
	dequeue and return element from list
	requeue an element on list

`VOID push(LIST, ANY)` `ANY pop(LIST)`	push element on list pop and return element from list
`VOID setel(LIST, INT, ANY)` `ANY getel(LIST, INT)`	array element assignment array element selection
`forlist (LIST, ANY_V, INT_V) { }`	loop through all elements of list

LifeLines provides general purpose lists that can be accessed as queues, stacks or arrays. A list must be declared with the *list* function before it can be used.

A list can have any number of elements. *Empty* returns *true* if the list has no elements and *false* otherwise. *Length* returns the length of the list. The only parameter to both is a list.

Enqueue, *dequeue* and *requeue* provide queue access to a list. *Enqueue* adds an element to the back of a queue, *dequeue* removes and returns the element from the front of a queue, and *requeue* adds an element to the front of a queue. The first parameter to all three is a list, and the second parameter to *enqueue* and *requeue* is the value to be added to the queue and can be any value.

Push and *pop* provide stack access to a list. *Push* pushes an element on the stack, and *pop* removes and returns the most recently pushed element from the stack. The first parameter to both is a list, and the second parameter to *push* is the value to be pushed on the stack and can be of any type.

Setel and *getel* provide array access to a list. *Setel* sets a value of an array element, and *getel* returns the value of an array element. The first parameter to both is a list; the second parameter to both is an integer index into the array; and the third parameter to *setel* is the value to assign to the array element and can be of any type. Array elements are indexed starting at one. Unassigned elements are assumed to be *null* (0). Arrays automatically grow in size to accommodate the largest index value that is used.

Forlist is an iterator that loops through the element in a list. Its first parameter is a *LIST* expression; its second parameter is a variable that iterates through the list elements; and its third parameter is an integer counter variable that counts the list elements starting at one.

Table Functions

`VOID table(TABLE_V)`
`VOID insert(TABLE, STRING, ANY)`
`ANY lookup(TABLE, STRING)`
	declare a table
	insert entry in table
	lookup and return entry from table

These functions provide general purpose, keyed tables. A table must be declared with the *table* function before it can be used.

Insert adds an object and its key to a table. Its first parameter is a table; the second parameter is the object's key; and the third parameter is the object itself. The key must be a string and the object can be any value. If there already is an object in the table with that key, the old object is replaced with the new.

Lookup retrieves an object from a table. Its first parameter is a table, and the second parameter is the object's key. The function returns the object with that key from the table; if there is no such object, *null* is returned.

GEDCOM Node Functions

`STRING xref(NODE)`
`STRING tag(NODE)`
`STRING value(NODE)`
`NODE parent(NODE)`
`NODE child(NODE)`
`NODE sibling(NODE)`
	cross reference index of
	tag of
	value of
	parent node of
	first child of
	next sibling of

`NODE savenode(NODE)`	copy a node structure
`fornodes (NODE, NODE_V) { }` `traverse (NODE, NODE_V, INT_V) { }`	loop through child nodes loop through all descendent nodes

These functions provide access to the components of a GEDCOM node. All take a GEDCOM node as their only parameter, and each returns a different value associated with the node.

Xref returns the cross reference index of the node, if any; *tag* returns the tag of the node; and *value* returns the value of the node, if any. If there is no cross reference, *xref* returns *null*; if there is no value, *value* returns *null*.

Parent returns the parent node of the node, if any; *child* returns the first child node of the node, if any; and *sibling* returns the next sibling node of the node, if any. Whenever there is no such related node, these functions return *null*. These three functions allow simple navigation through a GEDCOM node tree.

Savenode makes a copy of the node, and the substructure of nodes below the node, that is passed to it. Beware: the memory used to make the copy is never returned to the system.

Fornodes is an iterator that loops through the child nodes of a GEDCOM node. Its first argument is a node expression, and its second parameter is a variable that iterates through each direct child node of the first node.

Traverse is an iterator providing a general method for traversing GEDCOM trees. Its first parameter is a node expression; its second parameter is a variable that iterates over every node under the first node in a top down, left to right manner; and its third parameter is a variable that is set to the level of the current node in the iteration.

Event and Date Functions

`STRING date(EVENT)`
`STRING place(EVENT)`
`STRING year(EVENT)`
`STRING long(EVENT)`
`STRING short(EVENT)`
`EVENT gettoday()`
	date of, value of first *DATE* line
	place of, value of first *PLAC* line
	year or, 1st string of 3–4 digits in 1st *DATE* line
	date and place, values of 1st *DATE* and *PLAC* lines
	date and place of, abbreviated from
	returns the "event" of the current date

`VOID dayformat(INT)`
`VOID monthformat(INT)`
`VOID dateformat(INT)`
`STRING stddate(EVENT)`
	set day format for stddate calls
	set month format for stddate calls
	set date format for stddate calls
	date of, in current format

These functions extract information about the dates and places of events.

Date returns the value of the first *DATE* line in an event, a node in a GEDCOM record tree. *Date* finds the first *DATE* line one level deeper than the event node. *Place* returns the value of the first *PLAC* line in an event. *Year* returns the first three or four digit number in the value of the first *DATE* line in an event; this number is assumed to be the year of the event.

Long returns the verbatim values of the first *DATE* and *PLAC* lines in an event, catenated together and separated by a comma. *Short* abbreviates information from the first *DATE* and *PLAC* lines, catenates the shortened information together with a comma separator and returns it. An abbreviated date is its year; an abbreviated place is the last component in the value, further abbreviated if the component has an entry in the place abbreviation table.

Gettoday creates an event that has today's date in the *DATE* line.

The last four functions are used to format dates in a variety of ways. *Dayformat*, *monthformat*, and *dateformat* select style options for formatting the day, month, and overall date structure; *stddate* returns dates in the selected style. The day format codes passed to *dayformat* are:

0	leave space before single digit days
1	use leading 0 before single digit days
2	no space or leading 0 before single digit days

The month format codes passed to *monthformat* are:

0	number with space before single digit months
1	number with leading zero before single digit months
2	number with no space or zero before single digit months
3	upper case abbreviation (eg, *JAN, FEB*)
4	capitalized abbreviation (eg, *Jan, Feb*)
5	upper case full word (eg, *JANUARY, FEBRUARY*)
6	capitalized full word (eg, *January, February*)

The full date formats passed to *stddate* are:

0	da mo yr	6	modayr
1	mo da, yr	7	damoyr
2	mo/da/yr	8	yr mo da
3	da/mo/yr	9	yr/mo/da
4	mo-da-yr	10	yr-mo-da
5	da-mo-yr	11	yrmoda

Value Extraction Functions

`VOID extractdate(NODE, INT_V, INT_V, INT_V)`	extract a date
`VOID extractnames(NODE, LIST_V, INT_V, INT_V)`	extract a name
`VOID extractplaces(NODE, LIST_V, INT_V)`	extract a place
`VOID extracttokens(STRING, LIST_V, INT_V, STRING)`	extract tokens

Value extraction functions read the values of certain lines and return those values in extracted form.

Extractdate extracts date values from either an event node or *DATE* node. The first parameter must be a node; if its tag is *DATE*, the date is extracted from the value of that node; if its tag is not *DATE*, the date is extracted from the first *DATE* line one level below the argument node. The remaining three arguments are variables. The first is assigned the integer value of the extracted day; the second is assigned the integer value of the extracted month; and the third is assigned the integer value of the extracted year.

Extractnames extracts name components from a *NAME* line. Its first argument is either an *INDI* or a *NAME* node. If it is a *NAME* line, the components are extracted from the value of that node; if it is an *INDI* line, the components are extracted from the value of the first *NAME* line in the person record. The second argument is a list that will hold the extracted components. The third argument is an integer variable that is set to the number of extracted components. The fourth argument is a variable that is set to the index (starting at one) of the surname component; the / characters are removed from around the surname component. If there is no surname this argument variable is set to zero.

Extractplaces extracts place components from a *PLAC* node. The first argument is a node; if its tag is *PLAC*, the places are extracted from the value of the node; if its tag is not *PLAC*, places are extracted from the first *PLAC* line one level below the argument node. The second parameter is a list that will hold the extracted components. The third argument is an integer variable that is set to the number of extracted components. Place components are defined by the comma-separated portions of the *PLAC* value; leading and trailing white space is removed from the components, while all internal white space is retained.

Extracttokens extracts tokens from a string and places them in a list. The first argument is the string to extract tokens from. The second argument is the list to hold the tokens. The third argument is an integer variable that is set to the number of tokens extracted. The fourth parameter is the string of deliminater characters that *extracttokens* uses to break the input string into tokens.

User Interaction Functions

`VOID getindi(INDI_V [,STRING])`	identify person through user interface
`VOID getindiset(SET_V [,STRING])`	identify set of persons through user interface
`VOID getfam(FAM_V)`	identify family through user interface
`VOID getint(INT_V [,STRING])`	get integer through user interface
`VOID getstr(STRING_V [,STRING])`	get string through user interface
`INDI choosechild(INDI|FAM)`	select child of person/family thru user interface
`FAM choosefam(INDI)`	select family person is in as spouse
`INDI chooseindi(SET)`	select person from set of persons
`INDI choosespouse(INDI)`	select spouse of person
`SET choosesubset(SET)`	select a subset of persons from set of persons
`INT menuchoose(LIST [,STRING])`	select from a list of options

These functions interact with the user to get information needed by the program.

Getindi asks the user to identify a person. The first argument is a variable that is set to the person. The second is an optional string to use as a prompt. *Getindiset* asks the user to identify a set of persons. *Getfam* asks the user identify a family. *Getint* and *getstr* ask the user enter an integer and string, respectively.

Choosechild asks the user select a child of a family or person; its single argument is a person or family; it return the child. *Choosefam* has the user select a family that a person is in as a spouse; its argument is a person; it returns the family. *Chooseindi* has the user select one person from a set of persons; its argument in a set of persons; it returns the chosen person. *Choosespouse* has the user select a spouse of a person; its argument is a person; it returns the chosen spouse. *Choosesubset* has the user select a subset of persons from a set of persons; its argument is the chosen subset.

Menuchoose allows the user to select from an arbitrary menu. The first argument is a list of strings making up the items in the menu; the second, optional argument is a prompt string for the menu; *menuchoose* returns the integer index of the item selected by the user; if the user doesn't select an item, zero is return.

String Functions

`STRING lower(STRING)`
`STRING upper(STRING)`
`STRING capitalize(STRING)`
`STRING trim(STRING, INT)`
`STRING rjustify(STRING, INT)`
	convert to lower case
	convert to upper case
	capitalize first letter
	trim to length
	right justify in field

`STRING save(STRING)`
`STRING strsave(STRING)`
`STRING concat(STRING [,STRING]+)`
`STRING strconcat(STRING [,STRING]+)`
`INT strlen(STRING)`
	save and return copy of string
	same as save function
	catenate two strings
	catenate two strings
	number of characters in string

`STRING substring(STRING, INT, INT)` `INT index(STRING, STRING, INT)`	substring function index function

`STRING d(INT)`
`STRING card(INT)`
`STRING ord(INT)`
`STRING alpha(INT)`
`STRING roman(INT)`
	number as decimal string
	number in cardinal form (*one*, *two*, ...)
	number in ordinal form (*first*, *second*, ...)
	convert number to Latin letter (*a*, *b*, ...)
	number in Roman numeral form (*i*, *ii*, ...)

`STRING strsoundex(STRING)`	find SOUNDEX value of arbitrary string
`INT strtoint(STRING)` `INT atoi(STRING`	convert numeric string to integer convert numeric string to integer

`INT strcmp(STRING, STRING)`
`BOOL eqstr(STRING, STRING)`
`BOOL nestr(STRING, STRING)`
	general string compare
	compare strings for equality
	compare strings for inequality

These functions provide string handling. Many of them use an approach to memory management chosen to minimize memory use. A function using this approach constructs its output string in its own string buffer, reusing that buffer each time it is called. When a function using this approach returns a string value it returns its buffer. In consequence the strings returned by these functions must be either used or saved before the function is called again.

Lower and *upper* convert the letters in their arguments to lower or upper case, respectively. *Capitalize* converts the first character of the argument, if it is a letter, to upper case. *Lower* and *upper* use the buffer return method; *capitalize* operates on and returns it argument.

Trim shortens a string to the length specified by the second parameter. If the string is already of that length or shorter the string is not changed. *Rjustify* right justifies a string into another string of the length specified by the second parameter. If the original string is shorter than the justified string, blanks are inserted to the left of the original string; if the string is longer than the justified string, the original string is truncated on the right. *Trim* uses the buffer return method; *rjustify* creates and returns a new string.

Save creates a copy of the argument string and returns it. This function is required because built-in functions that return strings use the buffer return method; if a string is to be used repeatedly or long after it is returned from a function, it should first be saved by using the *save* function. *Strsave* is the same function as *save*.

Concat and *strconcat* catenate strings and return the result. They are identical functions. They may take two to 32 string arguments; *null* arguments are allowed. The arguments are catenated together into a single, newly allocated string, which is returned.

Strlen returns the length of the string argument.

Substring returns a substring of the first argument string. The second and third arguments are the indices of the first and last characters in the argument string to use to form the substring. The indexes are relative one. *Substring* uses the buffer return method.

Index returns the character index of the nth occurrence of a substring within a string. The index is the relative one character offset to the beginning of the substring. The first argument is the string; the second argument is the substring; and the third argument is the occurrence number.

D, *card*, *ord*, *alpha* and *roman* convert integers to strings. *D* converts an integer to a numeric string; *card* converts an integer to a cardinal number string (*eg*, *one*, *two*, *three*); *ord* converts an integer to an ordinal number (*eg*, *first*, *second*, *third*); *alpha* converts an integer to a letter (*eg*, *a*, *b*, *c*); and *roman* converts an integer to a Roman numeral (*eg*, *i*, *ii*, *iii*).

Strsoundex converts an arbitrary string to a SOUNDEX value. Non-ASCII text characters are ignored in the string.

Strtoint converts a numeric string to an integer. *Atoi* is identical to *strtoint*.

Strcmp compares two strings and returns an integer that is less than zero, equal to zero, or greater than zero, if, respectively, the first string is lexicographically less than, equal to, or greater than the second string. *Eqstr* and *nestr* return whether two strings are equal or not equal, respectively.

Output Mode Functions

`VOID linemode()`	use line output mode
`VOID pagemode(INT, INT)`	use page output mode with given page size
`VOID col(INT)`	position to column in output
`VOID row(INT)`	position to row in output
`VOID pos(INT, INT)`	position to (row, col) coordinate in output
`VOID pageout()`	output page buffer
`STRING nl()`	newline character
`STRING sp()`	space character
`STRING qt()`	double quote character
`VOID newfile(STRING, BOOL)`	send program output to this file
`STRING outfile()`	return name of current program output file
`VOID copyfile(STRING)`	copy file contents to program output file
`VOID print(STRING [,STRING]*)`	print string to standard output window

Reports can be generated in two modes, line mode and page mode. *Linemode* selects line mode and *pagemode* selects page mode; line mode is the default. The first parameter to *pagemode* is the number of rows per page; the second parameter is the number of columns per page. When in the line mode report output is written directly to the output file as the program runs, line by line. When in page mode output is buffered into pages which are written to the output file when *pageout* is called. Page mode is useful for generating charts (eg, pedigree charts or box charts) where it is convenient to compute the two-dimensional location of output.

Col positions output to the given column. If the current column is greater than the argument, *col* positions output to the given column on the next line. *Col* works in both modes.

Row positions output to the first character in the given row; *row* can only be used in page mode.

Pos positions output to a specified row and column coordinate; the first argument specifies the row, and the second specifies the column. *Pos* can only be used in page mode.

Nl write a new line character to the output file; *sp* writes a space character to the output file; and *qt* writes a quote character to the output file. Note that *n* and \” can be used within string values to represent the newline and double quote characters.

Newfile specifies the name of the report output file. Its first argument is the file’s name; its second argument is an append flag – if its value is non-zero the report appends to this file; if its value is zero the report overwrites the contents of the file. *Newfile* can be called many times; this allows a single report program to generate many report output files during one execution. Programs are not required to use *newfile*; if it is not used then LifeLines automatically asks for the name of the report output file.

Outfile returns the name of the current report output file.

Copyfile copies the contents of a file to the report output file; its argument is a string whose value is the name of a file; if the file name is not absolute nor relative, then the *LLPROGRAMS* environment variable, if set, will be used to search for the file; the file is opened and its contents copied to the report output file.

Print prints its argument string to the standard output window; *print* may have one to 32 arguments.

Person Set Functions and GEDCOM Extraction

`VOID indiset(SET_V)`
`SET addtoset(SET, INDI, ANY)`
`SET deletefromset(SET, INDI, BOOL)`
`INT lengthset(SET)`
	declare a set variable
	add a person to a set
	remove a person from a set
	size of a set

`SET union(SET, SET)`
`SET intersect(SET, SET)`
`SET difference(SET, SET)`
	union of two sets
	intersection of two sets
	difference of two sets

`SET parentset(SET)`
`SET childset(SET)`
`SET spouseset(SET)`
`SET siblingset(SET)`
	set of all parents
	set of all children
	set of all spouses
	set of all siblings

`SET ancestorset(SET)`
`SET descendentset(SET)`
`SET descendantset(SET)`
	set of all ancestors
	set of all descendents
	same as descendentset; spelling

`SET uniqueset(SET)`
`VOID namesort(SET)`
`VOID keysort(SET)`
`VOID valuesort(SET)`
	remove duplicates from set
	sort indiset by name
	sort indiset by key values
	sort indiset by auxiliary values

`VOID genindiset(STRING, SET)` `VOID gengedcom(SET)`	generate indiset from GEDCOM name string generate GEDCOM file from person set
`forindiset(SET, INDI_V, ANY_V, INT_V) { }`	loop through all persons in person set

These functions allow you to manipulate person sets. A person set is a potentially large set of persons; each person may have an arbitrary value associated with him/her. A person set must be declared with the *indiset* function before it can be used.

Addtoset adds a person to a set. The first argument is the set; the second argument is the person; and the third argument may be any value. The same person may be added to a set more than once, each time with a different value. *Deletefromset* removes a person from a set. The first argument is the set; the second argument is the person; if the third parameter is *true* all of the person's entries are removed from the set; if *false* only the first entry is removed. *Lengthset* returns the number of persons in a person set.

Union, *intersect* and *difference* return the set union, set intersection and set difference, respectively, of two person sets. Each functions takes two person sets as arguments and returns a third person set. The functions do not affect the values of their two argument sets.

Parentset, *childset*, *spouseset* and *siblingset* return the set of all parents, set of all children, set of all spouses and set of all siblings, respectively, of the set of persons in their argument. In all cases there is no change to the argument person set.

Ancestorset returns the set all ancestors of all persons in the argument set. *Descendentset* returns the set of all descendents of all persons in the argument set. *Descendantset* is the same as *descendentset*; it allows an alternate spelling.

Uniqueset sorts a person set by key value and then removes all entries with duplicate keys; the input set is modified and returned.

Namesort, *keysort* and *valuesort* sort a set of persons by name, by key and by associated value, respectively.

Each person in a person set has an associated value. When a person is added to a set with *addtoset*, the value is explicitly assigned. When new sets are created by other functions, a number of rules are used to associate values with persons as they are added to the new sets. For *parentset*, *childset* and *spouseset* the values are copied from the *first* input set person that causes the new person to be added to the set. For *union*, *intersect* and *difference*, the values are copied from the values in the first input set, except in the case of *union*, when persons are taken from the second set alone, in which case the values come from there. For *ancestorset* and *descendantset* the value is set to the number of generations the new person is away from the *first* person in the input set that the new person is related to. If the new person is related to more than one person in the input set, the value is set for the nearest relationship; that is, the value is as low as possible. *Valuesort* sorts a person set by the values of these auxiliary values.

Genindiset generates the set of persons that matche a string whose value is a name in GEDCOM format. *Genindiset* uses the same algorithm that matches names entered at the browse prompt or by the user interaction *getindiset* function.

Gengedcom generates GEDCOM format output, to the report output file, of all persons in the argument person set. The output contains a person record for each person in the set, and all the family records that link at least two of the persons in the set together.

Forindiset is an iterator that loops through each person in an indiset. The first parameter is an indiset. The second parameter is a variable that iterates through each person in the set. The third parameter iterates through the values associated with the persons. The fourth parameter is an integer variable that counts the iterations.

Record Update Functions

`NODE createnode(STRING, STRING)`	create a GEDCOM node
`VOID addnode(NODE, NODE, NODE)`	add a node to a GEDCOM tree
`VOID deletenode(NODE)`	delete a node from a GEDCOM tree

These functions allow you to modify an internal GEDCOM node tree.

Createnode creates a GEDCOM node; the two arguments are tag and value strings, respectively; the value string can be *null*. *Addnode* adds a node to a node tree. The first argument is the new node; the second is the node in the tree that becomes the parent of the new node; the third is the node in the tree that becomes the previous sibling of the new node; this argument is *null* if the new node is to become the first child of the parent. *Deletenode* removes a node from a node tree.

These functions change the internal form of a node tree; they *do not* modify original records in the database. These functions may be changed or extended in the future to allow database changes. Note: *deletenode* has a memory leak.

Record Linking Functions

`BOOLEAN reference(STRING)`
`NODE dereference(STRING)`
`NODE getrecord(STRING)`
	determine if string is a cross reference
	reference cross reference or key to node tree
	same as dereference

These functions allow you to recognize values that are cross references and to read the records they refer to. *Reference* returns *true* if its string argument is a cross reference value, that is, the internal key of one of the records in the database. *Dereference* returns the node tree of the record referred to by its cross reference string argument. *Getrecord* is a synonym for *dereference*.

Miscellaneous Functions

`VOID lock(INDI|FAM)` `VOID unlock(INDI|FAM)`	lock a person or family in memory unlock a person or family from memory
`STRING database()` `STRING version()`	return name of current database return version of LifeLines program
`VOID system(STRING)`	execute string as a UNIX shell command

Lock and *unlock* are used to lock a person or family into RAM memory, and to unlock a person or family from RAM memory, respectively.

Database returns the name of the current database, useful in titling reports. *Version* returns the version of the running LifeLines program, *eg*, *3.0.2*.

System executes its string argument as a UNIX shell command by calling the UNIX *system* system call.