To: Distribution

From: J. Falksen, P.L. Lyon

Date: June 1, 1978

Subject: Document SubSystem/LSS

The Document subsystem is an office-oriented mechanism for producing documents. It consists of a forms compiler, an editor, and a specialized command interface to compose.

A document is described as a "form". Entities useable within it are termed "levels". This grows out of the requirement of having level 1 headings, level 2 headings, etc. It is used in a more general sense; a "level" could be named something like "note" or "index".

The document system maintains 20 counters for reference in the document description. These counters are nested, that is, using a "level" which is associated to counter 4 causes this counter to be incremented by 1 and counters 5-20 to be set to zero. Each counter can be converted to one of five representations.

These commands make up the document subsystem:

make_form	compile a form description segment
set_form	make an form description segment current
newdoc	create a new document, then begin editing
olddoc	edit a section of an existing document
prdoc	print a document or a section (via compose)
setdoc	specify the current document
typos	check for possible typing errors
expdoc	do speedtype expansion
form	active function referenced within compose to get the document processing done

newdoc, olddoc, and prdoc share the concept of a current document. newdoc will set the current document. olddoc and prdoc can set and/or change the current document. setdoc does nothing but set the current document.

This MTB is not finished, it is being shelved. It is being published to document the work which was done in the design of such a system. There were not enough resources available to work it into a coherant, installable system. Some good ideas may be useable when manpower can be allocated to think about this type of facility at length.

A document description segment consists of 1 or more document descriptions of this general form:

```
            ..form name,alias,...

        [ ..#dd                ] ...
        [ marked_string        ]

        [ ..file etc.          ]

        [ ..data etc.          ] ...

        [ ..list etc.          ] ...

        [ ..start              ]
        [ marked_string        ]

        [ ..comp string        ]

        [ ..conv xx ...        ]

        [[ ..toc              ]]
        [[ marked_string      ]]

        { ..level etc.         } ...
```

Marked strings

A marked string is a sequence of compose data and marks. These are the marks available:

<<	give a <
<d>	give title d, 0-9
<dc>	give title d capitalized
<du>	give title d underlined
<duc>	give title d caps/underline
<dd>	give counter dd, 01-20
<#dd>	give copy string dd

CONTROL: ..#dd

This control defines a "copy" string for use at any point beyond it in the description file. dd is any two digits 00 - 99. It may be redefined at any time in the file.

CONTROL: ..file

This control defines the control file which is to be built by newdoc for a document. It must have one of these two general forms:

```
          xxxx<0>xxxx
 or       <#dd>
```

The contents of the copy string must have the form xxx<0>xxx.

The default is:

```
          .sr Save_Parameter "%Parameter%"
          ..level 1
          <0>.ur .ifi %Save_Parameter%
          ..level 0
```

The <0> is the only mark allowed. It represents the place where the .sr lines are to be inserted.

CONTROL: ..data

This control defines a datum which is needed to specialize the form. It has this general form:

$$\texttt{..data name} \left[\texttt{,} \left[(\underline{n}) \right] \left\{ \begin{matrix} \underline{\texttt{dd}} \\ \texttt{l}\underline{\texttt{dd}} \\ \texttt{c}\underline{\texttt{dd}} \\ \texttt{r}\underline{\texttt{dd}} \end{matrix} \right\} \right] \texttt{...}$$

$$\left[\texttt{..default xxxxx} \right]$$

$$\left\{ \begin{matrix} \texttt{..select yyyyy} \\ \left[\texttt{..or zzzzz} \right] \texttt{...} \\ \left[\texttt{marked\_string} \right] \end{matrix} \right\} \texttt{...} \quad \Bigg| \quad \left[\texttt{marked\_string} \right]$$

The name on the ..data line is the name which will be asked for by newdoc. The portion of this up to the first blank will be available as <0> in the associated marked_string's.

If any additional parameters are present, they represent lengths. dd is the maximum length allowed.

```
          ..data pa,34
```

means that the reply to asking for pa cannot exceed 34 characters.

```
          ..data title,25,25,20
```

means that the reply to asking for title is to be broken into 3 parts of max length 25, 25, and 20 respectively.

A repetition count may preceed any length specification. The above example could also be written:

```
          ..data title,(2)25,20
```

If newdoc cannot fit the reply into the specified area(s), it will tell the user what is left over and ask again.

If the length specification has the leading "lcr" then the result is to be padded out to the specified length with the data to be either left, center, or right in the field.

The ..default control specifies what the value is to be if the user replies with an empty line. If this is not specified, there is no default; the user MUST make a reply.

The ..select control specifies a valid reply string. If any ..select is specified, then the set of all of them for this datum constitutes the only responses acceptable. The associated marked_string is what to use if the given response it received. The ..or control allows a list of replies to be attached to a single_marked_string to save on typing.

If no ..select is specified, then a marked_string can still be specified. For example the name you want typed may contain spaces and so you want to use a different value for the variable name.

The default value for marked_string is:

```
          .sr <0> "<1>"
```

<1> thru <9> will have any internal " and * protected to keep compose happy. The <dd> form of mark is not allowed here.

For example, You have room in a header for 35 characters of title on 4 consecutive lines. On the first two lines, other fields need to be positioned beyond the title. This means that the first 2 lines must be of a fixed length. Also the first line is to be capitalized. This is how the statement could be entered:

```
          ..data title,135,135,35,35
          .sr title_1 "<1c>"
          .sr title_2 "<2>"
          .sr title_3 "<3>"
          .sr title_4 "<4>"
```

If a reply to "title:" were:

```
       Applications of fan-tailed swords to community tanks
```

then the corresponding lines in the control file would read:

```
          .sr title_1 "APPLICATIONS OF FAN-TAILED SWORDS  "
          .sr title_2 "to community tanks                 "
          .sr title_3 ""
          .sr title_4 ""
```

Or, you have to restrict a response to a certain set of values. The pre-printed form needs to have a mark in one of three boxes which indicates the security level.

```
          ..data security
          ..default controlled
          ..select proprietary
          ..or p
          .sr prop "X"
          .sr cont " "
          .sr free " "
          ..select controlled
          ..or c
          .sr prop " "
          .sr cont "X"
          .sr free " "
          ..select unrestricted
          ..or u
          .sr prop " "
          .sr cont " "
          .sr free "X"
```

CONTROL: ..list

This control defines a list datum. A list causes newdoc to accept multiple lines of response after the query.

$$\text{..list name} \left[, \begin{bmatrix} (\underline{n}) \\ [\underline{n}] \end{bmatrix} \begin{Bmatrix} \underline{\text{dd}} \\ \overline{\text{l}}\underline{\text{dd}} \\ \text{c}\underline{\text{dd}} \\ \text{r}\underline{\text{dd}} \end{Bmatrix} \right] \ldots$$

$$\left[\text{ marked\_string } \right]$$

Then name on the ..list line is the name which will be asked for before accepting the list. The portion of this up to the first blank will be available along with a numeric suffix as <0>. The sequence will be name_1 name_2 ... name_99 The number of items actually entered will be made in a variable name_ct.

Any additional parameters are of the same general form as for ..data. However, two kinds of repetitions are allowed. (n) means a required number. [n] means an optional number. Actually all after the first occurance of a [n] are considered optional.

The list of numbers represent the lengths for the list of replies. If the reply is larger than the specified length, it is truncated without comment.

The optional marked_string is the same as for ..data.

For example, you need a distribution list on a memo. You might define it as:

```
          ..list cc,(6)125,[93]25
```

This says that all replies are to be 25 characters long, the first 6 are not optional and must be padded. In response to:

```
          cc: (list, type "." to finish)
```

you reply:

```
          M. Snerd
          C. McMarthy
          J. Winters
          .
```

the corresponding lines in the control file will read:

```
          .sr cc_1 "M. Snerd                  "
          .sr cc_2 "C. McCarthy               "
          .sr cc_3 "J. Winters                "
          .sr cc_ct 03
          .sr cc_4 "                           "
          .sr cc_5 "                           "
          .sr cc_6 "                           "
```

CONTROL: ..start

This control specifies the control string necessary to begin processing of this form.

CONTROL: ..comp

This control specifies control arguments which must be given to compose by prdoc. For example, if a form needs to have page numbering

```
          4 of 27
```

there will have to be a pagenumber reference of

```
          %PageNo% of %lastpage%
```

and a final line of input which says

```
          .sr lastpage %PageNo%
```

However, this does not do any good unless the

```
          -pass 2
```

control is not given to compose. A form designer causes this to happen with

```
          ..comp -pass 2
```

CONTROL: ..conv

This control specifies what kind of conversions are to be applied to the counters. The allowable values for xx are:

ar	arabic
ur	uppercase roman
lr	lowercase roman
ua	uppercase alphabetic
la	lowercase alphabetic

If no ..conv is specified, then ar is assumed for all 20 counters.

CONTROL: ..toc

This control specifies the compose controls necessary to begin printing of the table of contents. The marked_string following is optional. If it is not present, this default is supplied:

```
   .sr PAGE_TAB %PageWidth% - 9
   .ur .htd TOC 10,%PAGE_TAB%" ."
   .htn ` TOC
   .brp 3
   .srm rl
   .spb 7
   .bbe 1
   ¦¦CONTENTS¦¦
   .spf 2
   .ur %FilaName%.toc
```

The controls in this string must be consistant with all controls specified in any ..toc present under a ..level.

CONTROL: ..level

This control specifies the name and description of a named level. It has this general format:

```
           ..level name

        [  ..counter n      ]

        [  ..delim x        ]

        [  ..title
           marked_string    ]

        [  ..notitle
           marked_string    ]

        [  ..toc
        [[ marked_string   ]]
```

The name on the ..level is the name by which this will be referenced. In the compose source it will be called via "..name". "name" may not be "form", "level", or "toc" as these names are reserved for commands to the form active function. All names defined on levels must also be added to the document system form.compin segment (with the compin suffix).

The ..counter control specifies that there is to be an associated counter. Whenever this level is used, one is added to this counter and all higher-numbered counters are zeroed.

The ..delim control specifies that there may be multiple parts to the title. The character x is what is to delimit them. If no delimiter is specified, or one is specified but there is no such character in the title presented, then the value is placed in <1> and <2> thru <9> are set to empty. There can be at most 9 parts in a title.

The ..title control specifies the compose string to be used if there is a parameter on the call to this level.

The ..notitle control specifies the compose string to be used if no parameter is on the call to this level.

If a marked_string is present with neither ..title nor ..notitle before it, it is used for both.

The ..toc control specifies that a table of contents entry is to be generated. It optionally supplies the control string which does this. If no marked_string is present, this is the default supplied:

```
          <to be determined>
```

For example:

```
          ..level manager
          ..delim ¦
          ..counter 2
          ..title
          .brp
          .sr section <02>
          .sr division "<3u>"
          .sr xxxxxx "<01>.<02>            "(1,10)
          .in 11
          .unl 10
          .ur %xxxxxx%<1c>, <2>
          Manager: <3> division since <4>.
          .spb
          .wrt tic <1>, <2>`%section%-%PageNo%
```

Making a call like this:

```
          ..manager Suggins¦Elwood P.¦Rocking Chair¦1934
```

could cause this to be the result:

```
          .sr section 8
          .sr division "Rocking Chair"
          .sr xxxxxx "16.8          "(1,10)
          .in 11
          .unl 10
          .ur %xxxxxx%SUGGINS, Elwood P.
          Manager: Rocking Chair division since 1934.
          .spb
          .ur .wrt tic Suggins, Elwood P.`%section%-%PagNo%
```

Using a Form

When newdoc creates a section, it initializes the segment with this line:

```
          ..form XXXX
```

where XXX is whatever you replied for the name of the form.

Any desired text and controls may be added after this line. In order to reference levels that are defined in the form you use either of these two forms:

```
          ..xxx
          ..xxx title string
```

xxx is tha name of the level. title string is the optional title field. If the level is so defined there may even be multiple parts to the title.

In some instances you will need to be able to set the counters to a certain value. This is done with this control:

```
          ..level n n ...
```

The first n is the number to be placed in the first counter, the second in the second, etc. If the value of n is "*" this indicates that the associated counter is to be left unchanged.

To set the first counter to 6 you would say:

```
          ..level 6
```

To set the third counter to 99 without disturbing the first 2:

```
          ..level * * 99
```

All counters higher than the last one mentioned are set to zero.

The table of contents (toc) is being created as the document is being processed. This means that it must be printed last. When it comes time to print it you use this control:

```
          ..toc
```

Name: make_form

The make_form command converts a form description source into a form description segment.

Usage

```
     make_form path
```

where:

1. path
   is the pathname of the form description source. The suffix form is added if not present.

Name: newdoc

The newdoc command creates a new document and then begins editing the first section (in INPUT mode).

Usage

```
newdoc document {section_count {section_names}}
```

where:

1. document
   is the name of a document to be created.

2. section_count
   is the number of sections to be created with sequentially numbered names. The section names will be 2-digit numbers from 1 to this number. If this number is zero, no numbered sections will be created. If not specified, 1 is assumed.

3. section_name
   is the name of a non-numeric section, such as an appendix. This is limited to 10 characters.

Notes

newdoc will begin by asking for the form name. Then by referencing the description of this form, it will ask for all needed data items and create the control file. This file is named document.compin. It will also create a driver file which lists all sections, in order, which are created. This file is named document.XX.compin. This file is used by prdoc when the whole document is to be printed.

After each reply to a data name, newdoc will ask "OK?". You reply "y" or "yes" if it is. Any other reply will cause the datum to be asked for again.

If a list is asked for, then each line of reply represents one entry. When all entries have been entered, enter "." alone on a line. In this mode, each line is accepted as-is. No verification is asked for.

The editing which may be done is as is described under olddoc.

Examples

To create a document named MonMoth with 5 sections and 2 appendices you could call newdoc like this:

```
          newdoc MonMoth 5 A B
```

Name: olddoc

The olddoc command is used to edit a section of an existing document.

Usage

```
olddoc {document {section}}
```

where:

1. document
   is the name of the document to be processed. If not present or is a null string, then the current document name is used.

2. section
   is the name of the section to be processed. If document is supplied but section is not, olddoc assumes "01".

Notes

The editing which may be done in olddoc is exactly what may be done with ted. When running in the LSS environment, the commands available are:

```
a M P x p gp gP gd g= d q Q s =
```

Example

To input section 2 of the document described under newdoc you would call:

```
          olddoc "" 2
 or       olddoc MonMoth 2
```

Name: prdoc

The prdoc command prints either all of a document or one section.

Usage

```
prdoc {document {section}} {-control_args}
```

where:

1. document
   is the name of the document to be processed. If not present or is a null string, then the current document name is used.

2. section
   is the name of the section to be processed. If section is not supplied, prdoc assumes the whole document.

3. control_args
   can be chosen from the following:

   -from N, -fm N
   starts printed output at page N. This control argument is mutually exclusive with the -page control argument.

   -indent {n}, -in {n}
   adds {n} spaces at the left margin of the output. This space is in addition to any indentation given with indent-left controls in the text. The default value of {n} is 0.

   -page n|n,n ..., -pg n|n,n ...
   specifies a blank separated list of selected pages to be printed. Each member of the list must be a single page, {n}, or a range of pages, {n,n}. The page numbers given must steadily increase without duplication. At least one page must be specified. Up to one hundred (100) pages may be specified. This control argument is mutually exclusive with the -from control argument. The default for this feature is OFF.

The following options imply online printing:

-stop, -sp
: waits for a newline character (ASCII NL) from the user before beginning the first page of output to the terminal and after each page of output including the last page. Any other characters typed are ignored, thus any forms positioning and top-of-form notes for special forms are easily accomplished. The default for this feature is OFF.

-wait, -wt
: waits for a newline character (ASCII NL) before beginning the first page of output to the terminal, but not between pages (see the -stop control argument above). The default for this feature is OFF.

The following options imply the output goes to a segment and then is dprinted:

-destination STR, -ds STR
: labels the output with the string STR, which is used to determine where to deliver the output. If this control argument is not given, the default is the requestor's Project_id.

-header STR, -he STR
: identifies the output by the string STR. If this control argument is not given, the default is the requestor's Preson_id.

-queue N, -q N
: prints output in proirity queue N ($N \leq 3$). If this control argument is not given, queue 3 is assumed.

-request_type STR, -rqt STR
: places output in the queue for requests of the type identified by the string STR. If this argument is not given, the default rquest type is "printer".

Name: set_form

The set-form command sets the current form description segment to the name specified as an argument.

Usage

```
     set_form {path}
```

where:

1. path
   is the pathname of a form description segment to be made current. This name is pushed on a list.

Note

If path is not specified, then the top name is popped from the list. When newdoc, prdoc, or form is called, it uses the top name on the list. If the list is empty then "form_descriptions" is automatically pushed on to the list.

Name: setdoc

The setdoc command sets the name of the current document without doing any other processing.

Usage

```
setdoc document {section}
```

where:

1. document
   is the name of the document to be processed. If not present or is a null string, then the current document name is used.

2. section
   is the name of the section to be processed.