L-Soft international, Inc.

Advanced Topics Manual for LISTSERV 17.0

Table of contents

Information about this document

Manuals for LISTSERV are available in PDF and HTML at
https://www.lsoft.com/manuals/index.html

L-Soft invites comment on its documentation. Please feel free to send your comments by email to manuals@lsoft.com

Last Updated: 8 Aug 2024

Preface

Every effort has been made to ensure that this document is an accurate representation of the functionality of LISTSERV®. As with every software application, development continues after the documentation has gone to press so small inconsistencies may occur. We would appreciate any feedback on this manual. Send comments via email to: MANUALS@LSOFT.COM

The following documentation conventions have been used in this manual:

- Menus, options, icons, fields, and text boxes on the screen will be bold (e.g. the **Help** icon).
- Clickable buttons will be bold and within brackets (e.g. the **[OK]** button).
- Clickable links will be in blue and underlined (e.g. the Edit link).
- Directory names, commands, and examples of editing program files will appear in `Courier New` font.
- Some screen captures have been cropped and/or edited for emphasis or descriptive purposes.

Section 1 DBMS and Mail-Merge

In this section, you will find a brief description of LISTSERV's 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.

Note: Embedded mail merge is the default.

Contents:

1.1 Overview

1.1.1 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 under Windows, and Oracle's OCI interface (SQL*Net) on AIX, Linux, and Solaris (SPARC and x64). IBM DB2 is also supported natively via CLI.

L-Soft formally supports any Microsoft-supported version of **SQL Server** as a datastore for mailing lists. At the time of this writing, SQL Server 2014 (with Service Pack 3) and later are still under either Microsoft's mainstream or extended support. For more information, see Microsoft Lifecycle Policy.

Important: L-Soft does NOT support Microsoft Access (any version) as a datastore for mailing lists. Microsoft Access is not a full implementation of Microsoft SQL and thus lacks important functions required by LISTSERV.

L-Soft formally supports any Oracle-supported version of **Oracle Database** including or posterier to version 8i as a datastore for mailing lists. However, it should be noted that while Oracle offers limited lifetime ("sustaining") support for its DBMS products, as of this writing, only Oracle 12.1 and later remain under Extended Support or higher, and are thus eligible for updates/fixes/security alerts and so forth. L-Soft recommends that sites using Oracle Database should, at minimum, be using a version of Oracle Database under Extended Support or higher. For more information, see Oracle's Lifetime Support Policy.

L-Soft formally supports any IBM-supported version of **DB2** as a datastore for mailing lists. At this writing, DB2 version 9.7 and later remain under, at minimum, extended support. For more information, see DB2 Distributed end of support (EOS) dates.

Finally, L-Soft formally supports any in-support version of **MySQL Server (Enterprise or Community)** as a datastore for mailing lists. As of this writing, this means MySQL 5.7 and later remain under, at minimum, extended support. For more information, see Oracle's Lifetime Support Policy and find "Oracle's MySQL Releases" in the table of contents.

The MariaDB Foundation's **MariaDB** is also in use by some LISTSERV sites with mixed reviews as to compatibility (primarily connector issues). **L-Soft does not have an official policy regarding MariaDB at this time.** At present, MariaDB 10.2 and later remain under MariaDB's maintenance. For more information, see the Maintenance Policy for MariaDB, which details support end dates for various versions.

1.1.2 Mail-Merge

Documented Restriction: In order to use the mail-merge features, the site configuration variable EMBEDDED_MAIL_MERGE must be set to a value of 1 (that is, enabled). This is the default.

LISTSERV's mail-merge support allows you to send individually customized 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.

1.2 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 works with any modern version of LISTSERV Classic or Classic HPO, but the newer the version, the better, as older versions may not have enhancements and bug fixes included in newer versions. We strongly recommend upgrading to the latest release version of LISTSERV prior to enabling DBMS support.
- Mail-merge support requires LISTSERV Classic or Classic HPO.
- DBMS-enabled LISTSERV versions prior to 14.5 must set EMBEDDED_MAIL_MERGE set to 1 (enabled) in the site configuration file. This is the default setting in LISTSERV 14.5 and later.
- 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 ODBC (Unix), you will need to install unixODBC. At minimum you will need the base 64-bit unixODBC package. Some systems may also need unixODBC-devel, but in most cases, LISTSERV relinks correctly with only the base package installed.
  - For OCI, you need to install and configure the Oracle 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, butOracle databases can be accessed via ODBC.)
  - For DB2, you need to install the appropriate IBM JDBC drivers for your operating system and DB2 version.

- If you are using Microsoft Windows, you must be running a supported version of Windows. L-Soft drops support for Windows versions when Microsoft ends what they call "extended support". Our current LISTSERV installation kits for Windows 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. For a list of the Windows operating systems currently supported by L-Soft, please visit http://lsoft.com/products/listserv_os.asp .
- If using the DBMS interface, you may wish 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. If you want LISTSERV to be able to create tables on its own, you MUST grant CREATE TABLE to LISTSERV's DBMS username.
- A compiler (such as gcc) 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.

1.2.1 Selecting a Suitable DBMS Product

This section applies only to ODBC users (including Oracle users under Windows). OCI users (except under Windows) 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.

  This should not be an issue for SQL Server users, as ESCAPE support was added many years ago in version 6.5. However, the Microsoft Access DBMS product appears to have this restriction, and as such, 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 behavior. 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 behavior 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 planning to use the DBMS interface only 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.

1.3 Installation

Installation instructions have been broken down by operating system. Make sure that your current LISTSERV license is installed before you begin, and that you have made a backup of your LISTSERV directory tree.

1.3.1 Windows

Standard Windows kits include support for ODBC and mail-merge by default.

Documented Restriction: LISTSERV, being a 64-bit application, requires that you use the 64-bit ODBC management application built into Windows. Any ODBC DSNs created using the 32-bit version of the ODBC management application will be invisible to LISTSERV and won't be usable.

1.3.2 Unix

LISTSERV 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
Linux-S390	No	No	Yes
Linux (x86-64 platforms)	No	Yes	Yes
Solaris SPARC	Yes	Yes	Yes
Solaris x64	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 the share function names inside LISTSERV. (It should be noted that this is due to design decisions made by Oracle and IBM, not by L-Soft; L-Soft has no choice in the matter.)

The current LISTSERV for unix kits contain a Makefile which is set up to relink 'lsv' without any DBMS support by default. An "OS-specific flags" section s available where you can add or remove DBMS support simply by adding or removing the reference to the appropriate no*.o file. The default compilation flags for the unix operating systems supported by LISTSERV are

```
CFLAGS_AIX=
CFLAGS_LSV_AIX=nooci.o nocli.o -lldap -llber

CFLAGS_Solaris=-lsocket -lnsl
CFLAGS_LSV_Solaris=nooci.o nocli.o -lldap -m64

CFLAGS_Linux=
CFLAGS_LSV_Linux=nooci.o nocli.o -lldap -llber -lresolv
```

In the Makefile, the **`` CFLAGS_`uname` ``** macros are reserved for libraries needed by both LISTSERV itself and its associated utilities. The **`` CFLAGS_LSV_`uname` ``** macros contain the object files and libraries needed only by LISTSERV itself. In these examples we will concern ourselves only with the **`` CFLAGS_LSV_`uname` ``** macros.

For instance, if you are running LISTSERV under Solaris, the default flags line is

```
CFLAGS_LSV_Solaris=nooci.o nocli.o -lldap -m64
```

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

```
CFLAGS_LSV_Solaris=nocli.o -lldap -m64
```

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

```
CFLAGS_LSV_Solaris=nocli.o -lldap -m64
```

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

```
CFLAGS_LSV_Solaris=-lldap -m64
```

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

```
CFLAGS_LSV_Solaris=nooci.o -lldap -lodbc -m64
```

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).

Important: The **`-lldap`** and **`-m64`** flags specified are required for Solaris, and must not be removed from the CFLAGS_LSV_Solaris macro.

The other supported unixes are configured in a similar manner. For instance, linking unixODBC support into a Linux installation would look like this:

```
CFLAGS_LSV_Linux=nooci.o -lldap -llber -lresolv -lodbc
```

Important: Again, the **`-lldap, -llber,`** and **`-lresolv`** flags specified are required for Linux, and must not be removed from the CFLAGS_LSV_Linux macro.

Important: Do not attempt to relink with DBMS support unless you have the appropriate DBMS support (SQL*Net, JDBC, unixODBC) already installed on your machine. Without this support, the link option will fail.

1.3.3 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.

1.4 Post-installation tasks

Post-installation instructions have been broken down by feature. References to the “LISTSERV configuration” correspond to SITE.CFG under Windows, and go.user for unix systems.

Note: Do not forget to export configuration variables under unix.

1.4.1 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, whereas individual list owners may not have this information.

1.4.2 ODBC Interface (Windows)

(For unixODBC, please see Section 1.4.5 unixODBC (UODBC) Interface.)

To activate the Windows ODBC interface, you must first create a system-wide ODBC data source. This MUST be done with the 64-bit verson of the ODBC management application. Thankfully, this is the default under 64-bit Windows systems, and the icon can be found in the Administrative Tools folder from the Start Menu. Alternately, click the “Run” command, and type

```
%windir%\system32\odbcad32.exe
```

into the dialog box, then click OK.

Once the management app is open, select the System DSN tab, click [Add...], and then select the appropriate driver and fill out the driver-specific form.

(Under Windows Server 2016 and later, Microsoft have placed links for both the 32-bit and 64-bit versions of the ODBC interface into the Windows Administrative Tools start menu group, so it should no longer be necessary to use the workaround above.)

Important: L-Soft does not recommend supplying passwords in a system-wide DSN (if offered at all by the driver) because any Windows user has access to this information. LISTSERV should be configured to send the appropriate password, instead.

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 Section 1.4.7 Generic DBMS Post-Installation Tasks for instructions that are common to ODBC and OCI.

1.4.3 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: Under Windows, external authentication is based on the Windows userid under which the application (LISTSERV) is running. When you run LISTSERV interactively, you are usually not running LISTSERV under the same Windows 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 Section 1.4.7 Generic DBMS Post-Installation Tasks for instructions that are common to ODBC and OCI.

1.4.4 CLI Interface

This support is similar to the ODBC support documented in Section 1.4.2 ODBC Interface (Windows), but the configuration variables are called CLI_DSN, CLI_UID, and CLI_AUTH.

1.4.5 unixODBC (UODBC) Interface

The prerequisite for unixODBC 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 this interface is designed primarily to work with MySQL, and does not formally support other DBMS products accessible 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 in Section 1.8 Installing and configuring unixODBC with LISTSERV and MySQL, below.

1.4.6 Connecting to multiple simultaneous database sources

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 our first 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).

Note: When defining data sources under unix, remember that you must enclose the settings in double-quotes, and the configuration variables must be exported. For instance,

```
OCI_CONNECT=”oci-connect-string-whatever-it-is”
export OCI_CONNECT
```

and so forth.

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=.

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.

Finally, for unixODBC, you first define your default UODBC connection, then alternate UODBC data sources are defined with additional parameters of the form

UODBC_DSN_*server_alias*=
UODBC_UID_*server_alias*=
UODBC_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. The default data source can be as simple as the following (using ODBC as an example):

```
ODBC_DSN=NOLOGIN
ODBC_UID=NOLOGIN
ODBC_AUTH=NOLOGIN
```

Note: Remember that LISTSERV for Windows does not support Oracle (OCI) databases natively, but Oracle databases can be accessed via ODBC from Windows.

1.4.7 Generic DBMS Post-Installation Tasks

A number of additional configuration variables are available to alter the default behavior 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.

1.4.8 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.

1.5 Creating DBMS lists

1.5.1 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.

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.

1.5.2 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.

1.5.3 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).

1.5.4 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
reserved	4194304	see UTF-8 notes below

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

Notes on UTF-8: The "reserved" bit value 4194304 indicates that the subscriber's name in the list record is in UTF-8. Normally, LISTSERV sets this value automatically at subscribe time based on the character set found in the "real name" field. In this case, the flag should always be left in the state in which it is found.

When adding DBMS records using extra-LISTSERV methods, this flag should be set if you are using UTF-8 in the name field. If the name field is plain ASCII, then you do not need to set this flag. While it is unlikely that setting this flag to "on" for a plain ASCII name field will cause problems, it may cause up conversion to UTF-8, which may not be desired. The best practice is to leave this flag "off" for plain ASCII.

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 value for 14 August 2018 is 736919.

Documented Restriction: If you do not set the subscription date field for a new subscriber, LISTSERV will probe the address at least once, immediately, regardless of the Renewal= setting for the list and/or regardless of whether NORENEW is set for the address. If you do not want new subscribers to receive an initial subscription probe, be sure to set the subscription date!

The sixth number is a bitmask indicating to which topics the user is subscribed. 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).

1.5.5 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 Section 1.5.6 Preserving Options When Migrating from Non-DBMS to DBMS Lists if you are migrating from a “standard” LISTSERV list to a DBMS-type list.)

Note: 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: 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 = 1152. 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, i.e.,

```
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 14 August 2018, the value on that date is 736919, so the setting would now be:

```
4;1024;0;0;736919;16777215
```

Documented Restriction: If you do not set the subscription date field for a new subscriber, then LISTSERV will probe the address at least once, immediately, regardless of the Renewal= setting for the list and/or regardless of whether NORENEW is set for the address. If you do not want new subscribers to receive an initial subscription probe, be sure to set the subscription date!

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: 14 Aug 2018

The topics you subscribe to are: All
```

1.5.6 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, i.e., 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.

Warning: Do not change the value of the first character from 3 to 4 when migrating as this is guaranteed to cause breakage.

1.6 Using the Mail-Merge functions

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

- Using the web interface.
- By sending DISTRIBUTE jobs to LISTSERV.
- Using LISTSERV Maestro to construct and send complex mail-merge jobs.

The first two methods will be described in this document. For more information about using mail-merge jobs in LISTSERV Maestro, please see the LISTSERV Maestro documentation.

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.

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

1.6.1 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, 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 Windows)

`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).

1.6.2 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 Section 9 Relayed File Distribution & the DISTRIBUTE Command 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 2019 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:

- o Jobs based on a DBMS back-end with an arbitrary SELECT statement.
- o Jobs where the recipient data is extracted from an existing LISTSERV list (either a DBMS list or a traditional list).
- o 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: 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 2.2 of this manual) to split the command over multiple physical lines. See for instance the second example below.

1.6.2.1 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, i.e., 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: A “continuation card” was used in the DISTRIBUTE command line above, because the line was so long it wrapped. See Section 2.2 General Syntax Rules 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.

1.6.2.2 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 behavior 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).

1.6.2.3 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.

1.6.2.4 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:

```
20190704100221 BOUNCE jack@xyz.com 5.5.0 User Unknown

20190704100223 BOUNCE Joe.Doe@abc.def.com 552 5.2.2 Mailbox full
```

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 contain only 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.

LISTSERV’s BOUNCE records provide the date/time, bouncing email address, and information about why the message bounced, with a syntax of

```
20190329174013 BOUNCE USER@ZYX.COM x.x.x Bounce Message Here
```

For example:

```
20191107112809 BOUNCE BOGUSUSER@RERUN.IN.LSOFT.COM 5.1.1 Mailer
[192.168.254.101] said: “550 5.7.1
<bogususer@RERUN.IN.LSOFT.COM>... Relaying denied”
```

1.6.2.5 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.

1.6.3 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.

&*URLENCODE(); is a function 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.

1.6.3.1 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 customize the response on a per-list basis using the standard web template Customization features, which are described in the list owner's guide.

1.6.3.2 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, 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.

1.6.3.3 Advanced Substitutions

In addition to the above, two further built-in mail-merge substitutions and a related (optional) DISTRIBUTE keyword are available. They were 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).

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 additional substitution variables are available:

- **&*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: 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.

1.6.4 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 baseball 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.

1.6.4.1 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.

1.6.4.2 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.

1.7 LISTSERV LDAP and Dynamic Query List (DQL) support

These features are not available in LISTSERV Lite. They require LISTSERV Classic or Classic HPO.

LISTSERV is now able to interface to LDAP servers to authenticate user logins and to insert LDAP attributes in mail-merge distributions. For full information on LISTSERV/ LDAP integration, please see Section 7 LISTSERV and LDAP.

In addition, LISTSERV is now able to execute on-demand Dynamic Queries against either LDAP or traditional DBMS servers, and use the results for access control or mail delivery. For instance, an LDAP query against an Active Directory server could be used to grant list owner privileges to all members of a particular Windows Security Group. A DBMS query could be used to combine employee rosters for two departments and send a single copy of an announcement to each unique employee e-mail address. For more information on Dynamic Query Lists (DQL), please see Section 8 Dynamic Queries.

1.8 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.

Terminology note: The community-developed version of MySQL is now known as MariaDB. The instructions below apply to both MySQL and MariaDB.

1.8.1 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.

1.8.2 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 https://dev.mysql.com/downloads/connector/odbc/. (We strongly recommend using the GA ("Generally Available") release.) Documentation for the installation of MyODBC is found at https://dev.mysql.com/doc/connector-odbc/en/connector-odbc-installation.html.

1.8.3 unixODBC

unixODBC can be installed either as a package or from source. Both are available from http://www.unixodbc.org/, or if you have a package installer such as yum or apt, it is likely available for installation from one of the standard repositories. 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, please visit unixodbc.org at the hyperlink above, choose "Download", and the source and instructions are available there.

1.8.4 Creating a unixODBC System DSN for LISTSERV

Note: Please be aware that the version numbers for your installed versions of the MySQL ODBC Connector and unixODBC may differ from those below. Please verify the correct versions before attempting to configure LISTSERV's system DSN.

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 2.3.7
DRIVERS............: /etc/odbcinst.ini
SYSTEM DATA SOURCES: /etc/odbc.ini
FILE DATA SOURCES..: /etc/ODBCDataSources
USER DATA SOURCES..: /root/.odbc.ini
SQLULEN Size.......: 8
SQLLEN Size........: 8
SQLSETPOSIROW Size.: 8
```

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

1.8.5 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:~#
```

1.8.6 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
```

1.8.7 Connecting to External MySQL Databases

Note: Please be aware that the version number for your installed version of the MySQL ODBC Connector may differ from the below. Please verify the correct versions before attempting to configure LISTSERV's system DSN.

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
```

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.

1.8.8 Known Issues

- There is a KNOWN ISSUE with unixODBC and the MariaDB (MySQL) version 5.2 driver. LISTSERV throws the following error when attempting to access tables via unixODBC configured with the 5.2 driver:

```
31 Aug 2020 16:03:48 Connecting to UODBC data source DEFAULT...
31 Aug 2020 16:03:48 Connected to data source listserv.
SQL> SELECT EMAIL,NAME,MYLIST-L FROM LISTSERV WHERE {fn
UCASE(EMAIL)} LIKE ?
>>> S1001/4001: [MySQL][ODBC 5.2(w) Driver][mysqld-5.5.65-
MariaDB]Memory allocation error
31 Aug 2020 16:03:48 >>> Error X'0100003B' searching DBMS list
<<<
31 Aug 2020 16:03:48  -> Severity: Error
31 Aug 2020 16:03:48  -> Facility: DBMS interface
31 Aug 2020 16:03:48  -> Abstract: SQL error
```

It should be carefully noted that the error is occuring in the MariaDB driver and *not* in LISTSERV; LISTSERV is merely the messenger. Rolling back to the version 5.1 driver fixes the issue. L-Soft has not performed formal testing on MariaDB drivers posterior to version 5.2 and cannot at this time guarantee that later versions will work properly with LISTSERV.

1.8.8 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

It should be noted that the last two references are outdated, but the basic instructions haven't changed.

Section 2 Commands-Job Feature and CJLI Interpreter

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 (e.g. dataset concatenation with a blank DD card) have not been implemented, and new features have been implemented whenever required.

Contents:

2.1 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 (e.g. 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 three kind of cards in a job stream:

1. Control Cards – Starts with "//" in column 1 and are interpreted by the CJLI. Control cards are divided into three categories:
    - 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).
    - 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.
    - 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.

2. Command Cards – Contains 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.

3. Data Cards – These cards 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.

2.2 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. For instance,

   No space before comma:

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

   Space before comma:

   ```
   //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:

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

  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.

  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.

  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
  ```

2.3 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`

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 may be set with the JOB_STAT_DEFAULT site configuration variable.

`AFTER=`date-spec**`|`**time-spec**`|"`**date-spec time-spec**`"`**

Tells LISTSERV when to run the job, i.e., run it after the time specified. There are three valid formats:

`AFTER=`date

`AFTER=`time

`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=""
```

2.4 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.

2.5 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 one of 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 (i.e. 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 that could 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.

2.6 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.

2.7 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 mail file 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 LISTSERV on z/VM: 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 the standard LISTSERV documentation.

Section 3 Delayed Mailing List Posting

Since LISTSERV 1.8e (13.0), it has been possible to delay execution of DISTRIBUTE jobs until a given time, by use of the JOB card keyword AFTER=. However, this was not originally an option for regular postings to mailing lists.

When LISTSERV 16.0 was released in 2009, it became possible to schedule standard mailing list postings using an internal RFC822 mail header tag as follows:

```
X-LSVAfter: date time
```

This tag, like all other X-LSV tags inserted by LISTSERV, are deleted from the header before being sent to the recipient. The date and time parameters can be specified in any of the formats supported by LISTSERV, except those that contain white space. It is suggested that the format

```
yyyy-mm-dd hh:mm:ss
```

should be used. For example:

```
X-LSVAfter: 2019-11-26 13:41:00
```

Important: The value MUST NOT be enclosed in quotes or LISTSERV will not recognize it. Similarly, the value MUST NOT be a standard RFC822 date field. A setting like

```
X-LSVAfter: Mon, 21 Oct 2019 12:00:00 -0400
```

is invalid, and will not be recognized.

When the scheduled posting time arrives, LISTSERV will substitute the current local server date and time for the value in the original RFC822 Date: field.

The simple message below utilizes X-LSVAfter to delay an early-morning post until noon:

```
Date: Mon, 21 Oct 2019 09:14:00 -0400
From: joe.user@example.com
To: lunch@listserv.example.com
Subject: Lunchtime!
X-LSVAfter: 2019-10-21 12:00:00

It's time for lunch!
```

Note: This feature requires either a mail client that is capable of adding ad-hoc RFC822 headers into messages, or the ability to construct RFC821/822 email messages manually and feed them directly to SMTP for delivery to LISTSERV.

The complete RFC821 envelope for the above example would be something like

```
HELO
MAIL FROM:<joe.user@example.com>
RCPT TO:<lunch@listserv.example.com>
DATA
Date: Tue, 20 Oct 2009 09:14:00 -0400
```

```
From: joe.user@example.com
To: lunch@listserv.example.com
Subject: Lunchtime!
X-LSVAfter: 2009-10-20 12:00:00

It's time for lunch!
.
QUIT
```

The LISTSERV console log shows the following:

```
20 Aug 2019 09:12:37 Processing file 0077 from
MAILER@LISTSERV.EXAMPLE.COM
20 Aug 2019 09:12:37 File 0077 suspended until 20 Oct 2009
12:00:00
(…)
20 Aug 2019 12:00:00 Processing file 0077 from
MAILER@LISTSERV.EXAMPLE.COM
20 Aug 2019 12:00:00 Processing mail from joe.user@EXAMPLE.COM for
LUNCH
20 Aug 2019 12:00:00 Rebuilding HTML page for LUNCH...
20 Aug 2019 12:00:01 Message DISTRIBUTEd to 125 recipients.
```

Note: If LISTSERV finds an X-LSVAfter: header in the message that either is blank or does not match any of the standard date/time formats understood by the server, then the following will appear and the message will be processed immediately:

```
20 Oct 2009 09:26:35 Processing file 0078 from MAILER@LISTSERV.EXAMPLE.COM
>>> Invalid date/time specification in X-LSVAfter: tag - ignored
20 Oct 2009 09:26:35 Processing mail from joe.user@example.com for LUNCH
20 Oct 2009 09:26:35 Message DISTRIBUTEd to 125 recipients.
```

Tip: For details on scheduling a delayed posting using the Message Posting Interface, see the List Owner's Manual or the Site Manager's Operations Manual.

Section 4 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.

Contents:

4.1 What is a List Exit?

List "exits" are 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, i.e., 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 restart LISTSERV after making this change.

Note: 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 and your exit is called XYZ.CMD, you set LIST_EXITS to "XYZ".

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.

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 z/VM, the program must be called XYZ EXEC A1 (on LISTSERV's A-disk) and must be written in REXX.
- On unix, the program must be called XYZ and must be located in the 'home/' subdirectory (i.e., $LSVROOT/home). 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.[1]
- On Windows, the program must be called XYZ.CMD and must be located in the MAIN subdirectory. It must be written in the Windows 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** z/VM, where multiple parameters of arbitrary length and contents may be passed to a REXX program. On z/VM, the system independent convention is never used, because it is unnecessary and less efficient than the native method. z/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 unix, the file is not created when there is only one parameter and EXIT_INPUT defaults to 0. Since most exits have only a single parameter, in most cases this improves performance. Note that LISTSERV will take care of deleting the 'exit.input' file 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 unix. Under Windows systems, the first parameter is *only* passed through the 'exit.input' file. Again, this may simplify programming for simple exits.

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 z/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. 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 z/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 z/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: 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 z/VM users (it is not possible to provide an API for non-z/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

---

1. To be clear, the exit program **MUST NOT** have a file extension. If you write a MYEXIT exit program in perl, for instance, DO NOT name the exit MYEXIT.pl -- LISTSERV will not see it and you will see errors in the LISTSERV log each time an exit point is called for the list(s) in question. Thus the script simply should be named MYEXIT regardless of the language in which it is written. Since the script contains (or should contain) its own internal information about the interpreter being used to execute it (the "bang-hash" path in the first line), there is no overriding reason for the file to have an associative extension.

4.2 List Exit Points

The following list exit points are available.

4.2.1 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.

4.2.2 DELETE/SIGNOFF/CHANGE Entry Points

Name: CHG_REQ

z/VM Parameters: Three; originator's e-mail address '15'x target subscriber's e-mail address '15'x new e-mail address
Non-z/VM Parameters: Three; originator's e-mail address '20'x target subscriber's e-mail address '20'x new e-mail address
Return code: 0=Accept, 1=Reject
Description: This exit point is called for the CHANGE command when issued by a list subscriber. It is NOT called for CHANGE commands originating from the list owner or node administrator on behalf of a subscriber. This exit point allows you to reject the operation. If you return the value 1, LISTSERV rejects the operation.

Name: DEL_FILTER

z/VM Parameters: One or two; target e-mail address '15'x originator's address
Non-z/VM Parameters: One or two; target e-mail address '20'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

z/VM Parameters: Two; target e-mail address '15'x originator's address
Non-z/VM Parameters: Two; target e-mail address '20'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.

4.2.3 Other Exit Points

Name: SET_REQ

z/VM Parameters: Three; originator's address '15'x list of options to be altered '15'x target e-mail address
Non-z/VM Parameters: Three; originator's address '20'x list of options to be altered '20'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, i.e., 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: 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.

4.2.4 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 z/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
```

where keywords = 'KWD1 value of first keyword'||'15'x'KWD2 second keyword'||...

4.3 Local Command Definition (non-z/VM)

It is possible to define "local" LISTSERV commands on non-z/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 restart LISTSERV for the changes to take effect.

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.

4.4 SPAM_EXIT

This “exit point” 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 https://spamassassin.apache.org.

Note: 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.

4.4.1 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 . 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 an .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.

4.4.2 Practical Application

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'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.

4.4.2.1. 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 (https://regina-rexx.sourceforge.io/, Windows kit download available at https://sourceforge.net/projects/regina-rexx/files/regina-rexx/ . 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.

Note: At time of writing, the latest version of Regina REXX is 3.9.1; however, the saexit script was written under version 3.0 and is known to work under that version. The REXX.EXE available from L-Soft is version 3.0.

Step 3 of 4: Install and configure SAEXIT script.

- Download the L-Soft supplied sample script at one of the following URLs:

  Unix: https://ftp.lsoft.com/LISTSERV/UNIX/CONTRIB/SAEXIT.PL.gz

  Windows: https://ftp.lsoft.com/LISTSERV/Windows/CONTRIB/saexit.zip

- 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%
  ```

  Note: For convenience, the saexit.cmd file is included in saexit.zip (see link above).

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. Current maintenance for LISTSERV 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.

z/VM

The spam exit is also available on z/VM, but there is no version of spamc for this system. Since z/VM 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.

The configuration parameter SPAM_MAXSIZE can be used to automatically accept messages larger than a certain size without wasting cycles scanning them. 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:

z/VM: **`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.

4.5 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 version of LISTSERV 1.8d.

Tip: It should be noted that modern versions of LISTSERV reject UUEncoded files by default when they are sent to lists; see the documentation for the Attachments= list header keyword for more information. Thus this exit is technically obsolete, and is retained in the documentation only because it remains a fairly involved example of LISTSERV exit programming.

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.
 */

/***********************************************************
******************/
/*                  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.

Notes: 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).

The LISTSERV.CMSUT1 file that the exit searches 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 was not possible to block it with the original Attachments= list header keyword, and the exit was the only way to guard against it.

Modern versions of LISTSERV, with its integrated anti-virus scanning and ability to filter encoded inline attachments, removes the need for this particular exit, but the above remains a useful example of exit programming.

Tip: Windows users should be able to use more modern scripting languages, such as Perl or PowerShell, to write exits. The key is that any exit you write for Windows MUST be called from a CMD file as has been documented above, regardless of the language in which it is written.

Section 5 Advanced SMTP Topics

This section discusses several diverse topics at the SMTP mail server level that can affect LISTSERV's operations.

Section 5.1, *SMTPL-level spam control for Windows*, explains how to integrate spam filtering at the SMTPL.EXE "listener" level so that spam can be identified and rejected prior to reaching LISTSERV. Since SMTPL is multi-threaded, this can significantly increase single-threaded LISTSERV's throughput efficiency by the simple expedient of stopping spam before it ever gets to LISTSERV's inbound spool.

Section 5.2, *SMTP Worker Pool support*, allows a large LISTSERV site to run large lists on a server and also (with appropriate hardware) to set up high-speed, low-volume delivery nodes for command responses and the like.

5.1 SMTPL-level spam control for Windows

LISTSERV for Windows may use the `SMTPL.EXE` SMTP "listener" service to submit inbound email to a third-party spam scanning product. This feature allows sites with significant inbound spam problems to scan and reject spam before it reaches the main LISTSERV process, thus freeing LISTSERV itself to handle legitimate email.

Note: 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 https://spamassassin.apache.org.

Important: 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 regarding SpamAssassin. Although L-Soft's support department will gladly answer questions about the integration of SpamAssassin and SMTPL, we cannot answer questions about SpamAssassin itself.

5.1 Enabling

To enable SMTPL-level spam filtering, you must:

- Install a third-party spam filter.
- Provide a script that will scan messages using the third-party filter (if using SpamAssassin, you can use the L-Soft supplied script).
- Activate this script by making changes to the LISTSERV configuration (which is also used by SMTPL) and restarting SMTPL.

SMTPL will then scan every message it processes, with a few exceptions, and reject messages identified as spam.

5.2 Configuring LISTSERV to Use SpamAssasin

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:

- You are running LISTSERV 16.0 or later (preferably the Generally Available version, currently 17.0) with the version of `SMTPL.EXE` that shipped with it (version 1.0w) or later.
- 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 (i.e. you can run LISTSERV on Windows and SpamAssassin on unix).
- 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 lspamc/spamd. We will call this file testmsg.txt.

Step 1 of 4: Install and Test SpamAssassin Client

Download and install the lspamc.exe executable from L-Soft, and place it in a directory in LISTSERV's path – for instance, the LISTSERV\MAIN directory.

Note: `lspamc.exe` is slightly different from the **`spamc.exe`** provided for LISTSERV spam scanning. **`lspamc.exe`** should be used for SMTPL spam scanning.

To test the client, issue the following command (substituting the correct FQDN for your SpamAssassin server for "spamd.example.com"):

```
C:\> lspamc -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 REXX (if not already available)

Install a REXX interpreter, such as Regina REXX (https://regina-rexx.sourceforge.io/, Windows kit download available at https://sourceforge.net/projects/regina-rexx/files/regina-rexx/ . 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 lspamc.exe.

Note: At time of writing, the latest version of Regina REXX is 3.9.1; however, **`SASMTPL.REXX`** that you will download in the next step is incompatible with Regina REXX version 3.1 and later. The version of **`REXX.EXE`** downloadable from L-Soft is version 3.0, and is known to work with the script.

Step 3 of 4: Install and Configure the SASMTPL Script

- Download the L-Soft supplied sample script:

https://ftp.lsoft.com/LISTSERV/Windows/CONTRIB/SASMTPL.REXX.zip

- 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 (for instance, `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 (SASMTPL).

Step 4 of 4: Enable the SASMTPL Script

To enable the script, add the following lines to LISTSERV's configuration:

```
SMTP_SPAM_EXIT=!C:\LISTSERV\MAIN\REXX.EXE C:
\LISTSERV\MAIN\SASMTPL.REXX
SMTP_SPAM_THREADS=75
SMTP_SPAM_CHECK=[list of target hostnames to check]
```

Change the paths in `SMTP_SPAM_EXIT` to match your local installation. Ensure that the line begins with an exclamation point (!).

Normally, `SMTP_SPAM_CHECK` may be set to `%MYDOMAIN%`. If you host multiple domains on the same LISTSERV server, you will need to list each domain (space-separated).

Note: For the 75 concurrent spam scans configured in the example, 750M-1G of memory will be required. Depending on available resources, you may want to start lower.

Restart SMTPL to make the change take effect, then mail a known spam message to a test list to confirm that everything is working as it should.

5.2 SMTP Worker Pool support

The SMTP worker pool feature allows a site to run large lists on a server and also (with appropriate hardware) to set up high-speed, low-volume delivery nodes for command responses and the like.

SMTP worker pools sound quite complex but are very simple once you understand what they actually do. LISTSERV recognizes 27 mail delivery pools, called A through Z plus a default pool, called '-' (hyphen). By default, LISTSERV puts all outgoing mail in the default pool, and the workers process mail for all the pools. You can tell LISTSERV to put certain types of messages in different pools, as follows:

- The "Delivery-Pool= x,y" list header keyword (must be set by the LISTSERV maintainer) lets you define the delivery pool for postings to the list (x) and for administrative messages related to the list (y). The latter includes bounces, error monitoring reports, welcome messages, etc. It does NOT include command responses of any kind (even if there is just one command and it is for the list) or responses to postings (posting rejection, request for cookie, forward to editor, etc), which are considered to be command replies and always go to the default pool. If y is missing, x defines the pool for both postings and administrative mail.

- The DEFAULT_LIST_POOL keyword, which defines site defaults for the above (space separated).
- The POOL=x parameter for DISTRIBUTE.
- The DEFAULT_DIST_POOL keyword, which defines the default value for the above.

If you use any of the above, LISTSERV will start assigning outgoing mail to the various pools, while the default pool continues to receive command replies and spontaneous messages that are not tied to any particular list (system errors, etc). By default, all the workers are set to process mail from any of the pools, so it will not have any effect until you reconfigure the workers to only manage some of the pools:

```
SMTP_FORWARD_n=blah blah;POOL=xyzt
```

(use POOL=* for the default behaviour where all pools are processed). Therefore, each individual worker can be configured to manage either a specific pool, a series of pools, or all the pools (the latter is probably not useful). Note that you must always provide at least one worker to process the default pool (POOL=-), since you will always have messages in this pool. Obviously you don't need to provide workers for pools you do not use.

In the current implementation, the pools are one-letter prefixes to the xxx.MAIL filename, rather than separate directories. This is a little bit less efficient, but it makes it a lot simpler for a LISTSERV maintainer to visually check the contents of the queue. Here is a concrete example where we define pool L to be large lists, X to be a high-priority express service for small lists, and M to be the default pool for normal lists:

```
DEFAULT_LIST_POOL=M
SMTP_FORWARD_1=2*FAST.LSOFT.COM;FAST-
BACKUP.LSOFT.COM;POOL=X-
SMTP_FORWARD_2=REGULAR1.LSOFT.COM;BACKUP1.LSOFT.COM;POOL=M
SMTP_FORWARD_3=REGULAR2.LSOFT.COM;BACKUP2.LSOFT.COM;POOL=M
SMTP_FORWARD_4=5*BULK.LSOFT.COM;BACKUP3.LSOFT.COM;POOL=L
```

A simpler setup, where we create a fast channel for command replies, would be:

```
DEFAULT_LIST_POOL=M
SMTP_FORWARD_1=2*FAST.LSOFT.COM;FAST-
BACKUP.LSOFT.COM;POOL=-
SMTP_FORWARD_2=whatever we had before;POOL=M
```

This keeps list messages routed as before and routes command replies to a new, high-speed, low-volume SMTP MTA.

Incidentally, assigning a worker to a single pool is faster because it can then look for files called M*.MAIL and so forth. A worker that manages multiple pools needs to scan for *.MAIL and skip the files it does not service. Either way, introducing pools adds some measure of overhead corresponding to the time spent skipping over files which are not serviced. This would only be a problem in practice if you had a bunch of workers configured to skip most of the files.

Worker pools can get quite interesting for specialized applications when combined with another worker feature called “open time”. This feature lets you define that certain servers should only be used during certain times of the day:

```
SMTP_FORWARD_n=BULKBOX.XYZ.COM(09:00-17:00,19:00-
23:00);OTHERBOX.XYZ.COM
```

This defines a worker that will use BULKBOX during the times specified, and OTHERBOX the rest of the time. A typical use is to bring bulk delivery boxes to the rescue during peak interactive hours, while making sure not to use them when they are processing large postings. But you can also use it like this:

```
SMTP_FORWARD_n=SOMEBOX.XYZ.COM(23:00-06:00);POOL=W
```

This defines a worker that is only allowed to do useful work from 11pm to 6am. If this is the only worker that services pool W, you won't be seeing much activity in that pool outside these hours. Not probes, not welcome messages, nothing (but commands can still be sent and the command reply will be delivered). This could be a good way to shut off a very large list, or to cause a large mailing to start off at a very precise time. The workers are accurate to the minute. Likewise, the workers will stop on a dime (after finishing the current message) at the end of the opening interval. Currently there is no weekday support; however if ithis is needed, it can be implemented with a daily reboot.

Section 6 Feedback Loop Auto-Processing

LISTSERV is able to automatically process Feedback Loop reports, which are sent automatically by some ISPs, such as AOL and Yahoo!, to organizations that are on their whitelist. Once configured, LISTSERV automatically parses the reports and implements the actions required by the whitelist agreement. This helps preserve whitelist status and reduce the number of spam complaints from subscribers.

Use of this feature assumes you already have a feedback loop in place as part of your ISPs whitelist agreement and are able to redirect the feedback loop reports to a new address.

Contents:

6.1 Configuring the SPAM_FEEDBACK_PROBE Variable

The SPAM_FEEDBACK_PROBE variable defines the domains for which to enable automatic processing of Spam Feedback Loops.

There are two steps in configuring automatic feedback loop processing.

1. Define this configuration variable with the space-separated domain names for which to enable Spam Feedback Loops, for example AOL.COM or YAHOO.COM.

2. Create a LISTSERV list dedicated to feedback loop processing. It can be configured as you wish as long as the special addresses from which the spam reports are sent are authorized to post to the list. Test the list carefully to make sure that archiving is working properly and that the list will not send any replies back. Then, add

   Misc-Options= PROCESS_SPAM_FEEDBACK

   to the list configuration to activate automatic report processing.

If the message is recognized as a spam report and refers to a message that originated after you set the SPAM_FEEDBACK_PROBE configuration option, LISTSERV will:

1. Immediately delete the user from all local LISTSERV lists, on the first spam report. This does not include Dynamic Query Lists, which are managed outside LISTSERV.

2. Not send any notification to the user.

3. Add the user to the spam feedback list with the NOMAIL option to keep a record of who has lodged spam complaints. The subscription date is set to the date of the first complaint and the last activity date is set to the date of the last complaint.
4. Log a SPAM_COMPLAINT record to the list's change log, if one has been defined. You can use this information to remove the user from Dynamic Query Lists, if you have any such lists.

This ensures that no further mail will be sent to the user and minimizes the risk for further complaints (although experience has shown that people sometimes find old messages in their mailbox days or weeks later, which can lead to a handful of additional spam reports).

To define in the Web Interface, expand **Server Administration**, then select **Site Configuration**. Click the **Anti-Spam** button, and then define **SPAM_FEEDBACK_PROBE**. For example:

```
SPAM_FEEDBACK_PROBE=AOL.COM
```

By default, SPAM_FEEDBACK_PROBE is not set (that is, no domains are enabled for Feedback Loop processing).

6.2 Configuring the SPAM_FEEDBACK_ACTION Configuration Variable

The SPAM_FEEDBACK_ACTION variable determines what actions to take when LISTSERV receives spam complaints through Feedback Loops.

By default, when a spam complaint is received through a Feedback Loop, LISTSERV will:

1. Immediately delete the user from all local LISTSERV lists, on the first spam report. This does not include Dynamic Query Lists, which are managed outside LISTSERV.
2. Not send any notification to the user.
3. Add the user to the spam feedback list with the NOMAIL option to keep a record of who has lodged spam complaints. The subscription date is set to the date of the first complaint and the last activity date is set to the date of the last complaint.
4. Log a SPAM_COMPLAINT record to the list's change log, if one has been defined. You can use this information to remove the user from Dynamic Query Lists, if you have any such lists that may contain users.

To define in the Web Interface, expand **Server Administration**, then select **Site Configuration**. Click on the **Anti-Spam** button, and then define the value for **SPAM_FEEDBACK_ACTION**.

Possible definitions include (all space-separated actions are taken in turn):

- DELETE: The user is deleted from all LISTSERV lists.
- EXCLUDE: The user is added to the spam feedback lists with the NOMAIL option.
- SERVEOFF: The user is served off.
- SERVEDROP: The user is served off with the DROP option.

- NONE: No action is taken other than logging an entry to the list's change log (if defined).
- LOGALL: An advanced option for customers operating multiple LISTSERV instances, which collects a central copy of all spam reports and forces them to be logged on that instance even if the reports are for another instance.

The default value for SPAM_FEEDBACK_ACTION is DELETE EXCLUDE.

6.3 Configuring the SPAM_FEEDBACK_EXCLUDE Variables

It is possible to exclude subscribers who have complained about spam from your organization. At this time, the exclusion is only allowed via DISTRIBUTE; this does not include administrative messages.

Important: The SPAM_FEEDBACK_EXCLUDE variables are for advanced users and are only available when the LISTSERV web interface is accessed in **Expert Mode**.

To define in the Web Interface, expand **Server Administration**, then select **Site Configuration**. Click the **Anti-Spam** button, and then define the following:

- **SPAM_FEEDBACK_EXCLUDE_LIST** – This variable tells LISTSERV to exclude all of the subscribers of the Spam Feedback Loop list in question from future bulk mailings, even if subscribed to a list or targeted for a mailing via LISTSERV Maestro or home-made DISTRIBUTE job or any other channel. This does not blackhole administrative messages; therefore, the users can still receive confirmation cookies necessary to execute commands and so on.

  To enable this setting, enter the name of the Spam Feedback Loop list as the value of this configuration variable. For example, for a list named Spam-Feedback:

  ```
  SPAM_FEEDBACK_EXCLUDE_LIST=SPAM-FEEDBACK
  ```

  By default, SPAM_FEEDBACK_EXCLUDE_LIST is not set (that is, no Spam Feedback Loop lists are defined for subscriber exclusion).

Note: L-Soft does not recommend defining multiple Spam Feedback Loop lists. For maximum performance, this variable is only intended for use with one list. Then, you can ADD anyone that you want to exclude to this list. (Make sure to set Default-Options= for this list to include the NOMAIL setting.)

- **SPAM_FEEDBACK_EXCLUDE_ALLOW** – This variable enables LISTSERV instances on this CIDR range to query the exclusion list managed by this LISTSERV instance. Use this setting in the instance running the Spam Feedback Loop, and in this instance only. For example:

  ```
  SPAM_FEEDBACK_EXCLUDE_ALLOW=212.247.25.0/25
  ```

  By default, SPAM_FEEDBACK_EXCLUDE_ALLOW is not set (that is, no other LISTSERV instances are allowed to query this LISTSERV instance's exclusion list).

  SPAM_FEEDBACK_EXCLUDE_SERVER – This variable is intended to be set in any LISTSERV instance(s) which is/are sharing the feedback loop. **It should NOT be set**

on the LISTSERV instance where the feedback loop is running. The value is the hostname (plus optional TCPGUI port number) of the instance that operates the Spam Feedback Loops. The optional port number must match the TCPGUI port for the feedback loop instance if the default value for TCPGUI_PORT on that instance has been changed. Examples:

If TCPGUI_PORT is *not* set, or is set to 2306 (the default):

```
SPAM_FEEDBACK_EXCLUDE_SERVER=LISTSERV.XYZ.COM
```

If TCPGUI_PORT is set to any non-default value, e.g., 23006:

```
SPAM_FEEDBACK_EXCLUDE_SERVER=LISTSERV.XYZ.COM:23006
```

By default, SPAM_FEEDBACK_EXCLUDE_SERVER is not set.

6.4 Testing

L-Soft strongly advises a testing period during which individual spam reports are manually forwarded to the list. It is important to forward the messages in such a way that they are received by the list with the same special MIME structure that identifies spam reports. The sender is immaterial as long as it is authorized to post to the list.

By default, when a spam complaint is received through a Feedback Loop, LISTSERV will:

1. Immediately delete the user from all local LISTSERV lists, on the first spam report. This does not include Dynamic Query Lists, which are managed outside LISTSERV.
2. Not send any notification to the user.
3. Add the user to the spam feedback list with the NOMAIL option to keep a record of who has lodged spam complaints. The subscription date is set to the date of the first complaint and the last activity date is set to the date of the last complaint.
4. Log a SPAM_COMPLAINT record to the list's change log, if one has been defined. You can use this information to remove the user from Dynamic Query Lists, if you have any such lists that may contain users.

This ensures that no further mail will be sent to the user and minimizes the risk for further complaints.

You can verify the operation of the feature on the LISTSERV log. Once you are comfortable with your setup, you can redirect all of the ISPs spam reports to the list so that they are processed without manual forwarding.

6.5 Challenges and advice

You can expect a few challenges once you have activated automatic processing. Here is some advice on how to deal with the most common issues.

- Be patient. Initially, most incoming spam reports will refer to messages created before the SPAM_FEEDBACK_PROBE= setting was defined in the LISTSERV configuration. It may take a day or two for automatic processing to start kicking in. You will continue to see unprocessed messages for at least one month after enabling the feature. There is nothing you can do about it, other than wait for these old messages to have "expired" from the pipeline.

- Many false positives. L-Soft's experience has been that many spam reports are sent erroneously. We have seen list owners with certain ISP accounts report their own lists as spam! It is exceedingly easy for a user to generate a spam report without even realizing it; for instance, exiting the mail interface in a hurry will silently generate a spam report for any unread messages automatically sent to the junk mail folder by some ISP filters.
- Complaints from certain ISP users, such as AOL. For instance, the heavy-handed "one strike and you're out" rule is not popular with AOL users who have erroneously generated a spam report. If anything, the lack of notification is even less popular. AOL users may get upset and blame your organization. But, in reality, you do not have a choice in the matter. The terms of AOL's whitelist agreement require "one strike and you're out" and forbid notification. You have a choice between being heavy-handed and not being on the whitelist, which in practice would promptly cause all of your mail to AOL to be blocked and all AOL users to be deleted. Be prepared to write a FAQ for your list on this topic.
- Complaints from list owners. It is convenient for most list owners to refer to an organization-wide policy that in turn refers to the legal terms of an ISP's whitelist agreement, which most reasonable people will understand must be followed. But some list owners will complain that they are unable to re-add users that have been served off. Be prepared to make a policy on serving these users back. It is up to you as whitelist agreements are silent on that issue.
- Spam reports from reinstated users. You will find that many reinstated users continue to issue spam reports because they simply do not understand what causes these spam reports to be sent. Until they know what they must do differently, they will keep generating spam reports that hurt your deliverability – and blaming LISTSERV and you for the false positives. This is why it is important to keep archives of all your spam reports. Once the user is convinced that you did receive a new spam report, the problem will be transferred to ISP's help desk, which in most cases is able to help the customer disable these reports.
- Lax or tough? One of the decisions you will have to make is whether to reinstate all users simply for the asking, or only if they petition you in writing and agree not to send any further reports, or never. It is important to understand that every spam report is a petition by a customer to terminate your organization's access to that particular ISP. Of course, it takes a lot of reports for this to happen, but there is absolutely no difference between voluntary and involuntary spam reports. The risk, if you keep reinstating anyone who asks, is that you keep receiving spam reports from users who are unable to learn how to turn them off, and that your deliverability will suffer. By trying to help a handful of users who share their account or are just not very good with computers, you may end up hurting thousands of users who have meticulously followed instructions to turn off spam reports for your organization.

6.6 Future direction

L-Soft would like to receive customer input about the reinstatement issue.

At this point, L-Soft will not provide configuration options to loosen the “one shot and you’re out” rule, because it is a legal requirement of the AOL whitelist agreement. Usage of the automatic feedback loop processing feature is entirely optional; nobody is locked in to the new automatic processing feature, but it can only be used in compliance with AOL’s terms. L-Soft may revisit this position if AOL changes its whitelist terms. L-Soft believes that the way to move forward on this issue is to encourage AOL users to complain to AOL about the rule, rather than try and configure LISTSERV to ignore it. Like any business, AOL is likely to listen to its customers. AOL has historically been willing to listen to issues such as this one, at least once it saw that there was widespread concern and not just a handful of vocal complaints. Give AOL a chance to set this straight.

6.7 Known issues and restrictions

The following known issues and restrictions exist:

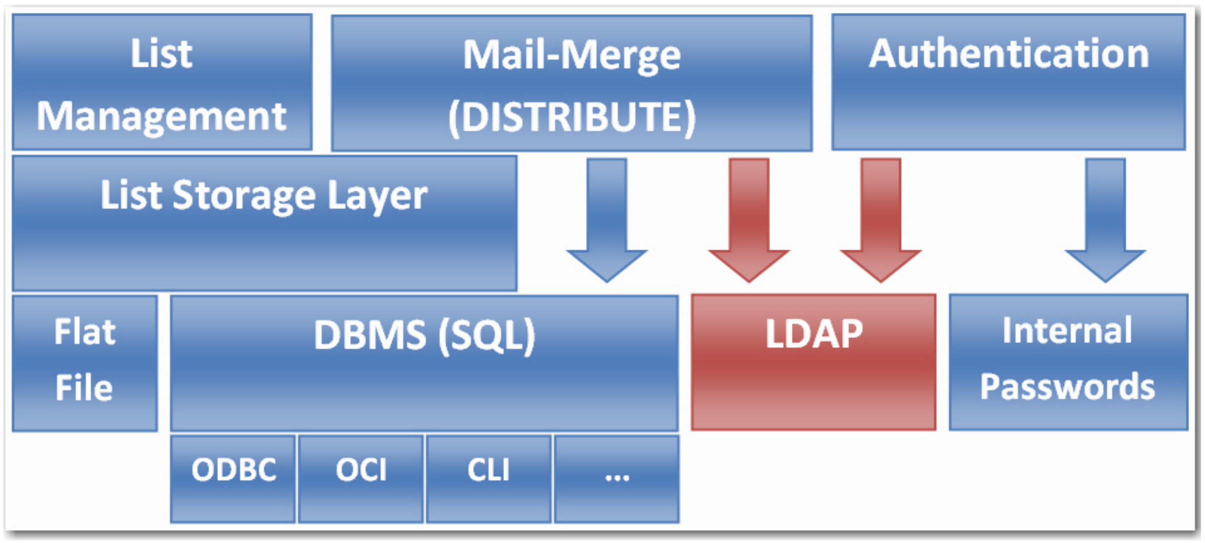
- Preamble inserted in original message. The little preamble LISTSERV inserts on spam reports it has successfully processed is often also inserted inside the original message. This is an artifact of the current implementation that may be removed in the future.

Section 7 LISTSERV and LDAP

This functionality is not available in LISTSERV Lite.

LISTSERV Classic and Classic HPO can interface to LDAP servers to authenticate user logins, to insert LDAP attributes in mail-merge distributions, as well as to implement Dynamic Queries (see Section 8 Dynamic Queries for more information).

The following diagram shows the LISTSERV LDAP architecture, in relation to other components:



Note: For clarity, Dynamic Query functions have been omitted from the diagram, but they also interface with the LDAP functionality.

The LDAP interface is at the same level as the DBMS interface – not at the level of the vendor-specific SQL drivers. Quite simply, LDAP servers do not "speak" SQL. To support LDAP, we had to teach the mail-merge and authentication modules to "speak" LDAP. This is why there is a new syntax for every LDAP-related function.

At this point, LISTSERV only queries LDAP directories. It will never try to make any changes, so it should not be given write access to the directory.

Contents:

7.1 Configuring LDAP in LISTSERV

The first step in using LDAP with LISTSERV is to add one or more LDAP servers in the LISTSERV site configuration. This can be done via the LISTSERV web administration interface (the preferred method), or alternately by adding the entries manually to SITE.CFG or 'go'.

Each LDAP server is given a nickname in the LISTSERV configuration, similarly to DBMS data sources. You can also configure one unnamed LDAP server, again like with DBMS data sources, but it is probably less confusing to assign a nickname to every LDAP server.

Three configuration variables must be defined for every LDAP server:

- `LDAP_SERVER_`*`nickname`*`=`*`hostname`*`[:`*`port`*`]`

  The hostname and optional port of the LDAP server. The exact format depends on your operating system and LDAP library; LISTSERV passes this string to the LDAP library as it is. On unix, SSL encryption is requested by prepending 'ldaps://' to the hostname. On Windows, the 'ldaps://' prefix is not available, but setting the port to 636 automatically requests SSL.

- `LDAP_UID_`*`nickname`*`=`*`userid`*

- `LDAP_AUTH_`*`nickname`*`=`*`password`*

  The userid and password that LISTSERV should use in order to login to the LDAP server. The exact format of the userid depends on your LDAP server. LISTSERV does not attempt to parse or reformat these variables. If the password is the empty string, most LDAP servers will perform an anonymous login. If both userid and password are the empty string, LISTSERV will attempt a default login, as defined by the LDAP library for your operating system. Under Windows, LISTSERV will be logged in with its current domain credentials (assuming it is connecting to an Active Directory server), and this usually provides sufficient access – try it before configuring a userid and password.

If the LDAP server is to be used to authenticate LISTSERV users, the following variables must also be defined:

- `LDAP_PW_BASE_`*`nickname`*`=`*`DN`*

  The 'distinguished name' that should be the 'base' for searches when LISTSERV looks for a user account (see below for an explanation of the authentication process). This can be used to restrict LISTSERV access to a particular organizational unit within the enterprise. If omitted, LISTSERV tries to guess the DN that will admit any Active Directory Windows account, but this is a difficult guess to make, and of course you may not even be connecting to Active Directory.

- `LDAP_PW_FILTER_`*`nickname`*`=`*`filter`*

  The LDAP 'filter' that should be used when looking up user accounts (if this filter returns at least one entry, LISTSERV allows the user to try and log in; otherwise, the login is rejected, even if the user would otherwise be able to log in to the LDAP server with the supplied credentials). Any occurrences of '%s' are replaced with the user's full e-mail address, while '%u' expands to just the userid and '%h' expands to the hostname. If omitted, LISTSERV uses a filter that is suitable for most Active Directory installations.

In addition, the following optional variables can be defined:

- `LDAP_DEFAULT_EMAIL_`*`nickname`*`=`*`attribute`*

  The name of the attribute that ordinarily specifies a user's e-mail address in this directory. This is used as a default value in searches and can be overridden. If omitted, it defaults to 'mail' (suitable for Active Directory).

- `LDAP_DEFAULT_NAME_`*`nickname`*`=`*`attribute`*

  The name of the attribute that ordinarily contains the user's full name. Defaults to 'name'.

7.2 Using LDAP for Mail-Merge

Because of its complex, machine-friendly syntax, LDAP is primarily suited for scripting. While it is relatively easy for a programmer to write a script that sends a weekly notice to every member of a particular department, it is not realistic to expect ordinary list owners or end-users to understand the intricacies of LDAP and devise working search filters. For instance, to select all users in an Exchange database, one would have to use the following filter:

```
(&(!(Alias=$null))(|(&(ObjectCategory=person)(ObjectClass=user)
(Database=$null)(ServerLegacyDN=$null))(&(ObjectCategory=person)
(ObjectClass=user)(!(Database=$null))(!
(ServerLegacyDN=$null)))))))
```

L-Soft expects that LDAP-based distributions will be created by customer-developed scripts – either intranet web scripts or traditional 'cron' jobs or scheduled tasks. At this point, there are no plans to provide a web interface page into which raw LDAP search filters could be entered.

To create an LDAP-based distribution, a script uses the DISTRIBUTE command and specifies an LDAP keyword as follows:

```
DISTRIBUTE … LDAP=YES(SERVER=nickname,E-
MAIL=attribute,PARTS=attribute)
```

The syntax of this keyword is essentially the same as for SQL-based distributions ("DBMS="):

- **`SERVER=`*`nickname`***

  Identifies the LDAP server to be queried. If omitted, the default (unnamed) LDAP server is used.

- **`E-MAIL=`*`attribute`***

  Identifies the name of the directory attribute containing the recipient's e-mail address. If omitted, the value of `LDAP_DEFAULT_EMAIL_nickname` is used.

- **`PARTS=`*`attribute`***

  The name of an optional directory attribute containing a list of message parts that the recipient subscribes to. Although this mail-merge feature is unlikely to be used with

LDAP, it is available if desired.

Note: The E-MAIL and (if enabled) PARTS attributes must be specified or the distribution will fail.

Similarly to SQL-based distributions, the ‘TO’ DD contains a list of LDAP search statements, rather than a list of actual recipients. Each line in the ‘TO’ DD can be one of the following statements:

- **BASE** ***DN***

  The ‘distinguished name’ of the ‘base’ of the LDAP search. Mandatory.

- **FILTER** ***search_filter***

  The LDAP search filter for the search. Mandatory.

- **ATTRS** ***attr1*** **[*attr2* […]]**

  A list of directory attributes of interest (used in the mail-merge). If omitted, all directory attributes are made available. Attribute names are not case-sensitive. The main purpose of this statement is to improve search performance if there are many irrelevant attributes in the directory.

- **SCOPE BASE|ONELEVEL|SUBTREE**

  Optionally changes the scope of the search from the default (SUBTREE).

- **SEARCH**

  Starts the search. This command allows multiple LDAP searches to be performed in the same distribution. If there is only one search, this command is optional – LISTSERV automatically starts the search when it reaches the end of the ‘TO’ DD.

  For instance, this search will select all Windows users in the EXAMPLE.COM domain with a valid e-mail address:

```
BASE CN=Users,DC=EXAMPLE,DC=COM
FILTER (&(objectcategory=person)(objectclass=user))
ATTRS Name Mail Phone
SEARCH
```

7.3 Using LDAP for Authentication

LISTSERV can be configured to use one or several LDAP servers for authentication (user login). You can choose to allow users without an LDAP account to log in with an internal LISTSERV password, or to restrict access to users with an LDAP account.

LDAP authentication is enabled by defining the following configuration variables:

- **LDAP_PW_SERVERS=*nickname1*** **[*nickname2* […]]**

The list of LDAP servers to be queried (in the specified order) for user accounts. Be sure to enter server nicknames, not hostnames.

- **`LDAP_PW_ONLY=0 or 1`** (default: 0)

  If set to 1, only users with an LDAP account are allowed to log in to LISTSERV; other users will only be able to access LISTSERV anonymously.

Warning: Make sure to test your LDAP settings before enabling this option, or you will not be able to undo it from the web interface! Enabling this option on a server that previously had external users is likely to result in significant confusion for the external users, whose passwords will no longer work.

- **`LDAP_PW_REQUIRE_SSL=0 or 1`** (default: 1)

  Whether or not LISTSERV should accept LDAP passwords transmitted to the web interface in plain text. By default, LISTSERV will only attempt to verify passwords transmitted over SSL.

Note: This option does not control LISTSERV's own use of SSL when communicating with the LDAP server. See the LDAP_SERVER_nickname variable.

- **`SIGNUP_REQUIRE_SSL=0 or 1`** (default: 0)

  Similar to the above, but affects all LISTSERV passwords, whether LDAP or internal. Can be used without enabling LDAP authentication.

7.3.1 The LDAP Authentication Process

When LDAP is enabled, LISTSERV goes through the following steps to log in a user:

1. The servers listed in LDAP_PW_SERVERS are examined in turn, in the order in which they were listed. For each server, LISTSERV executes the search configured with the LDAP_PW_BASE_nickname and LDAP_PW_FILTER_nickname variables. LISTSERV stops at the first successful search, or when there are no more LDAP servers to query.
2. If none of the searches were successful (no LDAP account exists for this user), LISTSERV:
   a) Rejects the login if LDAP_PW_ONLY=1.
   b) Switches to internal (non-LDAP) login if LDAP_PW_ONLY=0. The login will be validated against the user's internal LISTSERV password, if any, or the user will be prompted to create a LISTSERV password.
3. If an LDAP account was found for this user, LISTSERV:
   a) Rejects the login if LDAP_PW_REQUIRE_SSL=1 and the login request did not come over an SSL session. In this case, LISTSERV does not even try to verify the password.
   b) Verifies the password against the LDAP server where the account was found, and accepts or rejects the login as appropriate.

7.3.2 A Note on the “Require SSL” Option

The purpose of the “require SSL” option is to prevent ordinary, non-malicious users from jeopardizing their login credentials for their personal convenience, for instance by typing clear-text passwords in e-mail requests because it is faster than waiting for a confirmation ‘cookie’ at the particular Internet café where they are reading their mail. The “require SSL” option effectively disables these login attempts and forces users to log in using the web interface and SSL.

As LISTSERV does not directly process SSL sessions, it has no first-hand knowledge as to whether SSL was used to encrypt the login session or not. It is the web server that handles the SSL session with the user’s browser, notifies the LISTSERV web interface that SSL was used, and the web interface script in turn notifies LISTSERV that the password was not sent in clear text. LISTSERV has no way to verify this representation or guarantee that SSL was in fact used to transmit the password. This being said, there is no advantage for a malicious user in logging in to LISTSERV with his own credentials over an unencrypted connection. The malicious user’s interest is for other, non-malicious users to expose their passwords by sending them in clear text, so that the malicious user may gather them.

7.3.3 Using the Optional LDAP_PW_Bind Configuration Variable

The optional LDAP_PW_BIND site configuration variable is now available in all modern LISTSERV builds, i.e., post-16.0. This variable contains the string to format and use when logging on a user. The default is "%n", which works "out of the box" with Active Directory and, in most cases, with OpenLDAP. For other LDAP implementations, or if a different bind string is required for your local installation, it can be defined in this setting and may use the %u/%h/%s escapes.

Examples (for use with the "default" LDAP server):

Unix:	**`LDAP_PW_BIND="%n"`**
	`export LDAP_PW_BIND`

Windows:	**`LDAP_PW_BIND=%n`**

Additionally, this variable, like most of the other LDAP-related variables, can also take an optional server-name attribute, e.g.,

Unix:	**`LDAP_PW_BIND_MYSERVER="%n"`**
	`export LDAP_PW_BIND_MYSERVER`

Windows:	**`LDAP_PW_BIND_MYSERVER=%n`**

The above example assumes that LDAP_SERVER is set to include MYSERVER.

Important: The default value for LDAP_PW_BIND is "%n".

7.4 Using LDAP over SSL for the Solaris Operating System

Setting up LISTSERV to authenticate via LDAP with SSL can be challenging. There are several scenarios that are dependent primarily on the version of Solaris you are running.

Originally, Solaris 8 was supported with a native Solaris 8 kit, but L-Soft no longer produces a Solaris 8 kit.

There are two LISTSERV kits involved:

- Solaris 10 and later (SPARC)
- Solaris 10 and later (x64)

7.4.1 SCENARIO 1: Using the Solaris 8 LISTSERV Kit on any Version of Solaris

As noted, this scenario is obsolete as L-Soft no longer produces a native Solaris 8 kit.

7.4.2 SCENARIO 2: Solaris 8 with the Solaris 10 LISTSERV kit

This scenario assumes and REQUIRES that you are running Solaris 8 with Sun's LDAP library.

If you are running OpenLDAP on Solaris 8, you MUST use the Solaris 8 kit, and the Instructions for OpenLDAP.

In this scenario, you must first take the following steps:

1. Confirm that you have the 'certutil' utility on your system AND that it operates in the old cert7.db format. This utility does not come pre-installed and is unlikely to be in root's default path. This would be something you or a colleague installed manually at some point, presumably in /usr/local/bin, but it could be elsewhere.
2. If you do not have or cannot find certutil, download the "Directory Server Resource Kit 5.2.1" from Sun and install it. This contains a suitable certutil utility.

At this point, you may continue with the Generic Solaris Instructions found below.

7.4.3 SCENARIO 3: Solaris 9 with the Solaris 10 LISTSERV kit

This scenario will generally be unworkable because of certificate incompatibilities introduced by Sun in Solaris 9. Although Sun's LDAP for Solaris 9 requires certificates to be formatted in the older cert7.db format, the certutil utility that ships with Solaris 9 creates cert8.db format files.

Important: We strongly recommend that Solaris 9 users wishing to use LDAP with LISTSERV contact Sun directly for support on this issue. L-Soft is unable to provide support for this issue as it is a problem that only Sun can resolve.

If you have a Solaris 8 system available, you can use it to follow the Generic Solaris Instructions, below, and then FTP the resulting certificate files to your Solaris 9 system.

If you do not have a Solaris 8 system available, you could download and install the "Directory Server Resource Kit 5.2.1" from Sun, and install it in a temporary directory.

Important: If you install the Directory Server Resource Kit 5.2.1 under Solaris 9, then you **must NOT overwrite** anything that came with your Solaris 9 system.

Alternately, and perhaps preferably, Sun may have a conversion program or another way to help you with this incompatibility issue. Please contact Sun directly for more information.

7.4.4 SCENARIO 4: Solaris 10 with the Solaris 10 LISTERV kit

Solaris 10 does understand the cert8.db files created by its version of certutil. You will find certutil pre-installed in /usr/sfw/bin, so just follow the Generic Solaris Instructions. Note that /usr/sfw/bin is not in root's path.

7.4.5 Generic Solaris Instructions

1. Create the NSS DB only if not already done:

   ```
   # ls /var/ldap/*.db
   ```

   STOP NOW AND GO TO STEP 2 if there are already files in there.

   ```
   # <path-to-certutil>/certutil –N –d /var/ldap
   # ls /var/ldap/*.db
   # chmod 644 /var/ldap/*.db
   ```

   Do not provide a password. Just type RETURN twice. Verify that you created the right 'flavor' of database: cert7 for Solaris 8 and 9, cert8 for Solaris 10. Otherwise you need to start again with the right certutil.

2. Obtain the public SSL certificate for the LDAP server you want to connect to. This example assumes you used the standard PEM exchange format (base64-encoded ASCII), there are other formats that may require additional or different switches. We will assume that you have saved the certificate in a file called cert.txt.

3. Register the certificate as follows:

   ```
   # <path-to-certutil>/certutil –A –n "nickname" –d /var/ldap –a
   –t CT –i cert.txt
   ```

   The nickname is just a dummy name for your convenience in remembering what this certificate is for.

   That's it! You are now set up for LDAP over SSL to that particular server. Remember to specify port 636 in LISTSERV's configuration, for instance:

   ```
   LDAP_SERVER="ldap.example.com:636"
   ```

7.4.6 Instructions for OpenLDAP

This section contains the OpenLDAP instructions, which are the same regardless of the brand of unix you are running.

1. Obtain the public SSL certificate for the LDAP server you want to connect to, in PEM format (ASCII). If you receive the file in a different format, it is probably easier to ask the LDAP server administrator for a PEM file in ASCII than to try and convert it yourself. If you must convert the file, there are too many possible scenarios to cover here, but check the man pages for the openssl command.
2. Save this file in the home directory of the 'listserv' user as 'SSL.pem'. If working as root, make sure the 'listserv' user has at least read access.
3. Do:

```
$ cat > ~listserv/.ldaprc
TLS_CACERT /home/listserv/SSL.pem
<Ctrl-D>
```

   Substitute the path to the home directory for the 'listserv' user. You may or may not be able to use '~listserv', but an explicit path will always work. As before, if working as root, make sure the 'listserv' user has at least read access.

   That's it! Remember to specify ldaps access, for instance:

```
LDAP_SERVER="ldaps://ldap.example.com"
```

7.5 Dynamic Queries

Although Dynamic Queries are primarily based on the LDAP interface, they are described separately in Section 8 Dynamic Queries, since they support both LDAP and DBMS data stores.

7.6 Known issues and restrictions

The following known issues and restrictions exist:

- OpenLDAP library required to re-link on unix. Customers wishing to re-link 'lsv' on unix will have to install the OpenLDAP library, even if they do not want to use LDAP.
- Static library support not tested on all supported unix brands. Our goal is for supported unix builds with LDAP functionality to work on target systems that do not have the dynamic LDAP library, but we have not tested this on every system.

Section 8 Dynamic Queries

LISTSERV can execute on-demand Dynamic Queries against either LDAP or traditional DBMS servers, and use the results for access control and mail delivery. For instance, an LDAP query against an Active Directory server could be used to grant list owner privileges to all members of a particular Windows Security Group. A DBMS query could be used to combine employee rosters for two departments and send a single copy of an announcement to each unique employee e-mail address. Support for other types of data sources can also be provided by writing an exit (short script) to query the custom data source and return the results to LISTSERV.

Note: This section will not cover basic DBMS or LDAP configuration. Please see Section 7 LISTSERV and LDAP for information on how to configure LDAP servers in LISTSERV.

Contents:

8.1 Dynamic Query Concepts

A Dynamic Query is a pre-configured search against either an LDAP directory or a DBMS that returns, at a minimum, a series of e-mail addresses, and may also return additional information, such as the recipient’s full name or phone number. Queries can be used either for access control purposes (for instance, to grant all members of the CORP_HR Windows Security Group permission to post to the HR-NEWS list), or to provide additional recipients for a posting to a mailing list.

All dynamic queries are defined ahead of time by the LISTSERV administrator. List owners cannot create their own queries, but they can make use of pre-defined queries and supply parameters to these queries (see below for more information on security restrictions). The LISTSERV administrator determines how these parameters are used. For instance, the administrator could define a query that matches all members of a particular Windows Security Group, specified as a parameter. List owners could then provide the group name when they refer to the query. Queries whose definition contains no parameters cannot be altered by list owners.

Dynamic queries are executed when, and only when, LISTSERV needs access to the query results to complete a task. A dormant query consumes no resources. Note that LISTSERV does execute queries when verifying the syntax of list header changes. LISTSERV rejects any invalid queries and issues a warning when the query is valid, but returns no data, as this usually means that a parameter was spelled incorrectly.

Queries can be inserted in most list header keywords used for access control or to enumerate e-mail addresses. Example:

```
* Owner= J.Smith@example.com (Joe Smith)
* Owner= Query(DEPT,HR) (HR Department)
* Review= Private,Query(DEPT,HR),*@example.com
```

Queries can also be used to create dynamic sub-lists, which are covered in Section 8.5 Creating Dynamic Sub-Lists.

8.2 Defining Dynamic Queries

Each dynamic query must be given a nickname, which may not contain white-space or non-printable characters. List owners know the queries by their nicknames, and have no access to their actual definitions. A query is defined by a LISTSERV configuration variable called:

```
DYN_QUERY_nickname=kwd1 *kwd2 *…++
```

On unix, the nickname must be entered in upper case, and shell escape and quoting conventions must be respected. Keywords have the following syntax:

keyword=token

keyword='string that can contain white space'

keyword="string that can contain white space"

Quotes within a quoted string must be doubled. The following keywords are recognized for both LDAP and DBMS queries:

- **TYPE=LDAP | DBMS | EXIT** (default: **LDAP**)

  Whether the query uses LDAP, DBMS or a customer-supplied exit (script).

- **SERVER=*server_name*** (default: **DEFAULT**)

  The nickname of the LDAP or DBMS server against which to run the search. If omitted, the search is run against the default (unnamed) LDAP or DBMS server.

- **E-MAIL=*attr_name*** (default: see explanation)

  EMAIL=*attr_name* (default: see explanation)

  NAME=*attr*_name (default: see explanation)

  The name of the LDAP attribute or SQL column that contains the e-mail address or full name of the recipient, respectively. For LDAP, a default value can be specified in the LDAP server configuration (LDAP_DEFAULT_EMAIL_server and

LDAP_DEFAULT_NAME_server). There is no default value for DBMS sources. The e-mail attribute or column name must be supplied, either explicitly or through LDAP_DEFAULT_EMAIL; the name is optional.

The following keywords are recognized for LDAP queries only:

- **SCOPE=*scope*** (default: **SUBTREE**)

  The scope of the LDAP query.

- **BASE=DN**

  The 'distinguished name' of the search base. Mandatory; may refer to parameters.

- **FILTER=*filter***

  The search 'filter'. Mandatory; may refer to parameters.

- **ATTRS=*attr1 *attr2 *…++***

  Any additional attributes (besides the name and e-mail address) that should be returned by the query when it is used in the context of processing a posting to a mailing list. These additional attributes become available as mail-merge fields. This keyword is ignored when the query is executed for other purposes (access control, etc.)

These keywords are recognized for DBMS queries only:

- **DBMS=ODBC | OCI | CLI | UODBC** (default available if only one driver is configured)

  The name of the DBMS driver to use for the search. As with other DBMS functions, a default is provided if only one DBMS driver has been configured. Otherwise, this attribute is mandatory.

- **SEARCH=*SELECT_statement***

  The SELECT statement to execute. It must return at least the e-mail column, and must also return the name column if one was specified. It may return additional columns, which are made available as mail-merge fields when the query is used in the context of processing a posting to a mailing list. For optimal performance, you should list the e-mail and name columns first. This attribute is mandatory and may refer to parameters.

These keywords are recognized for TYPE=EXIT queries only:

- **EXITPARMn=arbitrary_data** (1 ≤ n ≤ 4)

  Four independent keywords are provided to pass arbitrary data or arguments to the exit. These keywords are made available to the exit script as it is called and may refer to parameters.

Important: For security reasons, keywords may not refer to parameters unless indicated otherwise. LISTSERV inserts all parameters as escaped text constants, to prevent them from changing the semantics of the search.

Parameters are specified as %1 for the first parameter, %2 for the second one, and so forth. The percent sign must be doubled.

8.3 Query Examples

These examples have been wrapped for legibility, but they must be entered as a single line in the LISTSERV configuration. Parameters are shown in red for emphasis only.

- Query matching Active Directory users in a particular organizational unit at EXAMPLE.COM:

```
SERVER=nickname BASE=OU=%1,DC=EXAMPLE,DC=COM
FILTER= '(&(objectcategory=person)(objectclass=user))'
```

- Query matching members of a Windows Security Group at EXAMPLE.COM:

```
SERVER=nickname BASE=CN=Users,DC=EXAMPLE,DC=COM
FILTER= '(&(objectcategory=person)(objectclass=user)
(memberof=CN=%1,CN=Users,DC=EXAMPLE,DC=COM))'
```

- Advanced example: query matching recipients from a DBMS table whose e-mail address is at a particular hostname.

```
TYPE=DBMS SERVER=nickname SEARCH='SELECT CUST_EMAIL,
CUST_NAME FROM SOMETABLE WHERE CUST_EMAIL LIKE CONCAT(''%%
@'',%1)' E-MAIL=CUST_EMAIL
```

Because LISTSERV inserts all parameters as escaped string constants, it is not possible to form the pattern '%@[parameter #1]' by entering '%%@%1' in the SELECT statement. Instead, the run-time CONCAT function is used. The single quotes surrounding the first parameter, '%%@', must be doubled because the entire SELECT statement is inside a quoted string. LISTSERV will supply the quotes around the %1 parameter.

Note: All of the quote marks used in these examples are **true single quotes** (ASCII hex 27), also called apostrophes. **Do not use smart quotes or double quotes in these queries.**

8.4 Using Dynamic Queries for Access Control

Once defined by the administrator, a dynamic query can be entered in most list header keywords used for access control or to enumerate e-mail addresses (see below for more information on security restrictions). Queries can be inserted wherever you would be allowed to enter an e-mail address, and have the following format:

```
Query(nickname*,parameter #1*,…++)
```

The nickname is mandatory and must refer to an existing, pre-defined dynamic query. Parameters are optional and must be enclosed in true single quotes (ASCII hex 27) if they contain white space or commas. Do not use smart quotes or double quotes. Smart quotes are not recognized by LISTSERV, and double quotes have a special meaning in list header keywords and are generally removed from the keyword.

Tips: *Remember that dynamic queries can fail at run time*, for instance because the LDAP server is not available or because the DBMS login credentials have expired. A dynamic query is not a suitable method for defining, for instance, the primary editor of a mailing list, but it is perfect for adding large groups of people as secondary editors.

Try to put the dynamic query last in access control keywords. LISTSERV will only execute the query if none of the prior access control patterns were a match. It is always more efficient for LISTSERV to match a pattern like '*@example.com' than to run a query against a DBMS or LDAP server.

8.5 Creating Dynamic Sub-Lists

Dynamic queries can also be used as "virtual sub-lists" to augment the membership of a list (see below for more information on security restrictions). To do so, simply insert the query in the "Sub-Lists=" keyword, or create a "Sub-Lists=" keyword with just the query:

```
* Sub-Lists= PROJECT_A,Query(DEPT,HR),PROJECT_B
```

Dynamic sub-lists work exactly like regular sub-lists, except that all recipients have the default list options. As always with sub-lists, duplicate recipients are eliminated.

8.5.1 Why not create plain dynamic lists instead?

It is not possible to create a regular list whose membership is defined by a dynamic query. Dynamic membership is only available by creating a super-list with a dynamic query as a sub-list, and the reasons for this design choice may not be obvious at first. In a nutshell, a plain dynamic list implementation would have led to reduced functionality for about the same learning curve. It may seem strange at first to define dynamic membership as a sub-list, but the only practical difference is the name of the list header keyword that carries the query and its parameters.

There are two main advantages to having the query in a sub-list rather than in the list itself. First, all the value-added LISTSERV features remain available – or rather, they can be made available if desired. A recipient from the dynamic query can subscribe to the list, change subscription options, be promoted to list editor, turn off mail receipt, and so forth. This would not be possible if the list membership were "frozen" and managed entirely outside of LISTSERV – for instance, in Active Directory. There is no function in Active Directory to switch to DIGEST mode or activate the NOMAIL option. This degree of freedom may not be desirable for internal lists and can be disabled easily with a list exit, or by preventing subscriptions to the super-list, but it can also be enabled. This provides more flexibility than an implementation where there is simply no way for a dynamic recipient to opt out.

Second, LISTSERV only explores sub-lists when it needs to, i.e. when a message is posted to the list, but explores and grooms top-level lists on a regular basis. Having the dynamic query as a sub-list minimizes the number of potentially resource-intensive queries. LISTSERV also knows that each sub-list has its own, independent list management process and will generally "do the right thing" with respect to managing, grooming and advertising the super-list vs. the sub-list. In a nutshell, LISTSERV will leave the sub-list recipients alone, as it always has when expanding sub-lists.

8.6 Creating a Dynamic Query Exit to Access Custom Data Sources

LISTSERV supports a dynamic query exit to provide support for custom data sources other than LDAP and SQL databases, or to query LDAP or SQL data sources in a specific manner not otherwise supported by LISTSERV. This functionality is implemented as a standard LISTSERV exit with the following operational requirements:

- **Registration:** The name of the dynamic query exit is registered in the DYN_QUERY_EXIT configuration variable.
- **Input Parameters:** LISTSERV converts the EXITPARM1 through EXITPARM4 keywords to plain-text counted format (see below) and concatenates them in order from 1 through 4 to form the exit input parameter. See sample decoding code below.

Warning: These keywords can contain arbitrary data. The exit script is responsible for escaping or removing harmful characters as required by its particular environment and programming language. These parameters should not be inserted 'as is' in shell command strings!

- **Output requirements, successful completion:** If it completes successfully, the exit must return the standard success return code for the operating system on which LISTSERV is running, and create a result file (see below).
- **Output requirements, unsuccessful completion:** Any error return code causes LISTSERV to fail the dynamic query. A result file does not need to be created in this case.

8.6.1 The Result File

The exit script must store the results of the dynamic query in a plain-text result file named `exit.results`. LISTSERV deletes this file before calling the exit.

Each record in the result file must be in Plain-Text Counted Format (see below). The first record specifies a name for each of the attributes returned by the exit. This first record must be supplied even if the query returns no data. The second and following records contain attributes for each entry returned by the query, in the same order as the first record. See the Example below.

If the query succeeds, the exit must return at least an e-mail address for each entry, and it may return additional attributes. As with LDAP and DBMS queries, the name of the attribute containing the e-mail address is defined with the E-MAIL= keyword when configuring the query.

8.6.2 Plain-Text Counted Format

A string is converted to plain-text counted format by prepending the length of the string, formatted as plain-text decimal characters, and an underscore. For instance, the string

“Hello” becomes “5_Hello” and the empty string becomes “0_”. Encoded strings are concatenated with no spaces or other delimiters in between. See the Example below.

8.6.3 Example

In this example, we will use a dynamic query called SAMPLE, which has been defined as follows:

```
TYPE=EXIT E-MAIL=MAIL EXITPARM1=Employees EXITPARM2=%1
```

The query is invoked as:

```
* Sub-Lists= Query(SAMPLE,HR)
```

LISTSERV calls the exit with the following input argument:

```
9_Employees2_HR0_0_
```

The exit writes the following in the plain-text file, exit.results:

```
4_MAIL4_NAME
15_joe@example.com8_Joe User
16_jane@example.com9_Jane User
```

If the query had returned no results, the exit would have written the header record only :

```
4_MAIL4_NAME
```

The following sample REXX code (written for Windows exit conventions) illustrates the process of extracting exit parameters 1 through 4 from the exit input argument and creating a hardcoded query result:

```
   /* Extract EXITPARM1..4 */
   Parse arg data
   ep1 = Parm()
   ep2 = Parm()
   ep3 = Parm()
   ep4 = Parm()

   /* Create hardcoded result set to illustrate encoding
   process */
   ofile = 'exit.results'
   Call Lineout ofile,N('MAIL')N('NAME')N('PHONE')
   Call Lineout ofile,N('mike@example.com')N('Mike Smith')
   N('555-125-9182')
   Call Lineout ofile,N('marie@example.com')N('Marie
   Jourdain')N('N/A')
   /* Exit with SUCCESS return code (OS-dependent) */
   Exit 0

N:
   /* ENCODE to plain-text counted format */
   Procedure
```

```
      Parse arg line
      Return Length(line)'_'line

   Parm:
      /* DECODE from plain-text counted format string
      ‘data’ (updated) */
      Procedure expose data
      Parse var data n'_'data
      If ^Datatype(n,'W') | n < 0 Then Exit 100 /* 100 = error
      on any OS */
      chunk = Left(data,n)
      data = Substr(data,n+1)
      Return chunk
```

8.7 Dynamic Queries and Security

Only the LISTSERV administrator can define new dynamic queries. Once a query has been defined, list owners can use them:

- For access-control purposes, without any special authorization from the LISTSERV administrator. In an access-control scenario, the result of the dynamic query is a simple yes/no/error response. The data returned by the query is processed only to the extent necessary to determine if the attempted action is authorized or not.
- For enumeration purposes, if authorized by the LISTSERV administrator (see below). The result of an enumeration is a list of e-mail addresses that LISTSERV may process further, for instance to send an administrative notice to the e-mail addresses enumerated in the “Notify=” list header keyword. In an enumeration scenario, LISTSERV only queries the e-mail and full name attributes of the directory (for DBMS queries, LISTSERV fetches all columns but discards all but the e-mail and name columns).
- To augment the membership of a mailing list, if authorized by the LISTSERV administrator (see below). In this scenario, all attributes are made available.

Note: There is no way for a list owner to obtain a list of available queries. List owners can only learn of the availability of pre-defined queries from the LISTSERV administrator. List owners are also unable to view the query definition.

To authorize the use of a dynamic query in a list header keyword, the LISTSERV administrator must update the list header the first time the query is used. From then on, the list owner can take of day-to-day list header management, and in particular, may:

- Update other keywords in the list header than the one containing the query.
- Change non-query elements in the keyword containing the query. For instance, if the “Notify=” keyword was changed to contain a query, the list owner can add additional e-mail addresses to the keyword.
- Change the relative position of the query within the keyword. For instance, change the relative position of a query in a “Sub-Lists=” keyword, where sub-lists are processed in the specified order.

- Remove the query altogether. In that case, only the LISTSERV administrator can re-add the query.

Authorization is on a per-keyword basis, and includes all parameters. A list owner authorized to use the DEPT query with parameter HR in the “Notify=” keyword can only use it in this keyword, and with the HR parameter.

8.8 Known Issues and Restrictions

The following known issues and restrictions exist:

- Dynamic queries disable the special DBMS sub-list traverse feature. When sending a posting to a super-list configured to use mail-merge for regular postings and whose sub-list recipients are hosted in a database (“DBMS= Yes” or equivalent in the list header), a special feature was activated in order to make additional DBMS columns available to the posting. Instead of processing the distribution normally, LISTSERV traversed the sub-list tree and processed the recipients one sub-list at a time, recognizing that each sub-list might have its own set of columns coming out of different tables with different names. This feature is disabled when one or more dynamic queries are used, because they cannot be tied to a physical sub-list. This does not affect mail-merge data extracted from the dynamic query itself. This is a delicate scenario in the general case. L-Soft recommends not mixing “DBMS= Yes” and dynamic queries when mail-merge data must be extracted from both data stores.

Section 9 Relayed File Distribution and DISTRIBUTE

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 section is composed of three independent sections. The first one 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 next section gives a detailed technical description of RFDRs and assumes the reader is familiar with the basic concepts of the Commands-Jobs Language Interpreter (CJLI), described in Section 2 Commands-Job Feature and CJLI Interpreter. The final section is a more practical tutorial regarding the construction of RFDR jobs.

Note: 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.

The DISTRIBUTE command requires validation, so by default it cannot be used by any random entity (for instance, to generate spam). The validation is implemented as follows:

- 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
- The DISTRIBUTE command must be validated with the issuer's personal LISTSERV password (obtained with the PW ADD command).

Prior to the release of LISTSERV 1.8d (13) in 1999, the DISTRIBUTE command did NOT require validation. It is possible to "relax" the DISTRIBUTE command back to the previous (pre-1999) behavior by setting the DIST_SECURITY site configuration variable appropriately, but this is NOT RECOMMENDED. See the documentation for the DIST_SECURITY site configuration variable for details.

Contents:

9.1 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'.

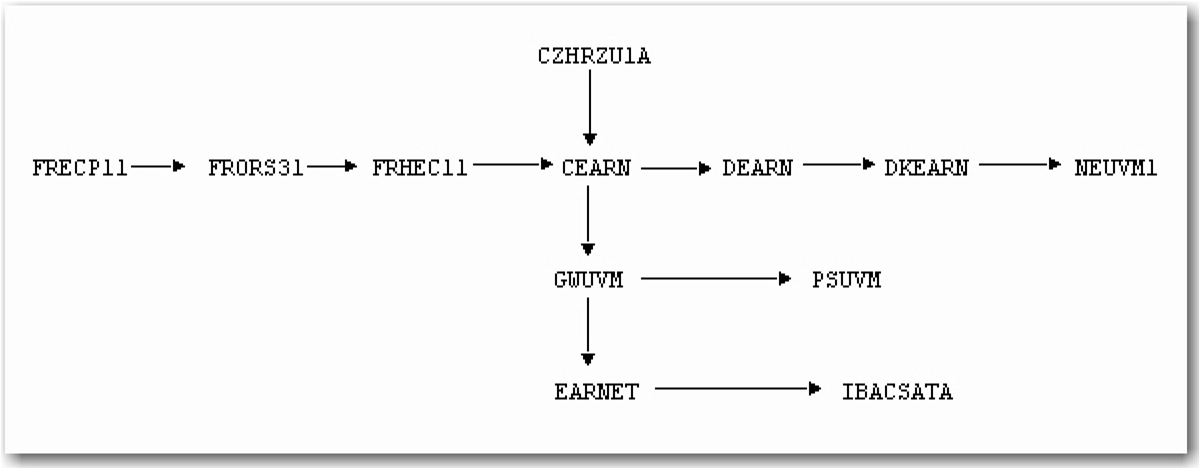
The server that initially receives the file from the sender will first distribute the file to the (possible) recipients of its local node, which does not generate any network traffic. It will then examine the list of non-local recipients and determine, for each of them, which of the L-Soft LISTSERV servers 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):


$$\begin{tikzcd}
 & & & \text{CZHRZU1A} \arrow[d] & & & \\
\text{FRECP11} \arrow[r] & \text{FRORS31} \arrow[r] & \text{FRHEC11} \arrow[r] & \text{CEARN} \arrow[r] \arrow[d] & \text{DEARN} \arrow[r] & \text{DKEARN} \arrow[r] & \text{NEUVM1} \\
 & & & \text{GWUVM} \arrow[r] \arrow[d] & \text{PSUVM} & & \\
 & & & \text{EARNET} \arrow[r] & \text{IBACSATA} & &
\end{tikzcd}$$

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.

---

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

9.2 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 Section 2 Commands-Job Feature and CJLI Interpreter if you are not familiar with the basic concepts of CJLI.

By default DISTRIBUTE may only be executed by a LISTSERV maintainer (defined in the POSTMASTER= variable in the site configuration file) or a "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, e.g. filename.filetype
- A DISTRIBUTE command with the appropriate arguments:

```
DISTribute <type> <source> <dest> <options> PW=<password>
```

Options for <type>:

- `MAIL` – Data is a mail message and recipients are defined by ‘<dest>’.
- `MAIL-MERGE` – Data is a mail-merge message.
- `POST` – (non-z/VM only) Used to send pre-approved messages to moderated (Send=Editor) mailing lists. Typically, this is used only by automated scripts and LISTSERV Maestro. For more information, see Section 9.4.2 Sending Pre-Approved Messages to Moderated Lists.
- `FILE` – (z/VM only) Data is not mail, recipients are defined by ‘<dest>’.
- `RFC822` – Data is mail, recipients are defined by the RFC822 To:/cc: fields.

<source> is:

- `DD=ddname` – Name of a Ddname holding the data to distribute (default: ‘DD=DATA’)

<dest> can be one of the following:

- `<TO> user1 <user2 <...>>` – List of recipients.
- `<TO> DD=ddname` – One recipient per line in a provided DD.

Available <options> are:

- `ACK=NONE|MAIL|MSG` – Acknowledgement level (default: ACK=NONE).
- `CANON=YES` – 'TO' list in canonical form.
- `DEBUG=YES` – Do not actually perform the distribution; returns debug path information.
- `INFORM=MAIL` – Send file delivery message to recipients via mail.
- `TRACE=YES` – Similar to DEBUG=YES, but the mail or file is actually distributed.
- `AV=YES[,FORCE]` – Check the message for viruses. The FORCE option overrides any maximum message size limit that is set.
- `DKIM=YES | NO` – Sign (or don't sign) the message with a DomainKeys signature. Default is DKIM=NO.
- `PRIO=* | NN` – Set a network transmission priority level for the file.
- `INFORM=MSG | MAIL` – For non-mail DISTRIBUTE only, specify how to inform users that they have received the file. Effective on z/VM only.

<options> that require postmaster-level privileges:

- `FROM=user` – File originator (RFC821 MAIL FROM:).
- `FROM=DD=ddname` – One line:'address name'.
- `PRE-APPROVED=YES` – Pre-approve message (with DISTRIBUTE POST only).

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

- A "type" of DISTRIBUTE must be defined as the first parameter to the DISTRIBUTE command. Specifying MAIL indicates that the file to be distributed is a mail file, 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 mail file 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.

  Other DISTRIBUTE "types" are MAIL-MERGE (described in the chapter below on DBMS and mail-merge), POST (a method used when messages are "pre-approved", typically by LISTSERV Maestro), FILE (obsolete except for z/VM servers), and RFC822 (where the data is mail and the recipients are defined by the RFC822 To: and cc: fields found in the data).

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

- DD=ddname is the ddname corresponding to the data to be transmitted (i.e. the file or mail file). 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, e.g. 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 that should not normally go to the user invoking DISTRIBUTE--e.g., it can be set to a special address set up specifically to handle errors for this particular DISTRIBUTE job.

  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 mail file. 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 z/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. This is a standard LISTSERV personal password, set with the PW ADD command.

o 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.

o 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 (i.e., 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, i.e. 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 z/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).

9.3 Creating and Sending a Mail RDFR ("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.

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 Section 9.1. 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:

```
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.

9.3.1 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:

- `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
```

o `PRIME=prime-time-specification`

This option controls the time period(s) during which LISTSERV should not process the job.

Note: To all intents and purposes, this option is obsolete. It was originally designed for mainframe environments in which job scheduling was critically important, so that LISTSERV could process large jobs during non-"prime" time and not be a load on the processor during "prime" time. Therefore, and perhaps counter-intuitively to some, it defines the time period during which LISTSERV SHOULD NOT process the job. In modern versions of LISTSERV, the AFTER= RFDR job card option exists to help with scheduling jobs that should be held until a particular time and then released. Therefore the use of PRIME= is deprecated for that type of job.

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

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.

However, you should also be aware that all jobs are checked for PRIME= time when they are received, and if they are within the allowed time window, they are executed immediately.

General Notes: The default value for seconds in a PRIME time specification is zero. Therefore, specifying "00:00-23:59" is equivalent to specifying "00:00:00-23:59:00", not "00:00:00-23:59:59", and, in this example, there is actually a one-minute "window" each night between 23:59 and 00:00 in which an arriving job could be processed. You may code the seconds in the time specification if it is vitally important that no mail be processed during that one-minute "window".

Also note that if running in Network 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"
```

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.

o `AFTER=time-spec`

The preferred method of delaying a message's distribution is to use the AFTER= JOB card option. As noted in Section 2.3 The JOB Control Card, you can specify

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

until which LISTSERV will hold the job.

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). Once the scheduled time arrives, the job will be processed without any consideration to how long it may take to run.

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.

9.3.2 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:

- o 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.

- o 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
```

- `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
```

- `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
```

- `AV=YES|NO`

  Requires LISTSERV Classic/Classic HPO, 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 (z/VM 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 occurs only at the originating site.

IMPORTANT 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.

In addition, 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
WDVPIVAlQEFQWzRcUFpYNTQoUF4pN0NDKTd9JEVJQ0FSLVNUQU5EQVJELU
FO
...
.EB
```

- o DKIM=NO|YES

Requires LISTSERV Classic/Classic HPO with LISTSERV's DomainKeys Identified Mail (DKIM) signing engine enabled (see Section 12 Using DomainKeys Identified Mail (DKIM with LISTSERV for implementation details).

This option controls whether or not LISTSERV will sign outbound messages with DKIM. The default is DKIM=NO.

If enabled, DKIM signing will fail in two cases:

- If running a LISTSERV version without DKIM support; or

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

Otherwise DKIM 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.

9.4 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.

9.4.1 "List-free" bounce processing for one-shot lists

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

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 using DISTRIBUTE MAIL-MERGE, even if the job is not a mail-merge job. The feature does not work with DISTRIBUTE MAIL. In addition, you MUST have EMBEDDED_MAIL_MERGE enabled, and you must use SMTP "workers" (i.e. 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: 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 https://ftp.lsoft.com/listserv/unix/CONTRIB). 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.

9.4.2 Sending Pre-Approved Messages to Moderated Lists

It is sometimes desirable to send mail to a LISTSERV list with **`Send=Editor,Confirm`**, **`Send=Owner,Confirm`**, or **`Send=`*`address`*`,Confirm`**, and have it go out immediately, rather than wait for the editor to confirm it. Thus, there needs to be a mechanism for password-confirming a posting to a list as a substitute for the e-mail confirmation handshake.

To provide this mechanism, the **`DISTRIBUTE POST`** command is available. The syntax is based on the existing **DISTRIBUTE MAIL** command, with certain important differences:

- The verb is **POST** instead of **MAIL**. Data and recipient are supplied as usual. For example, **TO xxx**, **TO DD=xxx**, or **implied DD**.
- Only one recipient (the address of the list to which you are posting) is allowed in the **TO DD**; anything else is will generate an error message. Automated scripts should guard against supplying more than one recipient.
- Although an FQDN is required in the **TO DD**, the hostname of this recipient is assumed to be an alias for the local host, and is ignored. The local-part must be a valid, existing list. An error message will be generated if this is not the case.
- The **PRE-APPROVED=NO|YES** option becomes available to **DISTRIBUTE POST**. The default is **NO** and it then works like a DISTRIBUTE MAIL job with the list as sole recipient, except that it is more efficient. You must be a list owner to use **PRE-APPROVED=YES**. **PRE-APPROVED=YES** cannot be used with DISTRIBUTE MAIL and will generate an error message if so used. If the list is moderated, and the sender address is not an editor, the e-mail sent in this way will be submitted for moderation. To clarify: this replaces confirmation not moderation approval. Make sure that the poster's address (in the 'From:' line) is authorized to post to the list, or the message may be rejected or forwarded to the moderator. **PRE-APPROVED=YES** authenticates the origin of the message, but does not bypass the normal authorization steps.
- **DISTRIBUTE POST** is not available under VM.
- Distribution options are ignored, since **DISTRIBUTE** does not deliver the message at this time but passes it on to the list posting mechanism.

It is important to understand that **DISTRIBUTE POST** is just a submission system. The posting may be processed immediately, it may be delayed until the time specified in the list header, it may be sent to the moderator if the owner is not allowed to post to the list (perhaps not very common, but there are such lists), the list could be on hold, a virus could be detected (if you did not use AV=YES), etc.

Following is a list of the **DISTRIBUTE** options supported or partially supported with `DISTRIBUTE POST`.

Full support:

- **CANON=, AV=, PRE-APPROVED=**

Partial support:

- **DD=:** `TO DD` must refer to a list.
- **DEBUG=YES**: The job will be executed with all verifications, but in the end the message will not be posted (this is the primary purpose of **DEBUG=YES**). The secondary purpose, to receive a report showing the intended message routing, number of recipients and so on, is not supported since **DISTRIBUTE POST** always sends 1 local message and forwards 0 jobs to other servers.

Other options are ignored as long as the syntax is correct. If there is a syntax error, the job fails.

Section 10 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.

Contents:

10.1 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, 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.

10.2 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.

Important: LCMDX.C is a **sample** program for TCPGUI programming. It was originally intended only as a demonstration tool for TCPGUI, but over the years many customers have discovered it can also be used for day-to-day LISTSERV maintenance tasks. That being said, **it should be noted that LCMDX sends LISTSERV login credentials in clear text,** because the TCPGUI protocol does not have any encryption/decryption features. TCPGUI is typically intended to be used locally, by scripts running on the LISTSERV machine itself (most importantly the WA CGI), and for internal traffic encryption is not considered necessary. **This means that if you use LCMDX on a remote machine to send commands to LISTSERV via the Internet, your login credentials technically could be compromised by a "man in the middle" exploit.**

One solution to this problem is to restrict use of LCMDX to machines on your local network. For instance, an engineer could use Remote Desktop to log into a Windows LISTSERV machine, and run LCMDX at the command line in his RDP session. Or for LISTSERV running on unix, use an SSH client to run LCMDX on the LISTSERV machine remotely. With an SSH command line client, the command you will use looks like this:

```
[me@unix ~]$ ssh listserv.myhost.com -X /usr/local/bin/lcmdx
listserv.myhost.com me@myhost.com mypassword thanks
```

You are then asked for your login password for the postmaster account you're using to run LCMDX (in the example, "me@myhost.com"), and then you will get the response to your command:

```
You're welcome!
```

Because you have used SSH to "wrap" the LCMDX command, you have now run LCMDX securely by opening a session to the LISTSERV machine and executing its locally-installed copy of LCMDX, instead of simply executing your own copy of LCMDX insecurely through the Internet. It's a little more complicated to add the SSH wrapper, of course, but it is much more secure.

It should be noted that the TCPGUI port, 2306, may be blocked with a firewall rule that restricts access to the LISTSERV machine only, or if you prefer, to the LISTSERV machine and certain select machines on your local network; as far as the LISTSERV web interface is concerned, as long as WA on the LISTSERV machine can connect to TCPGUI on the LISTSERV machine, there is no need for the TCPGUI port to be exposed to the Internet at all.

Note: Under Solaris, most network programs require you to pass the '`-lsocket -lnsl`' flags to the compiler when compiling. If this is not done, then compiling LCMDX.C under Solaris will fail with network-related library errors.

Important: When running the unix version of lcmdx, note that any command that includes an asterisk (e.g., "**`Q * FOR userid@host`**") will require that the asterisk be escaped. This is due to unix's shell expansion of wildcard characters.

Likewise, any command that requires double-quotes will require that you escape the double-quotes.

In both cases, it should be sufficient to prefix a backslash character ("\", ASCII 0x5C) to the character requiring an escape.

Under the Windows version of lcmdx, it should not be necessary to escape the asterisk wildcard, but *may* be necessary to escape double-quotes. See 10.4.1 Creating or Replacing a List Header for an example of the latter.

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 email address in LISTSERV in order to send commands via TCP/IP. Instructions for registering a password are in 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 email address of the user sending the command,
- password is the password registered with this LISTSERV host for that e- mail address, and
- command is the one-line LISTSERV command.

For example, if joan@example.com wants to set her subscription to the TALK list to digest, and her password on LISTSERV.EXAMPLE.COM (where this list is hosted) is "ABCDE", she can use the following command:

```
lcmdx LISTSERV.EXAMPLE.COM joan@example.com ABCDE SET TALK
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.

10.3 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 primary 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.

10.4 Advanced TCPGUI programming topics

The technique described in the previous section, 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.

10.4.1 Creating or Replacing a List Header

Multi-line commands cannot be sent through the TCPGUI interface. Therefore, the usual approach to send 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 single text string that contains every header line (including the leading asterisk), one after the other, formatted in a particular way:

- A count of the number of characters in the header line, to include the asterisk in column 1
- An underscore character (ASCII 0x5F; *don't use Unicode or Microsoft "smart characters"*)
- The header line itself

Finally, the last header line in the string should not have any trailing spaces.

Thus, if the command you would use to put your list through e-mail was:

```
PUT TEST LIST PW=ABCDE
* test
*
* Owner=joan@example.com
* Notebook=No
* Confidential=Yes
```

For Windows, the command you send through the TCPGUI interface would be:

```
X-STL TEST 6_* test1_*25_* Owner= joan@example.com13_*
Notebook=No18_* Confidential=Yes
```

For unix, you will have to escape the asterisks, as described in 10.2 Running LCMDX :

```
X-STL TEST 6_\* test1_\*25_* Owner= joan@example.com13_\*
Notebook=No18_\* Confidential=Yes
```

(Remember that this is a single-line command -- there should be no line-break characters other than at the very end of the line.) For clarity, ach of the five lines in the header are represented as follows:

Standard header line (emailed)	Number of characters	X-STL formatted header li (For unix, add the backslash for the noted above)
`* test`	6	`6_* test`
`*`	1	`1_*`
`* Owner=joan@example.com`	25	`25_* Owner= joan@example.com`
`* Notebook=No`	13	`13_* Notebook=No`
`* Confidential=Yes`	18	`18_* Confidential=Yes`

(The second line is, of course, a blank header line containing a single asterisk, so it is only one character long.)

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_*30_* 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" message.

10.4.2 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 ***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 address provided, 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.

10.4.3 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 previous case, you used a third-party application to send the file, whereas in this case you will either type the file manually or use cut and paste into your telnet client to perform the operation.

10.4.4 Commands that Respond Over Email

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

10.4.5 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 Joan Smith, 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 2020

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

There are also advanced options for the GET command which can be used with changelog files:

- GET with MSG and COLumns(...)

Finally, there is an alternate command to read and parse list headers in an application-friendly way:

- SHOW X-LISTKWD

10.4.5.1 QUERY

To QUERY subscriber options from an application you should use:

```
QUERY ***GUI*** listname [FOR 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. (If the user subscribed to the list prior to LISTSERV 1.8c, the subscription date is not stored and this line will not appear.)
- 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.

It should be noted that for scripting purposes, it is possible to use a wildcard rather than a single list name with this command:

```
QUERY ***GUI*** * [FOR address]
```

TIP: It should be noted that for reporting or command-line scripting purposes, it is possible to use a wildcard rather than a single list name with this command:

```
QUERY ***GUI*** * [FOR address]
```

This makes it possible to write a script that, for instance, can produce a simple list of all lists to which a given subscriber is subscribed. While this is not sufficient for GDPR-type purposes, such a script using the easily-parsed QUERY ***GUI*** command could be more useful to the system administrator than the only other alternative, which is the overly-verbose 'Query * FOR *address*' command.

While it is also possible to use the wildcard to produce web output via TCPGUI, one should do so with care, as the amount of data returned may be more than you wish to display on a single web page.

10.4.5.2 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.

10.4.5.3 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.
```

10.4.5.4 Changelog access

Please note that this feature is not available under LISTSERV Lite.

Changelogs may be accessed via TCPGUI by using the **`GET`** command options **`MSG`** and **`COLumns(...)`**. Both are designed primarily for GUI use, and while they have been documented, there was no attempt at making the syntax intuitive to end users, as they are not expected to use this feature. The design goal was a syntax that is easy to generate and parse.

The **`MSG`** option (named for compatibility with the **`REVIEW`** option by the same name) simply tells LISTSERV to return the contents of the file as “interactive messages,” which for TCPGUI means that the data will be returned as command output. Thus the option is effective only with text files, as this is not a binary channel.

The **`COLumns(...)`** option tells LISTSERV to filter records from the file before sending it. The syntax is as follows:

```
COLumns(colspec1 colfilter1 [colspec2 colfilter2[...]])
```

Where:

- ‘colspec’ defines the column to which the filter is to be applied. It can be either a-b (positions a to b, inclusive), a- (a through end of line), a.b (positions a to a+b-1) or Wa (blank-delimited word number a).
- ‘colfilter’ is either a-b, a-, -b or a. The latter is equivalent to a-a, whereas the two incomplete forms only check one of the bounds. There must not be any spaces surrounding the hyphen in a-b, and neither a nor b may contain spaces or parentheses.

It is possible to specify as many different columns as desired. Each column must match its respective filter in order for the record to be selected.

It is also possible to specify the same column multiple times (for this purpose, equivalent a-b and a.b specifications are considered the same column). This provides multiple acceptable ranges for the column in question.

It is important to understand how exactly comparisons take place:

- All operands are treated as case-insensitive strings, even if they happen to be valid numbers. Thus, 9.00 is greater than 200.

- Comparisons proceed from left to right and stop at the first differing character. There is no concept of column alignment; in particular, leading blanks are significant (this is not applicable if you defined the column as **`Wa`**).
- If the reference is shorter than the column, the comparison ends successfully when the last character in the reference has been compared. For instance, if you search a changelog for **`W1 200101-200102`**, effectively only the first 6 characters will be searched.
- If the column is shorter than the reference, it is considered to be padded with blanks. This usually fails the record.
- Another way to describe it would be to say that the reference, which is well-defined, takes precedence over the record column, which could vary widely. If the reference is too short, the remainder of the column is ignored. If the column is too short, it will usually not match.
- Comparisons succeed for their bounds, i.e. 2000-3000 will select both 2000 and 3000.
- Comparisons that are bound to fail will fail as expected. If you look for a string of 4 characters in a column that is defined to be only 3 wide, you will find no match and there will be no error.

If there are no matches, the file is not sent and the message “No matching records” is sent instead as command response. This message is suppressed when the **`MSG`** option is used; you simply get an empty response.

To select **`ADD`** or **`DELETE`** records in 2018 in a changelog, you could use **`COL(1-4 2018 W2 ADD W2 DELETE)`**.

Note that this feature works with any file, not just changelogs. In particular, it works with archived changelogs.

Example: To show all POST operations for MYLIST-L since 1 June 2017,

```
GET MYLIST-L CHANGELOG (MSG COLUMNS(W1 20170601- W2 POST)
```

The COLUMNS() option was enhanced in LISTSERV version 16.5 to make it easier to search changelog files for specific email addresses (primarily for the purpose of GDPR reporting). This was accomplished by adding a comparison sub-option (in addition to all of the existing comparison sub-options documented above) which allows searches using an optional data type followed by a pattern, as follows:

```
[^]=typepattern
```

Where:

- ‘^’ is an optional negation
- **`type`** is a mandatory single character indicating the type of comparison
- **`pattern`** is the text to match the requested column to; as usual, a match will select the line for inclusion, and no match will try the next column/pattern pair, if any

- No spaces are allowed in the operand

Type can be:

- '=': column must equal pattern (this is not a substring match)
- 'W': pattern must appear in column as a space-delimited word
- '*': wildcard matching of column to pattern

The '=' option is arguably redundant since **`COLUMNS(16- ==ADD)`** does the same thing as **`COLUMNS(16- ADD)`**, but it can come in handy if you want to match to a pattern containing a hyphen, or starting with an equal sign.

In a GDPR context, you might do:

```
GET listname.CHANGELOG (MSG COL(16- =Wemailaddr)
```

to pull all records out of the *listname*.CHANGELOG file that contain the space-delimited email address (represented by ***`emailaddr`*** in the example) anywhere from column 16 of the record to the end of the record. For an email address such as joe@example.com, this might include records such as

```
20170329114508 POST joe@EXAMPLE.COM This is the subject of
my email.
```

10.4.5.5. The SHOW X-LISTKWD command

Info: `SHOW X-LISTKWD` cannot be issued by list subscribers without higher privilege. Execution requires list owner or LISTSERV maintainer privileges and authentication (personal password, OK cookie confirmation). List owners who are not LISTSERV maintainers may execute the command only against lists they own.

Tip: `SHOW X-LISTKWD` can be used to display exactly what LISTSERV thinks is set in the list configuration for any given list. It can be quite useful as a diagnostic tool to determine why something is not working "as advertised" in your list configuration.

`SHOW X-LISTKWD` is another LISTSERV internal command, which can be useful for determining via a script what lists exist on the server, and/or what options are set for a specified list.

The formal syntax is

```
SHOW X-LISTKWD *

or

SHOW X-LISTKWD keyword[,keyword2,...keywordn] * | listname
[listname2,...listnamen]
```

For instance,

```
SHOW X-LISTKWD *
```

sends back a list of the lists on your server in a machine-readable format:

```
***LIST*** ODBCTEST
***LIST*** TEST-NONOTEBOOK
***LIST*** TEST
```

This is similar to the human-friendly output of the **LISTS** command, which would probably provide more information than you need:

```
ODBCTEST          OBDC test list
TEST              Test list
TEST-NONOTEBOOK   Test list with no notebook
```

But the command is much more powerful than this. Let's add "OWNER" as a parameter before the asterisk, and make the command

```
SHOW X-LISTKWD OWNER *

***LIST*** ODBCTEST
OWNER nathan@EXAMPLE.COM
***LIST*** TEST-NONOTEBOOK
OWNER nathan@EXAMPLE.COM
***LIST*** TEST
OWNER nathan@EXAMPLE.COM jdoe@EXAMPLE.COM
```

That's more like it; now you have a tool to provide an easily-parsed list of all of your list owners, by list. Note that if you have defined an owner address that contains characters that need to be escaped, e.g.,

```
Owner= "john.o'shaugnessy"@example.com
```

, that address will show up in the output *without* the double-quotes, e.g.,

```
***LIST*** TEST
OWNER nathan@EXAMPLE.COM john.o'shaugnessy@EXAMPLE.COM
```

But what if we want to know who all the owners, editors, and moderators are for *all* lists on the server? Simple, comma-separate the three keywords and use a wildcard instead of a list name:

```
SHOW X-LISTKWD OWNER,EDITOR,MODERATOR *

***LIST*** ODBCTEST
OWNER nathan@EXAMPLE.COM
***LIST*** TEST-NONOTEBOOK
OWNER nathan@EXAMPLE.COM
EDITOR nathan@EXAMPLE.COM
MODERATOR ALL nathan@EXAMPLE.COM
***LIST*** TEST
OWNER nathan@EXAMPLE.COM jdoe@EXAMPLE.COM
EDITOR jdoe@EXAMPLE.COM (TEST)
```

Important: When specifying multiple keywords with **`SHOW X-LISTKWD`**, there *cannot* be any spaces between the keywords and the commas. You MUST specify multiple keywords exactly as shown, i.e.,

```
keyword1,keyword2,keyword3
```

and so forth, for as many keywords as you are specifying. If any spaces are found in the keywords, the command fails with no output. This command was designed to be issued by an automated process, so it expects a rather rigid syntax. If your calls to SHOW X-LISTKWD are returning no data, check to ensure that you have not inadvertently put spaces between the keywords and their commas.

In summary,

```
SHOW X-LISTKWD OWNER,EDITOR,MODERATOR *
```

works, but

```
SHOW X-LISTKWD OWNER, EDITOR, MODERATOR *
SHOW X-LISTKWD OWNER , EDITOR , MODERATOR *
```

or any other combination of keywords, commas, and spaces, do not work.

Tip: When issued by a list owner who is not a LISTSERV maintainer, this command will output information only for the lists owned by that list owner

Note: If you have set the "Quiet:" parameter in the Owner= keyword, it will show up like this:

```
***LIST*** TEST
OWNER nathan@EXAMPLE.COM QUIET: jdoe@EXAMPLE.COM
```

If you want to issue the SHOW X-LISTKWD command against a single list, simply specify it instead of the wildcard:

```
SHOW X-LISTKWD OWNER,EDITOR,MODERATOR TEST

***LIST*** TEST
OWNER nathan@EXAMPLE.COM jdoe@EXAMPLE.COM
EDITOR jdoe@EXAMPLE.COM (TEST)
```

You may specify multiple lists (space-separated) if you choose:

```
SHOW X-LISTKWD OWNER,EDITOR,MODERATOR TEST ODBCTEST

***LIST*** TEST
OWNER nathan@EXAMPLE.COM jdoe@EXAMPLE.COM
EDITOR jdoe@EXAMPLE.COM (TEST)
***LIST*** ODBCTEST
OWNER nathan@EXAMPLE.COM
```

Tip: When you specify list names explicitly, the results will display in the same order in which you specified the list names.

The SHOW X-LISTKWD command works with any documented list header keyword. For instance:

```
SHOW X-LISTKWD NOTEBOOK,DIGEST TEST

   ***LIST*** TEST
   NOTEBOOK YES E:\LISTSERV\LISTS\TEST MONTHLY PRIVATE
   DIGEST YES SAME DAILY BOTTOM_BANNER

SHOW X-LISTKWD NOTEBOOK,DIGEST TEST

   ***LIST*** TEST
   DEFAULT-OPTIONS NOACK REPRO

SHOW X-LISTKWD SUBSCRIPTION *

   ***LIST*** ODBCTEST
   SUBSCRIPTION BY_OWNER
   ***LIST*** TEST-NONOTEBOOK
   SUBSCRIPTION CLOSED
   ***LIST*** TEST
   SUBSCRIPTION OPEN
```

and so forth. If a particular keyword is not set explicitly for a given list, there is simply no output for that keyword:

```
SHOW X-LISTKWD SUBSCRIPTION,NOTIFY *

   ***LIST*** ODBCTEST
   SUBSCRIPTION BY_OWNER
   ***LIST*** TEST-NONOTEBOOK
   SUBSCRIPTION CLOSED
   NOTIFY YES
   ***LIST*** TEST
   SUBSCRIPTION OPEN
```

In this case, only the TEST-NONOTEBOOK list has a setting for Notify. You can assume at that point that the other lists are operating with the default setting (for Notify=, it would be "Yes", so in reality, all three of the lists are set to "Notify= Yes").

10.4.6 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:

o ***NOPW*** -- the e-mail address does not have a password registered with LISTSERV

o ***BADPW*** -- the password provided in the TCPGUI command does not match the password registered for the given e-mail address.

10.5 LCMDX.C

Note: This code is also available for download from https://ftp.lsoft.com/CONTRIB/lcmdx.c . A precompiled executable for Windows may be downloaded from https://ftp.lsoft.com/CONTRIB/lcmdx-intel.exe .

To compile an lcmdx executable for unix, simply download the code to a convenient directory and issue the command

```
[me@unixbox ~]$ sudo gcc -O lcmdx.c -o lcmdx
```

at the command prompt. This presumes that your compiler is 'gcc' and you have the correct level of permission to execute it. Once you have lcmdx compiled, you can either keep it in the same directory where you have compiled it, or you can copy it into /usr/local/bin or some other directory in your $PATH. You will also want to set appropriate ownership and permissions for lcmdx. For instance, `**sudo chmod 755 lcmdx; chown root:root lcmdx**' will result in the following:

```
-rwxr-xr-x.  1 root     root 13384 Apr 11 10:42 lcmdx
```

which should allow any user to execute lcmdx. This may be more extreme than you prefer; the only constraint is that the ownership and permissions must be set so lcmdx can be executed by whomever needs to run lcmdx.

Note: Under Solaris, most network programs require you to pass the '`-lsocket -lnsl`' flags to the compiler when compiling. If this is not done, then compiling LCMDX.C under Solaris will fail with network-related library errors.

```
/*******************************************************************
*************
 *
           *
 * 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
```

Section 11 Archive Search Functions using Email

This section is an introduction to the LISTSERV archive search functions using email, and it is intended to be a reference document for general users with little or no knowledge of database search systems. It does not contain any technical information that general users would have to worry about.

This section will discuss the syntax and operational characteristics of the LISTSERV database subsystem. It is assumed that the reader is familiar with his or her email 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 Section 11.3 The SEARCH Command.

Contents:

11.1 Basic search session

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
-> 6 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

To order a copy of these posting, send the following command:

   GETPOST EASE-HOME 58-59 66 74-76

>>> 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 tried 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
                          ^^^^^^^
```

Note: 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.

Next, use the GETPOST command to order the specific posts you wanted to read. For instance, if you want to read posts numbered 66 and 74 through 76, 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
```

LISTSERV would then respond with the desired postings.

For the non-z/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.

11.2 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.

11.3 The SEARCH command

This section will introduce the formal syntax of the SEARCH command.

Note: The minimum abbreviation of the SEARCH command is "S".

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

11.3.1 Basic Search Function

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: The mailing list name doesn't have to be in uppercase.

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 behavior, 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, enclose your search string in double-quotes instead of single quotes, for example:

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

Note: 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 behavior 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 overridden.

Historical Note: Starting in 1.8c, this represented a change from the way the "locate" clause was implemented in LISTSERV 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
```

ɔwing commands are strictly equivalent:

```
.rch (wooden chair) or (plastic chair not blue) in CHAIRS
.rch chair (wooden or (plastic not blue)) in CHAIRS
.rch chair (wooden or (plastic but not blue)) in CHAIRS
.rch chair NEAR (wooden OR (plastic AND NOT blue)) in CHAIRS
```

11.3.2 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
```

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.

11.3.3 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 CHAIRS 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
```

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)
```

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:

```
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. 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?
```

11.3.4 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 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.

```
 > 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,
```

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.

11.3.5 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:

o (Preferred) Narrow your search parameters as explained in Section 11.2 Narrowing the Search 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.

11.3.6 Specifying the Last "n" Posts as a Range

It is also 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@example.com posted. You would send

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

11.3.7 Exact Syntax Description

This section describes the exact syntax of the "SEARCH" command in technical terms.

11.3.7.1 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.

11.3.7.2 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
```

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>>.

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

11.3.7.2 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 either of 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
```

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
```

Note: 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 commands. 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)
```

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-Land)
```

11.3.7.3 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))
```

11.3.7.4 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
```

Section 12 Using DomainKeys Identified Mail (DKIM) with LISTSERV

This feature is not available in LISTSERV Lite.

In order for DKIM support to work, we assume that DKIM support has already been configured in DNS for the domains you will be signing for, per the DomainKeys Identified Mail documentation. If not, general instructions are provided below.

DKIM support is available for LISTSERV Classic and HPO, on all operating systems except for z/VM. It is not available in LISTSERV Lite.

Important: Support for DKIM replaced support for the old Yahoo DomainKeys system, which is now deprecated, beginning with LISTSERV 16.0-2017a.

Most LISTSERV sites already using DomainKeys authentication will find this to be a transparent change when upgrading from an earlier version, with no need to make any adjustment to your current settings. However, see below.

Important: Sites running older versions of LISTSERV which supported the Yahoo DomainKeys specification will wish to review their existing key pair before upgrading to LISTSERV 16.0-2017a or later, as key lengths which were sufficient for DomainKeys may be too short for DKIM. Per RFC 6376 “DomainKeys Identified Mail (DKIM) Signatures”, Section 3.3.3, “Signers MUST use RSA keys of at least 1024 bits for long-lived keys”, whereas many DomainKeys sites may be using keys of 512 or 768 bits.

In addition, RFC 8301 updates RFC 6376 and states in Section 3.1 that “DKIM supports multiple digital signature algorithms. Two algorithms are defined by this specification at this time: rsa-sha1 and rsa-sha256. Signers MUST sign using rsa-sha256. Verifiers MUST be able to verify using rsa-sha256. rsa-sha1 MUST NOT be used for signing or verifying.”

Bottom line: L-Soft strongly recommends that all DKIM keys MUST be 1024 bits or more, and they MUST be SHA-256 (AKA SHA-2) keys. Keys created with the SHA-1 algorithm are NOT supported by the DKIM specification, and MUST NOT be used. Key pairs NOT meeting these specifications are used strictly at your own risk.

Contents:

12.1 Creating DKIM keys and configuring DNS

Information describing the creation of DKIM keys and the configuration of DNS to enable DKIM signing is found at the DKIM website. Please see http://www.dkim.org/#specifications for the official DKIM documentation.

12.1.1 Creating a DKIM Key Pair

It is quite simple to create a DKIM key pair. There are websites where you can enter the basic information (selector and domain name) and the website will generate the key pair for you. However, it is questionable whether such sites will actually guarantee the confidentiality of the public keys they generate, so this may or may not be the best route for your site.

The simplest way to generate a DKIM key pair is to log into a unix machine that has OpenSSL installed, and issue the following commands in a terminal window.

Important: Please determine what version of OpenSSL you are using before creating your keys. This can be done by issuing the command

```
openssl version
```

at an operating system shell prompt.

If your OpenSSL version is 1.x:

```
$ openssl genrsa -out rsa.private 1024

$ openssl rsa -in rsa.private -out rsa.public -pubout -
outform PEM
```

If your OpenSSL version is 3.x:

```
$ openssl genrsa -out rsa.private -traditional 1024

$ openssl rsa -in rsa.private -out rsa.public -pubout -
outform PEM
```

In either case, running these commands should result in two files being created: rsa.private and rsa.public. The private key will be a PKCS#1 private key which starts with

```
-----BEGIN RSA PRIVATE KEY-----
```

and ends with

```
-----END RSA PRIVATE KEY-----
```

Important: If your keys start with "BEGIN PRIVATE KEY" and end with "END PRIVATE KEY" (that is, with no "RSA") then you have generated PKCS#8 keys which LISTSERV cannot use and which will throw an error when LISTSERV attempts to read them. This is due to creating the keys using OpenSSL 3.x without the "-traditional" flag as documented above.

Note that you *cannot* simply edit the banners to fix this. PKCS#8 keys will have to be regenerated from scratch as PKCS#1 "traditional" keys.

The rsa.private file contains your private key, which will be used below to create the DKIM file for LISTSERV; the rsa.public file contains the corresponding public key, which will be used to create the DNS TXT record you need for DKIM.

12.2 Creating DNS records for DKIM

Many of our customers have hosted DNS, that is, the domain registrar from which they have purchased their corporate domain(s) also hosts their DNS zones, and these customers typically edit their zone file via a web-based GUI. In that case, for this and for each of the following examples, simply enter the appropriate information in the GUI and follow the registrar’s instructions to save and propagate it.

Other customers (generally large corporations or academic institutions) are more likely to run their own DNS servers, and will have to edit the appropriate zone file in the usual way.

In either case, this section is intended only to provide examples of the information you will need to create your DKIM records, and general DNS advice applicable to both cases above. Editing zone files is beyond the scope of this document and customers in the first case, above, should consult their ISP’s support for assistance, whereas customers in the second case are urged to consult the DNS/BIND documentation for their particular implementation of DNS for guidance.

Important: Please be aware that the examples provided below are not intended to be used “as-is”; you must substitute the correct information for your site or DKIM will not work. In particular, please note that the public and private keys in these examples are purposefully invalid and cannot be used to create live DKIM records.

12.2.1 Creating a DKIM TXT selector record in DNS

Creating a DKIM TXT record can be done in various ways. If you run your own DNS, simply edit your forward zone file to include a TXT record. We will assume for this exercise that the LISTSERV host name is “listserv.example.com”. You will need to enter the following information:

Host:	**`default._domainkey.listserv`**
TXT Value:	**`v=DKIM1;k=rsa;p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQ KBgQDcARWuStG7G33L+M5jqjiCbhfKBlgxIMC8Of5ODONOTUSE THISKEYITISANEXAMPLEONLY91O1RigBB/C+UXzPO+N1+hZ55Z XS8MPGPgaV9VM1EysEdyfm2Y/rn935GGJwtm67fz+6dyKkCAzL sMjR5DvcxxlMzf6Gs9TrX7PBNwIDAQAB`**

TTL:	Your preference, but typically 1 hour

Notes:

1. When creating a new record in a DNS zone file, the host name usually is not fully-qualified. If you are editing the zone file for the example.com zone, it should not be necessary to enter the fully-qualified domain name in the "host" section. Be sure to check the documentation for whatever DNS you are running if this is unclear.
2. We are assuming a DKIM selector value of "default". For the purposes of DKIM authentication, external sites will always check DNS for a TXT record belonging to "selector"._domainkey."hostname". In our example, external sites would be looking for the TXT record belonging to default._domainkey.listserv.example.com . Note that the underscore before "domainkey" is required.
3. The TXT value should not break and wrap as shown. It should be one continuous line of text. The value of "p=" is the text of the public key from between the lines

   ```
   -----BEGIN RSA PUBLIC KEY-----
   ```
   and
   ```
   -----END RSA PUBLIC KEY-----
   ```

   Those lines should not be included as they are not part of the public key.
4. If your key has banners reading "`BEGIN PUBLIC KEY`" and "`END PUBLIC KEY`", you have generated PKCS#8 keys and must start over and generate PCKS#1 keys as explained in the preceding chapter.

12.2.2 Creating a DKIM TXT policy record in DNS

Note: The policy record is an artefact of the original Yahoo! DomainKeys implementation and is no longer required in any way by the current RFC 6376 "DomainKeys Identified Mail (DKIM) Signatures" standard. ***You may freely ignore this section, but it remains for historical purposes.***

Again using our "listserv.example.com" example, you will enter the following information:

Host:	`_domainkey.listserv`
TXT Value:	`o=~`
TTL:	Your preference, but typically 1 hour

The "Host:" field is slightly different this time. The policy record does not require the selector "default", so we leave it off. Note that the underscore before "domainkey" is required.

The "TXT Value:" field contains the policy to be applied to DKIM lookups. The value we've provided above means that "some" outbound mail from (in the example case) listserv.example.com will be signed with DKIM. This is the default, and L-Soft's recommended setting.

12.3 LISTSERV configuration

LISTSERV's DKIM support is configured by doing two things.

1. Supply one or more private keys.

Each private key is stored as a text file in LISTSERV's main or home directory (that is, the directory where the *.list files are) and must be named xxx**`.dkim`**, where xxx is the arbitrary name you choose to give the key. If you only use one key, it is recommended to name it **`default.dkim`**.

The file is created in the usual openssl/RSA format, with one minor modification. Here is an example (not a real key, don't use it):

```
d=listserv.example.com;  s=default
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQDcARWuStG7G33L+M5jqjiCbhfKBlgxIMC8Of5OQaM00v83IRuk
jSq4pPvAhsHKSCacVCHp91O1RigBB/C+UXzPO+N1+hZ55ZXS8MPGPgaV9VM1EysE
dyfm2Y/rn935GGJwtm67fz+6dyKkCAzLsMjR5DvcxxlMzf6Gs9TrX7PBNwIDAQAB
AoGBALY1V8WARe+XNzqlmBnHMwIjOCSj2Irnu3io90vM5OStE56PFxvTptxCGBc+
BGYKF6BFtcjWhEeQETW5Y9PcHWbj3O2OSrhk9sPQHZCW46J0IVpP0vRHyrK4o+zX
CbHkFEJZFSBN2IquUR5m9Yqb5dqQPRf/7lGAQpVrd03wiX4RAkEA8jRE3CFfh7I5
idx1q2ohBEh2rPHioDONOTUSETHISKEYITISANEXAMPLEONLYPhcwoDjQQ/EqIUS
wezkWNX2zQJBAOiJGr7tzHY2Cg4ftfl1DJYXNkRtsR4ZoVsgcjhPVTLScfG7nOFL
pMCKE5ChYFkbYmh5knhOsYrZgBqPDxe8MBMCQFY3dv+pPZlPPx4tBRIUwFYG+X/M
xvGpwDhMaYIm5fmlwBLCBnHt8Z+kEGVwKbabVUkcLHUmYjOe0zOHAS4CVE0CQHSA
9MCCHfV//6ux4Zd5OHQebxb7qki9aKVibTefL72FyIbni6MpJgM9aq4E3GPon3Ze
qq7SJou9izxDPrmSlLcCQBG0OYhOQWank6kWaziTY/K93vGyHQOqUM425iLQdWWu
DHj08akKRILiTXhUYgQA9/fE/ncalK4ChvsVG0bqXZ0=
-----END RSA PRIVATE KEY-----
```

The first line in the file must include a specification for the 'd=' and 's=' parameters of the DomainKeys signature (in whatever order, as long as they are both there). Per the DKIM documentation, these variables specify the domain for which you are signing ("d=") and the "selector" that is used to form the query for the public key ("s="). For instance, let's say that your public key is registered as follows in the DNS:

```
default._domainkey.listserv.example.com IN TXT “g=; k=rsa;
p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDcARWuStG7G33L+M5jqj
iCbhfKBlgxIMC8Of5ODONOTUSETHISKEYITISANEXAMPLEONLY91O1RigBB/
C+UXzPO+N1+hZ55ZXS8MPGPgaV9VM1EysEdyfm2Y/rn935GGJwtm67fz+6dy
KkCAzLsMjR5DvcxxlMzf6Gs9TrX7PBNwIDAQAB”
```

The selector is "default" and the domain is "listserv.example.com".

Important: Again, please remember that the public and private keys in these examples are purposefully invalid and cannot be used to create a live DKIM configuration for LISTSERV.

2. Supply a DKIM_SIGN Configuration Variable

In your site configuration file, add a DKIM_SIGN= variable containing a blank-separated list of domains that you are able and willing to sign for. You can use wildcards, but only of the form '**`*.EXAMPLE.COM`**'. You can't use, for instance, '**`SALES.EXAMPLE.*`**'. For each entry in the list, specify the key to be used, as follows:

```
DKIM_SIGN=EXAMPLE.COM  *.EXAMPLE.COM  EXAMPLE.CA(CA)  *.EXAMPLE.CA(CA)
```

In the example we have been using above, our DKIM_SIGN variable would be

```
DKIM_SIGN=LISTSERV.EXAMPLE.COM
```

(Under unix, don't forget to export DKIM_SIGN .)

By default, the key called DEFAULT is used (if one exists). So, in the sample above, the key for EXAMPLE.COM will be fetched from DEFAULT.DKIM whereas the key for EXAMPLE.CA will come out of CA.DKIM.

12.4 Starting LISTSERV with DKIM support

LISTSERV loads the keys at startup and makes simple verifications.

```
26 Jul 2019 14:14:26 Loading DomainKeys private keys...
26 Jul 2019 14:14:26 -> Loaded DEFAULT (d=EXAMPLE.COM;
s=TEST; RSA-1024)
26 Jul 2019 14:14:26 -> Loaded CA (d=EXAMPLE.CA; s=TEST;
RSA-1024)
26 Jul 2019 14:14:26 DKIM support enabled
26 Jul 2019 14:14:26 DKIM Accelerator enabled
```

In particular, the 'd=' parameter in the key must match or be a parent of the domain you want to sign for. Thus, the key for EXAMPLE.COM can be used to sign for EXAMPLE.COM and *.EXAMPLE.COM, but not for EXAMPLE.CA. LISTSERV will skip any invalid entries. Keys are kept in memory so you can have as many as you want.

If there is no DKIM_SIGN variable or if you are running a LISTSERV version without DKIM support, LISTSERV does not attempt to load any keys and the DKIM feature is bypassed.

12.5 Using DKIM with LISTSERV

By default (**`DKIM_SIGN_ALL=0`**), LISTSERV does not sign any messages using DKIM other than those for which DKIM signing is expicitly requested by the caller, for instance, DISTRIBUTE jobs with an explicit "**`DKIM=YES`**" parameter in the JOB card. List mail and non-list administrative messages will not be signed when DKIM_SIGN_ALL is left at the default value.

However, because of its relationship to the DMARC protocol, you will probably want to have LISTSERV sign every message that it generates, regardless of its source. Setting **`DKIM_SIGN_ALL=1`** in the site configuration file tells LISTSERV to try to sign every message for which it has a suitable private key, as defined in the DKIM_SIGN configuration parameter (see above).

(If setting DKIM_SIGN_ALL in the go.user file under Unix, please also ensure that the variable is exported.)

Once you have enabled DKIM signing with **`DKIM_SIGN_ALL=1`**, the behavior is as follows:

With mailing lists:

- Incoming DomainKeys signatures submitted to a mailing list will be suppressed unless "**`Misc-Options= KEEP_DKIM_SIGNATURE`**" is set in the list configuration.

  In general, you will not need (or want) to use the KEEP_DKIM_SIGNATURE option. As

DKIM is specified today, signatures DO NOT survive posting to mailing lists (LISTSERV or otherwise), so LISTSERV removes them by default to avoid triggering alerts for subscribers whose mail hosts have implemented the stricter forms of DKIM. Therefore, if used at all, the KEEP_DKIM_SIGNATURE option should be used judiciously and with caution.

- When DKIM signing is enabled at the server level (**`DKIM_SIGN_ALL=1`**), the default is that all list mail (including administrative mail) will be signed. It is possible to override the default and disable DKIM signing for individual lists (typically for debugging purposes) by using the "**`Misc-Options= NO_DKIM_SIGNATURE`**" setting in the list configuration. It is not recommended to run with this option set during normal operation.

In DISTRIBUTE and DISTRIBUTE MAIL-MERGE jobs:

A **`DKIM=NO|YES`** option is available for the DISTRIBUTE command (default: NO). This will fail if running a LISTSERV version without DKIM support, but otherwise it 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.

In other types of messages:

When DKIM signing is enabled as described above, LISTSERV will to try to sign every message for which it has a suitable private key, as defined in the DKIM_SIGN configuration parameter.

12.6 Restrictions and implementation choices

If DKIM signing is enabled (**`DKIM_SIGN_ALL=1`**), a message that already has a DKIM signature when it arrives at LISTSERV will have that signature replaced by one generated by LISTSERV.

LISTSERV can be configured to retain the old signature of such messages via the list -level keyword setting "**`Misc-Options= KEEP_DKIM_SIGNATURE`**", though (as noted above) this is rarely, if ever, recommended. With this setting, only the original signature will be included in the distributed message. LISTSERV won't add its own signature in this case, as double DKIM signatures are disallowed in most cases and, even when allowed, may not be handled correctly by all implementations.

DKIM can be used to sign mail-merge messages, but in that case LISTSERV's Embedded Mail Merge (EMM) feature MUST be enabled. Using EMM is the only way to guarantee that the signing engine will see the exact text being sent to the recipient, and that the signature will match. EMM is normally enabled by default, but can be disabled at the server level. Check the setting of EMBEDDED_MAIL_MERGE= in the site configuration if you believe this may be an issue.

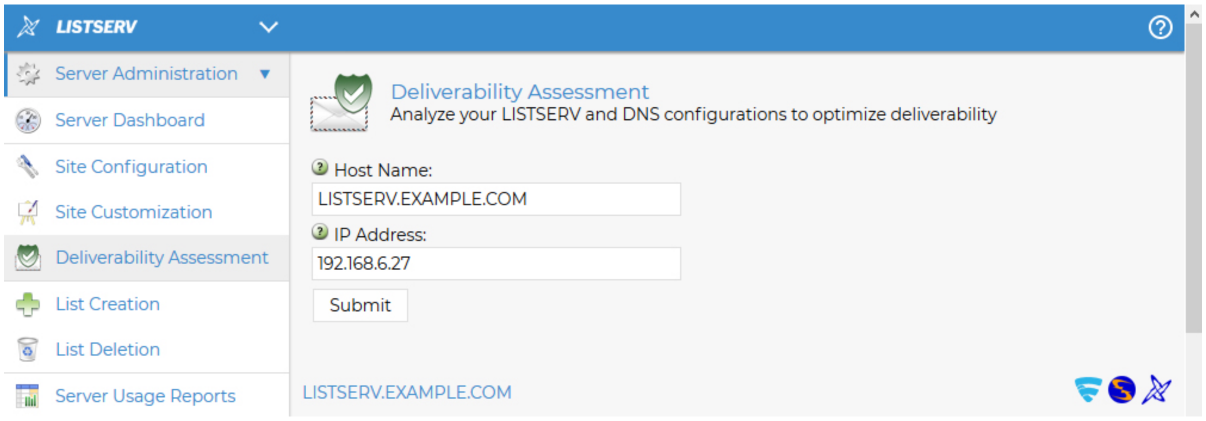
12.7 Testing DKIM

Once you have created your DNS entries and LISTSERV configuration for DKIM, you will want to test it.

12.7.1 LISTSERV's Deliverability Assessment report

LISTSERV has a built-in Deliverability Assessment report which can be reached in the web interface at **Server Administration** -> **Site Configuration** -> **Deliverability Assessment**. The first screen looks like the following:

Figure 12-1 Deliverability Assessment - Initial Screen



Deliverability Assessment
Analyze your LISTSERV and DNS configurations to optimize deliverability

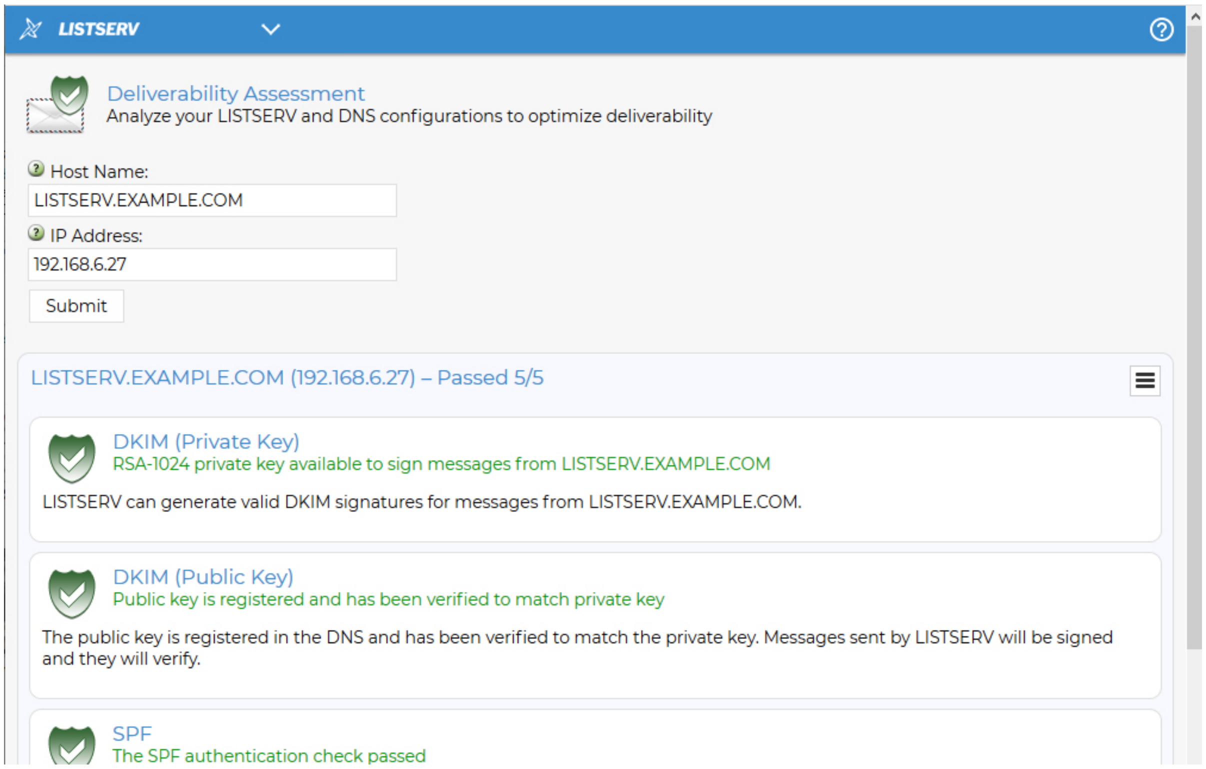
Host Name:
LISTSERV.EXAMPLE.COM

IP Address:
192.168.6.27

LISTSERV.EXAMPLE.COM

Unless you have multiple domains set up in LISTSERV, there should be no reason to change the pre-populated values. If they are correct, simply click **Submit**. This will yield the following report:

Figure 12-2 Deliverability Assessment - Report Screen



Deliverability Assessment
Analyze your LISTSERV and DNS configurations to optimize deliverability

Host Name:
LISTSERV.EXAMPLE.COM

IP Address:
192.168.6.27

LISTSERV.EXAMPLE.COM (192.168.6.27) – Passed 5/5

DKIM (Private Key)
RSA-1024 private key available to sign messages from LISTSERV.EXAMPLE.COM
LISTSERV can generate valid DKIM signatures for messages from LISTSERV.EXAMPLE.COM.

DKIM (Public Key)
Public key is registered and has been verified to match private key
The public key is registered in the DNS and has been verified to match the private key. Messages sent by LISTSERV will be signed and they will verify.

SPF
The SPF authentication check passed

The green shields indicate that, so far as LISTSERV is concerned, you have properly configured the DKIM DNS entry, and LISTSERV itself is properly configured to sign outbound messages with DKIM. If either or both of the shields are not green, you need to recheck your DNS entry and LISTSERV configuration, and correct any errors before running the report again.

12.7.2 Testing the DNS entries

Once you have created the DKIM DNS entries, you can check them with NSLOOKUP or DIG to ensure that they are being served properly by the DNS server:

If you use NSLOOKUP to check the record after you create it, you'll see something like this:

```
> set type=txt
> default._domainkey.listserv.example.com
Server:  google-public-dns-a.google.com
Address:  8.8.8.8

Non-authoritative answer:
default._domainkey.listserv.example.com       text =

        “v=DKIM1;
p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDwl88WzSt
oOTznXCeMJF+J0XDaxrRYgl42hx+zZ4MUAaI8ZCsaozbK4RiCjAt8TUSf4Ju
En8mqTK
MfL4Rxf0SEcgEPyJSq1j9AxU3e8ERx5GXj2kJw6lOxIa+Fh0WTcKNImgKz9g
MUUL7ls
LnBPghNCCdUvnmYpLhS2HMR1EFrDwIDAQAB”
>
```

and

```
> set type=txt
> _domainkey.listserv.example.com
Server:  google-public-dns-a.google.com
Address:  8.8.8.8

Non-authoritative answer:
_domainkey.listserv.example.com       text =

        “o=~”
>
```

To test the functionality of the DKIM DNS entries, you will probably want to use an online service. One such service is MXToolBox.com. They provide a DKIM Lookup tool that is both free and easy to use, located at https://mxtoolbox.com/dkim.aspx . Simply enter the domain name (for example, “listserv.example.com”) and the selector (for example, “default”) into the two text boxes on that page and click the button for DKIM Lookup. This presents an in-depth report that you can use to verify whether or not your DKIM TXT record has been properly created.

There are many other testers out there; the MXToolBox tester is simply the one that we tend to use at L-Soft. Other testers include https://www.mail-tester.com/spf-dkim-check and https://www.dmarcanalyzer.com/dkim/dkim-check/ ; many more are available via a web search.

12.7.3 Testing DKIM signatures on email

Finally, the simplest way to test that LISTSERV is actually signing emails properly is to create a test list, add an external test account to it (e.g., a GMail account) and send mail to the list.

In the mail headers for the message received by the test account, you'll see something like the following:

```
DKIM-Signature: v=1; a=rsa-sha256; d=LISTSERV.EXAMPLE.COM;
s=DEFAULT; c=relaxed/relaxed; bh=wCPfXJT/
+EjG2NI/0kOFZQI3luKHV0YjC+ZO6gi9sW8=; i=@LISTSERV.TD.COM;
 h=Date:Sender:From:Subject:To;
 b=b9p4Vj9NSsMxTIRwhO1oRYTYovn8UT/
```

This is the DKIM signature added by LISTSERV as the message was processed and distributed.

Note: The signature will not appear to match either of your DKIM keys; this is because the signature is generated on a per-message basis using the private key in the appropriate *.DKIM file you created and installed earlier.

Above the DKIM-Signature header will be a header showing that the message was properly authenticated:

```
Authentication-Results: mx.google.com;
       dkim=pass header.i=@LISTSERV.EXAMPLE.COM
header.s=DEFAULT header.b=b9p4Vj9N;
       spf=pass (google.com: domain of owner-nolist-test-
20160331-set*john*-doe**gmail*-com@listserv.example.com
designates 192.168.6.27 as permitted sender)
smtp.mailfrom=owner-nolist-TEST-20160331-SET*john*-
doe**GMAIL*-COM@listserv.example.com;
       dmarc=pass (p=NONE sp=NONE dis=NONE)
header.from=example.com
```

Since we used a GMail account, the above is how Google reports the results. In this case, the message not only passed DKIM testing, but also SPF and DMARC (meaning that this particular LISTSERV server is very well provisioned for mail reputation). Other ISPs will produce similar test results, although the formatting may vary depending on the mail product used on the receiving end.

It should be noted that Yahoo, which originally promulgated the now-deprecated DomainKeys standard, will also produce results for their standard, even though LISTSERV uses DKIM:

```
Authentication-Results: mta4010.rog.mail.bf1.yahoo.com
from=listserv.example.com; domainkeys=neutral (no sig);
from= LISTSERV.EXAMPLE.COM; dkim=pass (ok)
```

However, Yahoo is also authenticating against the new DKIM standard, so as long as DKIM gets a "pass", it doesn't matter that you did not provide a DomainKeys signature.

Section 13 DMARC and LISTSERV

In 2014, starting with Yahoo! and quickly followed by AOL, several of the large ISPs implemented aggressive DMARC authentication policies that created giant headaches. Stuck between a rock and a hard place with no advance warning, list administrators were forced to resort to a variety of circumventions, from deleting all Yahoo and AOL subscribers, to putting the list address in the "From" field, thus causing all private replies to be broadcast to the list without warning and inviting mail clients to submit out-of-office messages to the entire list. The solution L-Soft developed at that time for LISTSERV does not require anyone to make compromises or lose any functionality -- not even the Yahoo and AOL subscribers.

This section discusses the DMARC authentication protocol and how LISTSERV complies with it.

Note: DMARC rewriting is ***not available*** for z/VM because because there is no DNS query support in the z/VM version.

Contents:

13.1 What is DMARC?

DMARC is an acronym which stands for **D**omain-based **M**essage **A**uthentication, **R**eporting & **C**onformance.

Bulk email needs to be properly authenticated to meet data security and deliverability needs. It is essential for every site, and ISPs and corporations worldwide are implementing DMARC to help solve these problems.

But what, exactly, is DMARC?

DMARC is a technical standard that "marries" several different, pre-existing security protocols to create a consistent method of proving that mail claiming to originate from your domain actually did so originate. Your authentication practices are published in DNS for anyone to query, along with what to do with mail that fails the authentication checks. It also allows reporting on what actions were taken on mail claiming to be from your domain.

Effectively, DMARC builds on the following existing protocols:

- SPF (Sender Policy Framework, defined in RFC 7208 and related standards
- DKIM (Domain Keys Identified Mail, an overview of which is found in RFC 5585, and which we discuss in conjunction with LISTSERV in Section 12 of this document

- o Standard DNS records such as A (address) and MX (mail exchanger) records

It is beyond the scope of this document to explain in detail how DMARC works, as there are many such explanations available on the Internet. However, since 2014, LISTSERV has recognized DMARC and is able to comply with it.

13.2 What is Binding Operational Directive (BOD) 18-01?

Binding Operational Directive 18-01, or BOD 18-01 for short, is a US Department of Homeland Security directive aimed at Federal executive branch agencies, issued on 16 October 2017. It is "binding" in the sense that it is a "compulsory direction" to those agencies "for purposes of safeguarding federal information and information systems." DHS believed so strongly in the need for DMARC to be adopted that, in BOD 18-01, it stated unambiguously

“Within one year [October 16, 2018] of BOD issuance, set[] a DMARC policy of “reject” for all second-level domains and mail-sending hosts.”

Federal agencies were, predictably, slow to adopt the policies demanded by the directive. The research firm Agari Data reported that, as of July 15, 2018, "19 percent of executive branch domains still have no DMARC record and 26 percent have not progressed past the monitoring policy (p=none), leaving almost half of executive branch domains vulnerable to domain name spoofing."

LISTSERV's DMARC support provides its executive branch customers -- and others -- with a seamless DMARC-ready mailing list solution.

In passing, and unrelated to DMARC, it should also be noted that the LISTSERV web interface can enable HTTP Strict Transport Security, which directs browsers to only connect to the web interface using secure HTTPS connections. Unencrypted HTTP connections are automatically replaced with HTTPS connections at the browser level, preventing the transmission of unencrypted data to the server. **HSTS is also required by US BOD 18-01.** For more information, please see the WWW_HSTS_MAX_AGE site configuration keyword.

13.3 How does LISTSERV comply with DMARC?

When LISTSERV receives a posting to a mailing list, it queries DNS to see if the sending domain in the From: address has a DMARC record containing "p=reject" or "p=quarantine". There is nothing to configure, it is done automatically. If additional ISPs (besides Yahoo.com and AOL.com) decide to implement either the "p=reject" or "p=quarantine" rule, LISTSERV will automatically detect this and adjust its behavior accordingly.

As we have already noted in the previous section, the DHS BOD 18-01 requires US federal executive branch agencies to implement "p=reject", so all such agencies which have complied with the directive to date will be automatically covered as well.

Note: DMARC rewriting is ***not available*** for z/VM because because there is no DNS query support in the z/VM version.

Note: LISTSERV also rewrites addresses for hosts with "p=quarantine" because the definition of "p=quarantine" is unclear, but seems to all but guarantee messages from users at those hosts will end up in recipients' spam folders if the From: address is not rewritten.

The DMARC organization's FAQ states (verified 23 Feb 2021) the following:

> **What does a "quarantine" policy mean in a DMARC record?**
>
> Given the real-world, non-technical use of the term, quarantine means "set aside for additional processing". The definition is at the appreciation of the manager of the receiving email infrastructure. It may mean deliver to the "junk folder" but it may also mean hold in a database for further review by dedicated personnel, or simply add a specific tag to the message before delivery.

Because this definition is unclear, and there is no mechanism available for LISTSERV to obtain further information regarding how a specific host implements the quarantine, L-Soft made the design decision to treat "p=quarantine" identically to "p=reject".

If the message comes from such a domain, LISTSERV rewrites the "From:" address to the format

```
[token]-dmarc-request@LISTSERV.EXAMPLE.COM
```

(see example below). The poster's full name is left unchanged. Most mail clients do not show the actual address, only the name, so most users will not see any obvious difference.

If a list member decides to send a private reply to the sender (i.e. not to the list), via the re-written special address, it is routed back to LISTSERV, which forwards it to the original "From:" address. The token is unique to each sender and only works via the same LISTSERV instance.

Best of all, there is no need for anyone to change their habits and/or their mail folder filing rules for incoming mail.

It should be noted this is not a "full" implementation of DMARC as LISTSERV does not attempt to validate SPF records in DNS, nor does it attempt to verify DKIM signatures from any sending domains. It simply looks for a "p=reject" or "p=quarantine" in the originating domain's DMARC record, and automatically processes the From: address accordingly.

This is what it looks like in the LISTSERV log file:

```
>21 Apr 2014 12:21:56 Processing mail from user@YAHOO.COM for
TEST2
>21 Apr 2014 12:21:56 "From:" address rewritten due to p=reject
DMARC rule
>> Old address: user@YAHOO.COM
>> New address: 00000003f16b4808-dmarc-
request@LISTSERV.EXAMPLE.COM

>21 Apr 2014 12:22:36 From MAILER@LISTSERV. EXAMPLE.COM: X-ADMMAIL
00000003f16b4808-dmarc-request@LISTSERV.EXAMPLE.COM
>21 Apr 2014 12:22:36 -> Forwarded to user@YAHOO.COM
```

The original non-re-written address will be written to the list message archives files, if any. This keeps message threading and searching just like it was before. The original non-re-written address will also be written to the **`Reply-To:`** message header line if the list is configured for **`Reply-To= Sender`** or **`Reply-to= Both`**.

On Windows, handling of the tokenized forwarding address is automatic. On Unix platforms, admins will need to implement a wildcard alias for *-request@... email addresses. L-Soft Support has published suggestions for this for Postfix and sendmail.

13.4 Debugging DMARC errors in LISTSERV

Note: DMARC rewriting is ***not available*** for z/VM because because there is no DNS query support in the z/VM version.

The TRACE_DNS debug flag is useful for debugging problems with DMARC lookups from LISTSERV, as in conjunction with the "DEBUG QUERY TXT *hostname*" command, it can show you exactly what information LISTSERV is receiving (or not receiving) from its DNS servers.

To use the feature, issue the LISTSERV maintainer command "DEBUG FLAGS +TRACE_DNS". For instance (the interesting parts are highlighted in light grey):

```
Paused - enter a command:
debug flags +trace_dns
1 Feb 2017 15:10:05 From LISTSERV@LISTSERV.EXAMPLE.COM:
debug flags +trace_dns
* Debug flags for this session: 00000200
*
00000001 TRACE_DIST        [Trace DISTRIBUTE processing -
OFF]
00000002 TRACE_MIME        [Trace MIME parser - OFF]
00000004 TRACE_LISTS       [Trace (a few) list-related
functions - OFF]
00000008 TRACE_SPAM        [Trace spam filter calls - OFF]
00000010 TRACE_EMM         [Trace Embedded Mail-Merge
processor - OFF]
00000020 TRACE_DEV         [Temporary ad hoc tracing for
development use - OFF]
00000040 TRACE_FSAV        [Trace FSAV calls - OFF]
00000080 TRACE_LDAP_CALLS  [Trace LDAP library calls - OFF]
00000100 TRACE_LDAP_DATA   [Trace data obtained from LDAP -
OFF]
00000200 TRACE_DNS          Trace DNS lookups
08000000 HOLD_DISTBG       [Do not process background
DISTRIBUTE jobs - OFF]
10000000 HOLD_XB64         [Hold X-B64 jobs instead of
processing them - OFF]
20000000 KEEP_JOBFILES     [Keep successfully processed job
files - OFF]
40000000 TRACE_TCPGUI      [Additional TCPGUI tracing - OFF]
```

Next, enter the command "DEBUG QUERY TXT hostname" with whatever hostname is configured in DNS with the DMARC TXT record LISTSERV should be referencing.

```
Paused - enter a command:
debug query txt _dmarc.yahoo.com
1 Feb 2017 15:10:43 From LISTSERV@LISTSERV.EXAMPLE.COM:
debug query txt _dmarc.yahoo.com
1 Feb 2017 15:10:43 DBG> Looking up TXT records for
_DMARC.YAHOO.COM
1 Feb 2017 15:10:43 DBG> Trying 8.8.8.4
1 Feb 2017 15:10:43 DBG> One record found
* _DMARC.YAHOO.COM IN TXT "v=DMARC1; p=reject; pct=100;
rua=mailto:dmarc_y_rua@y
ahoo.com;"
```

If there is no DMARC record for the hostname you specify, the response will be different:

```
1 Feb 2017 15:21:37 DBG> Looking up TXT records for
_DMARC.EXAMPLE.COM
1 Feb 2017 15:21:37 DBG> Trying 8.8.8.4
1 Feb 2017 15:21:37 DBG> No record found
* DNS query for _DMARC.EXAMPLE.COM returned no TXT record.
```

Finally, turn off the debug flag:

```
Paused - enter a command:
debug flags -trace_dns
1 Feb 2017 15:11:00 From LISTSERV@LISTSERV.EXAMPLE.COM:
debug flags -trace_dns
* Debug flags for this session: 00000000
*
* 00000001 TRACE_DIST       [Trace DISTRIBUTE processing -
OFF]
* 00000002 TRACE_MIME       [Trace MIME parser - OFF]
* 00000004 TRACE_LISTS      [Trace (a few) list-related
functions - OFF]
* 00000008 TRACE_SPAM       [Trace spam filter calls - OFF]
* 00000010 TRACE_EMM        [Trace Embedded Mail-Merge
processor - OFF]
* 00000020 TRACE_DEV        [Temporary ad hoc tracing for
development use - OFF]
* 00000040 TRACE_FSAV       [Trace FSAV calls - OFF]
* 00000080 TRACE_LDAP_CALLS [Trace LDAP library calls - OFF]
* 00000100 TRACE_LDAP_DATA  [Trace data obtained from LDAP -
OFF]
* 00000200 TRACE_DNS        [Trace DNS lookups - OFF]
* 08000000 HOLD_DISTBG      [Do not process background
DISTRIBUTE jobs - OFF]
* 10000000 HOLD_XB64        [Hold X-B64 jobs instead of
processing them - OFF]
* 20000000 KEEP_JOBFILES    [Keep successfully processed job
files - OFF]
* 40000000 TRACE_TCPGUI     [Additional TCPGUI tracing -
```

```
        OFF]
```

These example commands were issued from the LISTSERV command line; note that the commands may also be issued by email or through the web interface's "LISTSERV command" pages.

13.5 DMARC_NO_REWRITE

This feature requires LISTSERV 16.5-2018a or later (build dates of 8 Dec 2018 or later).

Note: DMARC rewriting is ***not available*** for z/VM because because there is no DNS query support in the z/VM version.

Organizations which use DMARC "p=reject" or "p=quarantine" for their own domain, and where the outbound MTA used by LISTSERV (i.e., in its SMTP_FORWARD_n= settings) is authorized to send from the organization's domain, may use the new DMARC_NO_REWRITE configuration variable to prevent posters' addresses in their own domain from being rewritten.

Examples:

unix:	`DMARC_NO_REWRITE="EXAMPLE.COM *.EXAMPLE.COM"`
	`export DMARC_NO_REWRITE`

Win:	`DMARC_NO_REWRITE=EXAMPLE.COM *.EXAMPLE.COM`

The default is the null string, that is, LISTSERV *will* perform a DMARC rewrite for any domain configured with "p=reject" or "p=quarantine" in its DMARC DNS record.

Section 14 GDPR and LISTSERV

The European Union's General Data Protection Regulation (GDPR) has been in effect since May 25, 2018. GDPR doesn't just affect enterprises in the EU, though; theoretically at least, any entity that stores personal information pertaining to an EU citizen – regardless of whether or not that entity does business in the EU – is also subject to its provisions regarding personal data storage. If you're not sure about the basics of GDPR, you might want to take a look at our 2017 newsletter article entitled "Ready for GDPR? Test Your Knowledge, Get The Facts."

GDPR requires, among other things, that a company must be able to provide on demand a report in a common machine-readable format (such as XML) which lists every instance of a customer's personal data held by that company. For LISTSERV, that can be a tricky prospect, because personal data may be held in list archives, in changelogs, and of course in subscription lists themselves.

In response, L-Soft developed a PowerShell script which, using either the LCMD.EXE or LCMDX.EXE command interfaces that ship with the Windows version of LISTSERV, can pull the relevant data using standard LISTSERV commands and methods, and produce an XML report containing the results. While the script itself is Windows-specific, by using the LCMDX.EXE option (which communicates directly with the server's TCPGUI port), it is possible to generate reports from any unix-based LISTSERV site as well, so long as the site has the LISTSERV web interface enabled.

The script also works under PowerShell Core 6.1 and later, making it possible to run the script from Linux and MacOS workstations by using the unix version of 'lcmdx'.

Contents:

14.1 Technical requirements, downloading, etc.

14.1.1 Minimum PowerShell Version Required

The GDPRSCAN requires (on Windows) at least PowerShell Version 3.0 on the machine from which it will be executed. PowerShell 5.x or greater is preferred. For Linux and MacOS, you should install the latest version of PowerShell Core, which at this writing is 7.1.2.

14.1.2 PowerShell Execution Policy (Windows)

Note that the Microsoft Windows version of PowerShell has a default execution policy of "Restricted", that is, PowerShell accepts only interactive commands and will not run scripts. Typically this results in an error something like the following:

```
PS E:\listserv\main> .\gdprscan.ps1
.\gdprscan.ps1 : File E:\listserv\main\gdprscan.ps1 cannot be loaded because running
scripts is disabled on this system. For more information, see
about_Execution_Policies
at
http://go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ .\gdprscan.ps1
+ ~~~~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
```

In order to run the GDPRSCAN script, you must do one of two things:

- If you are running PowerShell from a standard Windows "run as administrator" command prompt, you can bypass the default execution policy by adding "-ExecutionPolicy Bypass" to the command, like this:

  ```
  E:\LISTSERV\MAIN>PowerShell -ExecutionPolicy Bypass -File .\gdprscan.ps1
  ```

- If you are running GDPRSCAN in a PowerShell console, you will need to elevate the execution policy level for the CurrentUser scope to at least "RemoteSigned". This can be done as follows:

  ```
  PS E:\listserv\main> Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
  ```

  You will be prompted to ensure that this is really what you want to do.

14.1.3 Downloading the script and associated files

The script is available for download from http://download.lsoft.com/downloads/gdprscan/gdprscan.zip and comes bundled with copies of LCMD.EXE, LCMDX.EXE, and the source code for LCMDX (lcmdx.c) for users' convenience. (We don't provide an executable copy of lcmdx for Linux/MacOS, because it's usually best to compile and link the code locally against your existing libraries. We'll provide instructions for doing that below.)

14.1.4 Prerequisites

- o LISTSERV change-logging is NOT enabled by default. Change-logging MUST be enabled in LISTSERV in order to provide changelog reports. For information on how to set up the system-level changelog, see this link. For information on how to set up list-level changelogs, see this link.
- o For Windows, a reasonably-recent version of PowerShell (5.x or later is preferred). For Linux or MacOS users, the latest version of PowerShell Core should be used.
- o LISTSERV 16.5 or later, with a build date of 9 Apr 2018 or later, is required in order to run

changelog reports. Earlier builds will produce a message in the XML stating that changelog reports cannot be run because the installed LISTSERV version does not support them.

- LISTSERV POSTMASTER-level access is required to run the comprehensive, server-level reports.
- List owners may use the script to run reports against lists they own. However, such a report may not fully meet the GDPR criteria if the target address is subscribed to lists on the server which are not owned by the script invoker.

14.1.5 Support

Customers with paid-up LISTSERV maintenance may obtain help and report problems with the script by emailing support@lsoft.com.

14.2 Windows: Installing, configuring, executing

14.2.1 Windows: Installing the script

Once you have downloaded the files, you can unpack them into any convenient directory, preferably one that is in your PATH. For instance, you may prefer to install them in a directory called C:\%USERPROFILE%\PROC, or even C:\PROC. If you are installing the files directly onto the LISTSERV machine, you may even unpack them into \LISTSERV\MAIN if you so choose. The most important point to observe is that LCMD.EXE and LCMDX.EXE must be available to the script, so they must be either placed in the same directory with the script or found somewhere in your PATH.

Note: It is likely that you will have to unblock the script before PowerShell will allow it to be executed. This can be done in one of two ways:

- Open a PowerShell prompt, change to the directory where you have unpacked the files, and execute “Unblock-File gdprscan.ps1”; or
- Right-click the gdprscan.ps1 file in Windows Explorer, choose “Properties”, click the “Unblock” button found at the bottom of the “General” tab

14.2.2 Windows: Named Pipes or TCPGUI?

You will need to decide whether GDPRSCAN will be using LCMD.EXE (which is a named-pipes interface) or LCMDX.EXE (which is a TCP/IP interface that talks to the TCPGUI port on the LISTSERV machine, in a similar manner to the way the WA interface works).

Note: If you are using the script to pull GDPR reports from a unix LISTSERV site, you *must* use LCMDX.EXE, as the named-pipes interface used by LCMD.EXE is only useable with the Windows version of LISTSERV.

The interface is chosen by specifying an optional value for the -Method (minimum abbreviation: -m) parameter on the command line. Either

```
-Method LCMDX
```

or

```
-Method LCMD
```

can be used. The default is “LCMDX”.

Typically, the named-pipes interface will work only if the machine running the script is in the same Windows domain as the LISTSERV server machine, and you have not explicitly disabled named pipes on either your Windows client machine or the LISTSERV server machine, and so long as named pipe connections are not otherwise blocked from either end. You may also need to make a minor configuration change to LISTSERV, as by default, named-pipe requests are presumed to be coming from invoking_userid@NODE – for instance, john.doe@LISTSERV.EXAMPLE.COM. This may or may not be what you want. You can use LISTSERV's CMDPIPE_HOSTNAME= site configuration variable to change the hostname side of the address. For instance, if your enterprise addresses take the form of local_part@EXAMPLE.COM, you could set CMDPIPE_HOSTHAME=EXAMPLE.COM and restart LISTSERV to pick up the change.

The advantage to using named pipes is that the LISTSERV named pipes interface is secure and does not require password authentication. If password authentication is desired, or if you are using the script to run GDPR reports on a machine that is external to your local Windows domain, or is running on a unix machine, you will have to use the LCMDX.EXE TCPGUI interface instead.

14.2.3 Windows: Where to write the XML report?

By default, GDPRSCAN will write the resulting XML report to your Windows desktop. This may or may not be optimal for you, so there is an option to change it. If you wanted to change it to your "My Documents" directory, simply use the -XMLpath (minimum abbreviation: -x) to set it accordingly:

```
-XMLPath 'C:\Users\youruserid\My Documents'
```

14.2.4 Windows: Executing the script

Once you have installed the script and made any needed changes to the LISTSERV configuration, you will execute the script like this (optional command line arguments shown in square brackets []):

With LCMD (Named Pipes):

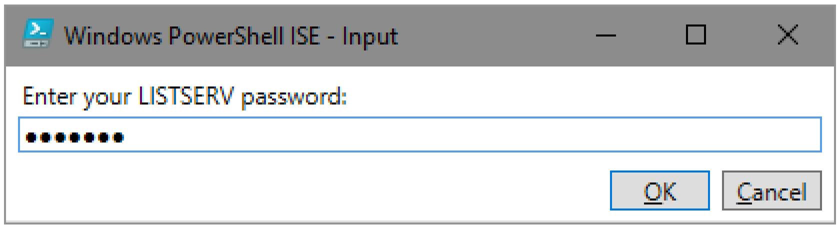
```
PS C:\PROC > .\gdprscan.ps1 -s listserv-hostname -t target-email [-d ALLlists |
SYSTEM | FULL]
```

With LCMDX (TCPGUI):

```
PS C:\PROC > .\gdprscan.ps1 -s listserv-hostname -t target-email -p postmaster-email
[-d ALLlists | SYSTEM | FULL]
```

When using the script with LCMDX, you will have to provide one additional pieces of information to the script after running it – the LISTSERV personal password corresponding to the postmaster-email address you are using. The password is obtained securely (see the examples below) and is stored as a secure string while the program is running.

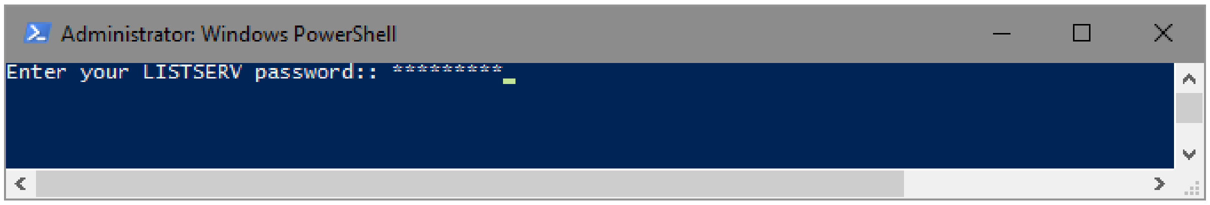
Example using the PowerShell ISE:



Enter your LISTSERV password:

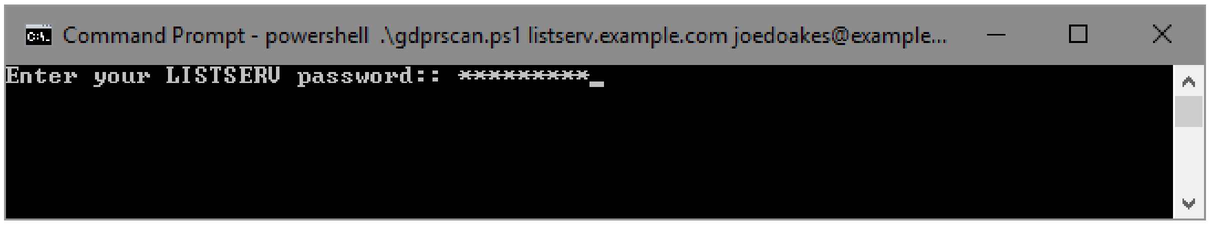
```
•••••••
```

Example if executed from a PowerShell prompt (not the ISE):



```
Enter your LISTSERV password:: *********_
```

Example if executed from a Windows command prompt:


Command Prompt - powershell .\gdprscan.ps1 listserv.example.com joedoakes@example...

```
Enter your LISTSERV password:: **********_
```

The script will continue to execute after you hit <return>.

The final command line argument is optional, defining the depth of the changelog scan. For a normal (default) scan, this argument is not used. Changelog scanning levels are defined as follows:

Default (no argument)	Only the *listname*.changelog files for the lists to which the target email address is subscribed are scanned.
ALLlists	(minimum abbreviation: "ALL") All *listname*.changelogs are scanned, regardless of whether the target email address is subscribed. (This may pick up information on lists to which the target email was subscribed in the past.)
SYSTEM	Same as the default, plus SYSTEM and NOLIST-* changelogs are scanned, if present.
FULL	All changelogs on the server are scanned.

If specified, these levels are mutually exclusive; only one may be specified per run.

The options are presented above in ascending order of how much time they will typically take to execute. On one L-Soft server, the ALL option resulted in a 51-minute-long scan for a single user; however, significant network latency may have contributed to that test. Another L-Soft server with a very large SYSTEM.CHANGELOG processed the ALL option for a single user in 15 minutes.

Typically, scanning changelogs other than those belonging to the list(s) to which the target address is subscribed is an expensive operation, there may be little if any personal information for the target address found in them, and it may simply not be desirable to run that deep of a scan.

GDPR does not require data controllers to spend an unlimited amount of time on requests, and therefore, L-Soft has left the decision on depth of scan up to the customer.

14.3 Linux/MacOS: Installing, configuring, executing

14.3.1 Linux/MacOS: Installing the script

Note: L-Soft no longer supports MacOS for new installations of LISTSERV. However, for customers who are still running recent versions of LISTSERV on MacOS, we have continued to publish these instructions. References to MacOS will be removed after the next full version of LISTSERV is released.

Installing the script on a Linux or MacOS machine presumes that you have already installed the latest version of Microsoft PowerShell Core, which at this writing is 7.1. Microsoft have published articles on that:

For Linux: https://docs.microsoft.com/en-us/powershell/scripting/install/installing-powershell-core-on-linux?view=powershell-7.1

For MacOS: https://docs.microsoft.com/en-us/powershell/scripting/install/installing-powershell-core-on-macos?view=powershell-7.1

You will also need to compile the lcmdx.c source code which is included in the gdprscan.zip bundle. This can be as simple as opening a terminal box and issuing the command

```
[root@linuxbox ~]# gcc -O lcmdx.c -o lcmdx
```

at the prompt. (Note that this also presumes that you have the 'gcc' compiler installed.)

If you are installing GDPRSCAN on a machine that is running LISTSERV, lcmdx may already be compiled; if not, check the $LSVROOT directory for lcmdx.c, and then simply run `make lcmdx' to compile and link the executable.

Once you have lcmdx compiled, you can either keep it in the same directory with gdprscan.ps1, or you can copy it into /usr/local/bin or some other directory in your $PATH. You will also want to set appropriate ownership and permissions for lcmdx. For instance, `chmod 755 lcmdx; chown root:root lcmdx' will result in the following:

```
-rwxr-xr-x.  1 root     root 13384 Apr 11 10:42 lcmdx
```

which should allow any user to execute lcmdx. This may be more extreme than you prefer; the only constraint is that the ownership and permissions must be set so lcmdx can be executed by whomever needs to run gdprscan.

14.3.2 Linux/MacOS: Where to write the XML report?

By default, GDPRSCAN will write the resulting XML report to your Windows desktop. This may or may not be optimal for you, so there is an option to change it. If you wanted to change it to your “Documents" directory, simply use the -XMLpath (minimum abbreviation: -

x) to set it accordingly:

```
-XMLPath '/home/myuserid/Documents'
```

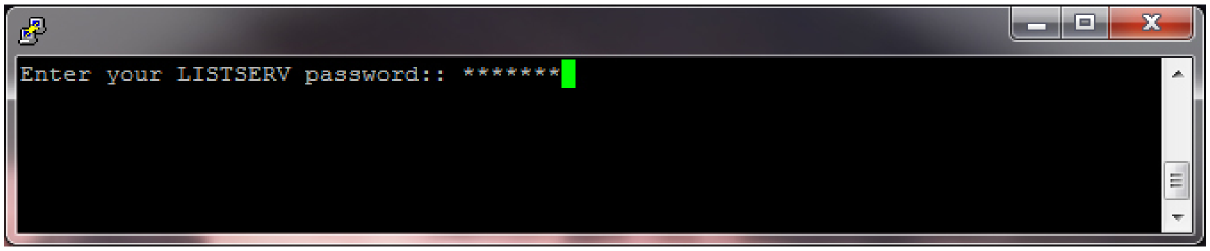
14.3.3 Linux/MacOS: Executing the script

Once you have installed the script, you will execute the script like this (optional command line arguments shown in square brackets []):

```
PS /home/you > ./gdprscan.ps1 -s listserv-hostname -t target-email -p postmaster-
email [-d ALLlists | SYSTEM | FULL]
```

You will have to provide one additional piece of information to the script after running it – the LISTSERV personal password corresponding to the postmaster-email address you are using. The password is obtained securely and is stored as a secure string while the program is running.

Example:



```
Enter your LISTSERV password:: *******
```

The script will continue to execute after you hit <return>.

The final command line argument is optional, defining the depth of the changelog scan. For a normal (default) scan, this argument is not used. Changelog scanning levels are defined as follows:

Default (no argument)	Only the *listname*.changelog files for the lists to which the target email address is subscribed are scanned.
ALLlists	(minimum abbreviation: “ALL”) All *listname*.changelogs are scanned, regardless of whether the target email address is subscribed. (This may pick up information on lists to which the target email was subscribed in the past.)
SYSTEM	Same as the default, plus SYSTEM and NOLIST-* changelogs are scanned, if present.
FULL	All changelogs on the server are scanned.

If specified, these levels are mutually exclusive; only one may be specified per run.

The options are presented above in ascending order of how much time they will typically take to execute. On one L-Soft server, the ALL option resulted in a 51-minute-long scan for a single user; however, significant network latency may have contributed to that test. Another L-Soft server with a very large SYSTEM.CHANGELOG processed the ALL option for a single user in 15 minutes.

Typically, scanning changelogs other than those belonging to the list(s) to which the target address is subscribed is an expensive operation, there may be little if any personal information for the target address found in them, and it may simply not be desirable to run that deep of a scan.

GDPR does not require data controllers to spend an unlimited amount of time on requests, and therefore, L-Soft has left the decision on depth of scan up to the customer.

14.4 Command line parameter reference

Command line parameters for GDPRSCAN are non-positional in nature. Each parameter requires an identifying flag. If no parameters are provided at run-time, certain basic assumptions are made, and you will be prompted for values for the parameters marked in the following table as "Mandatory". A basic sample command line would be something like

```
.\gdprscan.ps1 -s listserv.example.com -t joe@example.com -p admin@example.com -v f
```

While the parameters are non-positional, specifying them in the order shown without the parameter flags also works. However, if the flags are *not* used, you MUST provide a non-blank, valid value for each parameter. The only exception is for the -Server parameter; the server name may be specified in the first position without a flag if the rest of the parameters used are specified with flags. For instance,

```
.\gdprscan.ps1 listserv.example.com -v f -p admin@example.com -t joe@example.com
```

works, even though the parameters following the server hostname are specified "out of order."

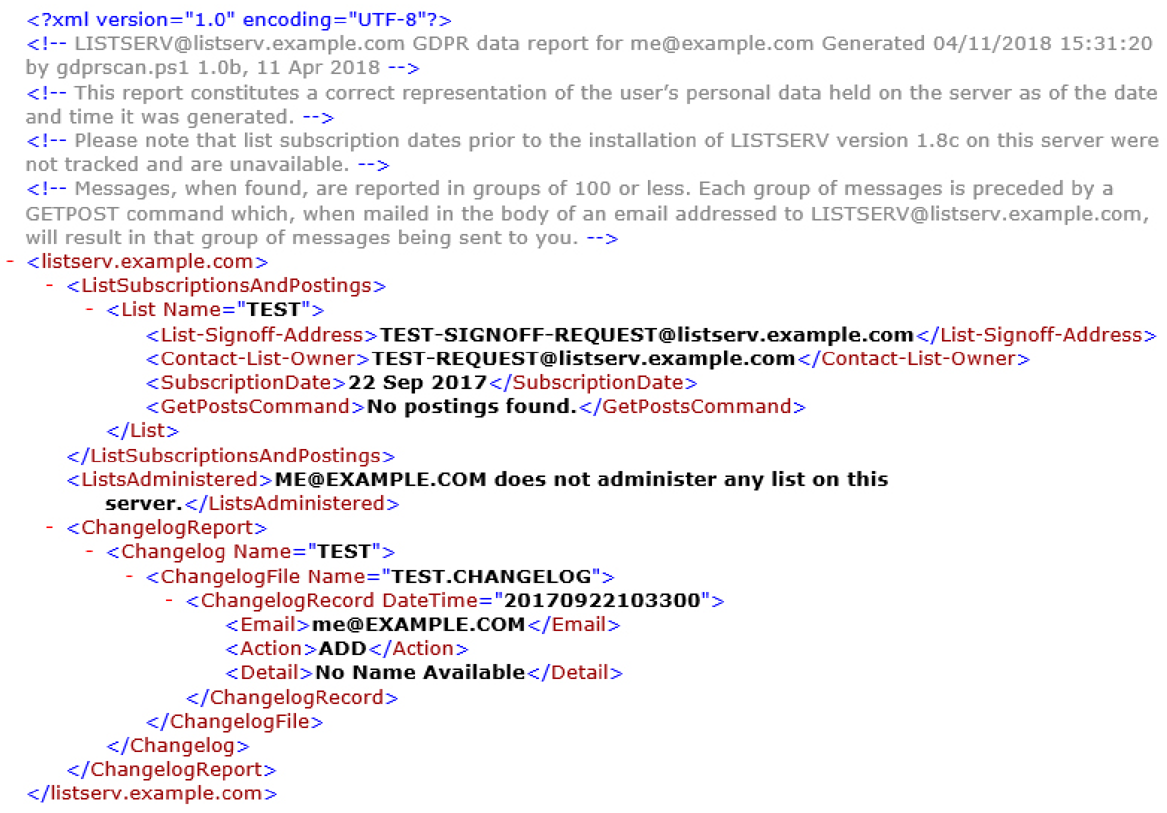
Parameter	**Alias**	**Default**	**Description**	**Mandatory?**
-Server	-s	none	For LCMDX, the fully-qualified domain name (FQDN) registered in DNS for the target LISTSERV server. For LCMD, the NETBIOS name of the LISTSERV machine. For LCMDX only: If the LISTSERV site configuration variable TCPGUI_PORT= has been configured to a value other than the default of 2306, you must also specify the port number here in the usual way, e.g., if you have set TCPGUI_PORT= 42306, then you must specify it thusly: **`-s listserv.example.com:42306`**	**YES**
-TargetEmail	-t	none	The email address to be searched.	**YES**
-PostmasterEmail	-p	none	The email address of the person running the report. Must be either a LISTSERV postmaster or a list owner. List owners have access only to information pertaining to their own lists.	**YES**
-VerboseOutput	-v	True	Determines whether to echo the report information back to the console screen. As the reports can become extremely verbose, it may be preferred to disable	No

			verbose output by setting it to “False”. Setting this parameter to False will result in only very basic console output, sufficient to assure the operator that the report is running. “T” and “F” are acceptable values and are case-insensitive.	
-ReportDepth	-d	none (basic changelog report)	One of “ALLlists”, “SYSTEM”, or “FULL”, depending on the depth of the changelog reporting desired. The default is to not specify a report depth, which results in a changelog report being run only on the list-level changelogs for the lists to which the target email is currently subscribed.	No
-Method	-m	LCMDX	Windows only. Either “LCMDX” (TCPGUI) or “LCMD” (named pipes). Windows users should see the section above entitled “Windows: Named Pipes or TCPGUI?” for more information. Linux and MacOS users will always use LCMDX and do not need to specify a value for this parameter.	No
-XMLPath	-x	none (Windows Desktop directory or Linux/Mac OS current directory)	The path to the location of the resulting XML report. The default is to not specify a value, which results in the report being written to the Windows desktop directory or to the Linux/MacOS current directory. If specified, the path should be enclosed in single quotes.	No

14.5 Output

14.5.1 Sample Output

A (very minimal) sample XML report generated with GDPRSCAN looks like this, when loaded into a browser:



```
<?xml version="1.0" encoding="UTF-8"?>
<!-- LISTSERV@listserv.example.com GDPR data report for me@example.com Generated 04/11/2018 15:31:20
by gdprscan.ps1 1.0b, 11 Apr 2018 -->
<!-- This report constitutes a correct representation of the user’s personal data held on the server as of the date
and time it was generated. -->
<!-- Please note that list subscription dates prior to the installation of LISTSERV version 1.8c on this server were
not tracked and are unavailable. -->
<!-- Messages, when found, are reported in groups of 100 or less. Each group of messages is preceded by a
GETPOST command which, when mailed in the body of an email addressed to LISTSERV@listserv.example.com,
will result in that group of messages being sent to you. -->
- <listserv.example.com>
  - <ListSubscriptionsAndPostings>
    - <List Name="TEST">
        <List-Signoff-Address>TEST-SIGNOFF-REQUEST@listserv.example.com</List-Signoff-Address>
        <Contact-List-Owner>TEST-REQUEST@listserv.example.com</Contact-List-Owner>
        <SubscriptionDate>22 Sep 2017</SubscriptionDate>
        <GetPostsCommand>No postings found.</GetPostsCommand>
      </List>
    </ListSubscriptionsAndPostings>
    <ListsAdministered>ME@EXAMPLE.COM does not administer any list on this
      server.</ListsAdministered>
  - <ChangelogReport>
    - <Changelog Name="TEST">
      - <ChangelogFile Name="TEST.CHANGELOG">
        - <ChangelogRecord DateTime="20170922103300">
            <Email>me@EXAMPLE.COM</Email>
            <Action>ADD</Action>
            <Detail>No Name Available</Detail>
          </ChangelogRecord>
        </ChangelogFile>
      </Changelog>
    </ChangelogReport>
  </listserv.example.com>
```

14.5.2 What data is included in the LISTSERV GDPR report?

Reports generated by GDPRSCAN are a "best effort" attempt to create a report containing all references to the requesting user as of the time the report is run. The report attempts only to determine the following:

- A list of the lists on the server to which the target address is currently subscribed
- A list of all postings found in each subscribed list's archives (if the list has archives) which were originated by the target address, including the post number, date/time, subject, and a GETPOST command for each 100 postings found for retrieval of those posts
- A list of all lists on the server for which the target address is currently a list owner, a list editor, and/or a list moderator
- A list of all list-level changelogs on the server which contain references to the target address
- A list of registration data held by LISTSERV, which contains the target address and/or the registered full name associated with it, along with the originating IP address recorded for the user’s last web interface login.
- (Optional) A list of all references found in the SYSTEM.CHANGELOG and any NOLIST-*.CHANGELOG files which exist on the server.

Note: There is no attempt made to search for the requesting user's email address in archived message bodies, nor is there any attempt made to "fuzzy match" the email address to other possible addresses used by the requesting user. GDPRSCAN also does not attempt to search archives of lists to which the requesting user may have been subscribed in the past, although typically that information will be exposed in POST records found in the list-level changelog reports, if changelogs exist for those lists.

Tip: Each invocation of GDPRSCAN is intended to produce output for one unique email address only. It would likely be possible to expand the script to cover other possibilities, but L-Soft believes that the script as it exists constitutes a reasonable search through LISTSERV data which does not potentially expose third-party personal information to the requestor.

14.6 Caveats and disclaimers

This information should not be considered as legal advice, and compliance is the responsibility of each organization.

L-Soft **strongly recommends** that each report be analyzed for any inappropriate data prior to being sent on to the requestor. L-Soft **does not guarantee or warrant** that any random piece of changelog data or subject line text from messages will not contain third-party personal information. **It is the sole responsibility of the person or organization generating the report to vet the report prior to sending it to the requestor.**

Use of the GDRPSCAN script and the associated LCMD/LCMDX LISTSERV interfaces constitutes the user's agreement to hold L-Soft international, Inc. harmless for any accidental or purposeful exposure of personal information consequent to its use.