R. Roach - [illegible]

To: Distribution

From: Paul Green
Bill Silver

Date: June 10, 1977

Subject: Lister, a WORDPRO Tool for List Processing

INTRODUCTION

This memorandum describes proposed new commands for list processing. These commands implement a simple data base manager that can be used in a word processing or office management environment. They are designed to interface with other word processing tools and will become an integral part of Multics WORDPRO.

Prototypes of these list processing commands exist today on Multics. They are used now to produce reports, mailing lists, and high quality form letters. It is the intention of the authors to have these commands become standard Multics products.

This memorandum contains the following sections:

Please send all comments and suggestions on this memorandum to the authors.

Send mail to: Paul Green
Honeywell Information Systems
575 Technology Square.
Cambridge, Mass. 02139

or send Multics mail at M.I.T or System M to:

Green.Multics

or call: (617) 492-9332
HVN 261-9332

LIST PROCESSING

A list is a series of names, words, numbers, etc. set forth in order. List processing involves the maintenance, sorting, and selection of items in a list and the production of documents that use this information.

List Processing and Word Processing

The definition of list processing given above is valid only in the context of word processing. Its definition may be quite different when used in other fields of computer science.

In the commercial marketplace, word processing products such as IBM Office System 6 and Wang Word Processor 10, 20, 30 have defined list processing to be a function of word processing and this definition has become an industry standard.

List Processing Functions

The main functions of list processing are:

- list maintenance, i.e., input and update of information in a list
- sorting
- selection
- report generation

An example of the use of list processing is a dental office that maintains a list of all patients serviced by that office. The data maintained for each patient might include the patient's name, address, phone number, date of last visit, etc. When a patient first visits the office, the patient is added to the list. This involves inputting information about this patient. This information may be updated later, for example, on each subsequent visit, the date of last visit is updated for that patient.

This dental patient list can be used to produce various documents. For example, a report listing the name, address, and phone number of all patients, sorted alphabetically by patient name. Or, a form letter reminding the patient to visit the office for a checkup. This letter is sent to those selected patients that have not visited the office for six months or more.

LISTER

Lister is a set of commands that maintain and process online lists of information.

Lister can be used to produce simple reports. Lister interfaces with the Multics mail facility and thus can be used to distribute information in an "electronic mail" environment. Lister interfaces with the WORDPRO formatter, runoff, and thus can be used to produce reports and form letters of the highest quality.

Lister Lists

The structure of a Lister list is simple. This simplicity is important since the people using Lister may not be computer professionals. The four elements in a Lister list are:

1. List: The whole list. A list is comprised of one header record and a series of data records. In the dental office example, the whole patient file is called the list.

2. Header Record: A list must contain one header record. It is the first record in the list. In the header record are specified the data record and field delimiter characters. Also specified are the names of the fields contained in the data records.

3. Data Records: A list may contain any number of data records. A data record is comprised of fields. In the dental office example, each data record in the list contains information about one patient.

4. Fields: Each data record is comprised of one or more fields. In the dental office example, each data record contains the fields: name, address, phone number, and date of last visit.

Lister Files

Lister uses three types of files. Each file type is identified by its entryname suffix. A description of these Lister files is given below:

1. listin: An ASCII file used to input and update a list. It is identified by the entryname suffix ".listin". Records in the list can be added, deleted, or updated simply by editing this file with a text editor. (See the create_list command documentation for a description of the structure of a listin file.)

2. lister: A file containing the list in a form that can be processed by Lister. It is identified by the entryname suffix ".lister". All merging, trimming, sorting, selection, and document processing is performed on lister files. A Lister command is provided to create a lister file from a listin file.

3. listform: A file that defines the format of a document produced from a list. It is identified by the entryname suffix ".listform". Several listform files may be used with one list. One listform file may be used with several lists. (See the process_list command documentation for a description of the structure of a listform file.)

LISTER ENHANCEMENTS

Lister performs very special and necessary functions that complement the overall capabilities of Multics WORDPRO. Its simple list structure and its interface to runoff and the Multics mail facility are its most important features.

To some, however, its lack of complex structure and its lack of a query and update language may be considered serious shortcomings. To those, it must be said that Lister is not intended to compete with the Multics Data Base Manager. In fact, all attempts to "improve" Lister and make it more like a real data base manager should be resisted, simply to keep Lister simple.

There are, however, enhancements that can be made to Lister without altering its simple nature. Two such enhancements may be implemented in future versions of Lister and are described below:

- The format of lister files could be changed to make them indexed files (accessed via vfile_). Currently, lister files are not indexed files and are limited in size to one segment.

- A command interface for inputting and updating lister files could be provided. Listin files would no longer be needed.

SUMMARY OF LISTER COMMANDS

create_list	creates a lister file from a listin file.
expand_list	creates a listin file from a lister file.
merge_list	combines two lister files into a single lister file.
process_list	produces a document from selected records in a lister file. The document format is defined in a listform file. The documents may be output on a terminal, saved in a file, or mailed to other Multics users.
sort_list	sorts the records in a lister file.
trim_list	deletes selected records from a lister file.

MPM DOCUMENTATION

The remainder of this memorandum presents draft MPM documentation for the Lister commands.

Name: create_list, cls

The create_list command creates a lister file from a listin file. The operation performed by this command is the opposite of that performed by the expand_list command.

Usage

```
     create_list path
```

where:

1. path
   is the pathname of the listin file. If the entryname suffix ".listin" is not specified, then it is added. A lister file is created in the working directory with the same entryname as path, and with the entryname suffix ".listin" changed to ".lister". Any existing copy of this lister file is overwritten.

Notes

The creation of a lister file is the only Lister operation of performed on listin files. All other Lister operations are performed on lister files.

A listin file provides an ASCII representation of a list. It is used to input and update a list. Listin files are created and updated by using the WORDPRO text editor.

The format of a listin file is simple (see Appendix A). It consists of the following three parts:

1. Header Record: Specifies the record and field delimiter characters and the field names. The header record begins with the first line in the file. This line must contain just two characters. The first character is the record delimiter character that identifies the beginning of each data record. The second character is the field delimiter character that identifies the beginning of a field. They must be different characters. The record and field delimiter characters must be chosen from the following set of special characters:

```
                    !#$%&*=?@`¦~
```

Beginning on the second line is a list of the names of all fields that will be found in any data record. These field names must be separated by a comma. The last field name must be followed by a semicolon.

2. Data Records: The beginning of each data record is denoted by the record delimiter character. It is followed by a list of fields. A record may contain some or all of the fields defined in the header record. Fields not specified for a record are considered to be null. Duplicate fields are not allowed within a record.

3. Fields A field within a data record consists of two parts, the field name and the value of the field for that record. The field name must be preceded by the field delimiter character. Field names may contain alphabetic uppercase or lowercase characters, numeric characters, or the underscore "_" character. No other characters are allowed. One special field name "User_id" is reserved for "electronic mail" applications. It specifies a Multics user mailing address (Person_id Project_id).

   A field value is a string. It must be separated from its field name by at least one white space character (space, horizontal tab, or new line). The end of the field value string is denoted by a record delimiter character or a field delimiter character followed by a defined field name. All white space characters preceding or following the field value string will be removed from the string when the field is processed.

Name: expand_list, els

The expand_list command creates a listin file from a lister file. The number of records expanded is printed. The operation performed by this command is the opposite of that performed by the create_list command.

Usage

```
expand_list path {-control_args}
```

where:

1. path
   is the pathname of the lister file. If the entryname suffix ".lister" is not specified, then it is added. A listin file is created in the working directory with the same entryname as path, and with the entryname suffix ".lister" changed to ".listin". Any existing copy of this listin file is overwritten.

2. control_args
   can be chosen from the following:

   -brief, -bf
   suppresses the printing of the number of records expanded.

   -line_length n, -ll n
   specifies that the line length of the ASCII listin file is to be n characters. If this control argument is not specified, then a default line length of 132 characters is assumed.

   -compact, -cmp
   specifies that more than one field is placed on a line. A field is placed on a new line only if adding the field to the current line would exceed the specified line length. At least one field is placed on each line. If this control argument is not specified, then only one field is placed on each line.

Notes

The ASCII listin file created by this command has the following format:

- The first line contains the record and field delimiter characters.

- Beginning on the second line are the field names. They are separated by a comma and a space. A field name will be placed at the beginning of a new line if adding it to the current line would exceed the specified line length.

- Each record will begin with a line containing just the record delimiter character.

- Unless -compact is specified, each field is placed on a separate line and indented one space.

Name: merge_list, mls

The merge_list command combines two lister files into a single lister file. The number of records merged is printed. The merged lister file may be appended to, or may replace an existing lister file. The fields defined in the two lister files must be identical. The fields to be compared during the merge must be in ascending order. (See the sort_list command.) The field comparisons are performed without regard to case (uppercase letters compare equal to lowercase letters).

Usage

```
merge_list mas_path up_path {out_path} {-control_args}
```

where:

1. mas_path
   is the pathname of the master lister file. If the entryname suffix ".lister" is not specified, then it is added.

2. up_path
   is the pathname of the update lister file. If the entryname suffix ".lister" is not specified, then it is added.

3. out_path
   is the pathname of the output lister file. If the entryname suffix ".lister" is not specified, then it is added. If this argument is not specified, then the input lister file is replaced.

4. control_args
   can be chosen from the following:

   -brief, -bf
   suppresses the printing of the number of records merged.

   -field_names fn1 ... fni, -fn fn1 ... fni
   specifies that fields fn1 through fni are used as the controlling fields for the merge. Records are considered equal only if all fields are equal. The fields are compared without regard to case. If this control argument is not specified, then all fields are used to control the merge.

-add

copies into the output lister file all records from the master lister file plus all records from the update lister file. Thus records contained in both lister files are duplicated. The "add" operation is the default.

-and

copies into the output lister file those records in the master lister file that are also in the update lister file. Thus only duplicate records are copied. No records from the update lister file are copied.

-or

copies into the output lister file all records in either the master lister file or the update lister file. Duplicate records are copied only from the update lister file.

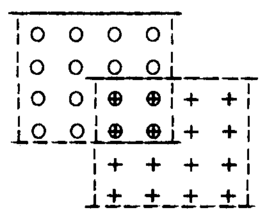
-sub

copies into the output lister file all records in the master lister file that are not also contained in the update lister file. Thus no duplicate records are copied and no records from the update lister file are copied.

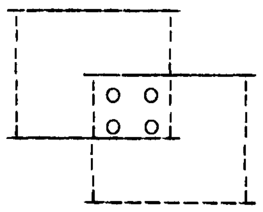
Notes

The diagrams below show how master and update lister files are merged for each of the four merge operations: "add", "and", "or", and "sub". In each diagram, the master lister file is represented by the upper left-hand "box", and its records are represented by "o" characters. The update lister file is represented by the lower right-hand "box", and its records are represented by "+" characters. The intersection of the two "boxes" represents duplicate records, i.e., records contained in both lister files. In each diagram, the records shown are those copied into the output lister file for that merge operation.

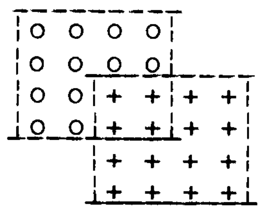
ADD



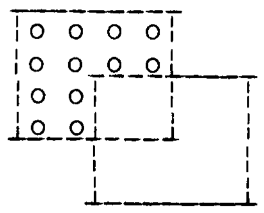
AND



OR



SUB



Name: process_list, pls

The process_list command produces a document from all or selected records in a lister file. The format of the document is defined in a listform file. The WORDPRO formatter, runoff, may be used to format the document. By default, the document is printed on the user's terminal. Alternatively, it may be saved in a file, or mailed to other Multics users.

Usage

```
process_list list_path {form_path} {-control_args}
```

where:

1. list_path
   is the pathname of the lister file to be processed. If the entryname suffix ".lister" is not specified, then it is added.

2. form_path
   is the pathname of the listform file that defines the format of the document. If the entryname suffix ".listform" is not specified, then it is added. If this argument is not specified, then a listform file will be used that has the same path as list_path, with the entryname suffix ".lister" changed to ".listform". (See the notes below for a description of the format of a listform file.)

3. control_args
   can be chosen from the following:

   -mail, -ml
   specifies that a document is produced for each record processed, and that each document is mailed online (via the Multics mail facility) to the Multics user specified in the field "User_id". Records that do not contain the field "User_id" are not processed.

   -output_file path, -of path
   specifies that the document produced by this command is saved in the segment specified by path. This control argument is in error if -mail or -runoff is specified.

-runoff, -rf
specifies that the document produced by this command is to be processed by runoff for final formatting. This control argument is the last control argument processed by process_list itself. All subsequent control arguments are treated as runoff control arguments and are passed to runoff.

-select "select expression", -sel "select expression"
specifies the records selected for processing. If this control argument is not specified, then all records in the list are processed. (See the trim_list command for a description of a "select expression".

-sort "sort string", -st "sort string"
sorts the records processed. The new ordering of the list is in effect only for the duration of the command. The lister file is not modified. If this control argument is not specified, then records are processed in the order in which they currently appear in the lister file. (See the sort_list command for a description of a "sort string".)

Notes

A listform file defines the format of a document. (see Appendix A). Information from the list may be copied into the document. Three section of a document may be defined. These three sections are:

1. BEFORE: This section is added to the document before any records are processed. It may contain any desired text, including runoff controls. The beginning of the BEFORE section is identified by a line consisting of the string "BEGIN BEFORE". The end of the BEFORE section is identified by a line consisting of the string "END BEFORE". If the BEFORE section is present, it must be the first section in the listform file.

2. AFTER: This section is added to the document after all records are processed. It may contain any desired text, including runoff controls. The beginning of the AFTER section is identified by a line consisting of the string "BEGIN AFTER". The end of the AFTER section is identified by a line consisting of the string "END AFTER". If the AFTER section is present, it must be the last section in the listform file.

3. RECORD: This section is added to the document for each record processed. It may contain any desired text, including runoff controls. It may also contain field value strings copied from the record being processed (see "Field Insertion" below). The beginning of the RECORD section is identified by a line consisting of the string "BEGIN RECORD". The end of the RECORD section is identified by a line consisting of the string "END RECORD".

Field Insertion

In order to insert information from the list into the document, a field name (enclosed in angle brackets) may be included in the text of the RECORD section. For example, <field_name>. For each record processed, that field name is replaced by the field's value string from that record. Any number of fields may be specified in a RECORD section.

An optional field width may also be specified. For example, <field_name,10> specifies that the value string of the field "field_name" is to occupy 10 character positions. If the current value string is less than the specified field width, then it is padded on the right with blanks. If the current value string is greater than the specified field width, then it is truncated on the right so its length is equal to the specified field width.

An optional field alignment may also be specified if a field width is specified. For example, <field_name,10,r> specifies that the value string of this field is to be right aligned within the 10 character field width. The alignment indicators "l" for left and "c" for center may also be specified. If no alignment is specified, then the value string is left aligned.

Angle Bracket Escapes

To place a single left angle bracket in the text enter two left angle brackets "<<". An unpaired right angle bracket is left as is in the text.

Name: sort_list, sls

The sort_list command sorts the records in the specified lister file. The records are sorted according to the fields specified in the "sort string" argument. (See the notes below.) Fields are sorted without regard to case, thus they will be sorted into alphabetical order and not ASCII order.

Usage

```
sort_list path "sort string"
```

where:

1. path
   is the pathname of the lister file sorted. If the entryname suffix ".lister" is not specified, then it is added.

2. "sort string"
   is one argument enclosed in quotes. It specifies how the records in the lister file are sorted. (See the notes below.)

Notes

A sort string specifies the record fields used to control the sort. It consists of one or more field specifications. The first field specification defines the primary sort field; the second field specification defines the secondary sort field; and so forth.

A field specification consists of two parts, a field name and an optional order control argument. The order control argument can be chosen from the following:

-ascending, -asc
: specifies that this field is to be sorted into ascending alphabetical order. If no order control argument is specified, then ascending order is assumed.

-descending, -dsc
: specifies that this field is to be sorted into descending alphabetical order.

Examples

```
"field1"

"field3 -dsc"

"field2 field1 -asc field3 -descending"
```

Name: trim_list, tls

The trim_list command deletes selected records from the specified lister file.

Usage

```
trim_list path "select expression"
```

where:

1. path
   is the pathname of the lister file being trimmed. If the entryname suffix ".lister" is not specified, then it is added.

2. "select expression"
   is one argument enclosed in quotes. It specifies the records selected for deletion. (See the notes below.)

Notes

A select expression specifies the records processed by a Lister command. Each data record in the specified lister file is tested to ascertain whether or not the record fulfills the selection criteria. Those that do are processed.

A select expression consists of one or more field comparisons. A field comparison involves comparing a test string to the specified field's value string in the current record. The comparison is made without regard to case, i.e., uppercase letters compare equal to lowercase letters. A field comparison is specified as follows:

```
field name  {comparison operator}  test string
```

The field name is the name of a field contained in the list. The reserved field name ":any" may be used to specify _any_ field in the record."

The optional comparison operator specifies what comparison is performed. The opposite comparison is performed if the comparison operator is preceded by "not". If no comparison operator is specified, then "contains" is assumed. The Lister comparison operators are:

contains	test string is contained in the field value string.
equal	test string is equal to the field value string.
greater	test string is greater than the field value string.
less	test string is less than the field value string.
nequal	test string is numerically equal to the field value string.
ngreater	test string is numerically greater than the field value string.
nless	test string is numerically less than the field value string.

The test string is the string that is compared to the field value string. The special test string ":null" is used to test whether or not the field is null, i.e., missing from the current record.

Several field comparisons may be specified in a select expression. Field comparisons are combined by the logical operators "and" and "or". The select expression will be evaluated from left to right with "and" operators evaluated first. Parenthesis may be used to specify the exact order of evaluation.

Examples

```
"field1 example"

"field1 contains example"

"field1 not foo"

"field3 not equal ""foo bar"""

"field2 less test"

"field1 not ngreater 123"

"field1 equal test and field2 nless 10"

"((field1 equal test) and (field2 nless 10)) or (:any :null)"
```

Appendix A
Sample Lister Files

patients.listin

```
$=
name,fname,lname,street,city,state,zip,phone,date,comment;
$
 =name John Doe
 =fname John
 =lname Doe
 =street 71 Pine Street
 =city Boston
 =state Massachusetts
 =zip 02020
 =phone (617) 991-7654
 =date 770520
 =comment you and your family well, including your pet rabbit Whitey
$
 =name Jane Smith
 =fname Jane
 =lname Smith
 =street 898 Smith Avenue
 =city Needham
 =state Massachusetts
 =zip 02112
 =phone (617) 992-4567
 =date 750713
 =comment you well after your seventeen month island hopping
cruise in the Lesser Antilles
$
 =name Francis Jones
 =fname Francis
 =lname Jones
 =street PO BOX 999
 =city Cambridge
 =state Massachusetts
 =zip 02139
 =phone (617) 888-7869
 =date 770131
 =comment you well
```

Appendix A
Sample Lister Files

addresses.listform

```
BEGIN BEFORE

     Dental Patient Addresses

END BEFORE


BEGIN RECORD

<name>
<street>
<city>, <state> <zip>
Tel:  <phone>
END RECORD


BEGIN AFTER

     Smiles Associates
END AFTER
```

Address Report

```
      Dental Patient Addresses

John Doe
71 Pine Street
Boston, Massachusetts 02020
Tel:  (617) 991-7654


Francis Jones
PO BOX 999
Cambridge, Massachusetts 02139
Tel:  (617) 888-7869


Jane Smith
898 Smith Avenue
Needham, Massachusetts 02112
Tel:  (617) 888-7869

      Smiles Associates
```

Appendix A
Sample Lister Files

letter.listform

```
BEGIN RECORD
.pl 40
.ll 55
.in 30
.nf
Smiles Associates
1001 Painless Avenue
Boston, Mass. 02003
(617) 987-6000
.sp 2
.in
<name>
<street>
<city>, <state> <zip>
.fi
.in

Dear <fname>:

     I hope this letter finds <comment>.
It has been over six months since your last visit to our office.
Please call and make an appointment to have a checkup.

.in 30
Keep smiling,


J. Puller, D.M.D.
.pa
END RECORD
```

Sample Letter

Smiles Associates
1001 Painless Avenue
Boston, Mass. 02003
(617) 987-6000

Jane Smith
898 Smith Avenue
Needham, Massachusetts 02112

Dear Jane:

I hope this letter finds you well after your seventeen month island hopping cruise in the Lesser Antilles. It has been over six months since your last visit to our office. Please call and make an appointment to have a checkup.

Keep smiling,

J. Puller, D.M.D.