Statement of intent: L-Soft to develop LISTSERV for unix

August 22nd, 1993

The reference number of this document is GM-9308-1.

Abstract: Due to strong demand from the academic market, L-Soft international intends to develop a unix® version of its LISTSERV product, in addition to the VMS™ and Windows NT™ versions that were already planned. In this document, you will learn more about L-Soft's plans for the unix® version - what to expect and when, how the unix® code will be developed, and so on.

Statement of intent

In order to better meet the needs of the academic community, L-Soft international intends to develop a unix® version of its LISTSERV product, which will provide:

- **Full compatibility** with the other implementations, upon final completion and with the exceptions noted below and in the remainder of this document.
- Staged availability, with roughly the same milestones as for the VMS™ version. See below for more information on the expected time scale.
- No NJE support, due to the lack of robust NJE implementations for unix®.
- System/hardware support based on actual customer demand. Within reason, we will port the code to any unix® flavour for which there is a sufficient customer base. We will not, however, support 16-bit machines or other small systems with "tiny" unixes.
- We may provide no support, or only partial support, for systems which are not "8-bit clean". In particular, 7-bit systems will not be able to participate in the global LISTSERV backbone.

This statement of intent adds to rather than subtracts from our existing plans. Support for VMS™ and Windows NT™ remains a strategic goal for the short and mid terms, respectively, and our plans regarding these operating systems are unchanged by the present announcement. We are simply adding unix® to the list of supported environments.

Mandatory legalese:

> *L-Soft international ("Licensor") makes no warranty, express or implied, in this announcement. In particular, Licensor does not guarantee the results of the development described in the present document in any respect, nor does Licensor represent or warrant that the development will ever be completed or produce usable and/or functional software, nor, in fact, that the development will ever be undertaken or even attempted. In no event shall Licensor be liable for damage consequential or incidental to non-completion, late completion, or failure of the development to meet the expectations of any particular customer, even if Licensor had been advised of the possibility of such damage.*

L-Soft international does not endorse or approve the use of any of the product names or trademarks appearing in this document.

Background information and development overview

The current (VM) version of LISTSERV is comprised of three main components:

1. The so-called "P-REXX Library", a low-level function library callable from PASCAL which provides, among other things, dynamic string manipulation functions similar to those available in

the REXX language. The P-REXX library is mostly system-independent and, with the exception of certain file manipulation functions, it can be easily ported to any system supporting 32-bit integers and pointers. The file system functions will require additional work on unix® but can still be ported. Under VM, the P-REXX library is implemented in S/370™ assembler and PASCAL (about 10,000 lines of code).

2. "Application" PASCAL code, most of which interfaces with the operating system only through the P-REXX library. A number of VM-specific interfaces are, however, provided by system-specific PASCAL code, plus some assembler interfaces which we will ignore for the purposes of this introduction, as they exploit VM-specific features which are not relevant to other environments. This PASCAL code totals about 34,000 lines.

3. REXX code, which like your average REXX program issues a system command every 5 lines and is thus not portable to any other system (the reason for having REXX code is that LISTSERV was originally written in REXX). There are about 19,000 lines of REXX, of which only 13,000 would need to be ported (the rest corresponds to initialization code, user exits, and functions such as the UDD or the RSCS line monitor which will not be ported to other systems).

L-Soft's strategy is to use the exact same source code for the system-independent parts of all implementations. This obviously decreases development costs, but above all it ensures full compatibility and consistency across the supported systems. So we will convert the remaining REXX code to PASCAL, port the library to all supported systems, and use that for all the system-independent code. System specific code will be implemented in the language which is most practical for the target system. A simple PASCAL preprocessor/translator will be developed to convert VS PASCAL language extensions to VAX™ PASCAL equivalents, so that the VM source code can be used directly under VMS™. For unix®, we will have to use a different technique, which will be described in the next section.

Once the system-specific code is written and the library has been ported, we should be in a position to build a first prototype, which will be fully compatible with the VM version for whatever functions are entirely implemented in PASCAL at that time, but will not support any of the functions that are still written in REXX on VM. This first prototype is unlikely to be robust enough to be released as a product, and it might still be missing a number of important functions, so "staged availability" will probably begin with the next prototype, which we expect to release 3-6 months later together with a new version of the VM code. This first usable version (code name "patchwork") is expected to be on the same level as the VM version for list-related functions, but it will only implement a limited subset of the file server functions and will not include any of the database functions. In other words, it will probably offer about the same level of functionality as the public domain list managers, but with full compatibility with LISTSERV and a higher degree of flexibility for list management.

As the conversion work progresses on VM, we will keep releasing new versions of the code for unix® and VMS™ with gradually increasing functionality, until all environments are at the same level and we are in a position to offer full functionality and compatibility across the entire range. The Windows NT™ development will start about 6 months before the planned availability of this final version, or earlier if there is enough demand; at any rate, it will be ready on time for the release of the common version.

Porting the PASCAL code to unix - technical overview

The only problem with this approach is that PASCAL compilers are not available for all unix® systems, and that they are at any rate not bundled with unix® and must be purchased separately. In order to solve this problem, we will need to develop a tool capable of somehow translating our PASCAL code to object code for the target systems. We plan to use a combination of the preprocessor we have already mentioned, and the public domain **`p2c`** (PASCAL-to-C) translator, modified as appropriate. While VS PASCAL is not supported by p2c, a feasibility study has shown that it should be possible and in fact relatively easy for the preprocessor to turn the original VS PASCAL code into something p2c can handle. If it turns out that we have overlooked a VS PASCAL feature which p2c does not support under any other name, we will try to find a way around it; if

necessary, we will just modify p2c to support it. We do not expect any serious problem with this high-level syntax translation.

p2c however is not a full-blown compiler and does not guarantee working code for all possible valid input programs. We are aware of this restriction and there is not much we can do at this point to ascertain its potential impact, since we would need to have at least a couple thousand lines of pre-processed VS PASCAL code, plus all include files and all necessary P-REXX library calls, to be able to make any serious test. This does not, however, mean that we have to act on blind faith. We have the source code of p2c, we can make changes to the VS PASCAL source to avoid constructs which do not translate properly to C, we can make changes to the p2c output before compiling it and, if everything else fails, we have a backup plan.

But we are reasonably confident that we will not run into insurmountable problems, because our PASCAL code is pretty straightforward, setting aside the techniques used to call system functions and bypass type checking (which p2c has no problem translating to C, since they map directly to simple C constructs). Most of LISTSERV is simple integer arithmetics, array scanning, straightforward control logic and function calls. We do not use **`write`** or **`read`**, built-in string operations other than for string constants, we do not have 3 levels of nested procedures sharing the same identifiers, we do not do any I/O via PASCAL primitives, do not use overlays, *etc.* This means we do not need to worry about compatibility problems between PASCAL **`read`** instructions and C **`sscanf()`** equivalents, about I/O error event handlers, about the mapping of built-in string manipulation primitives, and the like. If p2c does not work on our more complex input programs and we can neither fix p2c nor adjust the source program's syntax to bypass the problem, we can still make changes to the p2c output manually as a last resort.

And even if none of this were possible, we will still have the possibility to use genuine PASCAL compilers where they are available. In particular, there is a PASCAL compiler for DEC's™ Alpha AXP™ systems running OSF/1® that is compatible with the compiler we will be using for the VMS™ development. So even in the very worst scenario, we would still be able to generate object code for at least one of the major brands of unix® systems.

Rationale for not rewriting LISTSERV in C and abandoning PASCAL

Actually, the general advice L-Soft received from unix® experts in the academic community is that the VM version should be frozen, the VMS™ and Windows NT™ developments should be abandoned, and L-Soft should instead concentrate on rewriting everything in C or perl for unix®, because that is what everyone will be running in a few years and there is little or no need to support other systems if a unix® version is available, as all universities have unix® systems nowadays. While this would obviously provide a solution satisfactory to most unix® users, there are two indisputable problems with this approach (in addition to the controversial character of speculations on the future of unix®):

1. Rewriting the code in another language will take a lot more time (and cost a lot more money) than simply writing a preprocessor and using an existing compiler where one is available; in addition, it is much easier to convert REXX code to PASCAL using the specially designed P-REXX library than to standard C code based on the standard unix® library. In the best case, L-Soft would be throwing away 34,000 lines of working, fully tested and debugged code, which furthermore were guaranteed to be 100% compatible with the existing LISTSERV servers. The money to rewrite everything from scratch has to be found somewhere.

2. Most of the revenues L-Soft derives from its LISTSERV product come from service licenses paid by the existing VM sites (as opposed to right-to-use licenses from new customers). Since BITNET sites have been granted free indefinite access to version 1.7 of LISTSERV, L-Soft would not be able to make them pay yearly right-to-use charges for the existing version if development were to be frozen. The only way for L-Soft to collect license fees from its existing customer base is to keep improving the software its customers are using, *i.e.* the VM version.

So abandoning the existing PASCAL code and freezing the VM development to concentrate on writing a unix® version of LISTSERV would significantly increase L-Soft's development costs, while at

the same time making it lose virtually all the income it is now deriving from LISTSERV - not to mention the risk L-Soft would be taking by betting the product's future on the success of a single computing environment, for which free competing packages are furthermore available to anyone via anonymous FTP. Our cost estimate under this scenario calls for about 70-80,000 lines of C code (the current version of LISTSERV totals over 70,000 lines), representing the work of 4 FTE + one manager for about 18 months plus touch-up, with a total cost of $300-400,000 - a significantly higher figure than what we expect to spend to port LISTSERV to three new environments, while at the same time continuing to receive license fees from existing customers for new releases of the VM version.

Planned availability dates and development chart

In order to give you a better picture of the development plan and time scale, we have put all the information in one chart - or rather, three charts, one for each year from 1993 to 1995, to avoid page length problems. These charts call for a few comments before you read them:

- The VM development and REXX to PASCAL conversion is an ongoing effort. When you see "Development" in the VM column, it means we will be working on whatever is most urgently needed for the development work on the other systems.

- The reason the unix® development is initially one quarter late is that it is much easier to test the result of the necessary restructuration and conversion of the VM code on VMS™, using a genuine, trustworthy PASCAL compiler and what is generally considered to be the best debugging environment in the industry, than on a unix® system with p2c. If we find a bug on VMS™ at this stage, we know it is our code and not a glitch in p2c, and we will get the level of diagnostics and support one can expect from a quality compiler. Once all the problems have been ironed out, we will try out the code on unix® with the knowledge that all problems are most likely due to either our pre-processor, p2c, or the unix® specific code, and not to some VM idiosyncrasy we overlooked in the system-independent code.

- The time scale is based on L-Soft's current employment plans, on our contractual obligations and those of our employees, and on the requirements of the other L-Soft product lines and development projects. It accounts for the fact that the VM version of LISTSERV will continue to be improved during the course of the porting effort; in other words, this is not the time required to make a unix® version of release 1.7f with all development activities frozen, but the time we expect to need to be in a position to offer release 2.0a on four different environments. Finally, as in any other real world situation, we may slip a deadline due to other projects or plain bad luck, and conversely we may hire more staff for a new project and end up with some extra manpower we can use to speed up some of the items in the table.

1993	VM	VMS	unix	Windows NT
3Q93	Release **1.8a**	—		—
4Q93	Development	Port P-REXX library		—

1994	VM	VMS	unix	Windows NT
1Q94	Development	Begin work on system specific code	Touch-up library (file system calls)	—
2Q94	Release **1.8b**	First prototype (not released)	Begin work on system specific code	—

3Q94	Development	Touch-up system specific code; conduct beta-test while waiting for VM development	First prototype (not released); fix potential p2c problems	—
4Q94	Development	**Patchwork** stage released	Wait for early feedback on VMS *Patchwork*	—

1995	**VM**	**VMS**	**unix**	**Windows NT**
1Q95	Release **1.8c** with new file server system	**Libris** stage released - full file server functions, no database	**Patchwork** stage released early in quarter, then **Libris**	—
2Q95	Development	Mostly waiting for VM development; ideal time to start work on NT		Port P-REXX library and start writing system specific code
3Q95	Release **1.8d** with database functions in PASCAL	**Compendium** stage released (with database functions) 95% compatibility with VM		Finish system specific code; build and test prototype
4Q95 – 1Q96	Release **2.0a** All common functions now in PASCAL	**Stronghold** stage released **Full compatibility across the entire range**		
