L-Soft international, Inc.

Developers Guide
for
LISTSERV®, version 14.5

22 February 2006
LISTSERV 14.5 Release

The reference number of this document is 0602-MD-02.

All of L-Soft's manuals for LISTSERV are available in ascii-text format via LISTSERV and in popular word-processing formats via **ftp.lsoft.com**. They are also available on the World Wide Web at the following URL:

URL: http://www.lsoft.com/manuals/index.html

L-Soft invites comment on its manuals. Please feel free to send your comments via e-mail to **MANUALS@LSOFT.COM**, and mention which manual you are commenting on.

"Hot fix" revisions to this and other L-Soft manuals are posted as they are made to the master document, on the announcement-only mailing list:

LSOFT-DOC-UPDATES@PEACH.EASE.LSOFT.COM

A word about formatting: This manual was written in Microsoft Word 2000, and originally formatted to be printed on 8-1/2"x11" paper on a Hewlett-Packard LaserJet 4M+. When printing the manual on a different type of printer, or converting to a different word-processing program, it is highly likely that the formatting and pagination will change and it will be necessary to update the table of contents and figures as well as the index prior to printing. The author has taken great pains to ensure that the pagination and formatting works properly with the particular printer mentioned above, and cannot be held responsible for what is, in the end, a limitation of the software used to produce the manual.

Reference Number 0602-MD-02

Table of Contents

1. LISTSERV's Archive Search Functions 6
   - *1.1. Preface* 6
   - *1.2. A basic search session* 6
   - *1.3. Narrowing the search* 7
   - *1.4. The SEARCH command* 7
     - 1.4.1. Basic search functions 7
     - 1.4.2. Date specifications 10
     - 1.4.3. Keyword search specifications 10
     - 1.4.4. Phonetic search 12
     - 1.4.5. What to do about "100 matches (more available)." 13
     - 1.4.6. Specifying the last *n* posts as a range 14
     - 1.4.7. Exact syntax description 14
2. LISTSERV "Commands-Job" Feature and CJLI Interpreter ... 19
   - *2.1. Introduction* 19
   - *2.2. Overview -- the JOB entity* 19
   - *2.3. Control Cards -- general syntax rules* 20
   - *2.4. The JOB control card* 22
   - *2.5. The EOJ control card* 24
   - *2.6. The DD control card* 24
   - *2.7. The //*MSG control card* 25
   - *2.8. Special considerations* 25
3. Relayed File Distribution and the DISTRIBUTE Command 27
   - *3.1. Introduction* 27
   - *3.2. What is Relayed File Distribution?* 27
   - *3.3. A technical description of Relayed File Distribution Requests* 28
   - *3.4. Creating and sending a mail RFDR ("DISTRIBUTE") job* 31
     - 3.4.1. RFDR job options 35
     - 3.4.2. DISTRIBUTE Command Options 37
   - *3.5. Advanced LISTSERV applications using DISTRIBUTE* 40
     - 3.5.1. "List-free" bounce processing for one-shot lists (1.8d and following) 41
4. DBMS and mail-merge support 42
   - *4.1. Overview* 42
     - 4.1.1. DBMS support 42
     - 4.1.2. Why require Oracle 8 or higher? 43
     - 4.1.3. Mail-merge 43
   - *4.2. Pre-installation tasks* 44
     - 4.2.1. Selecting a suitable DBMS product 45
   - *4.3. Installation* 47
     - 4.3.1. Windows NT/2000/XP 47
     - 4.3.2. OpenVMS Alpha 47
     - 4.3.3. Unix 48
     - 4.3.4. Verification 49
   - *4.4. Post-installation tasks* 49
     - 4.4.1. Mail-merge 49
     - 4.4.2. ODBC interface 50
     - 4.4.3. OCI interface 50

4.4.4. CLI interface 50
4.4.5. unixODBC (UODBC) interface 50
4.4.6. Connecting to multiple simultaneous database sources 51
4.4.7. Generic DBMS post-installation tasks 52
4.4.8. Performance options 53
4.5. Creating DBMS lists 54
4.5.1. Configuring a list to use the DBMS 54
4.5.2. Importing subscribers into a DBMS list 55
4.5.3. ADD IMPORT benchmark 56
4.5.4. Updating DBMS lists from an external application 57
4.5.5. Format of the OPTIONS column 57
4.5.6. Sample OPTIONS column settings 58
4.5.7. Preserving options when migrating from non-DBMS to DBMS lists 60
4.6. Using the mail-merge functions 60
4.6.1. Using the web interface 60
4.6.2. Sending DISTRIBUTE jobs to LISTSERV 61
4.6.3. Using substitutions 65
4.6.3.1. New substitutions available in LISTSERV 1.8e-2003a "level set" release 68
4.6.4. Using conditional blocks 69
4.7. Installing and configuring unixODBC with LISTSERV and MySQL 72
4.7.1. MySQL DBMS 72
4.7.1. MyODBC Driver 72
4.7.3. unixODBC 72
4.7.4. Creating a unixODBC System DSN for LISTSERV 73
4.7.5. Creating a MySQL User and Database/Schema for LISTSERV 73
4.7.6. Configure LISTSERV support for unixODBC 74
4.7.7. Connecting to External MySQL Databases 74
4.7.8. References 74
5. List Exits 76
5.1. What is a "list exit"? 76
5.2. List Exit Points 80
5.2.1. ADD/SUBSCRIBE exit points 80
5.2.2. DELETE/SIGNOFF/CHANGE entry points 81
5.2.3. Other exit points 82
5.2.4. General remarks 83
5.3. Local command definition (non-VM) 83
5.4. SPAM_EXIT (LISTSERV 14.3 and following) 84
5.4.1. Formal documentation 84
5.4.2. Practical application 85
5.5. A practical example: HAPPY99 88
6. The LISTSERV TCPGUI interface 93
6.1. Setting up the TCPGUI interface 93
6.2. Running `lcmdx` 93
6.3. Sending LISTSERV commands directly from your application 94
6.4. Advanced TCPGUI Programming Issues 95
6.4.1. Creating or replacing a list header 96
6.4.2. Adding or replacing a password 97
6.4.3. Bulk operations 97
6.4.4. Commands that respond over e-mail 97
6.4.5. Application-friendly commands 98
6.4.6. Error handling 100

6.5. LCMDX.C 100
Revision History 104
Index 105

1. LISTSERV's Archive Search Functions

DOCUMENTATION NOTE

The information in this chapter applies to the "SEARCH" command introduced in LISTSERV 1.8c. It does not contain information pertaining to the old VM "DATABASE" command syntax, nor to LISTSERV DBMS support introduced in version 1.8d.

1.1. Preface

This chapter is an introduction to the LISTSERV archive search functions first implemented in L-Soft's LISTSERV 1.8c and later. It is intended to be a reference document for general users with little or no knowledge of database systems. It does not contain any technical information that general users would have to worry about.

Note: For LISTSERV on VM only, this chapter can be used with current versions (1.8c or later) of LISTSERV. For 1.8b or earlier VM servers, the "INFO DATABase" command will retrieve the manual for the old command syntax. The old database documentation is also available at L-Soft's ftp site in the file

ftp://ftp.lsoft.com/documents/old-listdb.memo

This chapter will discuss the syntax and operational characteristics of the LISTSERV database subsystem. It is assumed that the reader is familiar with his or her e-mail client and familiar with sending commands to a LISTSERV server.

If you just need a "quick start", read the next two sections for basic instructions. If you want a detailed tutorial on how to use the SEARCH command itself, you might want to skim the next two sections and then start reading with the section entitled "The SEARCH Command".

1.2. A basic search session

Let's say that you are looking for messages in the EASE-HOME mailing list that pertain to the list header keyword "Digest=".

To search for the term "Digest=" in the EASE-HOME list on HOME.EASE.LSOFT.COM, create a new mail message addressed to LISTSERV@HOME.EASE.LSOFT.COM and in the body (not the subject) of the message, simply type:

Search 'Digest=' in EASE-HOME

LISTSERV might respond to you with the following:

```
+-----------------------------------------------------------------+
| > Search 'Digest=' in EASE-HOME                                 |
| -> 10 matches.                                                  |
|                                                                 |
| Item # Date     Time    Recs Subject                            |
| ------ ----     ----    ---- -------                            |
| 000058 96/01/26 14:44   41   What happened                      |
| 000059 96/01/26 18:14   38   Re: What happened                  |
| 000066 96/02/02 22:51   31   Digest Problem                     |
| 000074 96/02/03 15:01   75   Re: Digest Problem                 |
| 000075 96/02/03 18:52   49   Re: Digest Problem                 |
| 000076 96/02/03 16:27   52   Re: Digest Problem                 |
| 000112 96/02/13 23:37   29   not receiving mail                 |
| 000126 96/02/25 20:20   63   error/bounce msg posted to list How?  |
```

```
| 000172 96/03/13 09:11   12   Digest Mailing Time                    |
| 000483 96/06/22 17:36   34   Header Info                            |
|                                                                     |
| To order a copy of these postings, send the following command:      |
|                                                                     |
|             GETPOST EASE-HOME 58-59 66 74-76 112 126 172 483        |
|                                                                     |
| >>> Item #58 (26 Jan 1996 14:44) - What happened                    |
|    I never touched the Limits= command or the notebook= All I did   |
| was try and add: Digest= Yes,Daily                                  |
|                  ^^^^^^^                                            |
| I have tryed this several times with the same reply message:        |
|                                                                     |
| >>> Item #59 (26 Jan 1996 18:14) - Re: What happened                |
| >   I never touched the Limits= command or the notebook= All I did  |
| >was try and add: Digest= Yes,Daily                                 |
|                   ^^^^^^^                                           |
+---------------------------------------------------------------------+
```

Figure 1.1. Sample SEARCH output (further output deleted)

Note that LISTSERV includes excerpts from the indexed postings showing the context of the search term(s). We've deleted all but the first 2 in the example above to save space.

You would then use the GETPOST command to order the specific posts you wanted to read. For instance, we want to read posts numbered 66, 74 through 76, and 126. You would make another new message (or reply to the response from LISTSERV without quoting the text) and type in the body:

```
GETPOST EASE-HOME 66 74-76 126
```

LISTSERV would then respond with the desired postings. For the non-VM servers, GETPOST is analogous to the old database command "PRINT". There is no corresponding command for the old database command INDEX, since the response to a SEARCH command includes the index of matching postings.

1.3. Narrowing the search

It is possible to add further parameters to your search in order to narrow it. You can limit a search by date with a "since. . . " predicate. Likewise, you can limit by sender and/or by the subject line with a "where . . ." predicate. For instance:

```
Search 'Digest=' in LSTOWN-L since 94/01/01
Search 'Digest=' in LSTOWN-L where sender contains 'Thomas'
Search * in LSTOWN-L where sender is ERIC@SEARN
Search * in LSTOWN-L since 94/01/01 where subject contains 'Digest'
```

are all valid search commands that will (depending on how well you've crafted your predicate) dramatically reduce the number of entries returned to you.

1.4. The SEARCH command

This chapter will introduce the formal syntax of the SEARCH command. (Note: The minimum abbreviation of this command is "S".)

The syntax of this command is a bit complex, and will be introduced step by step.

1.4.1. Basic search functions

The two most important things you have to indicate when you search list archives are:

1. The name of the list whose archives you want to search.

2. What you want to search the individual documents for.

The name of the list to be searched is specified after the words or phrases to be sought and is prefixed with an IN keyword. For example, we might do this:

```
Search Rosemary in MOVIES
```

This would select all the entries from list "MOVIES" containing the string "ROSEMARY".

Now if you just wanted to see the list of all the movies you can see, you could have used an asterisk as search argument to select all the entries in the list:

```
Search * in MOVIES
```

Note that the mailing list name doesn't have to be uppercased.

If you want to "narrow" your previous search, i.e. perform additional tests on the documents that have been previously selected, you must omit the IN keyword. In that case, the search will be applied to the previous "hits" and will create a new "hit list".

But in most cases, we will want to search for something longer than one word, for example part of a "key" sentence.

```
Search Hardware problem with a 4381 in IBMFORUM
```

Another problem is that we might not remember the exact original sentence. This is not very important, since LISTSERV will search each word individually: in the above example, any entry that contained the words "hardware", "problem", "with", "a" and "4381" would have matched the search, even if the words appeared in a different order.

But what if the original document had "4381-13" in it, instead of "4381"? This is again no problem, as LISTSERV does not require the word to be surrounded by blanks to find a match. Case is also ignored when performing the search operation. That is, "problem" would have found a match on "problems"... and "with" would have found a match on "without" or "withstand"! This may sound like inconsistent behaviour, but you should keep in mind that it is always possible to "narrow down" a search operation. However, once a document has been excluded from the list of "hits", it is very difficult to bring it back.

Now what if I want to search for an exact string? For example, I am interested in the string "in C". It is very likely that just any document in the database will contain both a "in" and the letter C. But what I am interested in is things which have been written, or programmed, or implemented, "in C". In that case, it is possible to force LISTSERV to group words together by quoting them, as in:

```
Search 'in C' in UTILITY
```

This method can also be used to insert extra blanks between or before words: leading and trailing blanks are normally removed automatically, but they are preserved inside quoted strings. Please note that quotes must be doubled when specified inside quoted strings, as in:

```
Search 'Rosemary''s baby' in MOVIES
```

The search for 'in C' resulted in over fifty hits, because a match was erroneously found against "in clear", "in core", etc. However, I do not want to search for 'in C ' because there might be hits with "in C." or "in C," in the database and I don't want to miss them. If the search respected the capital C, it would no longer find all those irrelevant hits. To do this, you must enclose your search string in double-quotes instead of single quotes, for example:

```
Search "in C" in UTILITY
```

Note that single quotes should not be doubled inside double-quoted strings, and vice-versa. Only quotes of the same type as the string should be doubled.

It is important to understand the difference between the two types of quoting. If you request a search for 'TEXT', you will find a match on "TEXT", "Text", "text" or even "teXt". This is the same behaviour as unquoted text. However, if you request a search for "TEXT", it will only find a match on "TEXT", not on "text" or "Text".

Quoting is also the only way to search for a reserved keyword like "IN": if you tried "Search in in UTILITY", LISTSERV would report that database "IN" does not exist and would reject the command. This is because the keyword IN indicates the end of your search arguments. If you quote it, however, it will not be recognized and will be searched as you wanted it done. Similarly, if you want to search for an asterisk, you will have to quote it since

```
Search *
```

indicates that all entries should be selected.

Now the problem is that there may be sentences starting with a capital I, e.g. "In C, it would be coded this way:". How can I catch these sentences?

Actually, you have been using "complex search expressions" from the beginning without even being aware of it. When you specified a search on

```
Hardware problem with a 4381
```

, you had in fact been asking LISTSERV for: "Hardware NEAR problem NEAR with NEAR a NEAR 4381". The "NEAR" is implicit, but it may be overriden.

[Note that this is a change from the way the "locate" clause was implemented in 1.8b and earlier. Earlier versions used a default of "AND" instead of "NEAR" between discrete search terms. The difference is that

```
Search JOE SMITH in XYZ-L
```

looks for JOE and SMITH close to each other rather than simply looking for instances of both terms in the entire document. You can still use AND explicitly. Note that 'a NEAR b NEAR c' is defined as '(a NEAR b) AND (b NEAR c)', so the NEAR operator is not fully commutative.]

You may even use parenthesis if needed:

```
Search ("in C" or "In C") and program in UTILITY
```

The "NEAR" can still be implied, as in:

```
+-------------------------------------------------------------------+
| Search wooden chair (blue or green) in CHAIRS                     |
| Search (wooden chair) or (plastic chair) in CHAIRS                |
| Search plastic chair (blue or green but not streaked) in CHAIRS   |
|                                                                   |
| The following commands are strictly equivalent:                   |
|  Search (wooden chair) or (plastic chair not blue) in CHAIRS      |
|  Search chair (wooden or (plastic not blue)) in CHAIRS            |
|  Search chair (wooden or (plastic but not blue)) in CHAIRS        |
|  Search chair NEAR (wooden OR (plastic AND NOT blue)) in CHAIRS   |
+-------------------------------------------------------------------+
```

Figure 1.2. Sample SEARCH commands using complex document search arguments

1.4.2. Date specifications

Since each document has been assigned a "date/time" field, it is possible to select documents based on this date field. This is accomplished by appending "date search rules" to the search expression, as in:

```
+-------------------------------------------------------------------+
| Search problem (serious or severe) in BBOARD since july           |
| Search problem in BBOARD since oct 85                             |
| Search symptom in BBOARD since 12/28                              |
| Search error report from 12 january to august in BBOARD           |
| Search user complaint until 18 sept in BBOARD                     |
| Search data check since today 11:53 in EREP                       |
+-------------------------------------------------------------------+
```

Figure 1.3. Sample SEARCH commands using date search arguments

The default values for omitted arguments are always chosen so as to exclude as few entries as possible. For example, "July" would mean "1 July 00:00:00" in a SINCE specification, and "31 July 23:59:59" in an UNTIL clause. The only exception is the year field, which always defaults to the current year.

1.4.3. Keyword search specifications

The last thing you may wish to search is the "keywords" list. For example, you might want to select those plastic chairs which cost less than 50 dollars. It is assumed that the price will vary often (maybe almost daily), and that it is therefore kept externally from the document describing the chair. Thus, you would have a "Price" keyword which you could search in the following way:

```
Search plastic chair in CHAIRS where price < 50
```

You may of course use complex expressions (with parenthesis) in the WHERE clause. There are new comparison operators available for this clause, like IS, CONTAINS, all the usual arithmetical comparison operators, and some more. However, the AND operation is no longer implied, but it can still be specified explicitly of course:

```
Search plastic chair in CHAIR where price < 50 and avail > 4
```

The problem now is that, as the search commands become more and more complex, they will no longer fit in a single line. To solve this problem, we begin the command with the string "// " (two front-slashes and a space) and follow it with the SEARCH command and the search specifications. Any database command ending in a comma indicates that more is to follow on the next line. This process can be repeated several times if desired.

```
+-------------------------------------------------------------------+
| // Search chair (wooden or (blue or green but not streaked)) ,    |
|        in CHAIRS ,                                                |
|        where price < 50 & avail > 4                               |
```

```
|                                                                    |
| // Search chair (wooden or (blue or green but not streaked)) ,     |
| in CHAIRS where price < 50 & avail > 4                             |
|                                                                    |
| // Search chair (wooden or ( ,                                     |
| blue or green but not streaked) ,                                  |
| ) ,                                                                |
| in CHAIRS where price < 50 & avail > 4                             |
+--------------------------------------------------------------------+
```

Figure 1.4. Sample SEARCH commands with continuation lines: All these commands are strictly identical, although the first one is obviously more legible. |

The only "trick" about this continuation line business is that you should always keep quoted strings on a single line. The process of identifying continuation lines and concatenating them afterwards may cause unwanted blanks to be inserted in the command line, which is no problem outside a quoted string since blanks are ignored, but might cause erroneous results in a quoted string.

If you want to search for several possible values in a given keyword, you do not have to repeat the keyword name and operator:

```
+--------------------------------------------------------------------+
| // Search * in BBOARD where ,                                      |
| subject contains (PC or (Personal and computer))                   |
|                                                                    |
| is strictly equivalent to:                                         |
|                                                                    |
| // Search * in BBOARD where ,                                      |
| subject contains PC or ,                                           |
| (subject contains Personal and subject contains computer)          |
+--------------------------------------------------------------------+
```

Figure 1.5. Sample use of "factorization"

However, it should be noted that this "factorization" is performed according to the rules of logic, which may not necessarily match those of English grammar. This removes any possible ambiguity as to the meaning of these clauses. Let's consider the following example:

```
machine does not contain (IBM and DEC)
```

This clause will get translated into:

```
machine does not contain IBM and machine does not contain DEC
```

In English you would probably say "machine contains neither IBM nor DEC" . This is how LISTSERV will understand it. However, if you read the clause aloud, you will probably not pronounce the parenthesis and will end up saying "machine does not contain IBM and DEC", in other words, "machine does not contain both IBM and DEC" , which is a totally different thing (and would most probably be true all the time). The "English meaning" could be obtained with the following clause:

```
not (machine contains (IBM and DEC))
```

In the former case, the negative "does not contain" operator is inserted inside the parenthesis. In the latter, only "contains" is moved, and the negation remains outside.

```
+--------------------------------------------------------------------+
| // Search gateway problem ,                                        |
|    in BBOARD ,                                                     |
|    since sept 86 ,                                                 |
|    where sender contains (john or paul but not mick) ,             |
|    and subject does not contain lost                               |
```

```
|                                                                      |
| -> 5 matches.                                                        |
|                                                                      |
| Item #   Date   Time  Recs   Subject                                 |
| ------   ----   ----  ----   -------                                 |
| 000012 87/10/18 13:09   12   The gateway has stopped working         |
| 000017 87/08/24 09:18    9   Glory glory alleluja! Again!!!          |
| 000018 87/10/18 13:09    8   You know what? It WORKS!!!              |
| 000024 87/10/18 13:09    7   Guess what happened today?              |
| 000205 87/10/04 16:59    9   Who's going to babysit it today?        |
|                                                                      |
|                                                                      |
| You  might now wish to narrow your search down to exclude postings   |
| whose subject contains "work".  For instance,                        |
|                                                                      |
| // Search gateway problem ,                                          |
|   in BBOARD ,                                                        |
|   since sept 86 ,                                                    |
|   where sender contains (john or paul but not mick) ,                |
|   and subject does not contain (lost or work)                        |
|                                                                      |
| -> 3 matches.                                                        |
|                                                                      |
| Item #   Date   Time  Recs   Subject                                 |
| ------   ----   ----  ----   -------                                 |
| 000017 87/08/24 09:18    9   Glory glory alleluja! Again!!!          |
| 000024 87/10/18 13:09    7   Guess what happened today?              |
| 000205 87/10/04 16:59    9   Who's going to babysit it today?        |
+----------------------------------------------------------------------+
```

Figure 1.6. Sample SEARCH commands with keyword search clauses

1.4.4. Phonetic search

There may be cases where you are looking for a certain value of a keyword, the exact spelling of which you cannot remember. In these cases, it may be useful to try a phonetic search. A phonetic search will yield a match for anything that "sounds like" your search string, as dictated by a predefined algorithm which is of course not perfect. It may give a hit for something which does not actually sound like your search string, or, more rarely, omit a keyword which did sound like what you entered. The main reasons for this are that the algorithm must be fast to execute on the machine and therefore not too sophisticated, and that the way a given word is pronounced depends on the idiom in which the word was written. For example, the phonetical transcription of the name "Landau" will be different in French, English, German and Russian. Thus, it is impossible to decide whether a word sounds like another if the language in which the words are pronounced is not known (and of course LISTSERV does not have, a priori, any way to know it).

Phonetic searches are performed through the use of the SOUNDS LIKE and DOES NOT SOUND LIKE operators, which are syntactically similar to CONTAINS and DOES NOT CONTAIN. That is, you could do something like:

```
Search * in PHONEBOOK where NAME sounds like WOLF
```

There is a little trick with the SOUNDS LIKE operator that you should be aware of. If your search string (WOLF in our above example) is a single word, it will be compared individually to all the words in the reference string (i.e. the data from the database), and will be considered a hit if it "sounds like" any of the words in the reference string. Thus, the search word "Ekohl" sounds like the reference string "Ecole Normale Superieure" because it matches the first word. If the search string contains more than one word, the search and reference strings will be compared phonetically as a whole (and "Ekohl Dzentrahll" will therefore not match "Ecole Normale Superieure"). Note that any search string containing more than a single word must be quoted, as explained in the previous sections of this chapter.

```
+-----------------------------------------------------------------+
| > Search * in BITEARN where site sounds like (COHRNEAL and      |
| LAPORRADRY)                                                     |
|                                                                 |
| -> 3 matches.                                                   |
|                                                                 |
| Ref# Conn  Nodeid   Site name                                   |
| ---- ----  ------   ---------                                  |
| 0292 87/03 CRNLASSP Cornell University Cornell Laboratory of Atomic|
| 0301 87/03 CRNLION  Cornell University Cornell Laboratory of Plasma|
| 0307 87/06 CRNLNUC  Cornell University Laboratory of Nuclear Studes|
|                                                                 |
| > Search * in BITEARN where SITE sounds like HOPTIKK            |
|                                                                 |
| -> 2 matches.                                                   |
|                                                                 |
| Ref# Conn Nodeid Site name                                      |
| ---- ---- ------ ---------                                      |
| 0751 87/09 FRIHAP31 Assistance Publique - Hopitaux de Paris     |
| 2120 87/04 UOROPT University of Rochester The Institute of Optics  |
|                                                                 |
| > Search * in BITEARN where SITE sounds like SCHIKAGO           |
|                                                                 |
| -> 1 match.                                                     |
|                                                                 |
| Ref# Conn Nodeid Site name                                      |
| ---- ---- ------ ---------                                      |
| 0140 86/03 BMLSCK11 Studiecentrum voor Kernenergie (SCK/CEN), Mol, |
+-----------------------------------------------------------------+
```

Figure 1.7. Sample SEARCH commands involving phonetic match

In the figure above, the first command shows an example of accurate phonetic match, where the result is exactly what the user expected. In the second example, the user found what he was looking for ("Optics"), but an additional unwanted entry was selected. This is by far the most common case. The last command is a typical example of phonetic clash, where the algorithm did not translate the search string into phonetics as the user expected it, with the result that the desired name ("Chicago") was not found and that completely irrelevant entries were presented instead.

The phonetic matching algorithm used by LISTSERV is a slightly modified version of SOUNDEX -- a well-known algorithm that provides reasonably accurate matches at a very low CPU cost. Although it gives best results with the English language, for which it was originally designed, it is not too strongly tied to it and can still be used with other languages. It is of course absolutely impossible to write an program that would work for all the languages in the world, or even for the most widely used ones, since their interpretation of the most common combinations of letters are completely incompatible.

1.4.5. What to do about "100 matches (more available)."

LISTSERV limits the number of matching records in a given response to no more than 100. This is done primarily to stop hackers from tying up the server with SEARCH requests on lists with thousands of archived postings, but it also keeps the size of the response down to a manageable level. For instance, sending a "SEARCH *" command for an old, very large list could result in a response measuring in megabytes if not for the 100-record limitation.

There are a couple of different approaches to a solution:

- (Preferred) Narrow your search parameters as explained in 1.3 and following.

- Specify the first record for your search so that LISTSERV knows to start in a certain place. For instance, if you sent a search command like

  ```
  search * in lstown-l where sender contains nathan@example.com
  ```

  that resulted in more than 100 "hits", and the last four hits were something like

  ```
  +-------------------------------------------------------------------+
  | 007696 95/08/23 16:02   13   Re: How to send 'urgent' messages to  |
  | digest users?                                                     |
  | 007698 95/08/23 18:10   52   Re: Blocking expletives              |
  | 007699 95/08/23 20:00   41   Re: How to send 'urgent' messages to  |
  | digest users?                                                     |
  | 007716 95/08/25 10:34   33   Re: ignore subsequent lines to       |
  | listserv?                                                         |
  +-------------------------------------------------------------------+
  ```

  you could send a followup search command using the following syntax:

  ```
  search * in lstown-l.7716- where sender contains nathan@example.com
  ```

  to get the next 100 hits starting with message number 7716. Note that it is important to include the trailing hyphen after the starting message number, as otherwise you will get back a response containing only a reference to the message number you specified.

1.4.6. Specifying the last *n* posts as a range

Starting with LISTSERV 1.8d it is possible to specify that your search criteria be applied to only the last *n* posts in the archive. For instance, say you are only interested in checking the last 50 postings of LSTOWN-L to see if nathan@lsoft.com posted. You would send

```
search * in lstown-l.last50- where sender contains nathan@lsoft.com
```

Again you would have to use the hyphen after the number. The number *n* can be any integer, but as with any other search, if LISTSERV finds more than 100 hits only the first 100 will be returned to you. Thus it would be possible for *n* to be 1000, or even 10,000, but you will still have to narrow your search if you want more hits.

1.4.7. Exact syntax description

This section describes the exact syntax of the "SEARCH" command in technical terms. You can skip it if you are not interested in learning about the details of this command.

General syntax

```
+--------+-------------------------------------------------------+
|        |                                                       |
| Search |  search-rules  <optional-rules>                       |
|        |                                                       |
|        |  Optional rules are:                                  |
|        |                                                       |
|        |    date-rules                                         |
|        |    keyword-rules                                      |
+--------+-------------------------------------------------------+
```

The optional "date-rules" and "keyword-rules" arguments may appear in any order.

Date rules specification

You may optionally restrict the search to only those entries that lay within a given interval of time. This is accomplished by specifying one of the following date rules:

```
SINCE date-spec <time-spec>
FROM date-spec1 <time-spec1> TO date-spec2 <time-spec2>
UNTIL date-spec <time-spec>
```

The format of a "date-spec" is quite complex because of the number of different ways date/time specifications are usually expressed:

```
TODAY
yy
dd mm
<dd><->monthname<-><yy>
mm/yy
mm-yy
yy/mm/dd
yy-mm-dd
yyyymmdd
```

(**yyyymmdd** is accepted by LISTSERV version 1.8d and following.)

Month names can be abbreviated to any length. If there is an ambiguity, the first month in chronological order is retained. For example, "J" would mean "January", "JU" would be "June" and "JUL" would unambiguously select "July".

The format of a "time-spec" is simply <hh:mm<:ss>>.

```
+---------------------------------------------------------------------+
| FROM 14 july TO oct 97                                              |
| SINCE 96                                                            |
| UNTIL 23-JUN-97                                                     |
| SINCE today 11:30                                                   |
+---------------------------------------------------------------------+
```

Figure 1.8. Sample date clauses

NOTE: Case is irrelevant in date specifications. The keywords (SINCE, UNTIL, etc) have been capitalized only for better legibility, and can be entered in lower case if desired.

Keyword rules specification

You may request the actual document search to take place only for those entries which match a set of "keyword comparison" rules. The syntax is the following:

```
WHERE kwd-expression
WITH kwd-expression
```

"kwd-expression" is, generally speaking, an mathematical expression of keyword/value comparisons, possibly bound by logical operators. Comparison operators have a higher precedence than logical operators, that is, "A>10 AND B=20" is interpreted as "(A>10) AND (B=20)". The available comparison operators are listed below. All the operators appearing on a given line are synonyms.

```
+---------------------------------------------------------------------+
| = IS                                                                |
| ^= <> IS NOT                                                        |
```

```
| >                                                                  |
| <                                                                  |
| >=                                                                 |
| <=                                                                 |
| SOUNDS LIKE                                                        |
| DOES NOT SOUND LIKE                                                |
| CONTAINS                                                           |
| DOES NOT CONTAIN                                                   |
+--------------------------------------------------------------------+
```

Figure 1.9. Comparison operators for WHERE clauses

All these operators are self-explanatory, except the last two which allow you to search the keyword value for a given "substring". That is, "Sender contains jeff" would be true if the value of the "Sender" keyword was "Jeff Smith" or "Jeffrey Donaldson". The case is ignored during the comparison unless the search operand is double-quoted.

If no valid comparison operator is specified between two arguments, "IS" (identity) is assumed.

The available logical operators are:

```
+--------------------------------------------------------------------+
| ^ NOT                                                              |
| & AND BUT                                                          |
| | / OR                                                             |
+--------------------------------------------------------------------+
```

Figure 1.10. Logical (boolean) operators

Please note that the logical operators AND and OR have equal precedence and are evaluated left-to-right.

Finally, keywords and operators can be "factorized" when the same comparison is to be applied to a given keyword and a series of comparands. For example, you might enter:

```
Search * where sender contains ('CS Dept' and (Jack or Phil))
```

This is internally expanded to:

```
// SEARCH * WHERE sender CONTAINS 'CS Dept' AND ,
   (sender CONTAINS Jack OR sender CONTAINS Phil)
```

Please note that the expression must always be enclosed in parenthesis, even if it is a simple one:

```
Search * where sender contains (Joe or Morris)
```

This stems from the fact that comparison operators have a higher priority than logical (boolean) ones.

```
+--------------------------------------------------------------------+
| WHERE Sender is "Arthur Dent" ,                                    |
| and Subject does not contain tea                                   |
|                                                                    |
| WITH Refcode 8467272 and Location Roubaix                          |
|                                                                    |
| WITH (QTY > 100 | PRICE > 1000) & MAT = COPPER                     |
|                                                                    |
| Where Sender is (Atiaran@Land or Elena@Land) ,                     |
| and Subject contains ('Be true' but not Ur-Lord)                   |
+--------------------------------------------------------------------+
```

Figure 1.11. Sample WHERE clauses

Search rules specification

Finally, you must specify what is to be searched inside the document. If you do not want anything to be sought at all (e.g. if you are only selecting known items from the database), you can specify an asterisk as a placeholder to waive the search. Otherwise you must specify a mathematical expression where arguments are search strings, possibly bound by logical operators (see Figure 10 for a comprehensive list). The default operator is AND, so that a search for "INTERPRET STEM PROBLEM" will select all entries where "INTERPRET", "STEM" and "PROBLEM" can be found (not necessarily in the same line).

```
+-------------------------------------------------------------------+
| Search *                                                          |
|                                                                   |
| Search 'I/O' Error                                                |
|                                                                   |
| // Search Interpret (performance or tips ,                        |
| but not (bug or question))                                        |
+-------------------------------------------------------------------+
```

Figure 1.12. Sample document-search clauses

Reserved words and quoting

When to quote strings

Keyword names and search arguments need not be quoted, unless:

- They are formed of more than one word (search arguments only).
- They contain leading or trailing blanks (search arguments only).
- Their name matches one of the "reserved keywords" of the LISTSERV database system, and appears in a context where it can be mistaken for such. The "reserved keywords" are: FROM, IN, SINCE, TO, UNTIL, WHERE, WITH.
- They contain a parenthesis, logical operator or comparison operator symbol. More generally, you should quote any string that contains one of the following characters:

( ) < > = | & ^ /

Any non-quoted word will be stripped of leading and trailing blanks and converted to uppercase before the search.

Single-quoted strings

Strings quoted in single-quotes (') are converted to upper case and cause case to be ignored during the search. That is, they behave in the same manner as un-quoted strings as far as the search algorithm is concerned. As a rule of thumb, any string can be single-quoted if desired, even if it does not have to.

Single quotes must be doubled inside single-quoted strings, but double quotes should not:

```
Search '"T''amo, ripetilo, si caro accento' in OPERA
```

Double-quoted strings

Strings quoted in double-quotes (") are not converted to upper case. They result in a case-sensitive search, which means that you should never double-quote a string unless you want case to be respected during the search.

Double quotes must be doubled inside double-quoted strings, but single quotes should not:

```
Search """T'amo, ripetilo, si caro accento" in OPERA
```

2. LISTSERV "Commands-Job" Feature and CJLI Interpreter

2.1. Introduction

The "Commands-Job" feature of LISTSERV was designed in an attempt to allow for powerful inter-LISTSERV (and more generally, program-to-LISTSERV) command transmission with message redirection and multi-line arguments capability, while still allowing inexpert users to send commands to LISTSERV for execution in a very simple, intuitive" way.

The implementation of Commands-Jobs has therefore been split into two different layers: the 'core' command job language interpreter, with its exacting, powerful but stern control cards syntax, and the 'outer' interface to the user which provides the required default control cards whenever they have been omitted, translating an "intuitive" series of commands to execute into an actual commands-job that can be processed by the 'core' interpreter.

Since this documentation is primarily intended for postmasters and LISTSERV applications programmers, it will be oriented towards a description of the 'core' interpreter. The work of the 'outer interface' will only be mentioned for better understanding. The 'core' interpreter will be referred to as 'Commands Job Language Interpreter' (CJLI) in the following discussion.

Warning: if you are familiar with MVS and JCL, you will probably notice some similarity between command job control cards and JCL. This similarity is purposeful and was intended to make CJLI easier to understand for JCL adepts and to make MVS users more comfortable with CJLI (MVS users who do not have any mailing system such as UCLA mail and have difficulties sending/receiving messages are often forced to use (basic) CJLI control cards to send commands to LISTSERV). However, there are a number of differences between CJLI and JCL and you should not assume that a given JCL card has the same meaning and syntax as its CJLI counterpart. Some JCL features which were deemed to be unnecessary (eg dataset concatenation with a blank DD card) have not been implemented, and new features have been implemented whenever required.

2.2. Overview -- the JOB entity

As soon as the 'outer' interface detects that a file contains commands (as opposed to a mail or non-mail file intended for redistribution to a list), it passes it to the CJLI for execution. This physical file will contain one or more logical 'JOB's with interspersed comment lines. Each 'JOB' will contain zero or more commands, start with a "// JOB" card and end with a "// EOJ" card. The 'outer' interface will provide these cards if omitted, but this will be detailed later on. Anything before the first "// JOB" card, after the last "// EOJ" or between "// EOJ" and "// JOB" cards is ignored. Notably, mail headers and "Acknowledge-To:" fields (at the bottom of the mail file) would be ignored. A physical job file containing two jobs might look like this:

```
(any number of header lines, ignored)
 //jobname1 JOB options
 .
 .
 .
 //jobname1 EOJ
(any number of lines, ignored)
 //jobname2 JOB options
```

```
 .
 .
 .
 //jobname2 EOJ
(any number of lines before EOF, ignored)
```

Each job in the physical file is a completely independent entity which contains commands and the 'dataset' definitions required to execute the commands properly. Jobs are executed in sequential order; if a job does not execute successfully, the other jobs in the physical file are still executed, unless the job has been terminated by a 'global' error (eg error reading the physical file), in which case the CJLI terminates after transferring the file to the postmaster. Within a given job, commands are executed in sequential order regardless of the result of the previous commands.

Each job generates a separate 'output', which is sent to its recipients (see below) before the next sequential job is executed. This 'output' consists in a series of messages which are (unless specified otherwise -- see below) sent back as a single mail file.

There are basically three kind of cards in a job stream:

A. Control cards, which start with "//" in column 1 and are interpreted by the CJLI. Control cards further subdivide into three categories:

   1. Pre-execution control cards, of the form: "//label kwd args". "label" and "args" can be omitted but there must still be a blank between the "//" string and the keyword name, ie "// kwd". If CJLI does not recognize the keyword, it strips off the leading "//" and considers the card as a command-card (see below).

   2. Comments, of the form: "//* any_comment_text". These cards are ignored by CJLI. Note the blank between the asterisk and the first character of the comment text.

   3. Execution-time control cards, of the following format: "//*kwd args". Note that the keyword is concatenated to the "//*" string to differentiate this from a comment. If the keyword is not recognized, the card is treated as a comment control card and ignored. It is otherwise processed by CJLI at execution time, as if it were a normal command card, and in sequence order with the other command cards. These cards are actually commands which are only available from within a command job because they would be irrelevant outside of a job context.

B. Command cards, which contain the text of the actual commands to be executed. They are ignored by CJLI which passes them to the main LISTSERV command interpreter for execution. If the card starts with an asterisk, it is treated as a comment and ignored.

C. Data cards, which are assembled into a dataset under control of a "// DD" control card. They are removed from the job stream by CJLI and are kept in a separate pool for later reference at command execution time.

2.3. Control Cards -- general syntax rules

All control cards, except comment control cards, follow the same syntactic rules:

1. Control card starts with the string "//" in column 1.

2. Control cards can span any number of physical records: continuation cards can be defined by placing a comma at the end of the first physical line, and having the continuation card start with the string "// " (note the blank) in column 1. This process can be repeated any number of times. No blank is inserted between the end of the first card and the beginning of the continuation card; however, anything before the comma is kept. Examples:

```
//card1 DD "Some very long text which,
// requires a continuation card"
--> "Some very long text whichrequires a continuation card"

//card1 DD "Some very long text which ,
// requires a continuation card"
--> "Some very long text which requires a continuation
    card"
```

   Since this approach makes it impossible to "cut" a line which ends in a large string of blanks, an alternate method was designed for blanks-sensitive cutting. If the continuation card starts with the string "//+ " in column 1 instead of just "// ", the continuation card is not stripped of leading blanks and data from columns 5-80 is appended to the first card. Example:

```
//card1 DD "Some very long text which,
//+ requires a continuation card"
--> "Some very long text which requires a continuation
    card"
```

3. Control cards can contain a label of any length starting in column 3. This label is translated to uppercase. If the label is omitted, there must be a blank in column 3. The label can contain any character, except blank and the slash sign ("/").

4. The label is followed by at least one blank. The next word in the card is the "card name", which is translated to uppercase.

5. Arguments can be specified after the "card name", and must be separated from it by at least one blank. Arguments are separated by commas, and there must not be any blank before or after the comma. There are two categories of arguments:

   a. Positional arguments, which must appear in the correct order (which will usually depend on action being performed).

   b. Keywords, of the form "name=data". They can appear in any order and can be freely intermixed with positional keywords. They do not affect the sequence order of positional keywords. Keyword names are translated to uppercase, and can contain any character except blank, comma, double-quote or equal sign.

   For example,

```
//JOB1 JOB XDZ,ECHO=NO,FRECP11,PW=EMERALD
//JOB1 JOB PW=EMERALD,XDZ,FRECP11,ECHO=NO
//JOB1 JOB Echo=NO,pW=EMERALD,XDZ,FRECP11
```

   are three different wordings of the same arguments string.

There are furthermore two different forms of "data" for arguments:

a. Quoted data: in that case the data is enclosed in double quotes and can contain one or more blank delimited words as well as leading or trailing blanks. Quoted data is case-sensitive, and can contain any character except a double-quote.

b. Non-quoted data: in that case the data consists of a single word (i.e., blanks cannot appear in the data), which is translated to uppercase. Non-quoted data cannot contain blanks, commas or double-quotes.

In the above example, specifying 'Echo=No' is functionally identical to 'ECHO=NO', while 'Echo="No"' would leave the argument in mixed case. Quoted data is used for list of recipients, full-names, etc.

6. Comments can be included at the end of the card, and must be separated from the arguments by at least one blank. If you did not specify any argument string, and still want to place comments at the end of the control card, you must specify a null argument string before the comments by putting a ", " before the comment text, e.g.:

```
//JOB1 JOB , These are comments
//JOB1 JOB XDZ,FRECP11 These are comments too
```

2.4. The JOB control card

The JOB control card indicates the start of a new job and allows you to define several "execution options" for the job. The format of the JOB card is:

```
//jobname JOB options
```

'jobname' is the name you want to assign to the job. If you leave it blank, CJLI will default it to be your 'userid'.

Any job options must be comma-delimited with no leading or trailing spaces. For instance, you would find that

```
//MYJOB JOB ECHO=NO AFTER=19:00
```

is equivalent to "//MYJOB JOB ECHO=NO". The "AFTER=19:00" option will be treated as a comment (per item #6 in the preceding section, because it is preceded by a space) and ignored. Thus, the correct syntax for the above JOB card would be

```
//MYJOB JOB ECHO=NO,AFTER=19:00
```

The following options are available:

- **Echo=YES|NO**
  The default value (if the keyword is omitted) is YES and indicates that each command must be echoed to the job output before execution. The command is then prefixed with a "> ", and preceded by a blank line on the job output.

- **Reply-to=Sender|None|"u@n1 u@n2..."**
  The default value is "Sender" and indicates that the output of the job is to be sent to the sender of the job. "None" indicates that no output should be generated,

and is used whenever the sender is a server which is not programmed to parse the output of a LISTSERV job. Also it makes sure that no loop can ever occur due to an error in a job. "u@n1 u@n2..." can be used whenever replies are to be sent to a different person/list of persons. These persons will first receive a separate message telling them that they are receiving the output of another person's job.

- **Reply-via=Mail|Message|MSG**
  The default value, "Mail", indicates that the job output is to be sent to its recipients in the form of a mail file. "Message" and "MSG" both indicate that interactive messages are desired. Note that an attempt to send interactive messages to a node which cannot handle them will result in LISTSERV sending a piece of mail containing the text of the various messages.

- **Stat=YES|NO**
  (Starting with 1.8d) Determines whether or not server statistics ("Summary of resource utilization") for a given JOB are output in the command response to the invoker. The default is YES. A server-wide default (also starting with 1.8d) may be set with the **JOB_STAT_DEFAULT** site configuration variable.

- **AFTER=*date-spec*|*time-spec*|"*date-spec time-spec*"**
  (Starting with 1.8e) Tells LISTSERV when to run the job, ie, run it after the time specified. There are three valid formats:

  1. AFTER=date

  2. AFTER=time

  3. AFTER="date time"

  The first two formats may also be quoted if desired. The third MUST be quoted since it contains a space. In case 1, the time is 00:00:00; in case 2, the date is today. The only date format that is guaranteed to work is **yyyy-mm-dd** (other formats may also work, without guarantees). The time is specified in a 24-hour format either with or without seconds: **hh:mm** or **hh:mm:ss**.

  If the requested date is in the past, the job is executed immediately. Otherwise, it is placed on hold for execution at the exact specified time. If several jobs are submitted for the same execution time, the order in which they are executed is unpredictable. Generally, this would be a bad practice anyway since there is no guarantee that jobs arrive in the same order as you sent them. If on the other hand you schedule your jobs for one second after the previous one in the desired sequence, execution order will be respected, even if the execution of the first job causes the others to be simultaneously eligible for execution.

- **PW=password**
  Is the default password to use on commands where an explicit "PW=" keyword has not been specified. It makes it easier to write jobs performing several maintenance commands on the same distribution list, for example.

All other keywords, as well as positional parameters, are ignored. If an invalid value is specified for a valid keyword, the job is terminated by CJLI but the remaining jobs in the physical file are still executed.

If the JOB card is omitted, the 'outer' interface provides the following default JOB card:

```
//userid JOB Echo=Yes,Reply-to=Sender,Reply-via=Mail,Stat=Yes,PW=""
```

2.5. The EOJ control card

The EOJ card indicates the end of a job; its syntax is very simple:

```
//anything EOJ
```

where 'anything' can be any valid label and is completely ignored. The 'outer' interface provides an EOJ card at the end of the physical job file, as well as before a new JOB card, if none was provided by the user.

2.6. The DD control card

The DD control card allows you to define single or multi-line 'datasets' for use by the various commands in the job stream. The syntax of the DD card is the following:

```
//ddname DD "single-line-constant"
            *
            *,EOF
            *,EOF,Res=Disk
            *,EOF,Res=Storage
```

'**ddname**' is the name the dataset is to be given. It must follow the general label naming convention, cannot be omitted and cannot appear more than once in the job stream. That is, datasets cannot be concatenated by means of several DD cards.

"**single-line-constant**" denotes a single-line dataset, whose value is that of the first (quoted) argument. The length of this argument must not exceed 255 bytes.

'*****' denotes a multi-line dataset, whose successive lines immediately follow the DD card. Any number of lines can thus be included in a dataset, with a "/*" line indicating the end of the dataset. The JCL "DD DATA,DLM='xxxx'" is not implemented. Note that unlike JCL, CJLI does NOT end the dataset when a control card (ie one starting with "//") is encountered; the "/*" must always be specified.

'***,EOF**' denotes a multi-line dataset, whose successive lines immediately follow the DD card and end at the end of the PHYSICAL job file. This option is used when transmitting "unknown" data in a dataset which could a priori contain any kind of character string. Needless to say, there can be only one such dataset in the job file, and it must be the last dataset in the last job.

'**Res=**' indicates whether the dataset is to reside in storage ("**Res=Storage**") or on disk ("**Res=Disk**"). In some cases it may be necessary to keep a large dataset on disk to avoid running out of storage and to improve execution speed when a disk-file is to be generated anyway by the command using the dataset. The "**Res=**" keyword is therefore ignored on all datasets except the '***,EOF**' one (if present), and causes a disk file to be generated with the remainder of the input deck. Please note that not all commands will support the "**Res=Disk**" option: commands which do not expect to receive a large dataset as input will usually expect to find it in storage and report an error when the "**Res=Disk**" option is used. For example, DISTRIBUTE fully supports "**Res=Disk**" while DELETE doesn't.

An invalid dataset declaration causes the job to be terminated by CJLI with the remaining jobs in the physical file still being executed.

2.7. The //*MSG control card

The "**//*MSG**" execution-time control card allows you to selectively halt/resume 'typing on the job output' or to discard messages which have been previously output during execution of the job. Note that the latter option has no effect when the output of the job is sent as interactive messages, as it is intended to control mail job output.

The syntax of the "**//*MSG**" control card is:

```
//*MSG option1,option2,...
```

Valid options are: **ON, OFF, FLUSH**

FLUSH discards all messages previously sent to the job output and leaves the message-receipt status unchanged.

OFF turns message receipt off, as if "Reply-to=None" had been specified in the JOB card.

ON turns message receipt back on. This does NOT override a possible "Reply-to=None" in the JOB card, though.

2.8. Special considerations

This section contains more information on the trickiest parts of CJLI as well as some useful hints for application programmers.

LISTSERV treats anything mailed to the LISTSERV userid as a set of commands to execute. Thus, as soon as an unknown command is encountered in the job stream, the whole physical job file (not just the current job) is immediately flushed and discarded. This avoids 'executing' hundreds of unknown commands and sending back a huge job output when a regular mailfile is sent to the LISTSERV userid by someone who thought it would be distributed to a list. Note that errors from known commands do not cause termination of the job -- only completely unknown commands such as "Hiya!!" would terminate the job.

Although CJLI is based on the network standard 80-characters card images, LISTSERV accepts command jobs in several network formats, including Disk Dump and Netdata. In that case it will accept records of up to 255 characters as input, and you may find this very convenient when sending long commands to the server.

Alternatively, continuation cards can be used to split long commands into several 80-characters cards. In that case you must insert a "// " string before the command text so that CJLI considers it as a control card and performs the required concatenation; it will then realize that the "card name" is unknown and transform the card into a regular command card. If you opt for that method, you will find the "**//+**" continuation card feature very convenient for a program (but not for a human person). This method is used by LISTSERV when transmitting "DISTRIBUTE" commands to other LISTSERVs.

There is no limit at all on the final size of a control card, i.e., on the number of continuation cards you can specify; however, you must make sure that no line in any of the 'datasets' ever exceeds 255 characters. In particular, if the physical job file is sent in

Netdata format, you must make sure that the file lrecl is not higher than 255.

Note for VM servers: Due to internal coding considerations, it is recommended that physical job files be sent to LISTSERV in PUNCH format. This will make it easier for the 'outer' interface to detect the job file for what it is and will save some CPU time to the server, thereby improving job response time. Please keep in mind that DD lines longer than 80 characters cannot be sent in PUNCH format.

For more information on how to send commands to LISTSERV for execution, see LISTSERV MEMO.

3. Relayed File Distribution and the DISTRIBUTE Command

Note that DISTRIBUTE is also documented in RFC1429.

3.1. Introduction

The Relayed File Distribution (also known generically as DISTRIBUTE) feature was developed in an attempt to provide an efficient and network-resources-saving means whereby files and mail could be distributed to a large number of persons on the network by ANY network user, without having to resort to predefined distribution lists (which have other advantages but are basically static).

This chapter is composed of three independent sections. The first one (3.2) is a description of the distribution algorithm used by Relayed File Distribution. It is general enough to be accessible to inexpert computer users, but does not explain how to send a Relayed File Distribution Request (RFDR) to LISTSERV. The second section (3.3) gives a detailed technical description of RFDRs and assumes the reader is familiar with the basic concepts of the Commands-Jobs Language Interpreter (CJLI). More information about CJLI is available in Chapter 2, above. The third section (3.4 and 3.5) is a more practical tutorial regarding the construction of RFDR jobs. The general user who wishes only to learn how to set up simple RFDR jobs and who is not interested in the technical issues involved may therefore skip sections 3.2 and 3.3, moving directly to 3.4.

Please note that while "RFDR Job" is actually the correct nomenclature for a job sent out using Relayed File Distribution, the more common (if not as entirely correct) term "DISTRIBUTE Job" may also be used herein, interchangeably, to describe such jobs.

Starting with LISTSERV 1.8d, a new security validation feature has been added to the DISTRIBUTE command. The new feature changes the default behavior of DISTRIBUTE in two ways:

1. Only a LISTSERV maintainer (i.e., a user who is identified in LISTSERV's site configuration file as a **POSTMASTER=**) or a "trusted" user (identified in LISTSERV's site configuration file in the **DIST_ALLOWED_USERS=** variable) may issue the DISTRIBUTE command; and
2. The DISTRIBUTE command must be validated with the issuer's personal LISTSERV password (obtained with the PW ADD command).

The new default can be relaxed back to the previous (pre-1.8d) behavior by setting the new **DIST_SECURITY** site configuration variable appropriately. See Appendix C of the *Site Manager's Operations Manual* for details.

3.2. What is Relayed File Distribution?

Relayed File Distribution is a service provided by the ever-growing network of L-Soft LISTSERV servers to all users connected to the Internet who are allowed to send and receive files or messages. (The origins of the service were on BITNET and associated services where it was possible to send files, as opposed to messages.) The user desiring to send the message (whom we will call the 'sender') provides his nearest L-Soft LISTSERV host with a copy of the message to be distributed and the list of persons who are to receive it. He can optionally indicate the full name of these persons as well as his own full name, and they will be used in information messages and mail headers as appropriate. The LISTSERV hosts will then relay the file to each other as explained below and distribute the file to all the indicated recipients in the most efficient way they could 'manage'.

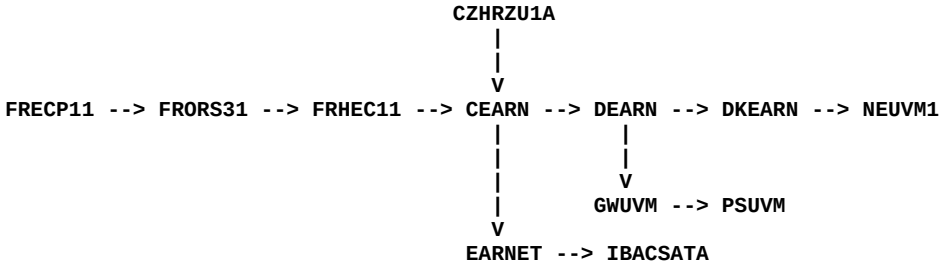
The server that initially receives the file from the sender will first distribute the file to the (possible) recipients of its local node, which does not generate any network traffic. It will then examine the list of non-local recipients and determine, for each of them, which of the L-Soft LISTSERV servers is nearest.[1] Whenever it finds itself to be the nearest server, it distributes the file directly, and removes the recipient from the list. It then uses a rather complex algorithm to "route" the remaining recipients through its nearest servers, and transmits the request to them. This will be best illustrated by an example.

User **X@FRECP11** sends a file distribution request for the following list of people:

Y@FRECP11,X@CEARN,Y@CEARN,X@CZHRZU1A,X@NEUVM1,X@IBACSATA,
X@EARNET, X@PSUVM

We will assume that a L-Soft LISTSERV server has been installed at the following sites:
FRECP11, CEARN, DEARN, DKEARN, EARNET

Finally we will assume the following topology, which will not necessarily reflect the actual network topology at the time you read this chapter (it has been simplified and nodes which do not participate in the transfer have been removed)



The file is distributed by LISTSERV@FRECP11 to Y@FRECP11, and then forwarded to LISTSERV@CEARN, which will distribute to the two CEARN recipients and to X@CZHRZU1A. LISTSERV@CEARN then forwards one copy of the file to LISTSERV@DEARN and one to LISTSERV@EARNET, with the appropriate list of recipients. LISTSERV@EARNET distributes to the EARNET recipient and to X@IBACSATA. LISTSERV@DEARN distributes to X@PSUVM, and also to X@NEUVM1. LISTSERV@DKEARN does not receive anything because it would have served only a unique recipient, and in that case a direct send can only be better.

The file has therefore crossed a total of 11 links. If it had been sent the normal way from FRECP11, it would have had to cross 32 links, i.e. thrice more. The reduction is very important because we assumed that a server is installed at all the central nodes in the network, which is not necessarily the case in practice.

3.3. A technical description of Relayed File Distribution Requests

Relayed File Distribution Requests ("RFDR Jobs") must imperatively be transmitted as a commands-job file. It is recommended that you read Chapter 2 of this manual if you are not familiar with the basic concepts of CJLI.

Beginning with 1.8d, by default DISTRIBUTE may only be executed by a LISTSERV maintainer (defined in the **POSTMASTER=** variable in the site configuration file) or a

[1] Under LISTSERV Classic only. DISTRIBUTE jobs sent to LISTSERV Lite servers are handled exclusively by the local server and the DISTRIBUTE backbone is not employed.

"trusted" user (identified in LISTSERV's site configuration file in the **DIST_ALLOWED_USERS=** variable) and requires a password (the invoker's personal LISTSERV password).

The job sent to the server must contain:

- A JOB card with the "Echo=No" option to avoid tracing the unique command to the job output. The "Reply-via=" keyword can be set to "Message" if so desired, although this is not recommended. The job name can be anything you want, eg filename.filetype

- A DISTRIBUTE command with the appropriate arguments:

```
DISTribute < <MAIL> <DD=ddname> <FROM=u@n | FROM=DD=ddname>
             <ACK=NOne | MAIL | MSG> <TO DD=ddname | TO u@n1
                 <...>> >
             <PRIOR=* | nn> <INFORM=MSG | MAIL>
             PW=password
```

  All the keywords (except **PW=**) can be omitted, but must be specified in the indicated order. The "PRIOR" and "INFORM" keywords are indepedent from the others and can appear anywhere in the command line. Beginning with LISTSERV 1.8d the **DISTRIBUTE** command requires a password (**PW=password** , above) for validation purposes. A description of the keywords follows:

  - MAIL indicates that the file to be distributed is a mailfile, to be sent to the MAILER at the destination node. The contents of the file are proofread and the "mail origin" is verified so that users cannot send forged mail. MAIL-type distribution is 100% transparent to the mail recipient, which is not the case with file distribution -- a file would come from the LISTSERV userid instead of the sender's userid. The mailfile must contain a blank "To:" line where the actual name and network address of each recipient will be automatically inserted as the file is distributed to him.

    All DISTRIBUTE jobs sent from non-VM servers will be MAIL-type DISTRIBUTE jobs. Non-MAIL DISTRIBUTE is available only on VM.

  - DD=ddname is the ddname corresponding to the data to be transmitted (ie the file or mailfile). The default is "DD=DATA"

  - FROM=xxxx indicates either the network address of the file sender or the name of a dataset of which the first line indicates the network address and full name of the file sender, eg FROM=DD=XXX and //XXX DD "JPB@BIGNODE John P. Brown". The default is FROM= your-userid.

    This field contains the address you want bounces to go to (the RFC821 MAIL FROM: address) and it can be used to redirect bounces[2] that should not normally go to the user invoking DISTRIBUTE--eg, it can be set to a special address set up specifically to handle errors for this particular DISTRIBUTE job.

    Under 1.8c and earlier, only LISTSERV maintainers were allowed to use the FROM= field. General users were not allowed to specify an origin different from

[2] Assuming that the bouncing system is compliant with Internet standards and uses the RFC821 MAIL FROM: address for bounces.

their own network address, so the FROM= option would have had no effect unless the user wanted to specify his full name. Under 1.8d and later, general users (those who are not otherwise defined in the **DIST_ALLOWED_USERS** site configuration parameter) do not have access to DISTRIBUTE to begin with.

You do not have to indicate your name when distributing MAIL-type files -- put your name in the "From:"/"Sender:"/whatever field, as appropriate.

- ACK= indicates the amount of acknowledgement you want to get. The default is NOne and indicates you do not want to receive messages as the file is distributed. MAIL indicates mail acknowledgements while MSG indicate interactive messages.

- TO indicates the list of recipients for the file or mailfile. It can either be a list of network addresses ("TO u@n1 u@n2...."), which must not cause the physical command line to exceed 255 characters (use continuation cards in that case), or a ddname of which each line is a "userid@node <full name>" pair. The former is best suited to file distribution while the latter should be preferred for MAIL-type distribution since the recipient's full name will be inserted in the "To:" field of the mail file. The default is "TO DD=TO". Note that distributing a file to a LISTSERV userid will cause it to be interpreted as a command job for execution or a PUT request, as appropriate. Releases 1.5b and earlier did not accept such a destination for DISTRIBUTEd files and ignored the recipient.

- PRIOR is the network transmission priority you want to assign to the file. If specified, it can be either "*" or an integer number between 0 and 99, inclusive. "*" indicates that the file priority is left up to LISTSERV, which will use an internal algorithm to assign a reasonable priority to the file according to its number of records.

- INFORM (effective on VM only) specifies the media by which you want users to be informed of the arrival of the file. The default is MSG for interactive messages -- specify MAIL if you want LISTSERV to notify recipients via mail. Note that this option is ignored when MAIL distribution has been selected.

- Finally, a password is required to validate the command invoker for versions 1.8d and higher of LISTSERV. This is a standard LISTSERV personal password, set with the **PW ADD** command (q.q.v.).

- A dataset for the "FROM=DD=" keyword, if specified -- the DD "text" syntax is recommended since the dataset will contain only a single line of data.

- A dataset for the "TO DD=" keyword, if specified -- the DD * syntax is best suited to this dataset. This dataset contains network addresses (imperatively) and per-address options, one address/options per line. The per-address options are a real-name field and the keywords BSMTP or PROBE (BSMTP and PROBE are mutually-exclusive; only one can be specified per address). For instance a TO dataset could contain lines like

```
janecustomer@abc.com
jacktechie@def.edu Jack Techie
johndoe@ghi.org BSMTP
joe@example.com PROBE
sue@example.edu Sue User BSMTP
petergunn@example.edu Pete Gunn PROBE
```

When the BSMTP option is specified, LISTSERV will combine the address along with any other addresses for which BSMTP is specified into a multi-recipient BSMTP envelope (which is much more efficient, since not using BSMTP tells LISTSERV to create a separate SMTP envelope for each address). For general DISTRIBUTE jobs (ie, non-mail-merge jobs), it is probably most efficient to specify BSMTP for all addresses in the job.

When the PROBE option is specified, LISTSERV will process the address as a PROBE. This is most useful when using a backend list or a changelog file for error processing (see the section on "Advanced LISTSERV applications using DISTRIBUTE", below, for more information on these features; also see the sections on LISTSERV's address probing in the *Site Manager's Operations Manual* and/or in the *List Owner's Manual*). PROBE should not be used on one-off DISTRIBUTE jobs which do not require specialized error processing, as it has no particular usefulness otherwise.

Again, BSMTP and PROBE are mutually-exclusive; you may specify one or the other but not both.

- Finally, a dataset for the "DD=" keyword, to contain the mail message or file. This should be the last dataset in the file and the DD *,EOF syntax should be used to ensure that the dataset is not prematurely ended by a possible "/*" card in the data. On VM, for storage space saving and performance considerations, the "Res=Disk" option should be specified on that DD card, ie: DD *,EOF,Res=Disk. Statistics have shown that this decreases the CPU time required to process the request by a factor of six.

Mail-type RFDR under VM must IMPERATIVELY use raw-image (PUNCH) format for the mail dataset, since the header will be scanned and modified by the server. File-type RFDRs can use any type of network file format -- the contents of the "data" dataset will be sent "as is" to the recipients, without anything added on top or bottom of it. The only difference will be the RSCS file origin -- the file will come from the LISTSERV userid instead of the sender's userid. The file class, spool fileid and DIST-code are preserved, but the FORM is changed to QU-DIST to trigger the "quiet file transfer" feature installed in some RSCSs in the network.

Non-BITNET recipients are routed to their gateway, as determined by the DOMAIN NAMES file. The first server in the chain that finds itself unable to determine the gateway distributes the file directly. Whenever non-mail file is distributed to a non-BITNET user, LISTSERV generates a standard mail envelope with the current date, sender's name and address, recipient's address and transfers it to the mailer. Any possible rejection mail will be sent directly to the sender by the mailing system (and not to the LISTSERV userid).

3.4. Creating and sending a mail RFDR ("DISTRIBUTE") job

The instructions below are for sending plain-text messages to a one-time list via LISTSERV. If you wish to send MIME-encoded mail messages (for instance, to send encoded binaries or HTML mail), you must create and add the appropriate MIME headers along with the file (in the case of an encoded binary) to be sent.

The syntax described here is for 1.8d running with the default of **`DIST_SECURITY=1`** in the site configuration. We have not attempted to discuss the differences between 1.8d and previous versions of LISTSERV in writing RFDR jobs.

A very basic RFDR job is shown below. This job would be sent to **LISTSERV@NODE**, where **NODE** is the **NODE=** site configuration setting for your local LISTSERV server.

```
//WIDGET1 JOB
DISTRIBUTE MAIL PW=XXXXXXXX
//To DD *
janecustomer@abc.com BSMTP
email-address1@domain.com BSMTP
email-address2@domain2.com BSMTP
/*
//Data DD *,EOF
Date:      Tue, 13 Jan 1998 12:25:00 -0400
From:      joe.techie@xyz.com
Subject:   Release of XYZ's WIDGET Model 2
Reply-to:  joe.techie@xyz.com
To:        Widget Customers <customers@widget.com>

Your text here
```

Below is a detailed description of each element of the job.

```
//WIDGET1 JOB
```

The JOB "card" is the required first line of any RFDR job. It signifies to LISTSERV that everything that follows in this particular message is part of the same job. In this case we have called the job WIDGET1. Each RFDR job should have a name, but if for some reason you don't want it to, you simply format the JOB card with a space between "**//**" and "**JOB**".

```
DISTRIBUTE MAIL PW=XXXXXXXX
```

This is the actual DISTRIBUTE command that tells LISTSERV that this is an RFDR job, and that specifically you want LISTSERV to distribute a piece of mail. **PW=XXXXXXXX** is part of the security mechanism described in 3.1. You replace **XXXXXXXX** with your personal LISTSERV password that you've previously set with LISTSERV's PW ADD command.

```
//To DD *
janecustomer@abc.com BSMTP
email-address1@domain.com BSMTP
email-address2@domain2.com BSMTP
/*
```

This part of the job tells LISTSERV to whom you want the mail distributed. There are a few options for the addresses themselves as you've probably noticed. The basic syntax for an address in the "To DD" is as follows:

userid@host.domain [Personal Name] [BSMTP | PROBE]

for instance, any of the following are correct:[3]

[3] Technically the syntax "**janecustomer@abc.com Jane Customer BSMTP**" is also correct, but since the personal name field will be ignored in a BSMTP job, there is no need to define it.

```
janecustomer@abc.com Jane Customer
janecustomer@abc.com BSMTP
janecustomer@abc.com PROBE
janecustomer@abc.com Jane Customer PROBE
```

The only part of this syntax that is required is the first token (userid@host.domain). You can follow it with the user's personal name, if available. Then you can add the token "BSMTP", which tells LISTSERV to use "Batch SMTP" processing for this address, or the token "PROBE", which tells LISTSERV to send the message to the specified address as a passive PROBE. Please note carefully that the BSMTP and PROBE options cannot be used together for the same address.

If you use the BSMTP option, it isn't necessary to specify a "personal name" as LISTSERV won't use it for anything. BSMTP jobs are sent out in the same manner as regular LISTSERV list mail for users who have the FULLBSMTP user option set; that is, the To: line of the mail contains the address you have specified on the To: line in the data (message text) portion of the RFDR job. This is the most efficient way to send mail via DISTRIBUTE as it "bundles" recipients into a few large outgoing jobs rather than creating a separate outgoing job for each recipient. If you need to "personalize" each message then you do not want to use the BSMTP option. However, most large RFDR jobs will probably be sent using BSMTP since it is more efficient.

If you do not use either BSMTP or PROBE and use just the e-mail address, e.g., **janecustomer@abc.com** or **janecustomer@abc.com Jane Customer**, the recipient's e-mail address will be specified in the To: field, regardless of what you define in the To: field header further down in the job. As noted above, this is an attractive way to do the mailings but uses FAR more resources since you are creating a separate outgoing copy of the message for each subscriber. The marketing benefits of personalized mail are obvious but not using BSMTP is something that you should test to see if the return is worth the cost. Utilizing BSMTP allows your job to be processed as if it were a traditional LISTSERV list and should be used when greatest efficiency is desired.

The "**/***" at the end of the "To DD" is required and may not be omitted. It tells LISTSERV where the DD ends. It must be on a line by itself. If it does not exist in the job stream, the job will fail.

```
//Data DD *,EOF
```

This card tells LISTSERV where the actual message begins. There is no end card for the "Data DD" as it ends at the end of the message you send the job in. This is important to note as it means that you must disable any signature file that normally is sent with your mail--otherwise it will appear as part of the RFDR job and be sent out!

```
Date:       Tue, 13 Jan 1998 12:25:00 -0400
From:       joe.techie@xyz.com
Subject:    Release of XYZ's WIDGET Model 2
Reply-to:   joe.techie@xyz.com
To:         Widget Customers <customers@widget.com>
```

Following the "Data DD" job card are the RFC822 message headers for your message. The only required headers here are Date:, From:, and To:. However, while you do not technically *need* to provide a Subject: or a Reply-To: field, you will probably want to do so for completeness' sake. There is no specified order for these header fields so you may arrange them as suits you best.

If you do not specify a value for the Date: line, but rather simply insert a "Date:" header without a following value, LISTSERV will insert its local time and date as of when it received the job for execution. In the following example:

```
Date:
From:       joe.techie@xyz.com
Subject:    Release of XYZ's WIDGET Model 2
Reply-to:   joe.techie@xyz.com
To:         Widget Customers <customers@widget.com>
```

LISTSERV will fill the Date: header in by itself. Otherwise LISTSERV will accept any Date: field you provide without comment. If on the other hand you do not at least specify a "Date:" header line, no date will be inserted at all.

If you do not specify a From: line, LISTSERV will insert a From: line containing the address of the command invoker (you!) unless you have specified a FROM= option in the DISTRIBUTE command (see either below or in the preceding technical section for specific DISTRIBUTE options), in which case the value of the DISTRIBUTE FROM= option will be inserted. It is probably wise in most cases to ensure that there is a From: line in the "Data DD" as otherwise the result may not be as you planned.

If you do not specify a To: line, LISTSERV will insert a To: line as follows:

```
To:           Multiple recipients <LISTSERV@NODE>
```

where "NODE" is the **NODE=** value for your LISTSERV server. As with the From: line, this is probably not what you want people to see, so you have a couple of options. For jobs which contain only BSMTP recipients, you should specify a To: value, which should point back to a real address in case someone decides to write back to it. For instance, in our example above,

```
To:         Widget Customers <customers@widget.com>
```

and we'll assume that "customers@widget.com" is a customer-service address where people can write, or even an existing LISTSERV list out of which you've pulled a subset of addresses for this particular RFDR job.

If you have all non-BSMTP and non-PROBE recipients, you can simply leave the To: field blank (as with Date:, above) and LISTSERV will fill the field in with the data you've provided in the "To DD". For instance, in the following example,

```
Date:
From:       joe.techie@xyz.com
Subject:    Release of XYZ's WIDGET Model 2
Reply-to:   joe.techie@xyz.com
To:
```

both the Date: and To: fields will be filled in by LISTSERV for you.

Please note that the blank line at the end of the RFC822 headers is not a misprint. According to RFC822 you MUST provide a blank line between the last header and the

beginning of the message body, and LISTSERV expects it to be there. If it is not there, the job will fail with an RFC822 parser error:

```
> DISTRIBUTE MAIL
Invalid RFC822 field found in mail header: "Your text here--forgot
blank line". Mail not delivered.
```

Finally, we get to the final element: the message body.

```
Your text here
```

This element is fairly self-explanatory; you simply add the actual body of the message you are sending out. Again, make sure that there is a blank line between the body of the message and the last RFC822 header as mentioned previously.

3.4.1. RFDR job options

There are three options that might be of use in the JOB card (the first line of the RFDR job as noted above). These options are:

1. **ECHO=NO**

This option suppresses the resource usage summary you would otherwise get upon completion of your delivery. Omitting it returns a mail message containing a summary like the following example to the e-mail address from which the RFDR job was sent.

```
Job "WIDGET1" started on 10 Feb 1998 13:33:34

> DISTRIBUTE MAIL

Job "WIDGET1" ended   on 10 Feb 1998 13:33:36

Summary of resource utilization
-------------------------------
 CPU time:        0.000 sec
 Overhead CPU:    0.030 sec
 CPU model:         100MHz Pentium (64M)
 Job origin:      joe.techie@XYZ.COM
```

To use this option and suppress the summary from being sent, simply type ECHO=NO at the end of the JOB card, e.g.,

```
//WIDGET1 JOB ECHO=NO
```

2. **PRIME=*prime-time-specification***

This option controls when the job runs, if you do not want it to run immediately.

This function of this option is identical to the function of the "Prime=" keyword setting for LISTSERV mailing lists, where "Prime=" defines (on a list by list basis) times during which LISTSERV *should not* process mail for the list. By default this option is set to "PRIME=Yes", meaning that the job can be processed at any time. If you set this option to "PRIME=No", it uses the value set globally by LISTSERV's PRIMETIME= site configuration variable, which by default is set to have no prime time (in other words, by default, mail can still be processed at any time even with "PRIME=No"). The PRIME=

option for RFDR jobs can be set to an explicit time definition if necessary. For instance, you might have a very large RFDR job (e.g., a newsletter) that should not be processed until after midnight (when network traffic is low and more machine resources are generally available).

RFDR jobs sent to LISTSERV during prime time are automatically held until non-prime time and then distributed normally, without requiring further intervention by anyone. For instance you could set the "PRIME=" option to

```
PRIME="MON-FRI: 00:00-23:59"
```

This means that you could send the job anytime during the regular work week--Monday morning through Friday afternoon--and know that it won't be distributed until Friday at midnight or shortly thereafter.

As another example, if you want the job only to be processed between midnight and 6 AM on weekends, you might set it to

```
PRIME="MON-FRI: 00:00-23:59; SAT-SUN: 06:00-23:59"
```

Note that the specifications for PRIME= must be enclosed in quotes and the format (spaces) should be identical to the examples.

VERY IMPORTANT: You should always leave yourself at the very least an hour processing window, and even more so for large jobs, since PRIME checks are done once each hour, on the hour, and very large jobs may require extended processing time. Also note that if running in Networked mode, even more time is required since jobs passed across the LISTSERV network for other servers to deliver also contain the PRIME setting, and depending on which time zone it is passed to, it may miss the "window" altogether. For example,

```
PRIME="MON-FRI: 00:00-23:59; SAT-SUN: 06:00-23:59"
```

leaves six hours of processing for your DISTRIBUTE job on Saturday and Sunday, i.e., 12 hours total processing time over the two days. So if the job does not get to, say, a server in Japan before the PRIME deadline on Saturday, it will be processed the next day during non-prime time. In actuality it is probably not necessary to use such small windows on the weekend, so

```
PRIME="MON-FRI: 00:00-23:59"
```

is probably all you would need for a job to be delivered over the weekend.

To use the PRIME option, simply type it following the JOB operand. If you need to specify ECHO=, note that it must follow PRIME= or a syntax error will result. For example:

```
//WIDGET1 JOB PRIME="MON-SUN: 09:00-23:59" ECHO=NO
```

or

```
//WIDGET1 JOB PRIME="MON-SUN: 09:00-23:59"
```

Please make sure that you test the PRIME= option setting with test jobs first to make sure that the proper syntax is being used before sending mission critical jobs to be

processed during off-prime hours.

3. **AFTER=*time-spec***

In LISTSERV 1.8e and later, a much simpler method of delaying a message's distribution is to use the **AFTER=** JOB card option. As noted above in chapter 2.4, you can specify

- a time, eg 01:00 (the release date is assumed to be "today")
- a date, eg 2001-12-26 (the release time is assumed to be 00:00:00)
- a date and time together, in double quotes, eg, "2001-12-26 01:00"

until which LISTSERV will hold the job. (See 2.4, above, for specifics on the syntax of times and dates.)

Using **AFTER=** instead of **PRIME=** has the added advantage of not specifying a "window" of time during which the job can be processed which is propagated along with the job to other servers (which can delay distribution if the window is too small).

Examples:

```
//WIDGET1 JOB AFTER=01:00 ECHO=NO
```

```
//WIDGET1 JOB AFTER=2001-12-26 ECHO=NO
```

```
//WIDGET1 JOB AFTER="2001-12-26 01:00" ECHO=NO
```

As with PRIME=, if ECHO= is specified, it should be the last token on the line.

3.4.2. DISTRIBUTE Command Options

There are six options that might be of use in the DISTRIBUTE command (the second line of the RFDR job as noted above). These options are:

1. **ACK=MAIL**

This option tells LISTSERV to confirm deliveries with an acknowledgement sent by mail (the default is not to send delivery confirmations, but rather to simply acknowledge receipt and processing of the job), as in the following example (note that **ECHO=YES** in the JOB card for this particular example):

```
Job "WIDGET1" started on 10 Feb 1998 13:33:34

> DISTRIBUTE MAIL
Mail delivered to janecustomer@abc.com

Job "WIDGET1" ended   on 10 Feb 1998 13:33:36

Summary of resource utilization
-------------------------------
 CPU time:        0.000 sec
 Overhead CPU:    0.030 sec
 CPU model:         100MHz Pentium (64M)
 Job origin:      joe.techie@XYZ.COM
```

Use of this option in the DISTRIBUTE command line is as follows:

```
DISTRIBUTE MAIL ACK=MAIL PW=XXXXXXXX
```

Other than the default **ACK=NONE**, the only other setting for this option is "**MSG**", which is obsolete except for NJE servers.

2. **DEBUG=YES**

This option produces a report showing how the various recipients will be routed over the LISTSERV backbone, but WITHOUT actually delivering the message. Before sending out your actual posting, you may want to send the RFDR job with this option enabled to see how the mail will be routed.

Examples of the use of this option in the DISTRIBUTE command line:

```
DISTRIBUTE MAIL ACK=MAIL DEBUG=YES PW=XXXXXXXX
```

```
DISTRIBUTE MAIL DEBUG=YES PW=XXXXXXXX
```

3. **TRACE=YES**

This option produces a report showing how the various recipients will be routed over the LISTSERV backbone, and actually delivers the message (similar to DEBUG=YES but the message is actually delivered).

Examples of the use of this option in the DISTRIBUTE command line:

```
DISTRIBUTE MAIL ACK=MAIL TRACE=YES PW=XXXXXXXX
```

```
DISTRIBUTE MAIL TRACE=YES PW=XXXXXXXX
```

4. **FROM=*netaddress***

This option can point to a particular address that will receive the bounced mail, notifications of delivery problems, etc. Specifically this option sets the RFC821 MAIL FROM: address to which all compliant mailers will bounce non-deliverable mail. The single parameter ***netaddress*** is an Internet address in the standard format (userid@host.domain).

Examples of the use of this option in the DISTRIBUTE command line:

```
DISTRIBUTE MAIL ACK=MAIL TRACE=YES FROM=john@example.com PW=XXXXXXXX
```

```
DISTRIBUTE MAIL FROM=john@example.com PW=XXXXXXXX
```

```
DISTRIBUTE MAIL ACK=MAIL FROM=john@example.com PW=XXXXXXXX
```

```
DISTRIBUTE MAIL TRACE=YES FROM=john@example.com PW=XXXXXXXX
```

5. **AV=YES|NO**

Requires LISTSERV Classic/Classic HPO 1.8e, with LISTSERV's anti-virus feature

enabled.

By specifying "AV=YES", you direct LISTSERV to check the message for viruses and terminate distribution if ANY errors occur. This includes internal virus scanner errors that do not necessarily indicate the presence of a virus, out of memory errors, invalid base64 data errors, you name it.

While very safe, there is obviously the drawback that critical messages may be suppressed due to problems within the virus scanner. It should be up to the customer to decide what to do in such cases. "AV=YES,FORCE" tells LISTSERV to stop ONLY if a virus has been found. The drawback is that viruses may be let through if there was a problem with the virus checker (this includes expired maintenance), insufficient memory, or any other problem that would otherwise have caused LISTSERV to terminate the distribution. For obvious reasons, therefore, L-Soft does not recommend the indiscriminate use of the FORCE option.

"AV=NO" is also available, but does not need to be explicitly coded. This is the default and the only valid option for DISTRIBUTE of an NJE FILE (VM/VMS only) and DISTRIBUTE RFC822 (although it is doubtful that this DISTRIBUTE type is used by anyone anymore).

This option is not forwarded. It is compatible with the backbone, but checking only occurs at the originating site.

IMPORTANT caveat for mail-merge: for performance reasons, virus checking is done only once, not for each recipient after substitution. It is theoretically possible for a virus to escape undetected if it is constructed from a combination of message text and substitution data, especially if the substitution comes from a database.

Another caveat is that certain types of sophisticated mail-merge jobs may incorrectly raise the invalid MIME data error. This could happen if the base64 data were fetched from a database, or if it were included in a .BB/.EB block. Again, all you need to remember is that LISTSERV scans the message template rather than the X million individual substituted messages, so viruses that do not exist in the template will not be detected. The expected usage for correct scanning is something like this:

```
.BB whatever

--blah boundary
content-type: application/msword; blah blah
content-transfer-encoding: base64

WDVPIVAlQEFQWzRcUFpYNTQoUF4pN0NDKTd9JEVJQ0FSLVNUQU5EQVJELUFO
...
.EB
```

6. **DKIM=NO|YES**

Requires LISTSERV Classic/Classic HPO 14.5, with LISTSERV's DomainKeys signing engine enabled (see *Using LISTSERV with DomainKeys* for implementation details).

A new `DKIM=NO|YES` option has been added to DISTRIBUTE (default: NO). This will fail in two cases:

- If running a LISTSERV version without DomainKeys support; or

- If running a LISTSERV version with DomainKeys support, but with the DomainKeys engine disabled.

Otherwise the DomainKeys signing always succeeds. Messages originating from domains for which LISTSERV has been configured to sign will be signed, while those originating from other domains won't be.

3.5. Advanced LISTSERV applications using DISTRIBUTE

Depending on your specific needs, it is possible for sites to "get creative" and use traditional LISTSERV lists as "back-ends" in conjunction with DISTRIBUTE jobs to handle bounces in the same way they are handled for a regular LISTSERV list, without ever using the list itself for posts.

Let's say that, to begin with, you collect addresses from a guestbook web page that you've set up for your company. You want to send out periodical updates to these people using RFDR "DISTRIBUTE" jobs but you know that you are going to have the usual 15-20% or more bad addresses from typos, pranksters, and plain old attrition. How do you find the bouncing addresses so you can purge them from your database without having to read each bounced message and determine what address actually bounced?

Simply create a traditional list with the appropriate error handling setting desired (see other sections of this manual and of the *List Owner's Manual for LISTSERV* for more information). Set "Change-Log= Yes" and "Auto-Delete= Yes,Full-Auto,Delay(0)" in the list's header ("Change-Log" requires LISTSERV 1.8d and later). Use an ADD IMPORT job as detailed in chapter 4.3 of the *List Owner's Manual* to add all of the addresses from your database to the list.

Then in your RFDR job, set the following in the FROM= option of the DISTRIBUTE command line:

```
DISTRIBUTE MAIL FROM=owner-listname@yourdomain.com PW=XXXXXXXX
```

where ***listname*** is the name of the list created. For instance, if the list was named **bouncelist1** and the LISTSERV host name was **LISTSERV.EXAMPLE.COM**, then the DISTRIBUTE command would be as follows:

```
DISTRIBUTE MAIL FROM=owner-bouncelist1@listserv.example.com PW=XXXXXXXX
```

Once you have the bounce list set up (and we'll continue to use our **bouncelist1** example), you simply check the **BOUNCELIST1.CHANGELOG** file for a listing of addresses LISTSERV has auto-deleted from **bouncelist1** since you sent out your RFDR job. You can then use this changelog file to update your database and clean out the bad addresses.

Do keep in mind, however, that in order for LISTSERV to take action on permanent bounces that come to the **owner-bouncelist1** address and are, furthermore, in a format understandable by LISTSERV, the bouncing address(es) must also be listed as subscribers in the back-end **bouncelist1** list (so do not forget to bulk add the addresses before sending the DISTRIBUTE job, as explained above).

To take this a step further, you could also use this same back-end bounce processing list as a front-end list to handle automatic subscribe and unsubscribe requests for each mailing, allowing for total flexibility for both yourself and the customers being mailed to.

3.5.1. "List-free" bounce processing for one-shot lists (1.8d and following)

This feature is not available for LISTSERV Lite, or for LISTSERV Classic running on VM.

Starting with LISTSERV 1.8d you can bounce-process a one-shot list without needing a back-end list for the bounces to come back to (as outlined above).

To use the "list-free" feature, you create and send a standard DISTRIBUTE MAIL-MERGE job with the **FROM=** address set to **OWNER-NOLIST-*xxx*@*hostname*** (where '***xxx***' can be anything and '***hostname***' is LISTSERV's host name), for instance **OWNER-NOLIST-MYDIST@LISTSERV.MYHOST.COM**. While '***xxx***' can be anything, **OWNER-NOLIST@whatever** doesn't work--you must provide '***-xxx***'. Within the job you then use **PROBE** for each address in the manner documented above.

NOTE: This feature *requires* that you send the job as a mail-merge message, that is, with DISTRIBUTE MAIL-MERGE, even if the job is not a mail-merge job. The feature does not work with DISTRIBUTE MAIL. This also requires that you MUST be running LSMTP Classic as your SMTP_FORWARD host, or, for LISTSERV 14.4 and later, you MUST have EMBEDDED_MAIL_MERGE enabled. You must also be using SMTP "workers", that is, you must have SMTP_FORWARD_n= variables set in your site configuration file so that LISTSERV sends mail to its outbound mailers in asynchronous mode.

After sending the job, you get a **NOLIST-*xxx*.CHANGELOG** in LISTSERV's A directory with entries for every bounce. (In our example above, this file would be called **NOLIST-MYLIST.CHANGELOG**.)The entries read **BOUNCE** followed by the e-mail address. As there is no monitoring and no decision to delete, the entry does not read **AUTODEL**, and thus bounce reporting for "list-free" DISTRIBUTE jobs is passive. By definition, however, you can't monitor one-shot jobs; monitoring (if desired) must be provided by collating all the one-shot jobs into a big monitoring database, or something similar (all jobs from client such and such, etc).

Note carefully that if you are running under unix with sendmail, you will have to patch sendmail in order for it to properly accept and route bouncing probe messages. As a convenience, L-Soft provides third-party sendmail patches for this purpose on its FTP site (in **ftp://ftp.lsoft.com/listserv/unix/CONTRIB**) but please be aware that L-Soft did not write and does not support these patches in any way; they are strictly for use at your own risk and you must contact the author(s) of the patches for help with them.

4. DBMS and mail-merge support

4.1. Overview

Two of the most significant enhancements in version 1.8d of LISTSERV were DBMS and mail-merge support. Some enhancements have been made in version 1.8e. In this chapter, you will find a brief description of the DBMS features, along with installation instructions and a few samples.

While the DBMS and mail-merge functions were designed to work together, they can also be used independently from each other. That is, you may find the DBMS support useful even if you have no need for mail-merge functionality, and likewise you can use the mail-merge functions without a DBMS back-end. Both functions require LISTSERV Classic or LISTSERV HPO, and are unavailable in LISTSERV Lite.

Prior to LISTSERV 14.4, mail-merge required that LSMTP Classic version 1.1b or later be installed as your outgoing mail transfer agent.

Starting with LISTSERV 14.4, outbound mail-merge traffic may be handled by any high-performance SMTP server as long as the new LISTSERV site configuration variable EMBEDDED_MAIL_MERGE is set to 1 (that is, enabled). Please see the LISTSERV 14.4 release notes for further information on this feature.

4.1.1. DBMS support

LISTSERV's DBMS support allows you to:

- Direct LISTSERV to store subscriber information in a DBMS, on a list by list basis. That is, you may have a mix of traditional LISTSERV lists and DBMS lists. Furthermore, you can adjust the layout of your DBMS lists to match existing or current applications. You can map each list to a private table if this is what makes sense for you, or you can put all the lists in the same table, place related lists in one table, etc. You can add as many columns as you want to store additional information about subscribers.

- Use the DBMS as a back-end for mail-merge jobs. LISTSERV can execute arbitrary SQL SELECT statements to extract recipients from your DBMS, and make related information (name, country, account number, etc.) available for mail-merge operations.

DBMS support is available through Microsoft's ODBC interface on Windows 2000 and greater, and Oracle's OCI interface on OpenVMS Alpha, Digital Unix, AIX and Solaris (SPARC only). Additionally, DB2 is now supported natively (ie via CLI) under the unixes which are supported by both DB2 and LISTSERV.[4]

L-Soft formally supports SQL Server 2000 as a datastore for mailing lists, and expects to be able to support SQL Server 2005 when sites begin to migrate to it.

L-Soft formally supports Oracle 8i, 9i, and 10g as a datastore for mailing lists.

[4] In LISTSERV 1.8e and following, it is possible to define multiple simultaneous connections to DBMS tables, and as a result, previous OCI support for Windows NT/2000 (which was provided to work around the earlier limitation of only a single ODBC connection at a time) was withdrawn with the release of LISTSERV 1.8e.

4.1.2. Why require Oracle 8 or higher?

While this is probably no longer an issue for most Oracle customers, we are leaving this section in for historical purposes.

Oracle introduced major changes to the OCI API in version 8. While many of the concepts remain similar, all the function calls have been renamed and most have a different calling sequence. Supporting both OCI 7 and OCI 8 would require more than a handful of **`#ifdef`** statements – we would need to develop and support a separate interface for OCI 7. In addition, we would need to license Oracle 7 on all supported systems, which would significantly increase our development costs. While we recognise that there is a large Oracle 7 installed base, a year or so from now most of them will have upgraded to Oracle 8, thus the cost for developing an OCI 7 interface and purchasing six Oracle 7 licenses would have to be amortised over a period of about a year, leading to much higher licensing costs for the OCI 7 version. Furthermore, it is possible to use LISTSERV's OCI interface with an Oracle 7 server: the only component which must be at version 8 is the client, i.e. SQL*Net. As Oracle-based LISTSERV installations typically run on a separate, dedicated system, this simply means that you need to purchase a version 8 or higher client license for the system in question. It is not necessary to upgrade the server to Oracle 8 or higher. Oracle 7 is also supported through the ODBC interface.

As noted above, L-Soft formally supports Oracle 8i, 9i, and 10g.

4.1.3. Mail-merge

Documented Restriction for LISTSERV 14.4 and following: LSMTP is no longer required in order to use LISTSERV's mail-merge functionality. However, in order to use mail-merge with non-LSMTP mailers, you must set the site configuration variable EMBEDDED_MAIL_MERGE to a value of 1 (that is enabled). For more information, see the LISTSERV 14.4 release notes.

Documented Restriction for LISTSERV 14.3 and earlier: Note that LISTSERV's mail merge functionality REQUIRES the use of LSMTP Classic as the outgoing MTA. Mail merge does not work with sendmail, qmail, Post.Office, Netscape Mail Server, Microsoft Exchange, PMDF, MX, or any other MTA except L-Soft's LSMTP Classic mailer.

Under unixes not supported by LSMTP Classic this may require that you set **`SMTP_FORWARD=`** accordingly in **`go.user`**, to point to a separate machine running LSMTP (for instance, a dedicated Windows NT LSMTP machine). Under OpenVMS or Windows NT can run LSMTP Classic either on the same machine (the preferred method), or on a separate machine if desired. The main point is that the outgoing mail-merge postings MUST be handled by LSMTP Classic. (LSMTP Lite does not support mail-merge.)

LISTSERV's mail-merge support allows you to send individually customised messages to large numbers of recipients with very high throughput. The mail-merge functions support:

- Simple substitutions, such as "Dear &firstname;".
- Conditional blocks, such as a birthday greeting sent when the message happens to coincide with the recipient's birthday, or a warning when the balance of the account is negative.

- Special facilities to send promotional banners to a randomly generated subset of the recipients. For instance, you can indicate that a first banner should be sent to a random subset of 200 recipients, while another banner is sent to a randomly selected (but distinct) series of 500 recipients, and others receive a third banner, or no banner at all.

- Easy support for "few of many" topic subscription, such as a service offering news about movie actors (many registered actors, while most people will only want news about a handful of them).

- Full integration with the DBMS interface, allowing recipients to be selected through arbitrary SELECT statements, while every column that can be converted to a character string is made available as a mail-merge field.

- A simple bounce processing and collection system – LISTSERV processes and decodes all bounces, and writes the failing addresses to a plain-text file. You can group related mailings in the same bounce file or use a separate file for each mailing, whichever makes the most sense in your context. As each message is sent in "probe" format, even non-standard bounces will be processed accurately, as long as the remote MTA sends bounces to the correct (RFC821 MAIL FROM:) address.

Please remember that for sites running LISTSERV 14.3 and earlier, mail-merge support requires version 1.1b of LSMTP Classic (LSMTP Lite does not support mail-merge operations). LSMTP is currently available for Windows NT and OpenVMS Alpha. Note that LSMTP and LISTSERV need not reside on the same hardware; by way of example, it would be possible to run the DBMS-enabled version of LISTSERV on any supported unix platform, with LSMTP Classic 1.1b running on Windows NT.

4.2. Pre-installation tasks

Before installing DBMS and mail-merge support, please review the following steps and make sure that your selected target system is ready to receive this update.

- DBMS support requires version 1.8d or later of LISTSERV Classic or Classic HPO (DB2 support requires version 1.8e or later).

- LISTSERV 14.4 and following: Mail-merge support requires LISTSERV Classic or Classic HPO, and either
  - version 1.1b or later of LSMTP Classic; or
  - EMBEDDED_MAIL_MERGE set to 1 in the site configuration file.

- LISTSERV 14.3 and earlier: Mail-merge support requires version 1.8d or later of LISTSERV Classic or Classic HPO, and version 1.1b or later of LSMTP Classic.

- If you are planning to use the DBMS interface, you must install vendor-supplied DBMS support files on the target machine before installing the LISTSERV update. For ODBC (Windows), the appropriate drivers are already installed as part of the operating system under supported versions of Windows. For OCI, you need to install and configure the Oracle8 client files (SQL*Net et al.) The OCI material is typically licensed and not freely redistributable, and thus does not come with the LISTSERV kit. (Note that OCI is not supported natively by LISTSERV under Windows, but can be accessed via ODBC.)

- If you are using Windows, you must be running at least Windows 2000 with Service Pack 4 applied. *Windows NT 4.0 is no longer supported.* The current Windows

installation kits query the operating system for the current version and service pack, and will abort the installation if you are not running the minimum required version. L-Soft no longer supports Windows NT. Windows 2000 (Server and Workstation), Windows 2003 Server, and Windows XP (Professional and Home) are currently supported.

- If you are planning to use the mail-merge functions with LSMTP, be sure that you are running LSMTP version 1.1b. Otherwise you must upgrade LSMTP. While you can upgrade LISTSERV first, it is best to start with LSMTP. Upgrading LSMTP requires a 1.1b license key from your sales representative.

- If using the DBMS interface, you may want to create a DBMS username for LISTSERV in advance, and grant it the CREATE SESSION (mandatory) and CREATE TABLE (optional) privileges. If you are planning to create all tables yourself, you should not grant CREATE TABLE to LISTSERV's DBMS username.

- Note that a compiler is required to use the OCI interface on unix systems. L-Soft may not legally ship pre-linked executables containing the SQL*Net library.

- A compiler is also required to use the CLI interface on AIX, for similar reasons.

4.2.1. Selecting a suitable DBMS product

This section applies only to ODBC users. OCI users will always be using Oracle, and CLI users will always be using DB2, neither of which exhibit any of the problems mentioned in this section.

While L-Soft does not ordinarily recommend or endorse specific hardware or software brands, some database products (especially low-end PC offerings) may not be suitable for use together with LISTSERV. Without advocating one brand over another, L-Soft recommends the use of a DBMS that does not exhibit the problems mentioned below. All error messages are in reference to the diagnostics printed by the LISTSERV ODBC interface during startup.

- **[FATAL] LIKE operator has no ESCAPE clause, errors will occur!**
  This error indicates that the DBMS does not support any kind of "escape clause" for the LIKE operator. In practice, it means that whenever LISTSERV attempts a search containing a percent sign or underscore, the results will be incorrect (you may also get an ODBC error). This makes the DBMS unusable as a data store for LISTSERV lists. However, if you only plan to use the DBMS for mail-merge jobs, this restriction may be immaterial as LISTSERV will only be executing the SQL statements that you provide in the mail-merge job.

  SQL Server users should note that ESCAPE support was added in version 6.5. L-Soft will not support SQL Server 6.0 or older as a data store for LISTSERV lists. The Microsoft Access DBMS product also appears to have this restriction and is not supported as a data store for LISTSERV lists.

- **[SEVERE] Max active stmt: 1 - expect uncommitted read & unrequested commits**
  With a maximum of one active statement per transaction, the ODBC interface is unable to carry out typical SELECT ... UPDATE ... UPDATE ... COMMIT sequences using a single transaction, because the SELECT remains active until the COMMIT and prevents the execution of the UPDATE statements. To bypass this problem, the ODBC interface will use two transactions for these sequences. However, the two

transactions will typically look like independent applications to the DBMS, and will suffer from "transaction isolation," a vital DBMS feature that permits shared database access by multiple unrelated applications. As LISTSERV expects that an update will be reflected in a subsequent search, whether it has been committed or not, the ODBC interface will be forced to commit updates before beginning a new SELECT, even when LISTSERV had not requested a commit. In addition, if the DBMS does not support row-level locking, the ODBC interface will hang when attempting to execute the UPDATE statement, because the table is locked by the transaction containing the SELECT! To avoid this, the ODBC interface may switch to the lowest level of transaction isolation, "uncommitted read." If LISTSERV is the only application writing to the tables containing the data, this will be of no consequence, otherwise LISTSERV may see uncommitted changes made by other applications.

SQL Server artificially exhibits this condition. In reality, SQL Server does support multiple statements per transaction, but its ODBC driver reports otherwise, and will only allow one statement per transaction unless using dynamic cursors (or a related option). Dynamic cursors, however, will only work (with SQL Server) if the table contains a unique index; otherwise, the cursor is downgraded and leads to the one-statement behaviour. It is possible that third-party ODBC drivers may allow and report unlimited statements per transaction.

- **[SEVERE] FOR UPDATE clause not supported, no locking will occur**
  Some entry-level DBMS products may not support locking at all, or may only support it through a proprietary interface (rather than via SQL commands). In that case, LISTSERV will be unable to lock the rows it is in the process of updating, which may lead to incorrect behavior if you have other applications updating the tables.

- **[SEVERE] '=' operator is case-insensitive, results may be incorrect!**
  With some DBMS products, the equality operator is case-insensitive. While LISTSERV is generally case-insensitive, it does of course have the ability to make case-sensitive searches, and will do so on occasion. LISTSERV's built-in data management functions typically use the e-mail address as a unique, case-sensitive, primary index into the list, and LISTSERV assumes that it can use the e-mail address as a kind of "row ID" if the need arises. LISTSERV is not programmed to "doubt" the equality of a successful search for an e-mail address it had previously retrieved, as this is algorithmically impossible with its built-in functions. While this is a severe error in that it can lead to incorrect results, in practice it only has limited impact.

  SQL Server usually has case-insensitive equality configured by default. While it is possible to change this setting, this can only be done by reinstalling the product from scratch, which is usually unacceptable. In addition, the setting is global and affects all tables, all users, etc. In practice, the impact does not warrant the re-installation of an existing system.

- **Cursor behaviour limits ability to commit before logical close**
  This warning indicates that the DBMS will "forget" the results of any active SELECT statement whenever a transaction is committed. As LISTSERV's built-in data management functions do not have this restriction, LISTSERV will often commit changes while a search is in progress. For instance, if you issue the command **`SET XYZ-L DIGEST FOR *@AOL.COM`**, LISTSERV will search for users matching the pattern **`*@AOL.COM`** and, for each match, update the subscription options, send an e-mail message, and commit. This way, if the command is interrupted and re-executed, users who already received an e-mail notice will not receive a second copy. When this warning is printed, the ODBC interface will ignore any commit request from LISTSERV that would abort an active search. The transaction will always be

committed when the LISTSERV command completes, so the final results will always be accurate. However, in extreme cases (such as a SET command updating every record in the database), the transaction might generate a very large amount of uncommitted data, and require a lot of rollback space.

SQL Server users should note that while the SQL Server ODBC driver will cause this message to be displayed unless a non-standard ODBC call has been issued in advance, in practice it has no operational impact as the ODBC interface will have to use a separate transaction for update activity anyway. Thus, the ODBC interface is free to commit whenever requested by LISTSERV. SQL Server itself is able to preserve active SELECT statements across a commit, it is the ODBC driver which requests the "close cursors on commit" behavior.

- **Multi-row operations are unavailable**
  This is a performance warning indicating that bulk ADD operations may be slowed down. In practice, this warning only occurs with DBMS packages that offer limited performance anyway, and can be safely ignored.

Except as specifically indicated above, any error marked as FATAL or SEVERE can potentially lead to incorrect results. If you are planning to use the DBMS as a data store for LISTSERV lists, FATAL errors are unacceptable and SEVERE errors need to be investigated carefully. If you are only planning to use the DBMS interface for mail-merge operations, both SEVERE and FATAL errors may be acceptable as LISTSERV will only be executing the SQL statements provided in the mail-merge job. The script or person providing these statements is then responsible for making the necessary adjustments to work around the DBMS restrictions.

4.3. Installation

Installation instructions have been broken down by operating system. Make sure that your 1.8e license is installed before you begin, and that you have made a backup of your LISTSERV directory tree. All kits are located on FTP.LSOFT.COM under the appropriate LISTSERV/*platform* directory (ie, LISTSERV/unix, LISTSERV/Windows, LISTSERV/VMS).

4.3.1. Windows NT/2000/XP

Standard Windows kits include support for ODBC and mail-merge by default. Since LISTSERV 1.8e has introduced support for multiple simultaneous ODBC connections, support for Oracle databases via OCI (which had been provided in 1.8d as a work-around for this problem) has been discontinued. Oracle databases may still be accessed via the ODBC driver for Oracle.

4.3.2. OpenVMS Alpha

The OpenVMS kits include mail-merge support and optional OCI support (selected at LINK time). Two kits are available:

- axp.zip for all Alpha systems.
- axp-ev56.zip for Alpha systems with an EV56 processor (this build uses the new byte instructions and must be used only on systems with an EV56 or EV6 processor, if in doubt use the other kit).

This is a standard VMSINSTAL kit. If you want to install the OCI interface, you must run the appropriate ORAUSER.COM procedure before installing the kit. Otherwise, you will not be asked whether you want to install OCI support.

If you do install OCI support, you will need to modify your LISTSERV startup procedures to execute ORAUSER.COM before starting LSV.EXE.

4.3.3. Unix

LISTSERV 1.8e through 14.4 for unix kits support Oracle and DB2, and LISTSERV 14.5 for unix kits support Oracle, DB2, and MySQL (via unixODBC) in a single, universal kit. This kit contains both a precompiled 'lsv' executable (which does not support any database), and a set of object files allowing you to link a new 'lsv' that supports any combination of databases for which you have a run-time environment on the machine running LISTSERV. The following object files are included:

`lsv.o`	Main object file for linking 'lsv'
`nooci.o`	Link with lsv.o to disable OCI (Oracle) support
`nocli.o`	Link with lsv.o to disable CLI (DB2) support
`nouodbc.o`	Link with lsv.o to disable unixODBC support

If a particular database is not available for your operating system, the corresponding noxxx.o file will have been pre-linked into lsv.o and will not be included in the kit. A table showing LISTSERV DBMS support under unix follows:

Unix variant	OCI Supported	CLI Supported	unixODBC Supported
AIX	Yes	Yes	Yes
BSDi (through 14.3 only)	No	No	n/a
FreeBSD	No	No	Yes
HP-UX	Yes	Yes	Yes
Irix (through 14.3 only)	No	No	n/a
Linux-S390	No	No	Yes
Linux (x86 platforms)	Yes	Yes	Yes
OSF/1 (aka Digital Unix/Tru64)	Yes	No	Yes
OSF/1 on EV6 architecture	Yes	No	Yes
Solaris SPARC	Yes	Yes	Yes
Solaris x86 (through 14.3 only)	Yes	Yes	n/a

Note, however, that you may relink LISTSERV with only the following combinations of DBMS support:

- OCI only
- CLI only
- unixODBC only
- OCI and CLI
- OCI and unixODBC

LISTSERV *cannot* be relinked with support for both CLI and unixODBC at the same time. This is due to the fact that the two implementations are quite similar and share function names inside LISTSERV.

The current LISTSERV for unix kits contain a Makefile which is set up to relink 'lsv' without any DBMS support by default. A new "OS-specific flags" section has been added where you can add or remove DBMS support simply by adding or removing the reference

to the appropriate no*.o file. For instance, if you are running LISTSERV 14.5 under Solaris, the default flags line is

```
CFLAGS_Solaris=nooci.o nocli.o -lsocket -lnsl
```

If you want to relink 'lsv' with Oracle support, simply change this line to

```
CFLAGS_Solaris=nocli.o -lsocket -lnsl
```

If you want to relink 'lsv' with DB2 support, you would change the line to

```
CFLAGS_Solaris=nooci.o -lsocket -lnsl
```

If you want both Oracle and DB2, remove both the **nooci.o** and **nocli.o** references:

```
CFLAGS_Solaris=-lsocket -lnsl
```

Relinking 'lsv' with unixODBC support is not quite as intuitive. You would use the following flags line:

```
CFLAGS_Solaris=nooci.o -lsocket -lnsl -lodbc
```

For unixODBC, you must leave CLI support enabled because CLI and unixODBC share internal function names in LISTSERV, as noted above. In addition, you must also link explicitly to the unixODBC library (the -lodbc flag).

(As noted in the Makefile, Solaris requires that -lsocket and -lnsl must be linked in all cases, so don't remove these references from CFLAGS_Solaris under any circumstances.)

The other supported unixes are configured in a similar manner.

PLEASE NOTE CAREFULLY that you should not relink with DBMS support unless you have the appropriate DBMS support (SQL*Net, DB2, unixODBC) installed on your machine. Without this support, the link option will fail.

4.3.4. Verification

When done, start LISTSERV normally and send a few test commands before proceeding with the post-installation tasks. This will ensure that the installation was successful and that, where applicable, the appropriate environment is configured for the OCI or CLI library. Note that the DBMS interface will not be enabled until you add the necessary authentication/login information to your configuration files.

4.4. Post-installation tasks

Post-installation instructions have been broken down by feature. References to the "LISTSERV configuration" correspond to SITE.CFG under Windows, SITE_CONFIG.DAT under OpenVMS and go.user for unix systems. Do not forget to export configuration variables under unix.

4.4.1. Mail-merge

For best performance, make sure that MAXBSMTP is set to at least 5000. If you want to allow individual list owners to send mail-merge messages for their respective lists, set the

DIST_OWNER_MAIL_MERGE configuration variable to the value 1. By default, this option is disabled. Note that the LISTSERV administrator is always allowed to send mail-merge messages; it is assumed that the administrator knows whether mail-merge is supported, either because LSMTP is being used for the outbound mail or because EMBEDDED_MAIL_MERGE is enabled, whereas individual list owners may not have this information.

4.4.2. ODBC interface

To activate the ODBC interface, you must first create a system-wide ODBC data source. Open the Control Panel, double-click on the "ODBC 32" icon, select the "System DSN" tab, click "Add..." then select the appropriate driver and fill out the driver-specific form. L-Soft does not recommend supplying passwords in a system-wide DSN (if offered at all by the driver) because any Windows NT user has access to this information.

For the purposes of this example, we will assume that the DSN you just created is called GREEN. You would then add the following lines to SITE.CFG:

```
ODBC_DSN=GREEN
ODBC_UID=...
ODBC_AUTH=...
```

Replace the ellipses with appropriate (DBMS-specific) authentication information. While officially called the "authentication string" in the ODBC specifications, ODBC_AUTH is often called "password" in vendor documentation.

See also "Generic DBMS post-installation tasks", below, for instructions that are common to ODBC and OCI.

4.4.3. OCI interface

To activate the OCI interface, you must add an OCI_CONNECT parameter to your LISTSERV configuration. Its value should be an OCI configuration string, typically a service name from TNSNAMES.ORA. In addition, you may need to add OCI_UID and OCI_PWD parameters. If you do not supply a userid and password, the OCI interface will login with external (i.e. operating system) authentication.

Note that, under Windows NT, external authentication is based on the NT userid under which the application (LISTSERV) is running. When you run LISTSERV interactively, you are usually not running LISTSERV under the same NT userid as when it is started as a service. Thus, if using external authentication, Oracle may fail to login when started as a service. To circumvent this problem, use standard authentication (IDENTIFIED BY password) or use the Control Panel to change the account under which the LISTSERV service is running.

See also "Generic DBMS post-installation tasks", below, for instructions that are common to ODBC and OCI.

4.4.4. CLI interface

This support is similar to the ODBC support documented in 4.4.2, above, but the configuration variables are called `CLI_DSN`, `CLI_UID`, and `CLI_AUTH` .

4.4.5. unixODBC (UODBC) interface

Starting with LISTSERV 14.5, unixODBC and MySQL are supported for use with LISTSERV's DBMS features.

The prerequisite for this support is that unixODBC *must* be installed on the unix machine that is running LISTSERV.

The LISTSERV implementation is similar to that for ODBC and CLI, except that the prefixes are UODBC_*. You define UODBC_DSN and so on in your configuration, and you use "DBMS= UODBC" or just "DBMS= Yes" in your lists and DISTRIBUTE jobs. For instance, the site configuration for a unixODBC datasource called GREEN might look like this:

```
UODBC_DSN="green"
UODBC_UID="listserv"
UODBC_AUTH="fiatlux"
export UODBC_DSN UODBC_UID UODBC_AUTH
```

It is possible to connect to all kinds of databases (as opposed to MySQL only) using unixODBC, but L-Soft has introduced this interface primarily to work with MySQL, and does not formally support other DBMS products accessed via unixODBC. We would be interested in hearing the results of tests conducted in the field with other DBMS products and would be willing to consider adding support based on those results.

Because the documentation for unixODBC is sparse and sometimes contradictory, we provide instructions for a simple installation and configuration of unixODBC below.

4.4.6. Connecting to multiple simultaneous database sources

The release version of LISTSERV 1.8d allowed only one database connection to be defined at a time, which was inconvenient for some sites which wished to integrate user data kept in multiple tables or in multiple locations. Under Windows NT/2000, it was possible under 1.8d to install a supplemental kit that had OCI support and sites could then define both an ODBC and an OCI connection at the same time. This did not prove to be efficient, and during 1.8e development functionality was added to allow multiple simultaneous database connections.

Alternate data sources are specified as follows:

In a List Header:

*** DBMS= Yes,...,SERVER(*server_alias*)**

In an ad-hoc DISTRIBUTE job:

DISTRIBUTE MAIL-MERGE DBMS=YES(EMAIL=EMAILADDR,...,SERVER=*server_alias*)

The SERVER specification is optional; it defaults to SERVER(DEFAULT), which for backward compatibility is identical with the server defined in ODBC_DSN= in the site configuration file.

When defining an alternate server, it is recommended that ***server_alias*** be defined as an alpha string as opposed to a numeric string, in order to avoid certain problems in some ODBC drivers. There is no character limit for the ***server_alias*** string, but for simplicity's sake it should not be overly long.

To use ODBC as an example, the default ODBC data source is defined in the site configuration file as before, using the site configuration variables ODBC_DSN= ODBC_UID=, ODBC_AUTH=, and so forth. (Obviously, if you are using OCI or CLI, you use the equivalent site configuration keywords for those interfaces instead).

Alternate ODBC data sources are then defined with additional parameters of the form

ODBC_DSN_*server_alias*=
ODBC_UID_*server_alias*=
ODBC_AUTH_*server_alias*=

For OCI, you first define your default OCI connection, then alternate OCI data sources are defined with additional parameters of the form

OCI_CONNECT_*server_alias*=
OCI_UID_*server_alias*=
OCI_PWD_*server_alias*=

Similar to ODBC, the default data source for OCI is OCI_CONNECT=, not OCI_CONNECT_DEFAULT=, in order to be backwards compatible with 1.8d syntax.

And finally, for CLI, you first define your default CLI connection, then alternate CLI data sources are defined with additional parameters of the form

CLI_DSN_*server_alias*=
CLI_UID_*server_alias*=
CLI_AUTH_*server_alias*=

A default data source MUST always be defined, otherwise the driver will disable itself. You do not have to use the default data source for anything and it does not even need to be a valid login, but it needs to be there as an indicator that you have configured ODBC/OCI/CLI and want it to be activated. The driver itself does not know which data source(s) will be accessed until it is called for the first DISTRIBUTE job or list posting after startup.

Please note carefully that not all database types are supported across all platforms. For instance, CLI is not currently supported natively under the Windows NT/2000 version of LISTSERV 1.8e, but note that CLI databases can be accessed via ODBC if need arises.

4.4.7. Generic DBMS post-installation tasks

A number of additional configuration variables are available to alter the default behaviour of the DBMS interface. In the following discussion, a "DBMS list" refers to a list whose membership data is stored in a DBMS, as opposed to a traditional LISTSERV list where LISTSERV keeps the membership data in the xxx.LIST file.

- **DBMS_DEFAULT_TABLE** (default: LISTSERV) – default value for the name of the table in which DBMS lists are stored. Specify either a valid table name, or an asterisk to name the table after the list.

- **DBMS_DEFAULT_EMAIL** (default: EMAIL) – default value for the name of the column containing the subscriber's e-mail address.

- **DBMS_DEFAULT_UEMAIL** (default: empty string) – default value for the name of the optional column containing an upper-case copy of the subscriber's e-mail address. If

set to the empty string, no such column is created or used. See below for more information on this performance option.

- **DBMS_DEFAULT_NAME** (default: NAME) – default value for the name of the column containing the subscriber's name.

- **DBMS_DEFAULT_OPTIONS** (default: *) – default value for the name of the column containing the subscriber's LISTSERV subscription options. An asterisk indicates that the column should be named after the list.

- **DBMS_NO_HOSTNAME_ALIASING** (default: 0) – when set to 1, disables hostname aliasing for increased performance with some DBMS products. L-Soft recommends using the default value of 0.

When using all the default options, DBMS lists will be kept in a table called LISTSERV, with the e-mail address in a column called EMAIL, the name in a column called NAME, and one additional column for each DBMS list. The name and e-mail address will be shared, that is, a change in the user's name for list A is automatically reflected in list B. Of course, each of the layout parameters can be overridden on a per-list basis.

4.4.8. Performance options

The DBMS interface offers two options to improve lookup performance and overcome a fundamental difficulty in obtaining good e-mail lookup performance from a traditional SQL DBMS. An Internet e-mail address, as stored by LISTSERV in the DBMS, has the following format:

userid@hostname

Both halves of the address present a performance challenge.

The ***userid*** is case-sensitive. Internet standards and current industry practice demand that the case of the userid be respected. Failure to do so will lead to undelivered mail. Thus, LISTSERV must store the userid in the DBMS exactly as it was provided to LISTSERV. People, on the other hand, are not used to making a difference between JOE and Joe. When asking LISTSERV to remove Joe from the list, they will expect JOE's subscription to be cancelled, not a message claiming that there is nobody named Joe on the list. Thus, LISTSERV must make a case-insensitive search when looking up an address:

```
SELECT ... WHERE UPPER(EMAIL) = 'JOE@XYZ.COM';
```

Unfortunately, even on an indexed column, this search will typically require a full table scan. While it would be technically possible for the DBMS to use the index together with the UPPER function, in practice this optimization has not been implemented, because it is not frequently required in a typical DBMS environment. To alleviate this problem, LISTSERV supports the optional use of an additional column, containing an upper case copy of the e-mail address. The above search can then be rewritten as:

```
SELECT ... WHERE UEMAIL = 'JOE@XYZ.COM';
```

This search will make use of any available indexes. The drawback is that the column takes up additional space and must be set and changed together with the e-mail address, or lookup results will be incorrect. If the table is to be updated by applications external to LISTSERV, L-Soft recommends adding a CHECK clause to make sure that

the UEMAIL column is always set correctly. If LISTSERV is the only writer, this is not necessary.

While the ***hostname*** is not case-sensitive, many sites use mail systems where users appear to have a different hostname based on a variety of technical factors. LISTSERV supports a feature called "hostname aliasing" which allows it to know that, for instance, *joe@classic.msn.com* and *joe@msn.com* are the same person. Thus, when your customer support department receives a complaint from *joe@msn.com* asking to be deleted from your mailing lists, LISTSERV will automatically delete *joe@classic.msn.com* rather than report that Joe is not subscribed to the mailing list. The drawback, however, is that LISTSERV must formulate the lookup as follows:

```
SELECT ... WHERE UEMAIL LIKE 'JOE@%';
```

LISTSERV then reviews the selected entries and determines whether they are a valid match for *joe@msn.com*. With a good DBMS, this will not introduce any performance problem. The DBMS will use the index for LIKE searches until the first wildcard is encountered, and in practice there will not be many entries left to scan. Some DBMS, however, may simply not use the index for LIKE searches. In this case, you may want to set the DBMS_NO_HOSTNAME_ALIASING configuration parameter to 1, to disable the hostname aliasing feature for DBMS lists. The drawback, of course, is that LISTSERV will not be able to match *joe@msn.com* to *joe@classic.msn.com*. L-Soft recommends upgrading to a more advanced DBMS product rather than disabling hostname aliasing.

4.5. Creating DBMS lists

4.5.1. Configuring a list to use the DBMS

This section assumes that the reader is familiar with the process of creating LISTSERV lists and updating their list header. Refer to the *Site Manager's Operations Manual* for more information.

The use of the DBMS as a data store for list membership information is controlled by the "DBMS=" list header keyword. To create a DBMS list, specify a "DBMS=" keyword as follows:

```
DBMS= Yes[,TABLE(xxx)][,EMAIL(xxx)][,NAME(xxx)][,UEMAIL(xxx)][,OPTIONS(xxx)]
```

Brackets indicate optional parameters. That is, "DBMS= Yes" defines a DBMS list with the default values for TABLE, EMAIL, NAME, UEMAIL and OPTIONS, as described in Generic DBMS post-installation tasks above. The EMAIL, NAME, UEMAIL and OPTIONS column should have a data type compatible with VARCHAR (not CHAR or other fixed-length data types). The UEMAIL column, if used, should be the primary key (at a minimum, it must be indexed), and should have the same width as EMAIL; the EMAIL column should not be indexed. You can set the column widths as you want, however LISTSERV will assume that EMAIL is at least 80 characters wide, and OPTIONS should allow at least 40 characters. If you let LISTSERV create the table, it will use a width of 128 for all columns.

LISTSERV 1.8d only: Under Windows NT/2000, it is possible to use both ODBC and OCI interfaces simultaneously. In this case, you can use "DBMS= OCI,..." or "DBMS= ODBC,..." to tell LISTSERV which interface to use for a particular list. You can also use "DBMS= Yes,..." if you add a DBMS_DEFAULT_INTERFACE parameter to SITE.CFG with the value ODBC or OCI.

Please note: OCI support for Windows NT/2000 has been discontinued in version 1.8e as LISTSERV now supports multiple simultaneous ODBC database connections. The above paragraph DOES NOT apply to LISTSERV 1.8e.

When migrating an existing list to use a DBMS, you are responsible for migrating the subscriber data to the DBMS, if necessary (in many cases, the subscriber data will already be in the DBMS, possibly in a slightly different format). Once you add the "DBMS= Yes" keyword, LISTSERV stops accessing subscriber data from the xxx.LIST file and uses the DBMS instead.

4.5.2. Importing subscribers into a DBMS list

Subscribers can be imported into a DBMS list using the **ADD IMPORT** command, also known as "bulk add." However, despite the name this is not an "import" operation in the database sense; it does not disable logging or rollback and is based on normal transactional operations. You may be able to obtain better performance using specialized import tools provided by your DBMS vendor.

While **ADD IMPORT** is very efficient with a traditional LISTSERV list, it still eliminates duplicates and, when appropriate, updates existing rows with the data in the **ADD IMPORT** job. That is, it does not assume that all the new data can simply be inserted into the table. In SQL terms, an **ADD IMPORT** job is a series of SELECT ... UPDATE and SELECT ... INSERT sequences. Even with an index, this kind of workload tends to have transaction rates in the dozens or at best hundreds per second, rather than thousands.

With hostname aliasing disabled, a UNIQUE constraint on the e-mail address, and ignoring case-sensitivity issues, it would be possible to just issue INSERT statements, let the DBMS reject those for which a row with the unique key already exists, and re-issue them as UPDATE statements. However, most people will use hostname aliasing, there can be no guarantee that a UNIQUE constraint is present, and case-sensitivity is not easily ignored. Instead, a new option was added to **ADD IMPORT**, directing LISTSERV to preload the existing e-mail keys in memory before starting the transaction. This allows LISTSERV to skip most of the SELECT statements and just issue a large number of INSERTs.

To activate this option, simply add **PRELOAD** after the word **IMPORT**:

```
ADD XYZ-L DD=NEWSUB IMPORT PRELOAD
//NEWSUB DD *
joe@xyz.com Joe Doe
Helen Doe <hdoe@abc.def.com>
...
/*
```

You should use this option whenever importing a new list into the DBMS, or whenever adding a very large number of new users. If you need to add 100 entries to a table with a million rows, it will actually be slower to preload the rows, at least if you have an index.

The **PRELOAD** option is not necessary for traditional LISTSERV lists and does not normally lead to a significant performance improvement. However, when importing a new list (no existing subscribers), it does reduce CPU usage somewhat, and was used in the benchmark suite in order to provide an apple to apple comparison.

4.5.3. ADD IMPORT benchmark

The purpose of this benchmark is to give you a rough idea of the maximal performance that can be expected from an ADD IMPORT job. The benchmark adds 100,000 users to an empty list, without duplicates. That is, while LISTSERV still needs to determine that there are no duplicates, the job ends up being a pure INSERT workload. On a 1-CPU 300MHz Pentium II system with one IDE drive and Windows NT 4.0, the results were as follows:

DBMS and options	Throughput	Percent of traditional list
SQL Server 6.0 Base	1,240/sec	9.7%
SQL Server 6.5 SP4	1,370/sec	10.7%
Oracle 8.0.4 (ODBC interface)	6,800/sec	53.1%
Oracle 8.0.4 (OCI interface)	7,300/sec	57.0%
Traditional list (without PRELOAD)	12,500/sec	97.7%
Traditional list (with PRELOAD)	12,800/sec	100%

Note: the UEMAIL column was not used in this benchmark, and the table was created with no index. In a typical DBMS import scenario, the index is not created until all the data has been imported, as this would significantly impact import performance. Hostname aliasing was enabled, which is the recommended setting.

Even with the slower DBMS products, bulk add times will usually be deemed satisfactory. However, traditional lists are always faster, because their data organization and access methods are optimized for LISTSERV's purposes. The difference is even more noticeable with a real-world workload containing duplicates and a higher proportion of SELECT statements, as illustrated by the following benchmark, in which 1% of the addresses in the ADD IMPORT job were duplicates (LISTSERV searched for the entry and updated the NAME field). Here an index was used on the UEMAIL column, as 1,000 full table scans would have led to unacceptable performance.

DBMS and options	Throughput	Percent of traditional list
SQL Server 6.5 SP4	101/sec	0.8%
Oracle 8.0.4	2,270/sec	18.5%
Traditional list	12,300/sec	100%

(DB2 benchmarks were not available at release time.)

Note again that an index was created as the recipients were added, whereas the first benchmark populated a table that would need to be indexed after the fact.

Whereas the first benchmark was a best-case scenario, 1% of duplicates could be considered the worst likely possibility. As you can see, duplicates have virtually no impact on LISTSERV's built-in data management functions, which were designed specifically for this purpose. The DBMS, on the other hand, are required to create an index on the fly, which is contrary to normal practice and significantly decreases performance. On the other hand, 1,000 full table scans would have taken even longer.

4.5.4. Updating DBMS lists from an external application

DBMS lists can be updated from an external application (by directly making changes to the DBMS tables) with essentially no restrictions and no particular precautions. If you are using an UEMAIL column, however, you must make sure to keep it synchronized with EMAIL, as LISTSERV will assume that, for every row, UEMAIL = UPPER(EMAIL). L-Soft recommends using a trigger procedure or adding a CHECK clause if using an UEMAIL column with an external application directly updating the table.

To delete a subscriber from a DBMS list, you can either delete the row or set the OPTIONS column (which is typically named after the list) to NULL. By definition, if the OPTIONS column is NULL, the user is not subscribed to the corresponding list. When LISTSERV deletes a subscriber from a DBMS list, it deletes the row if the table is named after the list (i.e. if you have the list in a separate, dedicated table). If the table is shared, it sets the OPTIONS column to NULL.

To add a subscriber to the list, simply set the OPTIONS column to the empty string (or insert a new row with OPTIONS set to the empty string). This subscribes the users with all the default options for the list in question.

Please note carefully that most if not all database applications make a distinction between a NULL value and the empty string. If you are having trouble making users added from an external application show up when you review the list, ensure that you are setting the OPTIONS column to "" (the empty string) rather than NULL.

Conversely, if you are having trouble deleting users via an external application, ensure that you are really setting the OPTIONS column to NULL rather than to the empty string.

You can change NAME at any time and without any special precautions. Likewise, you can change EMAIL at any time, as long as you update UEMAIL simultaneously (if you are using an UEMAIL column).

4.5.5. Format of the OPTIONS column

The simplest way to change subscription options is to use the TCP/IP command interface (the "TCPGUI" interface, see chapter 6) to submit a SET command. However, you can also do this by changing the value of the OPTIONS column, which is a series of numbers (in decimal form) separated with semicolons, as follows:

```
4;flags_1;flags_2;reserved;subscription_date;topics;...
```

The first number is a version identifier, and must always have the value 4. The second number, *flags_1*, is a bitmask defining a first set of subscription options, as follows:

Option name	Value (decimal)	May not be set together with...
ACK	1	MSGACK,NOACK
MSGACK	2	ACK,NOACK
NOACK	4	ACK,MSGACK
NOMAIL	8	–
DIGEST	16	INDEX
INDEX	32	DIGEST
NOFILES	64	–
REPRO	128	–
IETFHDR	256	*HDR

DUALHDR	512	*HDR
FULLHDR	1024	*HDR
xxx822header	2048	DUALHDR,IETFHDR,SUBJHDR
NORENEW	4096	–
reserved	8192	–
CONCEAL	16384	–
EDITOR	32768	NOPOST,REVIEW
REVIEW	65536	EDITOR,NOPOST
NOPOST	131072	EDITOR,REVIEW
MIME	262144	–
SUBJHDR	524288	*HDR
reserved	1048576	–
HTML	2097152	MIME must be set

(Options not found in the table, eg MAIL or POST, have a bit value of 0 and do not need to be set in order to be in force. To negate them you set the complimentary command, eg, NOMAIL or NOPOST, with the appropriate bitmask value from the table.)

Bit positions marked as "reserved" must be left unchanged if updating an existing entry, and set to zero when creating a new entry. They do not correspond to subscription options and are used for internal book-keeping purposes. Please note that certain options are incompatible with each other, or require that other options be enabled. If these restrictions are not observed, results will be unpredictable.

The third number, *flags_2*, is a bitmask reserved for future use. While no bits are currently defined, make sure to leave its value unchanged when updating an existing entry, and to set it to zero when creating a new entry. The fourth number is used internally by LISTSERV and should be handled similarly.

The fifth number is the date at which the user subscribed to the list. You should leave it unchanged when updating an existing entry. For new entries, you can either set it to zero (in which case LISTSERV will report the subscription date as unknown), or set it to the number of complete days (i.e. not counting today) elapsed since and including January 1, 0001. For verification, this number is congruent to zero modulo seven on Mondays. It is equal to the C language expression time(0)/86400 + 719162 or, for OpenVMS users, to the Smithsonian base date plus 678575. (If you are familiar with the REXX language, this value is strictly equal to the value returned by Date('B').) As an example, the value for 23 June 1999 is 729927.

The sixth number is a bitmask indicating which topics the user is subscribed to. The lowest order bit (value 1) corresponds to the pre-defined OTHER topic. The second lowest order bit (value 2) corresponds to the first topic in the "Topics=" keyword, and so forth. If you set this number to zero, the user will not be subscribed to any topics. To subscribe a user to all topics, use the value 16777215.

When updating an existing entry, make sure to preserve any information following the topics number (when creating a new entry, do not provide any such information).

4.5.6. Sample OPTIONS column settings

Sometimes using the list's pre-set defaults by setting the column value to the empty string is not acceptable or useful. In that case you can set the OPTIONS column values per the table above. This section provides a typical example of how the OPTIONS column values are used. (See also 4.5.7, below, if you are migrating from a "standard" LISTSERV list to a DBMS-type list.)

(Note carefully that if you are changing existing values, you should not change the values in the third and fourth positions, as they are reserved for LISTSERV's internal use. Only when adding new users should these positions be set to zero.)

If you want the simplest defaults (which is not recommended, because the default when set this way is to use SHORTHDR and at minimum you should set new users to FULLHDR), you set

4;0;0;0;0;0

in the OPTIONS column and the users are added to the list. (Remember that the OPTIONS column is normally named after the list; we are not talking about a table column that is explicitly named "OPTIONS".)

If you want the standard default options that LISTSERV normally assumes if you don't make any special settings in the list header, you would use

4;1024;0;0;0;0

This sets the user to MAIL FULLHDR NOREPRO NOACK, which are the standard defaults. You might want to change NOACK to ACK (or possibly NOREPRO to REPRO instead), in which case the value in the second position would be 1025 (for ACK) or 1132 (for REPRO).

Note that you simply add the numbers from the table together to combine option settings; if you want the user to be set to the FULLHDR and REPRO options, you set the value to 1024 + 128 = 1132. However, you cannot set it to a value such as 1536 (1024 + 512), which would be FULLHDR DUALHDR -- an impermissible combination as the two values are mutually exclusive.

If you want the user to be set to ALL topics (if topics are defined for the list) by default, you put the value 16777215 in the last position, ie,

4;1024;0;0;0;16777215

Finally, if you want the user's subscription date to be included, you find the value per the specification found above and add it in the fifth position. Assuming that today's date is 23 June 1999, the value on that date is 729927, so the setting would now be:

4;1024;0;0;729927;16777215

Then a QUERY command sent for this user (for example, JOE@XYZ.COM in a list called ODBCTEST) would result in the following output:

```
>QUERY ODBCTEST FOR JOE@XYZ.COM
Subscription options for Joe Doe <joe@XYZ.COM>, list ODBCTEST:

MAIL            You are sent individual postings as they are received
FULLHDR         Full (normal) mail headers
NOREPRO         You do not receive a copy of your own postings
NOACK           No acknowledgement of successfully processed postings

Subscription date: 23 Jun 1999

The topics you subscribe to are: All
```

4.5.7. Preserving options when migrating from non-DBMS to DBMS lists

When migrating an existing (standard) LISTSERV mailing list to the DBMS type, note that it is possible to preserve existing user options without having to figure out the bit values for every user's existing options by hand.

A standard LISTSERV subscriber entry is a 100-column string, of which the first eighty characters are the subscriber's e-mail address and "real name" field. Columns 81-100 contain an encoded string that holds the various user option settings, for instance:

```
3AEAQABMcBMe////
```

If you take columns 81-100 from an existing list file, insert a semicolon after the first character, and store this in the option field, all the settings will be successfully migrated. So for instance if you have '**3AEAQABMcBMe////**' in columns 81-100, you store '**3;AEAQABMcBMe////**' in the option field. Note that the column value is not automatically converted to '4;' format until the value is updated, ie, until a change is made to the user options or LISTSERV otherwise updates the value. LISTSERV will then use '4;' format when updating the value, but until then it will remain as it was and LISTSERV will be able to use it without any trouble*. Do not change the value of the first character from 3 to 4 when migrating as this is guaranteed to cause breakage.*

4.6. Using the mail-merge functions

The mail-merge functions can be used at three different levels:

1. Using the web interface.
2. By sending DISTRIBUTE jobs to LISTSERV.
3. By connecting to port 25 of LSMTP and sending mail-merge command streams.

The first two methods will be described in this document. The web interface is ideal when the mail-merge process is supervised by a person, whereas the second method is best suited to automated procedures. In most cases, the third method requires significantly more complex programming to duplicate functions which are provided by LISTSERV when using one of the first two methods.

Please read the description of the web interface for background information even if you are only interested in the DISTRIBUTE interface.

4.6.1. Using the web interface

The web interface can be used for mail-merge operations where the data source is either a DBMS or a traditional LISTSERV list. That is, if the addresses and names of the subscribers are in, say, a proprietary application that you have purchased or developed, you will need to write a script to extract the information from the application in question and create a DISTRIBUTE job.

To prepare a mail-merge job using a DBMS back-end, go to the following URL:

```
http://.../wa.exe?P1&0=M
```

or

```
http://.../wa?P1&0=M
```

(depending on your OS of course, the former for Windows and OpenVMS, the latter for unix).

You will be prompted to enter an arbitrary SELECT statement to select the recipients who should receive a copy of the message. Every column in the SELECT statement which can be mapped to a character string is then available as a substitution in the message, HTML-style. That is, if you have a column called ACCTNO with the customer's account number, you can substitute it in the text of your message using **&ACCTNO;** (a NULL value is mapped to the empty string).

Only the LISTSERV administrator and other users allowed to use DISTRIBUTE (as defined by the DIST_ALLOWED_USERS configuration variable) can use the above URL and send arbitrary DBMS-based mail-merge messages. This is because this method allows you to issue arbitrary SELECT statements, which are not limited to the membership of a particular list. List owners can use another URL to send mail-merge jobs to their respective lists:

http://.../wa.exe?P1&0=M&L=*listname* (for NT/2000 and OpenVMS)
http://.../wa?P1&0=M&L=*listname* (for unix)

This interface does not prompt you for a SELECT statement – it implicitly selects all the subscribers matching the MAIL/NOMAIL/DIGEST/INDEX subscription criteria. In addition, you can provide a boolean expression to further restrict the recipient list, such as:

(&age < 15) and (&country = Canada)

If the target list is a DBMS list, all the columns in the table which can be mapped to a character string are available as substitutions. If the target list is a traditional LISTSERV list, only &EMAIL and &NAME are available, in addition of course to all the special substitutions, such as &*DATE (see below).

4.6.2. Sending DISTRIBUTE jobs to LISTSERV

The purpose of this section is to describe the mail-merge enhancements to the DISTRIBUTE function, rather than DISTRIBUTE itself. While it will usually not be necessary to learn all the details of the DISTRIBUTE function to prepare mail-merge jobs, please refer to chapter 3, above, if you do need further information about DISTRIBUTE.

A traditional (non mail-merge) DISTRIBUTE job has the following format:

```
//XYZNEWS-215 JOB ECHO=NO
DISTRIBUTE MAIL FROM=owner-nolist-xyznews@xyznews.com PW=xxxxx
//TO DD *
joe@xyz.com Joe Doe
hdoe@abc.def.com Helen Doe
...
/*
//DATA DD *
Date: Sat, 4 Jul 1998 22:47:24 +0200
From: XYZnews editor <xyzed@xyznews.com>
Subject: XYZnews issue #215
To: XYZnews recipients

Welcome to issue #215 of XYZnews!
...
```

```
/*
```

The job must be mailed to LISTSERV, either from the LISTSERV administrator's address (not recommended) or from an address defined in the DIST_ALLOWED_USERS configuration variable. The personal LISTSERV password corresponding to the sending address must be provided, or the job will be rejected. The "ECHO=NO" option suppresses the message that is normally returned to indicate when the job starts and ends, on the assumption that this information is not wanted (if an error occurs, a message is sent anyway). XYZNEWS-215 is an arbitrary job name for problem tracking purposes.

There are three types of mail-merge DISTRIBUTE jobs:

1. Jobs based on a DBMS back-end with an arbitrary SELECT statement
2. Jobs where the recipient data is extracted from an existing LISTSERV list (either a DBMS list or a traditional list)
3. Jobs where the recipient data is totally external to LISTSERV and is provided in the job stream, for instance after being extracted from a proprietary customer database with no DBMS functionality

The only differences are the options provided on the DISTRIBUTE command line, and the format of the //TO section. The format of the message section is the same with all three types of mail-merge jobs.

Note that some DISTRIBUTE command lines can get quite long and may wrap in your mail client, causing an error when the job is processed by LISTSERV. To avoid this you can use one or more "continuation cards" (see chapter 2.3 of this manual) to split the command over multiple physical lines. See for instance the second example below.

DISTRIBUTE job with DBMS back-end

These jobs are the simplest and use the following syntax. (This example assumes that your database contains the fields referenced, ie, EMAILADDR, ACCTNO, NAME, etc.)

```
//XYZNEWS-215 JOB ECHO=NO
DISTRIBUTE MAIL-MERGE DBMS=YES FROM=owner-nolist-xyznews@xyznews.com PW=xxxxx
//TO DD *
SELECT EMAILADDR,ACCTNO,NAME FROM ...
WHERE ...
AND ...
...
/*
//DATA DD *
Date: &*DATE;
From: XYZnews editor <xyzed@xyznews.com>
Subject: XYZnews issue #215
To: &*TO;

Welcome to issue #215 of XYZnews!
...
/*
```

The DISTRIBUTE command line now reads DISTRIBUTE MAIL-MERGE DBMS=YES, and the recipient section contains one or more SELECT statements. If multiple SELECT statements are included, you must end them with a semicolon, and you may not have more than one statement per line. The &*DATE; and &*TO; substitutions are not required, and simply serve to illustrate the fact that substitutions can now be placed in the message text (including the mail headers).

With the syntax shown above, the first column in the SELECT statement must be the one containing the e-mail address. The other columns are made available as substitutions (&NAME, etc.) This is the recommended syntax when writing a script to prepare the jobs, as only the columns actually used for substitutions are transferred from the DBMS server. Sometimes, however, the script may not know in advance which columns will or will not be used. In this case, a SELECT * may be used, as follows:

```
//XYZNEWS-215 JOB ECHO=NO
// DISTRIBUTE MAIL-MERGE DBMS=YES(EMAIL=EMAILADDR) ,
FROM=owner-nolist-xyznews@xyznews.com PW=xxxxx
//TO DD *
SELECT * FROM ...
WHERE ...
AND ...
...
/*
//DATA DD *
Date: &*DATE;
From: XYZnews editor <xyzed@xyznews.com>
Subject: XYZnews issue #215
To: &*TO;

Welcome to issue #215 of XYZnews!
...
/*
```

(Note the use of a "continuation card" in the DISTRIBUTE command line above, used because the line was so long it wrapped. See chapter 2.3 of this manual for more information on JOB card syntax.)

The EMAIL=EMAILADDR option tells LISTSERV the name of the column containing the e-mail addresses. All the other columns are made available for substitutions, provided of course that they can be mapped to a character string. Note, however, that even large columns (LONG et al.) will be transferred from the DBMS server. As a rule, this syntax should be avoided whenever you would normally avoid doing a SELECT * against the table.

DISTRIBUTE job with existing LISTSERV list

These jobs use the following syntax:

```
//XYZNEWS-215 JOB ECHO=NO
DISTRIBUTE MAIL-MERGE DBMS=LIST(XYZLIST,MAIL,DIGEST,INDEX) PW=xxxxx
//TO DD "(&age < 15) and (&country = Canada)"
//DATA DD *
Date: &*DATE;
From: XYZnews editor <xyzed@xyznews.com>
Subject: XYZnews issue #215
To: &*TO;

Welcome to issue #215 of XYZnews!
...
/*
```

Please note carefully that the example assumes that the database table containing the list also contains "AGE" and "COUNTRY" fields for the conditional selection.

This job selects all the recipients from the XYZLIST list whose subscription options are either MAIL, DIGEST or INDEX and for which the expression **`(&age < 15) and (&country = Canada)`** is true. By default, if you specify only DBMS=LIST(XYZLIST), LISTSERV will only select recipients with the MAIL option. The options you can include in this fashion are MAIL, NOMAIL, DIGEST and INDEX. Note that the **`FROM=owner-`**

nolist-xyznews@xyznews.com option has been removed – bounces are automatically integrated with the XYZLIST bounce stream and do not need to be redirected to a change log. If desired, however, you can override this behaviour by providing a FROM= keyword.

In addition, you can specify a boolean expression in the //TO section. This expression is in the same format as the conditional expressions used in LISTSERV's mail template files (described in the list owner's guide). If you do not want to do any further filtering, just set //TO to the empty string, or to the value "1" (true), for instance:

```
//TO DD "1"
```

Please note carefully that for list-based mail-merge it is not sufficient to define "//TO DD *". This will result in the error "Implicit DD (TO or DATA) not found in job stream."

For a DBMS list, this syntax is functionally equivalent to a SELECT * job, that is, every column that can be mapped to a character string is available as a substitution. For a traditional LISTSERV list, only &EMAIL and &NAME are available. This can still have its uses, for instance, to send a mail-merge message to AOL recipients only, you could use:

```
//TO DD "&email =* '*@AOL.COM'"
```

(note the single quotes surrounding the selection criteria--these are required).

DISTRIBUTE job with external mail-merge data

To provide the mail-merge data as part of the DISTRIBUTE job stream, use the following syntax:

```
//XYZNEWS-215 JOB ECHO=NO
DISTRIBUTE MAIL-MERGE FROM=owner-nolist-xyznews@xyznews.com PW=xxxxx
//TO DD *
*XDFN NAME="Joe Doe" AGE=23 country="Canada"
joe@xyz.com PROBE
*XDFN name="Helen Doe" Age=47 country=USA
hdoe@abc.def.com
...
/*
//DATA DD *,EOF
Date: &*DATE;
From: XYZnews editor <xyzed@xyznews.com>
Subject: XYZnews issue #215
To: &*TO;

Welcome to issue #215 of XYZnews!
...
/*
```

With this syntax, all the substitutions are provided in the job, preceding the e-mail address they refer to. There is no DBMS access, no reference to an existing LISTSERV list and no filtering or selection of recipients – you are providing an exact recipient list. Note that when the DISTRIBUTE command's **FROM=** option is set to an **owner-xxx** address, LISTSERV generates the mail merge message as a passive probe of the recipient. The probe comes at no extra resource cost when sending a mail-merge message.

You can provide multiple *XDFN lines, which can have any number of keyword=value pairs. There must be no spaces either before or after the equal sign. Keywords are not case sensitive, so the case of the keyword name is not relevant. The value must be

enclosed in double quotes if it contains spaces, double quotes or backslash characters. To escape a backslash or double quote in such a quoted string, precede it with a backslash. While LISTSERV will support *XDFN lines of up to 64k, bear in mind that you will probably send the job to LISTSERV via e-mail, in which case a lower limit may apply depending on your mail system. It is good practice to keep *XDFN lines to 998 characters or less, as this is the maximum length guaranteed to be successfully transmitted over the SMTP protocol.

Automatic bounce processing

In most cases, you will want LISTSERV to process all the bounces automatically for you. If you are using the DBMS=LIST syntax, bounces are, by default, integrated with the regular bounce stream for the list in question, and you have access to the full range of LISTSERV bounce processing features (see the description of the "Auto-Delete=" keyword in the list owner's guide for more information). Otherwise, L-Soft recommends using the "change log" feature in order to keep track of bounces. This is accomplished by including the following keyword in the DISTRIBUTE command line:

FROM=owner-nolist-*logname*@*hostname*

Where *logname* is an arbitrary name for the "change log" in which LISTSERV will be recording bounce activity, and *hostname* is the host name of the machine on which LISTSERV is running. You can use a different log file for every message, one file for each set of related messages, or just one for all the mail-merge messages you send; the decision is left up to you. The file will be located in LISTSERV's main directory (the one where, among others, permvars.file is located) and will be called nolist-*logname*.changelog. You do not need to create a list called NOLIST-*logname* and, in fact, you must not do that. The change log is a standard text file containing entries of the form:

```
19980704100221 BOUNCE jack@xyz.com
19980704100223 BOUNCE Joe.Doe@abc.def.com
```

LISTSERV will process all bounces silently, and store the bouncing addresses in the change log. Note that there may be other entries in the change log – be sure to ignore any lines which do not contain BOUNCE in the second column (in practice, "nolist" change logs only contain BOUNCE entries, but this could change in a future version). Because mail-merge messages automatically use passive probing, bounce processing is extremely accurate, even if the target mail server normally returns bounces in an unintelligible format.

If you want to process bounces yourself, simply provide a FROM= keyword pointing to the desired target address.

Using DBMS or list-based jobs without mail-merge

It is possible to use the DBMS= keyword to extract and select recipients from either a DBMS or a LISTSERV list, without using the mail-merge functions. Simply use the formats shown above, substituting DISTRIBUTE MAIL for DISTRIBUTE MAIL-MERGE. Naturally, you are then unable to put substitutions in the message text, however the message may use significantly less bandwidth and will usually be delivered faster.

4.6.3. Using substitutions

Mail-merge substitutions work just like HTML substitutions – you embed them in the target message using a sequence such as:

&NAME;

If the substitution is not defined (for instance, using the example **&NAME;** above), no substitution is performed and the resulting message will contain the literal text '&NAME;'. This can happen (still using **&NAME;** as an example) if there is no column called NAME in the DBMS, and can also happen in individual merge messages if the column called NAME is null for a given user. (Remember that a null value is not the same as a blank value in most DBMS products.)

In addition to the substitutions provided through the DBMS, list or DISTRIBUTE job stream, a number of special substitutions are always available:

- **&*DATE;** is an RFC822 date field (without the "Date:" keyword) suitable for insertion in mail headers. This makes it easier to develop scripts that prepare DISTRIBUTE jobs, and there is no performance penalty as this is a "false substitution" – one that has the same value for all recipients and is pre-processed by LISTSERV. By the time the outbound mailer gets the message, it no longer contains a substitution.

- **&*WA_URL;** is the URL of the LISTSERV web interface script, up to and including the script name. You would typically add a question mark and parameter list afterwards. As it is a false substitution, there is no performance penalty for using it. If the web interface has not been installed, it translates to the empty string.

- **&*TO;** is the e-mail address of the current recipient, suitable for insertion in a "To:" field.

- **&*INDEX;** is a unique, random number ranging from 1 to the total number of recipients. It can be used in conditional blocks (see below) to send a particular message to a random sample of recipients – an invitation to preview a new web site or an ad banner, for instance. This special substitution can only be used in a conditional block, and cannot actually be inserted in the text of the message.

- **&*URLENCODE();** is a function introduced in LISTSERV 1.8e which can be used to assist you in passing URLs that contain spaces or special characters. The function encodes the passed value so that it is valid in a URL (replacing special characters with "%" followed by two hexadecimal digits forming the hexadecimal value of the character). For instance, this works for substitutions such as **&*URLENCODE(&ID;);** where **&ID;** represents a field extracted from a database. A specific example would be to encode a URL pointing to a file with spaces in its name, for instance, a file called Year 2000 figures.html . This file name could be converted for you and placed into a URL in your mail-merge message like this:

  http://www.example.com/statistics/&*URLENCODE(Year 2000 figures.html);

  resulting in the URL being correctly displayed as

  http://www.example.com/statistics/Year+2000+figures.html

  in your message.

- **&*TICKET(*listname*);** and **&*TICKET_URL(*listname*,*command*);** are special substitutions used to issue command tickets, which are described below.

Using command tickets

Documented restriction: While it is possible to tell LISTSERV to issue a command ticket to a list owner address, for security reasons list owners cannot use command tickets for authentication of commands sent for the lists they own. If a list owner attempts to use a ticketed command on a list he owns, LISTSERV will respond

```
For security reasons, list owners may not issue ticketed commands. The
ticket was otherwise valid - this is not an error on your part.
```

When testing ticketed commands it is thus necessary that the list owner do the testing from an account that is not listed in Owner= for the list in question.

Command tickets allow you to provide safe, authenticated, single-click subscribe, unsubscribe or SET commands in a mail-merge message. They are implemented through a two-step mechanism. In the first step, LISTSERV generates a cryptographically protected ticket through a mail-merge substitution. This ticket allows anyone to execute a limited set of commands on behalf of the user for whom it was issued, and only for the list for which the ticket was issued. Typically, you would use the substitutions to construct a URL, or perhaps a HTML form with a number of buttons triggering various changes in subscription options or status.

In the second step, the user activates the URL or form, which presents the ticket to LISTSERV for verification. If the ticket is valid and has not expired yet, the command is executed. There is no need to use passwords or OK confirmations. Tickets are safe because they are, by design, only ever mailed to the person for whom they are issued. They are also limited to a particular list, and may only be issued at the request of the owner of that list (who does not need to use tickets to affect the user's subscription to the list in question). Finally, tickets expire after a month, to limit the impact of incidents where printouts of e-mail messages were misplaced, etc.

A ticket allows the use of SUBSCRIBE, SIGNOFF and SET commands, with arbitrary parameters. However, the name of the list is extracted from the ticket, and may not be changed. In fact, you do not specify the name of the list in the command. The simplest way to format a ticket URL is as follows:

```
<a href="&*TICKET_URL(XYZ-L,SET NOMAIL);">Turn off mail receipt</a>
```

This substitution expands into a URL which will execute a SET XYZ-L NOMAIL command when activated. To use a ticket in a form with several buttons, you could do as follows:

```
<form action="&*WA_URL;">
<input type=hidden name=TICKET value="&*TICKET(XYZ-L);">
<input type=hidden name=L value="XYZ-L">
<input type=submit name=c value="Unsubscribe">
<input type=submit name=c value="SET NOMAIL">
<input type=submit name=c value="SET MAIL">
</form>
```

You could also use an image map pointing to a script of your own making, a Java applet, etc. To execute the command, simply redirect the browser to:

http://.../wa.exe?TICKET=*ticket*&c=*command*&L=*listname*

The list name is optional: it is not required to use the ticket, but can be provided to direct the LISTSERV web interface to use the HTML templates for the list in question. You can customise the response on a per-list basis using the standard web template customisation features, which are described in the list owner's guide.

Performance considerations with command tickets

Unlike OK cookies, command tickets do not use up any disk space or require any kind of house-keeping. You can use them as often as you wish without ending up with gigabytes of pending cookie data on your LISTSERV server, as would be the case with OK cookies and a large daily newsletter offering multiple one-click commands. However, being cryptographically protected, they do require some amount of CPU time for generation, without which they would not provide much protection at all. So, do not worry if LISTSERV does not acknowledge execution of your 500,000 recipient job after 30 seconds as usual.

LISTSERV will generate the tickets only once, even if you refer to the same ticket multiple times in the message. Thus, you do not need to worry about using tickets in a single big form rather than multiple smaller forms, or once at the top and once at the bottom of the message. While it does cost a small amount of resources for the ticket to be merged with the message text in multiple locations, the CPU-intensive ticket generation only occurs once.

If you have a very large list which must be delivered very quickly at a specific time, as can often be the case in the news industry, command tickets may present a challenge. In this case, the best solution is to make use of the SMTP worker pool feature[5], rather than the "PRIME=" job option, to schedule the delivery of the message. Whereas "PRIME=" will hold the entire job and require you to estimate execution time in order to open the execution window long enough in advance, the SMTP worker pool feature can be used to create a worker pool which does not begin delivery until a specific time. You can then execute the job long in advance, but suspend delivery until the desired time.

4.6.3.1. New substitutions available in LISTSERV 1.8e-2003a "level set" release

Your LISTSERV 1.8e installation must have a build date no earlier than 26 Sep 2003 in order to use these features. To check your build date, issue a SHOW LICENSE command to LISTSERV. These substitutions are also available in LISTSERV 14.3 and following.

In addition to the above, two new built-in mail-merge substitutions and a related (optional) DISTRIBUTE keyword are available starting with the LISTSERV 1.8e-2003a "level set" release. These have been added to solve concerns about "(no name available)" appearing in the To: field, and at the same time the flaw in using:

```
To: "&NAME;" <&*TO;>
```

which is not correct for all possible values of &NAME.

This feature adds one optional DISTRIBUTE keyword:

```
NAMEFIELD=xxx
```

For instance,

```
DISTRIBUTE MAIL-MERGE DBMS=LIST(XYZLIST) NAMEFIELD=NAME PW=xxxxx
```

This indicates the name of the XDFN/DBMS field containing the name of the recipient. If absent, the name is unknown (see below).

[5] Described in the LISTSERV Tuning Guide, available separately (contact support@lsoft.com for a copy).

In the case of DBMS=LIST, the default value of NAMEFIELD=xxx is set automatically from the "DBMS=" keyword and/or the system defaults found in SITE.CFG. Note that the correct syntax is NAMEFIELD=NAME, not NAMEFIELD=&NAME; or similar. NAMEFIELD=xxx is not ignored for a list distribution. Any available column name can be specified for NAMEFIELD, at the risk of making a mistake. The design assumption was that in some cases there might be several name columns in the table, for instance with different character sets or one with and one without accents. It was thought best to allow this to override the internal default, even if the default is correct. However, normally one should omit NAMEFIELD=xxx for a list distribution and LISTSERV will provide the correct value.

Two substitution variables are added.

- **&*NAME;** is replaced with the variable specified in the (new) DISTRIBUTE option NAMEFIELD=xxx. If unknown, the empty string is substituted as a constant. This is just a convenient way to refer to the name field in examples or generic jobs, regardless of what it is really called.

- **&*TOFIELD;** is a correct RFC822 to field (without the "To:") for the supplied name and e-mail address. If the name is unknown or missing, the result is the same as &*TO. A missing name is NULL, the empty string or '(no name available)'. To clarify, the correct use is:

  To: &*TOFIELD;

Note that there is a performance cost for this option. The RFC822 rules are somewhat time-consuming; additionally, this also requires parsing XDFN lines to extract the name field (when not needed, LISTSERV adds its own XDFN). Finally, the NAME field is passed even if it is only used for &*TOFIELD.

4.6.4. Using conditional blocks

In addition to simple substitutions, you can include "conditional blocks" in your mail-merge messages. A conditional block is a group of lines which is only included if certain conditions are met. Here is an example of a mail-merge message with conditional blocks:

```
Date: &*DATE;
From: birthdays@xyz.com
Subject: Happy birthday, &FNAME;!
To: &*TO;

Happy birthday!

...

To celebrate the occasion, XYZweb is pleased to credit your account with
$10.00 in birthday credits. You can spend these credits anywhere in our
online store, but there's a catch! They will expire in a week if you do
not use them! So wait no further and go to your personalized web page at:

&PERS_URL;

.* Special offer for people who turn 18
.bb &age = 18
Now that you are 18, you can finally do what you have been waiting for
all your life - sign up for your very own XYZweb online cash management
```

```
account! We are waiving the first year's fee if you apply within the next
2 weeks! Apply now at:

http://...

.eb
.* Special offer for younger children, but not around Christmas though!
.bb (&age <= 10) and (DEC not in &*DATE)
Congratulations! You have won an XYZweb teddy bear! Order it now by going
to:

http://...

.eb
.* Two randomly selected people every day get a free T-shirt
.* Note: &*index is randomized with every run. If we ran the job
.* twice, the prize would go to different people
.bb &*index <= 2
Congratulations! You have won a free XYZweb T-shirt!...

.eb
.* Another 10 randomly selected people get a free baseball cap
.bb (&*index > 2) and (&*index <= 12)
.* Make that a free pair of sunglasses in Texas!
.bb (&country = USA) and (&state = TX)
Congratulations! You have won a free pair of XYZweb sunglasses!...

.else
Congratulations! You have won a free XYZweb basecall cap!...

.eb
.eb

.* Plug our travel partner if user checked TRAVEL category on web
.* signup form
.bb TRAVEL
Are you by any chance making travel plans? If so, our partner, ZYX
Travels, have a special offer for you! Simply follow this URL for more
information:

http://...

.eb

.* Special for AOL users
.bb &*to =* "*@aol.com"
Did you know that you can access XYZweb's store directly from AOL?
Simply do...

.eb
.* That's it!
```

A conditional block starts with a .BB (begin block) statement, may include a .ELSE statement, and ends with .EB (end block). The .BB statement contains a conditional expression in the format used in LISTSERV mail templates (described in the list owner's guide). If it evaluates to TRUE, the text is included for the recipient in question, otherwise it is skipped.

Using &*INDEX to select random samples

Every recipient is assigned a unique, random number ranging from 1 to the total number of recipients. Or, in other words, every number from 1 to the total number of recipients is

assigned to a particular, randomly selected recipient. This number is called &*INDEX, and can be used in conditional blocks to select random samples of recipients. Typically, you will use a sequence such as the following:

```
.BB (&*index >= min) and (&*index < max)
```

The conditional block will be sent to all the recipients whose index ranges from *min* (inclusive) to *max* (exclusive). Thus, it will be sent to *max-min* recipients.

Using the PARTS option

While traditional LISTSERV lists provide support for up to 23 topics, there are cases where LISTSERV's topic functions are difficult to use. For instance, imagine a web site where people can subscribe to news about movie stars. Presumably, while there would be hundreds of actors to choose from, most people would only be interested in a handful of them. However, it is likely that a small number of people will be interested in dozens of actors!

LISTSERV's built-in topic support cannot be used, because there will be a lot more than 23 topics. The straightforward approach is to create one mailing list for each actor, and this will work well for people who are only interested in a few actors. However, the avid fans who subscribed to dozens of actors may resent the number of individual messages they receive from the service, and sign off. This would be unfortunate, as they probably represent a non-negligible fraction of the purchases of fan-related material.

The PARTS feature allows you to define any number of topics in the mail-merge message, using conditional blocks of the form:

```
.BB Maxine_Bahns
```

To determine whether a particular recipient is subscribed to news about this actress, LISTSERV checks whether the comma-separated PARTS field contains the topic Maxine_Bahns. For a mail-merge job using a DBMS back-end, the PARTS field is a DBMS column, whose name is specified either in the web interface form or in the "DBMS=" keyword on the DISTRIBUTE command, as follows:

```
DISTRIBUTE MAIL-MERGE DBMS=YES(EMAIL=EMAIL,PARTS=ACTORS)
```

In practice, you would probably maintain the mapping between subscriber and actors using a separate table and foreign keys, but LISTSERV cannot make use of this kind of data layout efficiently, as it would require issuing one SELECT statement per recipient. However, you could use a trigger procedure to update the comma-separated ACTORS column whenever a change is made to the mapping table.

In a mail-merge job with external data source (that is, a job which includes *XDFN declarations), you include the PARTS field in the job stream, with the following syntax:

```
*PARTS part1,part2,...
```

This declaration can come either before or after the *XDFN lines, however it can occur only once.

In some cases, you may want to send some information to people who subscribe to any of a variety of topics. You can do so using the &*PARTS substitution and the IN operator of the LISTSERV expression evaluator. LISTSERV sets &*PARTS from the PARTS field,

but replaces all commas with spaces in order to allow the use of IN/NOT IN. For instance, you could have the following:

```
.BB (Maxine_Bahns in &*PARTS) or (Edward_Burns in &*PARTS)
Maxine Bahns and Edward Burns have broken up! Millions of fans
were in turmoil as the news was announced today at...
.EB
```

You should use the simpler syntax whenever possible, because it allows LISTSERV to bypass the expression evaluator. While very fast, the expression evaluator does add some overhead considering that typical PARTS jobs will have scores of different topics, which must be evaluated for each of the hundreds of thousands of recipients. Thus, the likely order of magnitude of the number of evaluations is in the 1-10 million range.

4.7. Installing and configuring unixODBC with LISTSERV and MySQL

The following are instructions for a simple installation of unixODBC on a unix machine running LISTSERV where you will be using a local MySQL database as a datastore.

4.7.1. MySQL DBMS

If MySQL is not already installed on your unix machine, you will need to install it. If you are planning to access MySQL locally for LISTSERV's exclusive use, we recommend that you secure the installation for local access only, with a minimum of root access.

Current versions of MySQL and the accompanying documentation are available from http://www.mysql.com/, or you may have a version of it included in your unix distribution. If you have 'rpm' available, you can issue `rpm -q mysql' to check the install status. Otherwise, use your normal package installer to query its database of installed software.

4.7.1. MyODBC Driver

An ODBC driver is required to connect unixODBC to a database. The appropriate driver for MySQL is called MyODBC. As with MySQL, there is a chance that MyODBC may already be installed on your machine (particularly in Linux distributions), so check for it before installing.

MyODBC can be installed either as a package or from source, and is available from http://dev.mysql.com/downloads/connector/odbc/3.51.html. (We strongly recommend using the GA ("Generally Available") release, which is currently at version 3.51 -- *not* the "alpha" 5.0 release.) Documentation for the installation of MyODBC is found at http://dev.mysql.com/doc/refman/5.0/en/odbc-connector.html .

4.7.3. unixODBC

unixODBC can be installed either as a package or from source. Both are available from http://www.unixodbc.org/ . The decision of which installation method to use is left to the user. As with the other components, there is a chance that unixODBC may already be installed on your machine (particularly in Linux distributions). If it is, you can skip directly to the next section.

If you choose to install unixODBC from source:

1. Download the source from http://sourceforge.net/project/showfiles.php?group_id=1544. The latest version as of this writing is 2.2.11. The following instructions assume that this is the version you are installing.

2. Issue the following unix commands at the shell prompt:

```
gunzip unixODBC-2.2.11.tar.gz
tar xvf unixODBC-2.2.11.tar
```

3. Change into the directory created in step 2 and issue the following commands at the shell prompt:

```
./configure
make
make install
```

If you want to have more control over the installation, you can issue `./configure --help' to see what the options are. The available options are described in the INSTALL file found in this directory.

4. If you have not already installed the MyODBC driver, do so at this time.

4.7.4. Creating a unixODBC System DSN for LISTSERV

Next, create a system DSN for LISTSERV to use. Open /etc/odbc.ini in a text editor and add the following lines:

```
[ODBC Data Sources]
listserv    = MySQL ODBC 3.51 Driver DSN

[listserv]
Driver       = /usr/lib/libmyodbc3.so
Description  = MySQL ODBC 3.51 Driver DSN
Server       = localhost
Database     = listserv
Trace        = off
```

If you can't find odbc.ini under /etc, try issuing the command `odbcinst -j' . In a default installation, the output appears like this:

```
[home]root:~# odbcinst -j
unixODBC 2.2.9
DRIVERS............: /etc/odbcinst.ini
SYSTEM DATA SOURCES: /etc/odbc.ini
USER DATA SOURCES..: /root/.odbc.ini
```

The file you need is the one referenced by SYSTEM DATA SOURCES.

4.7.5. Creating a MySQL User and Database/Schema for LISTSERV

If you have not already created a 'listserv' database (sometimes called a "schema" in the MySQL documentation) or a 'listserv' user in your MySQL installation, perform the following commands at the shell prompt (where "privileged_user" is whatever user you have set up as the overall MySQL administrator, and "password" is whatever password you want to assign to the 'listserv' user):

```
[home]root:~# mysql -u privileged_user -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25 to server version: 3.23.58

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create database listserv;
Query OK, 1 row affected (0.00 sec)

mysql> use listserv
Database changed

mysql> grant all privileges on listserv to 'listserv' identified
by 'password';
Query OK, 0 rows affected (0.00 sec)

mysql> quit
Bye
[home]root:~#
```

4.7.6. Configure LISTSERV support for unixODBC

As the final step, make sure that you have linked LISTSERV's executable (lsv) with unixODBC support (see above for details). Then you can proceed to define connection information in LISTSERV's go.user file, for instance:

```
UODBC_DSN="listserv"
UODBC_UID="listserv"
UODBC_AUTH="my_password"
export UODBC_DSN UODBC_UID UODBC_AUTH
```

4.7.7. Connecting to External MySQL Databases

If the MySQL database you wish to connect to is running on a machine other than the one on which LISTSERV is installed, you will still need to install unixODBC and MyODBC on the LISTSERV machine in order to connect to the database, and then simply configure the system DSN to point to the machine in question. For instance,

```
[listserv]
Driver       = /usr/lib/libmyodbc3.so
Description  = MySQL ODBC 3.51 Driver DSN
Server       = mysqlbox.mydomain.com
Database     = listserv
Trace        = off
```

Note that the user defined in the UODBC_UID variable in go.user must:

- Have permission to log into the external database from the LISTSERV host machine;
- Have appropriate permissions on the external database.

4.7.8. References

If you are interested in more information on unixODBC, a few of the sites we used as references for these instructions were

http://www.unixODBC.org

http://www.easysoft.com/developer/interfaces/odbc/linux.html#configuring_unixodbc

http://caucuscare.com/tech_mysql.shtml

5. List Exits

Background for non-technical users: an "exit" is a program supplied by the customer to modify the behavior of a product (such as LISTSERV) in ways that the supplier of the product could not anticipate, or could not afford to support via standard commands or options. The product checks for the presence of the "exit" program and calls it on a number of occasions, called "exit points". In some cases, the "exit" program supplies an answer ("return code") to the main program, which adjusts its behavior accordingly. For instance, LISTSERV may ask an exit program "Is it ok to add JOE@XYZ.EDU to the ABC-L list?", and the program will answer yes or no, and possibly send a message to the user explaining why his subscription was accepted or rejected. In other cases, the "exit point" call is purely informative: the exit program gets a chance to do something, such as sending an informational message to a user, but does not return any answer. Because the exit is a computer program, it must be prepared by a technical person and installed by the LISTSERV maintainer.

5.1. What is a "list exit"?

Starting with version 1.8a, list "exits" have been available to control the major events associated with list maintenance. While the implementation of list exits is necessarily system dependent, the list exits themselves (i.e. the tasks that they may carry out, as opposed to how such tasks would be carried out on a particular operating system) are system independent.

To prepare a list exit, you must go through the following steps:

1. Create an appropriate exit program, as explained below. Let's assume that the program's name is XYZ. (Note that exit programs should be named with only alphanumeric characters, ie, A-Z, 0-9.)

2. Modify the LIST_EXITS configuration option in the LISTSERV site configuration file (create one if none was present in your configuration). This variable lists the names of all the "known" exits. For security reasons, LISTSERV will not call an exit which is not listed there. To enable XYZ and ABC as valid list exits, you would set LIST_EXITS to "XYZ ABC" (with or without the quotes, depending on your operating system). You must reboot LISTSERV after making this change.

   Note carefully that you do not code the extension of the exit program (if any) into the LIST_EXITS configuration variable. For instance if you are running under Windows NT and your exit is called XYZ.CMD, you set LIST_EXITS to "XYZ". Under OpenVMS, if your exit is called ABC.COM, you set LIST_EXITS to "ABC".

3. Modify the header of all the lists which should call the XYZ exit, and add "Exit= XYZ". This tells LISTSERV to call the XYZ exit at various exit points.

IMPORTANT SECURITY NOTE: Once an exit has been listed in the LIST_EXITS option, ANY list owner may activate it for his own lists. In other words, step 2 merely tells LISTSERV that the program is a "bona fide" exit. There is no mechanism to restrict the use of an exit to a particular list, because it is very easy to implement this in the exit itself, by checking that the name of the list is what you expect or allow.

LISTSERV exits receive one or more parameters as input, and may provide a numeric and (in a few cases) supplemental string result as output. Each operating system has its own set of numeric return codes for various kinds of failures, but LISTSERV always uses the same internal return code system for its exits - anything else would quickly become

unmanageable. The value 0 always means "success" or "normal/default action". Positive values indicate various non-default actions, depending on the particular exit point. Negative values indicate system errors. Exit programs are responsible for doing their own error reporting, since LISTSERV has no way to know which errors they may or may not run into.

The location, name, programming language and calling conventions of the exit program vary from one operating system to another. Let's examine the basics first:

- On VM, the program must be called XYZ EXEC A1 (on LISTSERV's A-disk) and must be written in REXX.

- On OpenVMS™, the program must be called XYZ.COM on LISTSERV's [.MAIN] subdirectory (LISTSERV_LISTS_DIR) and must be written in DCL.

- On unix®, the program must be called XYZ and must be located in the 'home/' subdirectory (i.e., $A). To distinguish them from the standard L-Soft-provided scripts and executables, exit programs must always be named in upper case. Thus, the program must be called XYZ and not xyz. It can be written in any supported language and LISTSERV must have execute permission.

- On Windows NT™, the program must be called XYZ.CMD and must be located in the MAIN subdirectory. It must be written in the NT batch language.

- On Windows 95™, the program must be called XYZ.BAT and must be located in the MAIN subdirectory. It must be written in the DOS batch language.

Naturally, you are free to call a program in another language from your exit program. The programming language restriction only applies to the exit program itself.

IMPORTANT: Even though the exit program is usually called from LISTSERV's root directory, it should not make any assumption about its current directory. For optimal portability, the program should always use absolute pathnames to access the various files it might need. For instance, list files should be accessed (if at all) as $A/ or A: or %A%\, and so forth.

It is particularly important to note that an intermediary script file (for instance a CMD file under Windows) must normally make use of absolute paths, including an absolute path to the interpreter that will run the program called by your exit. If your CMD file calls a perl file, your CMD file will need to look something like this (depending on the actual locations of the files in question, of course):

```
c:\bin\perl -w e:\listserv\main\MYEXIT.pl
```

Similarly, all files referenced in MYEXIT.pl (including, but not limited to, exit.input and exit.output) *must* be referenced with absolute paths.

Not using absolute paths in your programming is guaranteed to produce unexpected results, if indeed LISTSERV is even able to find your exit when the exit point is called.

The exit may receive one or more string parameters as input. Most operating systems provide a mechanism to pass one parameter to a script or program, with some restrictions. However, LISTSERV may need to pass several parameters, and the restrictions may not be acceptable. Thus, a system independent parameter passing convention had to be designed. This convention is used by all systems except VM, where

multiple parameters of arbitrary length and contents may be passed to a REXX program. On VM, the system independent convention is never used, because it is unnecessary and less efficient than the native method. VM exits use the standard PARSE ARG directive to retrieve their parameters.

The system independent convention uses a disk file called 'exit.input' in the same directory as the exit program. This is a standard text file, where each record is one input string parameter. This file is always created if there are 2 or more string parameters for the exit, or if the EXIT_INPUT configuration parameter is set to the value 1. In addition, it is always created on Windows™ operating systems. Under OpenVMS™ and unix®, the file is not created when there is only one parameter and EXIT_INPUT defaults to 0. Since most exits only have a single parameter, this improves performance in most cases. Note that LISTSERV will take care of deleting the 'exit.input' when appropriate.

Regardless of whether or not the 'exit.input' file is created, the first parameter is always passed to the exit using system-specific methods under OpenVMS™ and unix®. Under Windows™ systems, the first parameter is only passed through the 'exit.input' file. Again, this may simplify programming for simple exits.

IMPORTANT SECURITY NOTE: LISTSERV will always quote/escape the first parameter to prevent the recognition of special characters by the underlying operating system. However, your program should be very careful in its use of the parameters in any subsequent system call. For instance, if the parameter to your unix® exit is the string "a|b", LISTSERV will quote the vertical bar so that it does not result in the execution of the program 'b', and so that the value "a|b" is present in your argument vector. However, you must still be careful in the use of the arguments within your program, especially if you plan to launch a compiled program from a shell and pass it the same arguments. In that case L-Soft recommends the use of EXIT_INPUT = 1, which allows the second program to read its parameters safely from the 'exit.input' file.

For list exits, there is at least one input parameter, of the form (blank separated):

```
epname listname more
```

where 'epname' is the name of the entry point being called (SUB_FILTER, SUB_FAIL, etc), 'listname' is the name of the list, and 'more' depends on the particular exit point. Under VM, 'more' may contain '15'x characters delimiting additional parameters. Again, while 'epname' and 'listname' are unlikely to contain "tricky" characters, the same cannot be assumed about the remainder of the parameter string.

In most cases, your program will only handle a limited set of exit points. When it does not recognize the 'epname', it should exit with the numeric result 0, which tells LISTSERV to take the default action. To exit with the result 0, you can take a normal operating system dependent exit. In particular, success status codes are translated to 0 under OpenVMS™. To return any other numeric code, or to return a string code, you must use the system independent parameter return convention return below, except on VM where the operating system provides a suitable native convention for the return of one parameter of arbitrary length and contents. So, REXX programs return their results with a standard RETURN or EXIT statement. The first blank-delimited word is interpreted as the numeric result, and the rest, if any, is the string result. In other words, the return string is broken up into number and string in exactly the same manner as with a PARSE VAR RESULT NUMBER STRING instruction. On VM, the system independent return convention is not used, because it is unnecessary and less efficient than the native method.

The system independent convention is based on a file called 'exit.output', in the same

directory as the exit program. LISTSERV erases this file before calling your program, and reads it when it regains control. This file is a standard text file, which contains a number of directives. To set the numeric return code, you use the EXIT directive:

```
EXIT 2
```

This would set the numeric return code to 2. To set the string result, use:

```
EXIT-STRING This is the exit string
```

Note that you must ALWAYS set the numeric code if you want the string result to have any effect. The default numeric code is 0, which means "default behavior", and the default behavior never uses the string you supply. By definition, the default behavior is whatever LISTSERV would do if the exit were not present.

In addition to EXIT and EXIT-STRING, a number of other directives are available. For instance, you can tell LISTSERV to send a message to the originating user, to explain why the exit rejected his subscription request. These directives are processed sequentially until the end of the file. Note that the EXIT directives merely set the final exit codes. They do not interrupt the processing of the 'exit.output' file, which is always read to the end of the file. This means that, if you change your mind about the exit code, you can write a new EXIT instructions and LISTSERV will use the last value that you supplied.

Each directive has 0 or more mandatory parameters, and may support a number of optional parameters, which are always listed after the mandatory ones. Some parameters may be simple blank-delimited keywords or options, while others may contain arbitrary data. The exit should not need to provide placeholders for optional parameters, and above all it should be possible to add new optional parameters without requiring all exits to be rewritten. To solve this problem, each directive was given two forms: a simple form, where the entire directive fits in a single line, and an explicit form, where you indicate the number of parameters that you intend to provide, and each parameter follows on a line by its own. In the simple form, the mandatory parameters are filled from the data supplied on the single directive line, and all the optional parameters are set to their default value. Each blank delimited word supplies one parameter, until the first N-1 parameters have been set. The remainder is used for the last parameter. Here is an example of the simple form:

```
TELL jack@XYZ.COM The FOO-L list is only open to FOO Inc. employees.
```

Parameter 1 (mandatory): "jack@XYZ.COM"
Parameter 2 (mandatory): "The FOO-L list is only open to FOO Inc. employees."
Parameter 3 (optional): <default>

If, on the other hand, you want to send the message to more than one person, you need to use the explicit form. In the explicit form, you specify the parameters on separate lines and modify the directive to add the number of lines which you are passing to the processor. By way of example, here is the explicit form of the TELL directive shown above, where the number 2 is added to the directive, telling LISTSERV that there will be two lines of parameters following:

```
TELL2
jack@XYZ.COM cc: honcho@FOO.COM
The FOO-L list is only open to FOO Inc. employees.
```

If you wanted the message echoed to the LISTSERV console log, you would have to add

another line containing the ECHO parameter, and you would have to specify TELL3:

```
TELL3
jack@XYZ.COM cc: honcho@FOO.COM
The FOO-L list is only open to FOO Inc. employees.
ECHO
```

It is always safer to use the explicit form if you are not sure how many words you will have in the various parameters. The simple form is provided mostly for directives such as EXIT or EXIT-STRING which only take one parameter, and for hardcoded parameters.

Currently, the supported directives are as follows. The "VM API" indicates the corresponding REXX API for VM users (it is not possible to provide an API for non-VM systems because the exits run in a different virtual address space and may not call back into LISTSERV entry points).

Name: **EXIT, EXIT-CODE, RETURN**
P1: Numeric return code
Action: Sets numeric return code; does NOT abort exit.output processing!
VM API: EXIT/RETURN

Name: **EXIT-STRING**
P1: String result code
Action: Sets exit string result
VM API: EXIT/RETURN

Name: **TELL, LTELL**
P1: List of RFC822 addresses
P2: Message text
P3 (opt): Blank separated list of options (default = off)
- ECHO: echoes message to log
- RAGGED: flows but does not justify message

Action: Sends long (paragraph) message to specified users
VM API: LSVLTELL

Name: **TELLNF**
P1: List of RFC822 addresses
P2: Message text
P3 (opt): Blank separated list of options (default = off)
- ECHO: echoes message to log

Action: Sends unformatted message to specified users
VM API: LSVTELL

5.2. List Exit Points

The following list exit points are available as of Version 1.8d:

5.2.1. ADD/SUBSCRIBE exit points

Name: **SUB_FILTER**
Parameters: One; originator's e-mail address
Return code: 0=Accept, 1=Reject

Description: This exit point gives you a chance to reject a subscription request by returning a nonzero value. This adds to rather than replaces the "Filter=" keyword.

Name: **SUB_FAIL**
Parameters: One; originator's e-mail address
Return code: Ignored/reserved - always return 0

Description: This exit point is called whenever a subscription is rejected (in particular, it is called if you return 1 from SUB_FILTER). You may send additional messages to the user, log the event, and so on.

Name: **SUB_REQ**
Parameters: One; originator's e-mail address
Return code: Ignored/reserved - always return 0

Description: The user's request for subscription is being forwarded to the list owners. You may send additional messages to the user, log the event, and so on. To implement custom subscription/rejection, use the SUB_FILTER exit with "Subscription= Open". When SUB_REQ is called, it is too late to let the user through.

Name: **SUB_NEW, ADD_NEW**
Parameters: One; originator's e-mail address
Return code: Ignored/reserved - always return 0

Description: This exit point notifies you that the user has been successfully subscribed to the list (SUB_NEW is for the SUBSCRIBE command, ADD_NEW corresponds to the ADD command). You can send additional messages, log the event, and so on.

5.2.2. DELETE/SIGNOFF/CHANGE entry points

Name: **CHG_REQ**
Parameters: Three; originator's e-mail address '15'x target subscriber's e-mail address '15'x new e-mail address
Return code: 0=Accept, 1=Reject

Description: This exit point is called for the CHANGE command, and allows you to reject the operation. If you return the value 1, LISTSERV rejects the operation.

Name: **DEL_FILTER**
Parameters: One or two; target e-mail address '15'x originator's address
Return code: 0=Accept, 1=Reject, 2=Interrupt processing of command

Description: This exit point is called for all SIGNOFF commands, and for DELETE commands issued by a registered Node Administrator. It is NOT called for DELETE commands originating from the list owner. If you return the value 1, LISTSERV does not delete the target e-mail address but continues to look for more addresses matching the pattern provided, and the exit will be called again as appropriate. If you return the value 2, LISTSERV simply interrupts the processing of the SIGNOFF command and terminates the command with no further message (i.e., you have to

send your own explanation back to the command originator). If the request is rejected, you should check for the presence of the second parameter and send an explanatory message to that address, or to the target address if only one parameter was specified.

Name: **DEL_SIGNOFF**
Parameters: One; originator's e-mail address
Return code: 0=Continue, 1=Do not notify owner

Description: This exit point is called for the SIGNOFF command only, when a user has been successfully removed from the list. The farewell message, if any, has already been sent. You may issue additional messages, log the event, and so on. A return value of 1 directs LISTSERV not to mail the "SIGNOFF1" form to the list owners.

Name: **DEL_DELETE**
Parameters: Two; target e-mail address '15'x originator's address
Return code: 0=Continue, 1=Do not notify owner

Description: This exit point is called for the DELETE command only, after a user has been removed from the list. You may issue additional messages, log the event, and so on. A return value of 1 directs LISTSERV not to mail the "DELETE1" form to the list owners.

5.2.3. Other exit points

Name: **SET_REQ**
Parameters: Three; originator's address '15'x list of options to be altered '15'x target e-mail address
Return code: 0=Accept, 1=Reject, 2=Alter

Description: This exit point is called for all SET commands that do not originate from the list owner. The first parameter (originator's address) is the address you should use to send replies or informational messages to the command originator. The second parameter (list of options to be altered) is a blank-separated list of command options, in their canonical spelling. If topics have been specified, they are listed last, after the word 'TOPICS:', with the spelling provided by the user. Bear in mind that topic change requests are not necessarily a list of the new topics to be enabled, and may contain complex '+' or '-' directives. Finally, the third parameter (target e-mail address) is the address as it appears in the list, and is provided for the sake of completeness; in most cases you will not need to examine it. If you return the value 1, the command is rejected and no option is modified. A return value of 2 indicates that the list of options and/or topics should be altered before the changes are performed. The exit string must contain a replacement for the second input parameter, in the same format. LISTSERV will assume that any options or topics specified in this fashion are syntactically correct; while incorrect values will not cause any problem and will be properly rejected as invalid options, the error message(s) returned to the user may not be as helpful as they ordinarily would.

Name: **POST_FILTER**
Parameters: One; e-mail address of the poster.
Return code: 0=Accept, 1=Reject

Description: This exit point is called for messages posted to a list. It can be used to compare the e-mail address of the poster to a list of users who should (or should not) be allowed to post. It can further be used to scan the body of the e-mail message, ie, to accept or reject the message based on its content. The exit places the message to be checked in the file **$D/listserv.cmsut1** (where **$D** is the value assigned with **.SD D** in **system.cfg** or **CORPORAT SYSVARS**, **D=** in **go.sys**, or **D** in **system_config.dat**, depending on your OS platform; in any case, by default this is LISTSERV's TMP directory or minidisk).

A return value of 0 indicates that LISTSERV should continue processing the posting, while a return value of 1 indicates that LISTSERV should reject the posting.

Note carefully that your exit program MAY NOT edit or alter the **listserv.cmsut1** file as LISTSERV has already loaded it preparatory to processing it. The exit point ONLY allows you to accept or reject the message in toto.

5.2.4. General remarks

IMPORTANT: List exits may not make any change to the LIST file, because the command processor may have changes of its own to make to the file, or may have kept the file open for further processing. ***Under VM only,*** If you really have to change the file, use **CP MSG * CMS EXEC** to schedule a new call to the exit after command processing completes.

The template processor can be called from list exits. The syntax is:

```
Call LSVTMAIL recipients,template_name,listname,keywords
```

keywords = 'KWD1 value of first keyword'||'15'x'KWD2 second keyword'||...

Note the change in calling sequence from LSVIMAIL. LSVIMAIL was removed in version 1.8b and is no longer available.

5.3. Local command definition (non-VM)

It is possible to define "local" LISTSERV commands on non-VM systems. A "local" command is a locally developed extension to the LISTSERV command set, which can be installed without making any modification to LISTSERV itself. To install a local command, you must perform the following steps:

1. Create an exit program to implement the command, as described below. Let's assume the program is called ABC. Command and list exits share the same basic attributes and programming interface, and in particular they are located in the same directory and follow the same naming and calling conventions.

2. Choose a name for your local command. Names starting with a letter are reserved for L-Soft use; other names are reserved for customer use. You could call your command /ABC for instance.

3. Modify (or create) the file 'localcmd.file' in the main LISTSERV subdirectory (the same directory where the lists, exits and other LISTSERV files are located). You

must register the command in this file to define its existence to LISTSERV and indicate which exit should be called to execute the command. The format is:

```
/ABC 3 ABC DEF
```

/ABC is the full name of the command, 3 is the minimum abbreviation (allowing /AB or /ABC), ABC is the name of the exit program to execute, and DEF is an optional parameter to be passed to the exit (this allows multiple similar commands to be served by the same exit). NOTE CAREFULLY that the entries MUST be in UPPER CASE. If they are entered in lower case, LISTSERV will not recognize them.

4. Optionally, modify (or create) the file 'localcmd.helpfile' in the same directory to provide a brief (1-2 lines) description of your new command. This is a free form file whose contents are appended to the standard HELP message. If the command is important, you may want to mention it there.

You must reboot LISTSERV for the changes to take effect. This is a change from earlier versions.

The ABC program is called as an exit with two parameters. The first one takes the following form:

```
origin command arguments
```

where 'origin' is the address of the command originator, 'command' is the name of the command ('/ABC' in the present example), and 'arguments' are the command arguments, if any, provided by the user. The second parameter is the optional DEF parameter from 'localcmd.file'.

Typically, your program will parse the arguments, decide on a course of action, and issue a number of messages to the user. The exit code is immaterial; there is no particular course of action to select for command processing.

5.4. SPAM_EXIT (LISTSERV 14.3 and following)

Version 14.3 includes a LISTSERV “exit point” that allows you to use a third-party spam filter to scan messages processed by LISTSERV. Although this hook can in principle be used with any spam scanning product, all the examples and step-by-step instructions in this section will relate to *SpamAssassin*, a popular freeware spam filter that can be downloaded from http://spamassassin.apache.org . Please note that L-Soft did not author *SpamAssassin* and is unable to correct problems with the *SpamAssassin* product itself. L-Soft does not make any legal representations or warranties about *SpamAssassin.* Although L-Soft’s support department will be glad to answer questions about the integration of *SpamAssassin* and LISTSERV, we cannot answer questions about *SpamAssassin* itself.

5.4.1. Formal documentation

SPAM_EXIT is a standard LISTSERV exit, with all that this entails. The same OS-specific naming requirements used for regular LISTSERV exit points are enforced for the SPAM_EXIT exit point. However, SPAM_EXIT is defined separately in the site configuration file as it is a server-level exit rather than a list-level exit.

LISTSERV scans messages in the following sequence:

```
Virus scan -> SPAM_MAXSIZE test -> whitelist/blacklist ->
SPAM_EXIT -> future L-Soft supplied tests
```

The rationale for doing things in this order is that viruses are far more dangerous than spam, so LISTSERV wants to identify them as quickly as possible, and in particular before any whitelist rule has had the opportunity to accept them. Besides, virus scans are much faster than spam scans.

The exit is formally defined as follows:

Name: **SPAM_EXIT**
Parameters: One; `SCAN [listname] | REPORT`
Return code: 0=Accept, 1=Local whitelist, 2=Reject

Its primary input is the file `spam.tmp` in the D directory (typically, unix, `~listserv/tmp` ; Windows, `\LISTSERV\TMP` ; OpenVMS, `LISTSERV_ROOT:[TMP]`). This contains a copy of the whole message, header and body.

When using the SCAN parameter, the exit returns:

- 0: continue normally, per the LISTSERV exit standard. Currently, this means the message is always accepted in practice, but future L-Soft supplied tests would run.
- 1: local whitelist. Accept the message; do NOT run any further tests.
- 2: reject the message. The exit string must then contain an error message to be reported to the sender. LISTSERV will use a standard message if no exit string is supplied, but this standard message is vague since LISTSERV does not know what the exit does.

When using this exit, it is very important to test things carefully, since a mistake could mean that every message is rejected. If for instance the script is not found by the operating system due to a misspelling, and the operating system happens to return 2 in that case, then the message will be automatically rejected even though the message was never scanned. (Because Windows returns 1 for a misspelled exit name, we chose 2=reject instead of the usual 1=reject.)

Once configured, spam scans take place whenever a virus scan takes place and no virus was detected, with two exceptions:

- When downloading binary attachments via WA (virus scan only – spam filters are unlikely to do something meaningful with a .EXE file)
- For the `DISTRIBUTE AV=YES` programming interface.

The minimum implementation for the `REPORT` call is to do the same as `SCAN` does. In addition, it is desirable to create an output file called `spam.report` in the same directory where the input file `spam.tmp` is located. There is no special format for this output file, but it is a good idea to start with a line saying whether or not the message was identified as spam, and give the score if the spam filter uses a score system. LISTSERV does not process the report, it just ends up being shown to a human. If no `spam.report` file is created by the exit, LISTSERV will use the exit string as a one-line report. If there is no exit string, LISTSERV will generate a hard-coded message.

5.4.2. Practical application

Overview

To enable spam filtering in LISTSERV, you must install the third-party spam filter, provide a script that will scan messages using the third-party filter (if using *SpamAssassin*, you can use one of the L-Soft supplied scripts), and activate this script by making changes to the LISTSERV configuration. LISTSERV will then scan every message it processes, with a few exceptions, and reject messages identified as spam.

Optionally, you can also activate LISTSERV 14.3's built-in blacklist/whitelist functionality (see the release notes for version 14.3 for more information). This feature provides an additional level of spam filtering and can also improve performance significantly, because spam filters like *SpamAssassin* can take up to 5-20 seconds to scan a message.

Step-by-step instructions

This section contains step-by-step instructions for configuring LISTSERV to use SpamAssassin using one of the L-Soft supplied scripts. Throughout this section, we will make the following assumptions:

- *SpamAssassin* has already been installed and configured on a server that we will call `spamd.example.com`. This can, but does not have to be, the machine on which LISTSERV is installed. In particular, you can run LISTSERV on Windows and *SpamAssassin* on unix, and vice-versa.
- `spamd` has been started and is configured to accept incoming requests from the machine on which LISTSERV is installed.
- You have a test message file at your disposal to verify the operation of `spamc/spamd`. We will call this file `testmsg.txt`.

Step 1 of 4: Install and test *SpamAssassin* client.

Unix: Compile spamc on the LISTSERV host, then copy it to a directory in LISTSERV's path. To test it, use a command similar to the following:

```
$ spamc -c -d spamd.example.com < testmsg.txt
3.8/5.0
```

The flags you need to use may vary depending on your version of SpamAssassin and configuration. The response must be two numbers as shown above, but the numbers can be different than in the example (they are the *SpamAssassin* score of the test message). Any other response indicates an error. Refer to the `spamc` and `spamd` man pages for more information.

Windows: Download and install the spamc.exe executable from L-Soft, and place it in a directory in LISTSERV's path – for instance, the `LISTSERV\MAIN` directory.

To test the client, issue the following command:

```
C:\> spamc -c -d spamd.example.com < testmsg.txt
3.8/5.0
```

The response must be two numbers as shown above, but the numbers can be different than in the example (they are the *SpamAssassin* score of the test message). Any other response indicates an error. In that case, make sure that `spamd` is configured to allow connections from the LISTSERV host.

Step 2 of 4: Install perl or REXX (if not already available).

Unix: Install perl on the LISTSERV host, if not already installed.

Windows: Install a REXX interpreter, such as Regina REXX (http://regina-rexx.sourceforge.net/, Windows kit download available at http://prdownloads.sourceforge.net/regina-rexx/regina33.exe?download). When prompted to register .REXX as a path extension, you should do so. Alternatively, you can simply download REXX.EXE from L-Soft and place it in the same directory where you saved `spamc.exe`.

Step 3 of 4: Install and configure `SAEXIT` script.

- Download the L-Soft supplied sample script at one of the following URLs:

  Unix: ftp://ftp.lsoft.com/LISTSERV/UNIX/CONTRIB/SAEXIT
  Windows: ftp://ftp.lsoft.com/LISTSERV/Windows/CONTRIB/saexit.rexx

- Edit the script to configure, at a minimum, the address of your *SpamAssassin* server. You may also want to change the other parameters.
- Make any other changes that you deem appropriate.
- Save the script in LISTSERV's main directory (on unix, set execute permissions):

  Unix: `~listserv/home`
  Windows: `C:\LISTSERV\MAIN`

- You can call the script anything you want, but in this example we will assume that you have left the name unchanged (`SAEXIT`).
- **Windows:** if you have not registered `.REXX` as a path extension when installing it in step 2 or if you downloaded it from the L-Soft ftp site instead of installing the full kit, you will need to create a script called `saexit.cmd` in the C:\LISTSERV\MAIN directory containing the following three lines:

```
@echo off
rexx saexit.rexx %*
exit %ERRORLEVEL%
```

Step 4 of 4: Enable the `saexit` script.

To enable the script, add the following lines to LISTSERV's configuration:

Unix:
```
SPAM_EXIT="saexit"
export SPAM_EXIT
```

Windows:
```
SPAM_EXIT=SAEXIT
```

Restart LISTSERV to make the change take effect, then mail a spam message to a test list to confirm that everything is working as it should.

Restrictions

- The spam exit is a feature of LISTSERV Classic and HPO, and is not available with LISTSERV Lite. Maintenance is required.
- If you are using L-Soft's Anti-Virus Station (AVS) to provide virus protection to a server for which F-Secure Anti-Virus is not available, the spam exit must be

installed on the AVS server, not on the primary LISTSERV server. This is because message scanning is bypassed on a server that uses the AVS for this purpose.

- The spam exit is implemented within LISTSERV's message scanner, which is informally known as the "virus scanner," because this was its original purpose. If the message scanner is disabled, for instance by setting the ANTI_VIRUS configuration parameter to 0, or by failing to install the maintenance LAK, both virus scanning and spam scanning are disabled. If ANTI_VIRUS is unset, LISTSERV will enable the message scanner if either virus scanning or spam scanning is configured and available.

VM and VMS

The spam exit is also available on VM and VMS, but there is no version of `spamc` for these systems. Since VM and VMS hosts normally use the AVS for message scanning, spam protection can be provided by simply installing the spam exit on the AVS server.

Advanced configuration

At the list level, "Misc-Options= NO_SPAM_CHECK" can be used to disable spam scans for a particular list and its associated `xxx-request` address.

New statistical counters have been added for spam scans. They work just the same way as the virus counters.

A new configuration parameter, `SPAM_MAXSIZE`, can be used to automatically accept messages larger than a certain size. The rationale is that spam filters can take minutes to process very large messages, whereas spam messages are almost always very small. The following example sets the threshold to 512k:

VM:	`SPAM_MAXSIZE = 512`
VMS:	`SPAM_MAXSIZE "512"`
unix:	`SPAM_MAXSIZE=512` `export SPAM_MAXSIZE`
Win:	`SPAM_MAXSIZE=512`

The default value is 256k. If set to 0, all messages will be scanned, which again could take considerable time. Messages that are not scanned do not count as a spam scan in the LISTSERV statistics.

5.5. A practical example: HAPPY99

In late 1998/early 1999 a worm called HAPPY99 surfaced on the Internet. In an attempt to keep the worm off of lists, an L-Soft engineer wrote the following exit in Regina REXX to run under the Windows NT version of LISTSERV 1.8d.

Before proceeding, note this obligatory warning:

USE AT YOUR OWN RISK: THIS SOFTWARE IS PROVIDED ON AN 'AS IS' BASIS. L-SOFT DOES NOT MAKE ANY EXPRESS OR IMPLIED WARRANTY OF ANY KIND WHATSOEVER WITH RESPECT TO THE SOFTWARE, INCLUDING, WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Neither L-Soft nor any of its employees, officers or agents will be liable for any direct, indirect or consequential damages, even if L-Soft had been advised of the possibility of such damage. No support is available from L-Soft or from the

author for this program. If you give a copy of this program to someone else for their use, you must provide it with both the copyright notice and this paragraph intact. You are not authorized to make this program available for public access via anonymous FTP or by any other means of mass distribution.

First we wrote HAPPY99.REXX, which looks like this:

```
/* HAPPY99.REXX : Sample Windows NT list exit programming */
/* Added (0.1b) support for VBS.Freelink detection. */
versionno = '0.1b 1999/12/05'
/* By Nathan Brindle <nathan@lsoft.com> */
/* Copyright (c) 1999 L-Soft international, Inc. */
/* All rights reserved */
/*
   USE AT YOUR OWN RISK: THIS SOFTWARE IS PROVIDED ON AN 'AS IS' BASIS. L-SOFT
   DOES NOT MAKE  ANY EXPRESS OR IMPLIED WARRANTY OF  ANY KIND WHATSOEVER WITH
   RESPECT TO  THE SOFTWARE,  INCLUDING, WITHOUT  LIMITATION, ANY  WARRANTY OF
   MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Neither L-Soft nor any
   of  its employees,  officers  or  agents will  be  liable  for any  direct,
   indirect or consequential  damages, even if L-Soft had been  advised of the
   possibility of such damage. No support is available from L-Soft or from the
   author for this program. If you give a copy of this program to someone else
   for their use, you must provide  it with both the copyright notice and this
   paragraph intact. You are not authorized to make this program available for
   public access via anonymous FTP or by any other means of mass distribution.
*/

/*****************************************************************************/
/*                            WARNING WARNING WARNING                        */
/*****************************************************************************/
/* THIS PROGRAM IS NOT INTENDED TO BE USED WITHOUT MODIFICATION.  IT IS ONLY A
   SAMPLE OF HOW LIST EXIT PROGRAMMING MIGHT BE DONE UNDER WINDOWS NT.  WHILE
   THE EXAMPLES HAVE BEEN TESTED AND DO WORK, L-SOFT DOES NOT RECOMMEND SIMPLY
   INSTALLING THIS EXIT AND USING IT WITHOUT MODIFICATION AS THE RESULTS WILL
   PROBABLY NOT BE WHAT YOU EXPECT.  NOTE CAREFULLY THAT THERE IS NO SUPPORT
   AVAILABLE WHATSOEVER FOR THIS SAMPLE EXIT.
 */

/* List exits are documented in the Developer's Guide for LISTSERV, available
   at L-Soft's FTP and WWW sites.
 */

/*****************************************************************************/
/*                    USER CONFIGURATION AREA STARTS HERE                    */
/*                                                                           */
/* infile =  the full path (drive, directory, filename) pointing to          */
/*           the exit.input file.  This must be the path to \LISTSERV\MAIN   */
/*           and the filename must be 'exit.input'.                          */
/* outfile = the full path (drive, directory, filename) pointing to          */
/*           the exit.output file.  This must be the path to \LISTSERV\MAIN  */
/*           and the filename must be 'exit.output'.                         */
/* tmpdir  = the full path (drive and directory) pointing to LISTSERV's TMP  */
/*           directory.  Typically this is something like C:\LISTSERV\TMP ;  */
/*           in any case it is the directory pointed to by the .SD D setting */
/*           in SYSTEM.CFG.                                                  */
/* node =    the value from NODE= in your SITE.CFG file, i.e., the name of   */
/*           your LISTSERV machine in DNS.                                   */
/*****************************************************************************/
infile = 'e:\listserv\main\exit.input'
outfile = 'e:\listserv\main\exit.output'
tmpdir = 'E:\ListPlex\PEACH\TMP'
node = 'PEACH.EASE.LSOFT.COM' /* replace with NODE= value from site.cfg */
/*****************************************************************************/
/*                     USER CONFIGURATION AREA ENDS HERE                     */
/*****************************************************************************/

parms = linein(infile)

parse var parms epname listname more

if epname == 'POST_FILTER' then signal postfilter
```

```
/* otherwise, leave */
call lineout(outfile,'EXIT 0')
exit

/*********************************************************************/
/* END OF PROGRAM if no epname is recognized                         */
/*********************************************************************/

/*********************************************************************/
/* POST_FILTER                                                       */
/*********************************************************************/
postfilter:

  /* Note that you could conceivably reject messages based on the value of 'c',*/
  /* which is the total number of bytes in the message (including all headers, */
  /* attachments, etc.).                                                       */

  filtermsg = 'FALSE'
  c = chars(tmpdir'\listserv.cmsut1')
  msgin = charin(tmpdir'\listserv.cmsut1',1,c)
  call charout(tmpdir'\listserv.cmsut1')

  /* Grab Date: and Subject: headers */
  msgdate = ''
  msgsubject = ''
  p = pos("Date:",msgin)
  if p > 0 then do
                 e = pos('0d'x,msgin,p) - 1
                 msgdate = substr(msgin,p+5,(e-p)-4)
                 msgdate = strip(msgdate)
               end
  if msgdate == '' then msgdate = 'undated message'
                   else msgdate = 'message dated' msgdate
  p = pos("Subject:",msgin)
  if p > 0 then do
                 e = pos('0d'x,msgin,p) - 1
                 msgsubject = substr(msgin,p+8,(e-p)-7)
                 msgsubject = strip(msgsubject)
                 if left(msgsubject,1) == '0d'x then msgsubject = ''
               end
  if msgsubject == '' then msgsubject = 'with no subject'
                      else msgsubject = 'with subject "'||msgsubject||'"'

  /* Check for uuencoded Happy99.exe virus */
  filter = 'begin 644 Happy99.exe'
  p = pos(translate(filter),translate(msgin))
  if p > 0 then filtermsg = 'TRUE'
  say '>>>' p
  if filtermsg == 'TRUE' then do
    explanation = 'Your' msgdate msgsubject 'sent'
    explanation = explanation 'to the 'listname' mailing list has been'
    explanation = explanation 'rejected because it appears to contain the'
    explanation = explanation 'Happy99.EXE virus.  It is strongly suggested'
    explanation = explanation 'that you run an anti-virus program before'
    explanation = explanation 'posting again.'
  end
  if filtermsg == 'TRUE' then signal done

  /* deal with links.vbs (FreeLink) */

  filter = 'Have fun with these links.'
  p = pos(translate(filter),translate(msgin))
  if p > 0 then filtermsg = 'TRUE'
  say '>>>' p
  if filtermsg == 'TRUE' then do
    explanation = 'FREELINK: Your' msgdate msgsubject 'sent'
    explanation = explanation 'to the 'listname' mailing list has been'
    explanation = explanation 'rejected because it appears to contain the'
    explanation = explanation 'VBS.Freelink virus.  It is strongly suggested'
    explanation = explanation 'that you run an anti-virus program before'
    explanation = explanation 'posting again.'
  end
  if filtermsg == 'TRUE' then signal done

  /* More filtering conditionals could be put in here */
```

```
done:

  if filtermsg = 'FALSE' then call lineout(outfile,'EXIT 0')
  if filtermsg = 'FALSE' then exit

  /* Otherwise we've found reason to reject the message. */
  /* Construct an explicit TELL directive to inform the user.  The TELL */
  /* directive has 3 parameters, which will be passed one per line.     */
  call lineout(outfile,'TELL3')

  /* First parameter, the command originator, comes from the "more" */
  /* variable. */
  call lineout(outfile,word(more,1))

  /* Next we pass second parameter, which we've already built above */
  call lineout(outfile,explanation)

  /* Third (optional) parameter, echo to the LISTSERV log */
  call lineout(outfile,ECHO)

  /* Now, reject the message. */
  call lineout(outfile,'EXIT 1')

  /* Finished. */
  exit

/* end of program */
```

Then we wrote the following HAPPY99.CMD file, and placed it in the `\LISTSERV\MAIN` directory. This CMD file, when executed by LISTSERV, calls the Regina interpreter and in turn tells it to run the HAPPY99.REXX file we wrote above.

```
REM be sure to change the directories to point to the right places!
c:\util\reskit\rexx.exe e:\listserv\main\happy99.rexx
```

Next, SITE.CFG was modified and the line

```
LIST_EXITS=HAPPY99
```

was added. LISTSERV was then stopped and restarted to pick up the change.

Finally, the exit was enabled for certain lists by adding

```
* Exit= HAPPY99
```

to the list headers.

Note that the program makes no attempt to determine if the message actually contains a working copy of the virus. Its only function is to attempt to identify messages that MAY contain a working copy of the virus. Thus it may engender some false positives (but it's doubtful that anyone sending legitimate mail will send the specific strings searched for by the exit program).

Note further that the LISTSERV.CMSUT1 file that you search for evidence of the virus may not be edited "on the fly" to remove parts of the message--it is a read-only file for this purpose. You have only the option of accepting or rejecting the message in its entirety.

Given that the "classic" version of HAPPY99 is not sent as a MIME attachment, it is not possible to block it with the `Attachments=` list header keyword introduced in the 2000a level set release of LISTSERV 1.8d, and the exit is the only way to guard against it.

(LISTSERV 1.8e with its integrated anti-virus scanning and ability to filter uuencoded inline attachments removes the need for this particular exit, but the above remains a useful example of how to program an exit.)

6. The LISTSERV TCPGUI interface

The TCPGUI interface is the part of LISTSERV that listens for TCP/IP connections coming into the LISTSERV port on the LISTSERV host. To use the TCPGUI interface, you set up TCPGUI, then you send commands to it through TCP/IP. You have two options for sending commands to LISTSERV through TCP/IP:

- Use the **lcmdx** program from your operating system's command line, or
- Write a program to send the commands, using the code for **lcmdx** as a guide.

Most simple commands can be sent directly through the TCPGUI interface, but some advanced operations require special handling.

6.1. Setting up the TCPGUI interface

By default, LISTSERV listens to port 2306 on all the IP addresses assigned to the local host.

If it needs to be restricted to a given IP address, or if port 2306 is already in use by another application, LISTSERV must be set up so that it listens to the correct IP address and port.

To specify an IP address that LISTSERV should listen to, you need to add the **TCPGUI_IPADDR** parameter to your site configuration file.

For example, under Windows NT, you would add the following line to the SITE.CFG file:

TCPGUI_IPADDR=nnn.nnn.nn.nn

where **nnn.nnn.nn.nn** is the IP address assigned to the LISTSERV host.

You can instruct LISTSERV to listen to a different port by setting the **TCPGUI_PORT** parameter to an unused port number. For example:

TCPGUI_PORT=nnnn

Recall that you need to stop and restart LISTSERV for site configuration changes to take effect.

6.2. Running lcmdx

The **lcmdx** program (C language code at the end of the chapter) allows you to send a LISTSERV command from a command line (DOS, shell, or DCL prompt). Compile and link it (you may need to make minor changes to get it to compile, depending on your C compiler) to produce an executable.

You can run **lcmdx** from any computer that is connected via TCP/IP to the LISTSERV host (i.e. via the Internet or a TCP/IP-based intranet).

You need to have a password registered for your e-mail address in LISTSERV in order to send commands via TCP/IP. Instructions for registering a password are in section 2.12.8 of the List Owner's manual.

The format for the **lcmdx** command line is:

lcmdx *hostname*[:*port*] *address password command*

where

- ***hostname*** is the DNS name of the LISTSERV host,
- ***port*** is the LISTSERV port number (needed only if it is different from the default of 2306),
- ***address*** is the e-mail address of the user sending the command,
- ***password*** is the upper-case password registered with this LISTSERV host for that e-mail address, and
- ***command*** is the one-line LISTSERV command.

Note that the ***password*** parameter MUST imperatively be typed in UPPER CASE.

For example, if francoise@lsoft.com wants to set her subscription to the MSVC list to digest, and her password on PEACH.EASE.LSOFT.COM (where this list is hosted) is "ABCDE", she can use the following command:

```
lcmdx PEACH.EASE.LSOFT.COM francoise@lsoft.com ABCDE SET MSVC DIGEST
```

LISTSERV responds:

```
Your subscription options have been successfully updated. You are being
mailed a short report with the exact settings now in use for your
subscription. Please take a few moments to check that this is indeed
what you wanted.
```

Any one-line command can be submitted to LISTSERV in this way. To use **lcmdx** directly from your application, just spawn a subprocess or send the **lcmdx** command using whatever means is provided by the operating system under which your application is running.

6.3. Sending LISTSERV commands directly from your application

The **lcmdx** program is convenient, as it doesn't require programming to use, but it does not allow much flexibility in how your application can react to LISTSERV's responses to the commands sent. For greater flexibility, you can integrate the techniques used in **lcmdx** for communicating with LISTSERV directly into your application.

The **lcmdx** source code consists of three C functions:

- **receive** - used by **LSV_send_command** to receive a string of a given length from a socket
- **LSV_send_command** - the function that sends a command from LISTSERV and returns the answer; this function can be used directly by your application with few or no changes.
- **main** - the main function simply parses the command line for **lcmdx** into its component parts, passes them to **LSV_send_command**, and prints LISTSERV's response; your application would replace **main**.

The main change that you may want to make to the **LSV_send_command** function, will be to have it write the LISTSERV response to a string or an array of strings rather than to

a file, as is the case in **lcmdx**. What you actually do with LISTSERV's response will be dictated by the needs of your application.

The only other changes that might be required are changes that relate to how sockets are implemented on your operating system. For example, if you are using Windows sockets (WINSOCK), you would have to add the Windows sockets initialization code at the start of your application, calling and checking the status of the **WSAStartup** routine; and the cleanup code at the end, calling the routine **WSACleanup**. With Windows sockets, you would also have to include the header file **<winsock.h>** instead of **<sys/socket.h>**, **<netdb.h>**, and **<netinet/in.h>**; and replace the call to **close(ss)** with a call to **closesocket(ss)**.

The **LSV_send_command** function uses the following calling sequence:

```
int LSV_send_command(char *hostname, unsigned short port,
                     char *origin, char *pw, char *command, FILE *writeto)
```

where the return value is:

- 0 if the command was sent and the answer received without a problem, from a TCP/IP point of view - this does not indicate that the command itself was successfully executed by LISTSERV: your application needs to look at the answer received from LISTSERV to determine whether the command itself was successful;
- 1000 if a protocol error occurred while communicating with LISTSERV;
- a socket error code if a socket error occurred while communicating with LISTSERV - the socket errors should be described in the documentation for socket functions for your C compiler.

and the parameters are:

- **hostname** - a pointer to a character array containing the name of the LISTSERV host to which to send the command
- **port** - the port number to which to connect (use 2306 unless otherwise specified in the site configuration file of the LISTSERV host)
- **origin** - a pointer to a character array containing the e-mail address of the "originator" of the command
- **pw** - a pointer to a character array containing the LISTSERV password registered for the originator's e-mail address (must be UPPER CASE)
- **command** - a pointer to a character array containing the one-line command to send to LISTSERV
- **writeto** - a pointer to the file in which to write LISTSERV's response; as mentioned above, depending on the needs of your application, you may want to change this parameter and the code within **LSV_send_command** that writes to this file.

Of course, if you use **LSV_send_command** without modification, you will be opening and closing a connection to LISTSERV for each command you send. You can make your application more efficient by opening up a connection to LISTSERV and sending a series of commands before closing the connection again. The **LSV_send_command** function will work "out of the box", but an experienced programmer can use it as a guide for developing customized functions for working with the TCPGUI interface.

6.4. Advanced TCPGUI Programming Issues

The technique described above, using the **LSV_send_command** function to send commands over TCP/IP to LISTSERV, will work for most one-line LISTSERV commands, such as ADD, DELETE, SET, SHOW, etc. However, some commands require a different or modified approach:

- Creating or replacing a list header
- Adding or replacing a password
- Bulk operations
- Commands that respond over e-mail
- Application-friendly commands

There may also be some special error handling involved.

6.4.1. Creating or replacing a list header

Multi-line commands cannot be sent through the TCPGUI interface, therefore the usual approach for sending a list header to LISTSERV (the PUT command followed by the multiple lines of the header) cannot be used. Instead, TCPGUI has a special command for sending a list header: X-STL.

The syntax of the X-STL command is:

```
X-STL listname header-data
```

where header-data is a text string that contains every header line (including the leading asterisk), one after the other, each preceeded by a count of the characters on the line, followed by an underscore character. The last header line should not have any trailing spaces.

Thus, if the command you would use to put your list through e-mail was:

```
PUT listname PW=password
* test1
* Owner=francoise@lsoft.com
* Notebook=No
* Confidential=Yes
```

The command you send through the TCPGUI interface would be:

```
X-STL listname 6_* test1_*28_* Owner= francoise@lsoft.com13_*
Notebook=No18_* Confidential=Yes
```

(Remember that this is a one-line command.)

Obviously, you need to use the "create" password when creating a new header, and an owner's or postmaster's personal password when replacing an existing header.

NOTE: If double quotes are required (for instance, for the Prime= or Sender= keyword settings), you MUST escape them with a backslash character, thus:

```
lcmdx listserv.example.com xxxxxxxxx@example.com XXXXX X-STL
XXXXXXXXX 6_* test1_*31_* Owner= xxxxxxxxx@example.com13_*
Notebook=No18_* Confidential=Yes49_* Sender= \"test
<xxxxxxxxx@listserv.example.com>\"
```

(remember that the entire command must be on one line, not wrapped as is unavoidable in this document). Additionally, when counting characters for the line counts, the backslash-double quote combination counts as a single character. If the line count is wrong, you will get an "Error in header data stream" error.

6.4.2. Adding or replacing a password

You can't send a "**PW ADD**" or "**PW REP**" command through the TCPGUI interface. There is a special command for adding a password:

```
X-PWADD e-mail-address password
```

The **LSV_send_command** function generally requires a password, which you don't necessarily have when you're in the process of adding one. In this case, you can send the command anonymously through TCPGUI by using the anonymous e-mail address "@" as the "originator" in the call to **LSV_send_command** (but not in the **X-PWADD** command, obviously).

When LISTSERV receives the **X-PWADD** command, it sends e-mail to the given e-mail-address requesting confirmation. E-mail confirmation is the only way for LISTSERV to determine that the e-mail address provided is truly the correct address. Therefore, an application should never count on the password being immediately available. A message to the user, letting them know that they can continue after they have successfully confirmed the password registration, may be advisable.

6.4.3. Bulk operations

As noted above, the TCPGUI interface can only handle single-line commands. Therefore bulk operations, such as the bulk add and delete commands cannot be sent through TCPGUI, and can only be sent through the mail. The only ways to send bulk adds and deletes from an application are:

- Turn them into individual add or delete commands, and send each of these through the TCPGUI interface. If there are many subscribers to add or delete, this can extremely slow, and is therefore not recommended.
- Write a mail file containing the bulk add or delete job and use a local mail application to send it to LISTSERV.
- Open up a connection with the SMTP port on the LISTSERV host and use SMTP commands (documented in RFC821) to send a mail file (documented in RFC822) containing the bulk add or delete job. This is essentially the same thing as the previous bullet, except that in the latter case, you used a third-party application to send the file, whereas with this method you must write your own.

6.4.4. Commands that respond over e-mail

LISTSERV will accept almost any one-line command through the TCPGUI interface. The answer you receive through the TCPGUI interface, however, may not always be what you expected. Commands whose responses tend to be long will generally be sent through e-mail. LISTSERV will always send the response to the following commands back through e-mail:

- **INFO**
- **INDEX**
- **LISTS DETAILED**

- **LISTS GLOBAL**
- **LISTS SUMMARY**
- **GETPOST**
- **GET** for anything other than a list header

For these other commands, LISTSERV will send the response back through e-mail unless you use an option to make it come back through the TCPGUI interface:

- **GET *listname*** -- requires the "**(MSG**" option
- **REVIEW** -- requires the "**MSG**" option

6.4.5. Application-friendly commands

Some commands that can be sent through the TCPGUI interface and to which LISTSERV will send the response back through the TCPGUI interface nevertheless have responses that are human-friendly but not application-friendly. One such command is the **QUERY** command, which sends a response that looks like:

```
Subscription options for Francoise Becker , list MYLIST:

MAIL            You are sent individual postings as they are received
MIME            You prefer to receive messages in MIME format
SUBJECTHDR      Full (normal) mail headers with list name in message
subject
REPRO           You receive a copy of your own postings
NOACK           No acknowledgement of successfully processed postings

Subscription date: 12 Mar 1997

The topics you subscribe to are: Mytopic, Other
```

To assist the application developer, the LISTSERV provides alternate commands for the following:

- **QUERY**
- A special option for **QUERY**: **DEFSUB**
- **SCAN**

QUERY

To **QUERY** subscriber options from an application you should use:

```
QUERY ***GUI*** listname [FOR e-mail-address]
```

The response will look like the following:

```
***HDR*** e-mail-address
***NAME*** firstname lastname

***OPT*** option1
***OPT*** option2
...
***OPT*** optionN

***SUBDATE*** date
```

```
***TOPICS*** subscriber-topics
***TOPLIST*** list-topics
***HDR*** e-mail-address (next subscriber)
etc.

N matching entries found.
```

Where:

- The "*****HDR*****" line denotes the beginning of the subscription settings for a particular subscriber (recall that the **QUERY** command could yield information for multiple subscriptions) and identifies the e-mail address of the subscription.
- The "*****NAME*****" line provides the full name stored for that subscriber.
- The "*****OPT*****" lines each show one option set for the subscription. The first option would always be MAIL or NOMAIL. All the other options are only those options that are NOT the default options in LISTSERV (as opposed to the default options set for the particular list -- these do not apply here).
- The "*****SUBDATE*****" line contains the subscription date.
- The "*****TOPICS*****" line lists the topics which are selected for this subscription.
- The "*****TOPLIST*****" line lists all the topics that are available for the list, regardless of whether this particular subscription has them selected, except for "ALL" and "OTHER".
- These lines are repeated for each matching entry found.
- The last line gives a count of matching entries found.

QUERY DEFSUB

There is also a special **QUERY** command for obtaining default subscription options:

```
QUERY ***GUI*** ***DEFSUB*** listname
```

If this command is sent using an address that is subscribed to the given list, it works exactly the same as the "**QUERY ***GUI*****" command described above. If it is sent anonymously (see "Adding or replacing a password" above) or from an address which is not subscribed, then the first line in the response is:

```
***DEF***
```

And the remainder of the response is the same as the response to "**QUERY ***GUI*****" for a subscriber with the list's default subscription settings.

SCAN

To scan a list for a pattern, you should use

```
SCAN ***GUI*** listname pattern
```

The response will look like the following:

```
***MBX*** user1@host.domain
Firstname Lastname <user1@host.domain>
***MBX*** user2@host2.domain
"Full name w/ special characters" <user2@host2.domain>
etc.
***END***
```

SCAN: N matches.

6.4.6. Error handling

When the command you send produces an error the TCPGUI interface sends back the exact error that you would receive through the mail when that error is detected by LISTSERV (for example, if you use the wrong listname, or misspell a command). However, the TCPGUI interface does have some error messages specific to it, for when the error occurs within TCPGUI rather than LISTSERV. These are:

- *****NOPW***** -- the e-mail address does not have a password registered with LISTSERV
- *****BADPW***** -- the password provided in the TCPGUI command does not match the password registered for the given e-mail address.

6.5. LCMDX.C

```
/*****************************************************************************
 *                                                                           *
 * LISTSERV V2 - send command to LISTSERV on remote node via TCPGUI interface *
 *                                                                           *
 *       Copyright L-Soft international 1996-97 - All rights reserved        *
 *                                                                           *
 * Syntax:                                                                   *
 *                                                                           *
 *  lcmdx hostname[:port] address password command                           *
 *                                                                           *
 * Connects to 'hostname' on port 'port' (default=2306) using the LISTSERV   *
 * TCPGUI protocol, then executes the LISTSERV command 'command' from the    *
 * origin 'address'. 'password' is the personal LISTSERV password associated *
 * with the command origin ('address') - see the description of the PW ADD   *
 * command for more information on LISTSERV passwords. The reply from        *
 * LISTSERV is echoed to standard output (the command is executed            *
 * synchronously).                                                           *
 *                                                                           *
 *****************************************************************************/
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#ifdef INTUX
#include <net/errno.h>
#endif

#define DEFAULT_PORT    2306

#ifdef ultrix
        /* Use read() rather than recv() to bypass a bug in Ultrix */
#define recv(a, b, c, d) read(a, b, c)
#endif

static int receive(int ss, char *buf, int len)
{
        char *w, *e;
        int l;

        for (w = buf, e = buf + len; w < e;) {
                l = recv(ss, w, e - w, 0);
                if (l <= 0)
                        return(l);
```

```
                w += l;
        }
        return(len);
}

int LSV_send_command(char *hostname, unsigned short port, char *origin,
                     char *pw, char *command, FILE *writeto)
{
        char buf[256], *reply = 0, *cmd, *w, *r, *e;
        unsigned char *wb;
        int rc, ss, len, orglen, n;
        unsigned int ibuf[2];
        struct sockaddr_in sa_connect;
        struct hostent *H;

        /* Initialize */
        cmd = malloc(strlen(command) + strlen(pw) + 5);
        sprintf(cmd, "%s PW=%s", command, pw);
        orglen = strlen(origin);

        /* Create a socket */
        if ((ss = socket(AF_INET, SOCK_STREAM, 0)) < 0)
                goto Socket_Error;

        /* Prepare sa_connect structure */
        memset(&sa_connect, 0, sizeof(sa_connect));
        sa_connect.sin_family = AF_INET;
        sa_connect.sin_port = htons(port);
        if ((H = gethostbyname(hostname)) && H->h_addr_list[0])
                memcpy(&sa_connect.sin_addr, H->h_addr_list[0], 4);
        else
                goto Socket_Error;

        /* Connect to the TCPGUI port */
        if (connect(ss, (struct sockaddr *)&sa_connect,
                    sizeof(sa_connect)) < 0)
                goto Socket_Error;

        /* Send the protocol level request and the command header */
        wb = (unsigned char *)buf;
        len = strlen(cmd);
        n = len + orglen + 1;   /* Byte length                  */
        *wb++ = '1';            /* Protocol level: 1            */
        *wb++ = 'B';            /* Mode: binary                 */
        *wb++ = '\r';
        *wb++ = '\n';
        *wb++ = n / 256;        /* Request length byte 1        */
        *wb++ = n & 255;        /* Request length byte 2        */
        *wb++ = orglen;         /* Origin length: 1             */
        for (r = origin; *r;)
                *wb++ = (unsigned char)*r++;

        if (send(ss, buf, (char *)wb - buf, 0) < 0)
                goto Socket_Error;

        /* Await confirmation */
        for (w = buf;;) {
                n = recv(ss, w, buf + sizeof(buf) - w, 0);
                if (n <= 0)
                        goto Socket_Error;
                w += n;
                for (r = buf; r < w && *r != '\n'; r++);
                if (r != w)
                        break;
        }

        /* Anything other than 250 is an error */
        if (buf[0] != '2' || buf[1] != '5' || buf[2] != '0')
                goto Protocol_Error;

        /* Finish sending the command text */
        if (send(ss, cmd, len, 0) < 0)
```

```
                goto Socket_Error;

        /* Read the return code and reply length */
        if (receive(ss, (char *)ibuf, 8) <= 0)
                goto Socket_Error;

        /* Exit if the return code is not 0 */
        if (ntohl(ibuf[0]))
                goto Protocol_Error;

        /* Read the reply */
        len = ntohl(ibuf[1]);
        reply = malloc(len + 1);
        if (receive(ss, (char *)reply, len) <= 0)
                goto Socket_Error;

        /* Cut it into individual lines, and output it */
        for (r = reply, e = reply + len; r < e;) {
                for (w = r; w < e && *w != '\r'; w++);
                *w++ = '\0';
                fprintf(writeto, "%s\n", r);
                r = w;
                if (r < e && *r == '\n')
                        r++;
        }

        /* Close the socket and return */
        rc = 0;
        goto Done;

Protocol_Error:
        rc = 1000;
        goto Done;

Socket_Error:
        rc = errno;
        goto Done;

Done:
        free(cmd);
        if (reply)
                free(reply);
        if (ss >= 0)
                close(ss);
        return(rc);
}


#ifndef NO_MAIN
int main(int argc, char **argv)
{
        char cmd[8192], hostname[80], *w, *r;
        int rc, n;
        unsigned short port;

        /* Parse positional parameters */
        if (argc < 5) {
                printf("\
Syntax: lcmdx hostname[:port] address password command\n");
                return(EINVAL);
        }

        port = DEFAULT_PORT;
        for (r = argv[1], w = hostname; *r && *r != ':';)
                *w++ = *r++;
        *w = '\0';
        if (*r == ':')
                port = atoi(++r);

        for (n = 4, w = cmd; n < argc; n++) {
                if (w != cmd)
                        *w++ = ' ';
```

```
                for (r = argv[n]; *r; *w++ = *r++);
        }
        *w = '\0';

        /* Execute the command */
        rc = LSV_send_command(hostname, port, argv[2], argv[3], cmd,
                                        stdout);
        if (rc == 1000)
                printf("\
>>> Protocol error while communicating with LISTSERV.");
        else if (rc != 0)
                printf("\
>>> Error - unable to initiate communication with LISTSERV (errno=%d).\n", rc);

        return(rc);
}
#endif
```

Revision History

20011214-001 Initial beta of 1.8e Developer's Guide.
20020523-001 Initial release of 1.8e Developer's Guide.
20020529-001 Customer noticed that page numbers and link to index from TOC had disappeared; fixed. Also added info on &*URLENCODE() function to 4.6.3. Uploaded changes.
20020606-001 Added DBMS support table in 4.3.3. Uploaded changes.
20020924-001 5.3; local commands must be defined in upper case in localcmd.file and LISTSERV must be restarted to recognize them.
20021009-001 1.2, GETPOST command shown did not include listname parameter
20021009-002 4.3.3, instructions updated to reflect Makefile changes
20021011-001 1.8e-2002a
20030930-001 6.4.1, double quote characters in X-STL must be escaped with backslash
20031001-001 3.5.1, nolist changelog jobs must be sent as DISTRIBUTE MAIL-MERGE
20031106-001 5.1, added a TELL3 example showing use of ECHO
20040220-001 Uploaded changes.
20040430-001 5.1, Made it more clear that absolute paths are almost always necessary in list exit programming.
20040826-001 2.4, commas are required between options in JOB card
20040826-002 Added 4.6.3.1 to describe the new NAMEFIELD=, &*NAME;, and &*TOFIELD; MM options. This section will be subsumed in 4.6.3 at the next full version of LISTSERV.
20041207-001 SPAM_EXIT
20041207-002 4.6.4, corrected *PARTS example that had HTTP-like semicolons.
20050615-001 A number of changes throughout regarding LSMTP and mail-merge.
20050803-001 Clarification of Oracle versions supported.

Index

Aliases
owner-listname ... 39
BITNET ... 26, 30
Commands
ADD 26, 29, 31, 38, 46, 53, 54, 74, 75, 88, 90
CMS ... 77
CP ... 77
DISTribute ... 28
FOR ... 45, 57, 82, 83, 91
GET ... 90, 91
Info ... 5, 6
PUT ... 29, 89
PW .. 20, 22, 26, 28, 29, 31, 36, 37, 39, 59, 60, 61, 62, 66, 89, 90
SCAN ... 91, 92
SET ... 45, 55, 64, 65, 76, 87, 88
SHOW ... 88
SIGNOFF ... 65, 75, 76
List header keywords
Access Control Keywords
Attachments= ... 85
Filter= ... 74
Distribution Keywords
Digest= ... 5, 6
Prime= ... 34
Topics= ... 56
Error Handling Keywords
Auto-Delete= ... 38, 62
List Maintenance and Moderation Keywords
Notebook= ... 89
Owner= ... 64, 89
Other Keywords
DBMS= ... 49, 52, 60, 61, 62, 63, 66, 69
Limits= ... 6
Security Keywords
Change-Log= ... 38
Confidential= ... 89
Exit= ... 70, 85
PW= .. 20, 22, 28, 31, 36, 37, 39, 59, 60, 61, 62, 66, 89
Subscription Keywords
Subscription= ... 75
LISTSERV maintainer ... 26, 27, 28, 70
parameters .. 6, 12, 22, 48, 49, 50, 51, 52, 65, 70, 71, 72, 73, 74, 78, 85, 88
RFC1429 ... 26
RFC821 ... 28, 37, 43, 90
RFC822 ... 32, 33, 34, 38, 63, 74, 90
Site Configuration Keywords
DATABASE ... 5
DIST_ALLOWED_USERS .. 26, 28, 29, 58, 59
DIST_OWNER_MAIL_MERGE ... 48
DIST_SECURITY ... 26, 30
JOB_STAT_DEFAULT ... 22
LIST_EXITS ... 70, 85
MAILER ... 28
MAXBSMTP ... 48
NODE ... 30, 33, 83
POSTMASTER ... 26, 27
PRIMETIME ... 34
RSCS ... 30
SMTP_FORWARD ... 42
TCPGUI_IPADDR ... 86
TCPGUI_PORT ... 86