Rexx Language Association
2012 Rexx Language Symposium

Part Two: Transforming THE to be more than JUST an editor by using Rexx macros

Les Koehler
14 May 2012

Table of Contents

Abstract

I will present more of my Rexx macros that expand the basic operations of THE from those of a typical Eastern Orthodox Editor to a powerful tool that multiplies the productivity of the user. Last year's presentation included some THE basics as background information before presenting the tools. This year I will only present advanced tools. This will be a live demonstration. If time permits, and there is interest, I will explore the coding techniques used.

Notes about this presentation

This presentation was written before the actual online demonstration given during the Symposium.

I've attempted to document the main points I want to demonstrate, both in prose and screen shots. Of course I can't guarantee a one-for-one correlation. There may be either more or less here!

This is not an "Introduction to THE". Some of THE's basic features were presented last year and will not be repeated.

Command recall

The *?* command recalls the previous command held in the command ring buffer. Multiple question marks will retrieve the Nth previous command and the "+" argument retrieves the next command held in the buffer.

Unlike Xedit, in THE the buffer for holding commands is shared with all the files in the ring. This can be a great help when making the same change, or doing the same search, to multiple files.

Since I prefer the Xedit style, I added an option flag in the **smart_enter** code to implement command recall as separate macros: **cmdsave** and **retrieve.**

This allowed two other enhancements:

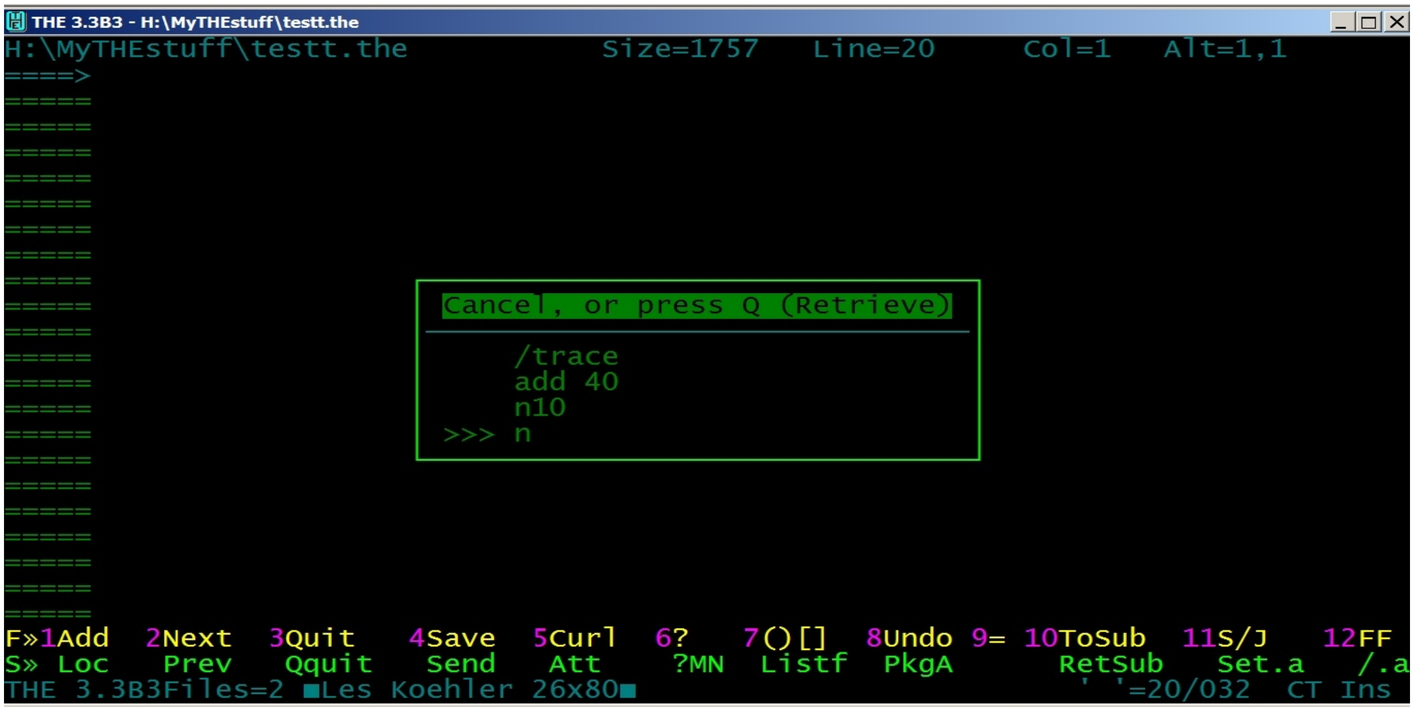
1. Treat each Directory as a unique file, even though they all appear to have the name: DIR.DIR.
2. Provide a Menu option. This idea came from emails exchanged with Wesley Miller, who's an X2 user and likes the menu that X2 presents.

   The Menu option comes in two flavors:

?MN	Displays a popup of saved commands to select from
?M	Displays the popup and sets the internal pointer to the item selected

Now command recall not only behaves like Xedit, but also has the menu behavior similar to that of X2.

Here is a trivial example I created in my usual playpen file so I could demonstrate the popup:



```
THE 3.3B3 - H:\MyTHEstuff\testt.the
H:\MyTHEstuff\testt.the          Size=1757    Line=20     Col=1    Alt=1,1
====>
=====
=====
=====
=====
=====
=====
=====
=====
=====                  Cancel, or press Q (Retrieve)
=====
=====                      /trace
=====                      add 40
=====                      n10
=====                  >>> n
=====
=====
=====
=====
=====
=====
F»1Add  2Next  3Quit   4Save  5Curl  6?   7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att   ?MN  Listf  PkgA       RetSub   Set.a   /.a
THE 3.3B3Files=2 ■Les Koehler 26x80■                    ' '=20/032  CT Ins
```

You select the command you want by using the arrow keys and then press ENTER. The command is put in the cmdline for further editing or execution, just as though you had typed it yourself.

Currently, the number of commands saved is 10.

Executing modified Rexx code without you having to SAVE it

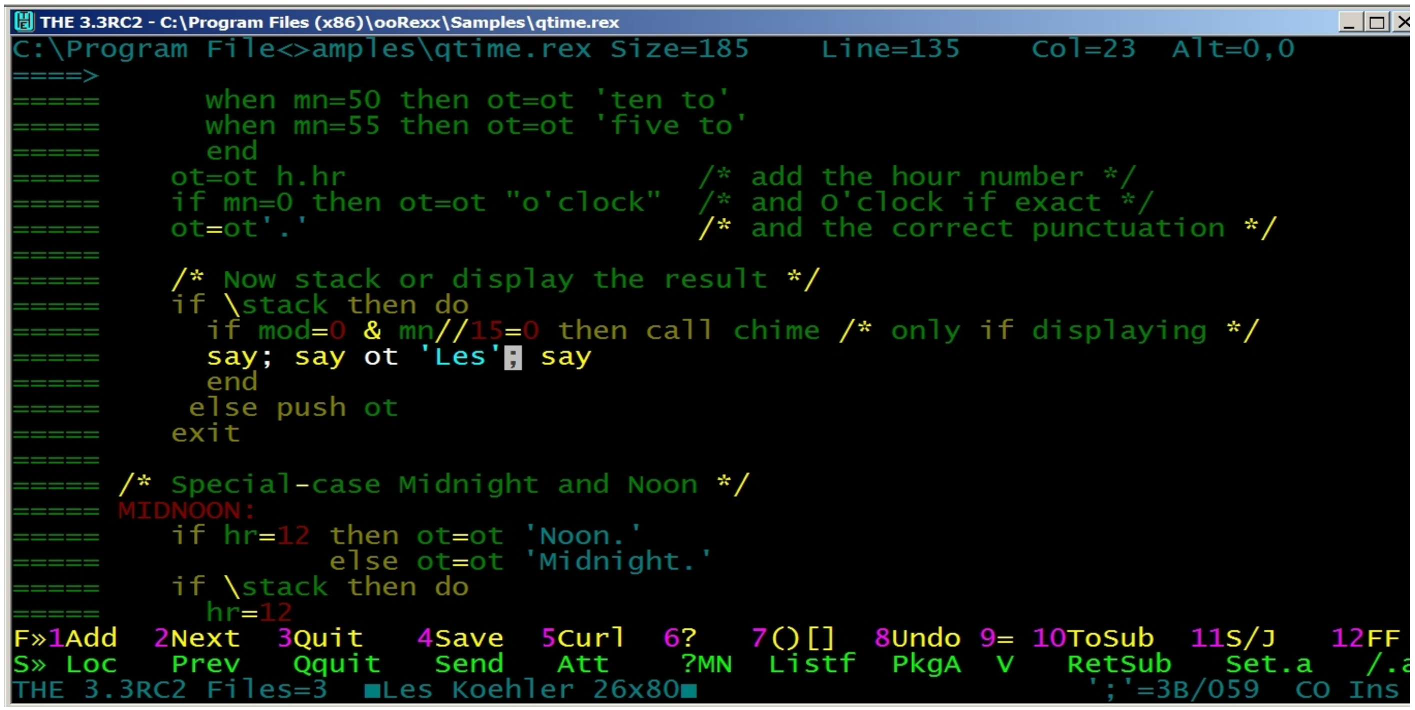
RUN executes the code that you're editing, without you having to save it. You can use it to experiment with changes to code or to simply execute some example you may have found. Output is captured by the command:

```
rexxoutput file
```

having been issued by the **profile**.

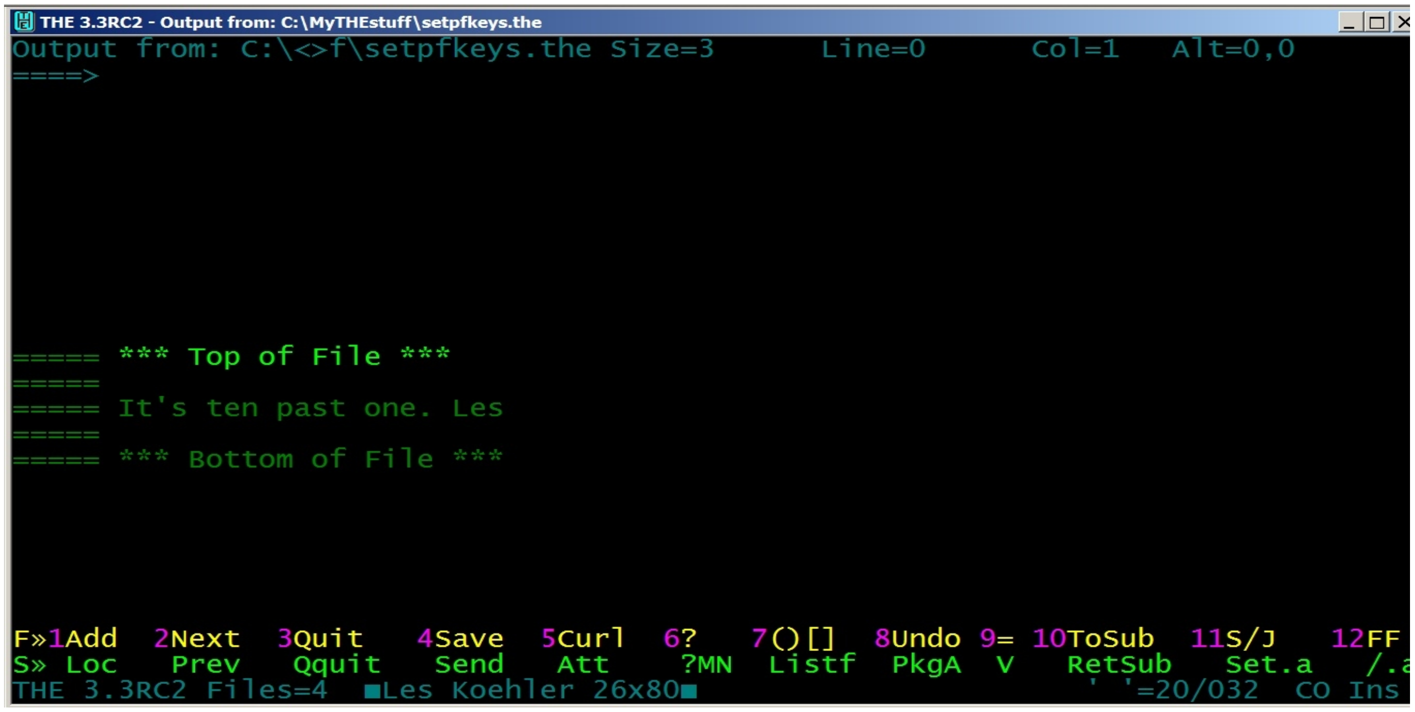
For example, here I've navigated to the ooRexx/Samples and edited **qtime.rex**

to add my name to the message:



```
THE 3.3RC2 - C:\Program Files (x86)\ooRexx\Samples\qtime.rex
C:\Program File<>amples\qtime.rex Size=185    Line=135    Col=23   Alt=0,0
====>
=====      when mn=50 then ot=ot 'ten to'
=====      when mn=55 then ot=ot 'five to'
=====      end
=====    ot=ot h.hr                     /* add the hour number */
=====    if mn=0 then ot=ot "o'clock"   /* and O'clock if exact */
=====    ot=ot'.'                       /* and the correct punctuation */
=====
=====    /* Now stack or display the result */
=====    if \stack then do
=====      if mod=0 & mn//15=0 then call chime /* only if displaying */
=====      say; say ot 'Les'; say
=====      end
=====     else push ot
=====    exit
=====
===== /* Special-case Midnight and Noon */
===== MIDNOON:
=====    if hr=12 then ot=ot 'Noon.'
=====             else ot=ot 'Midnight.'
=====    if \stack then do
=====      hr=12
F»1Add  2Next  3Quit   4Save  5Curl  6?   7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att    ?MN  Listf  PkgA  V   RetSub   Set.a   /.a
THE 3.3RC2 Files=3  ■Les Koehler 26x80■                    ';'=3B/059  CO Ins
```

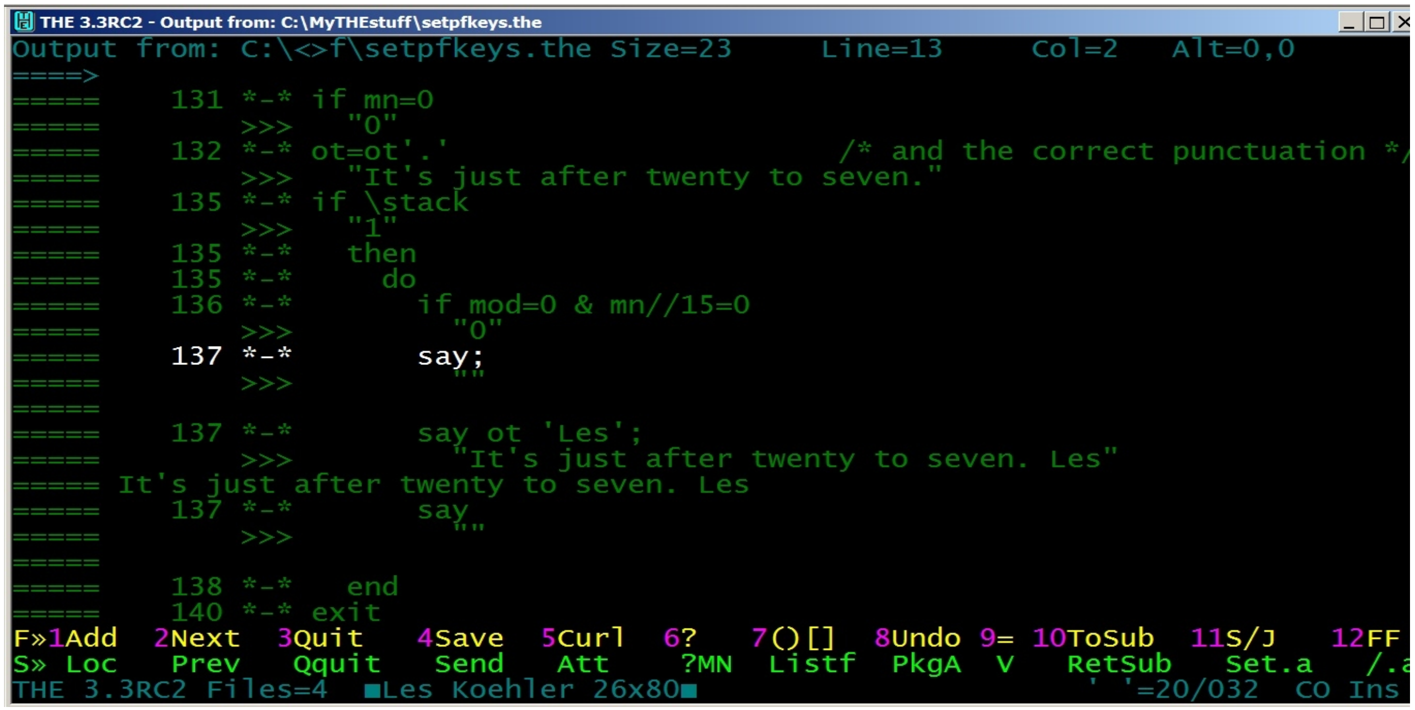
After I execute the **run** macro, THE captures the output and shows us:



```
THE 3.3RC2 - Output from: C:\MyTHEstuff\setpfkeys.the
Output from: C:\<>f\setpfkeys.the Size=3    Line=0     Col=1   Alt=0,0
====>









===== *** Top of File ***
=====
===== It's ten past one. Les
=====
===== *** Bottom of File ***






F»1Add  2Next  3Quit   4Save  5Curl  6?   7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att    ?MN  Listf  PkgA  V   RetSub   Set.a   /.a
THE 3.3RC2 Files=4  ■Les Koehler 26x80■                    ' '=20/032  CO Ins
```

It will try to ensure that the PATH is correctly set, but some BSF4OORexx code can confuse it.

If we had added a TRACE R statement, we might see something like:



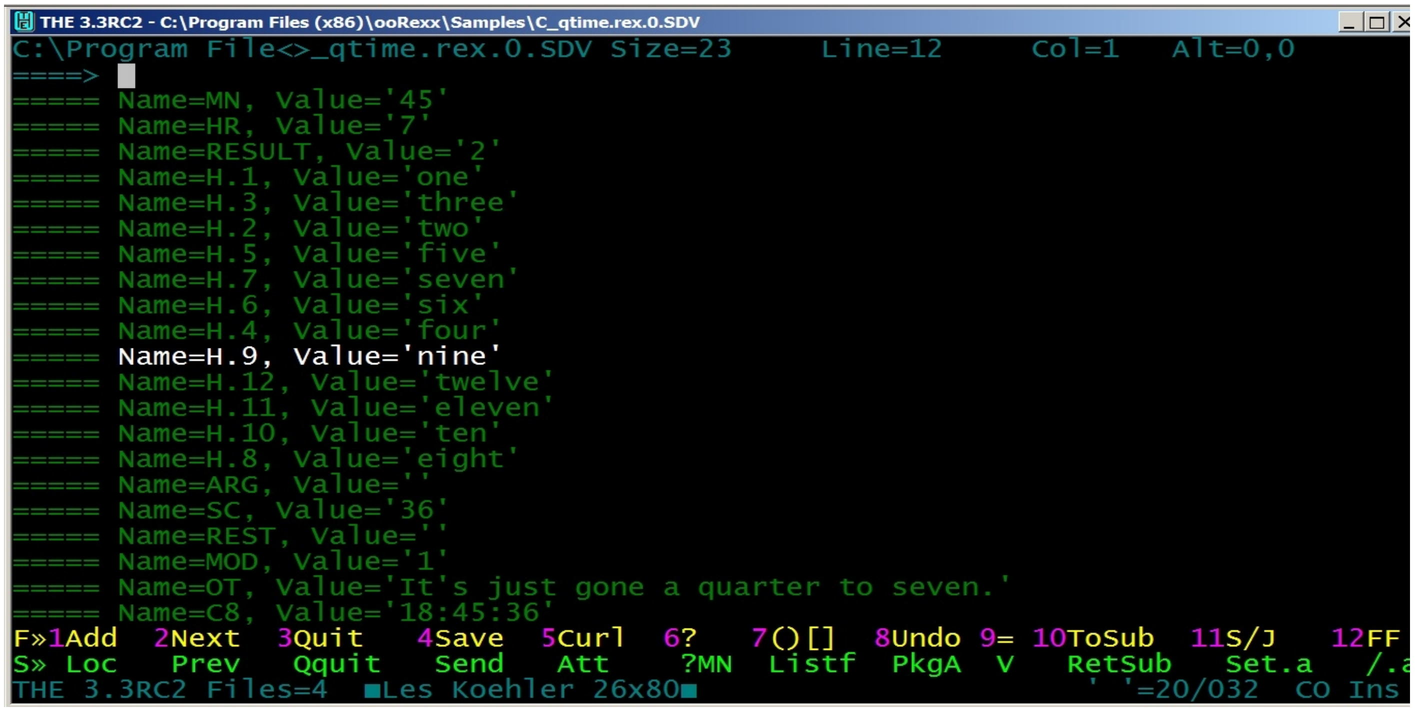
```
THE 3.3RC2 - Output from: C:\MyTHEstuff\setpfkeys.the
Output from: C:\<>f\setpfkeys.the Size=23     Line=13     Col=2   Alt=0,0
====>
=====    131 *-* if mn=0
             >>>   "0"
=====    132 *-* ot=ot'.'                        /* and the correct punctuation */
=====        >>>   "It's just after twenty to seven."
=====    135 *-* if \stack
=====        >>>   "1"
=====    135 *-*   then
=====    135 *-*     do
=====    136 *-*       if mod=0 & mn//15=0
=====        >>>         "0"
=====    137 *-*       say;
=====        >>>         ""
=====
=====    137 *-*       say ot 'Les';
=====        >>>         "It's just after twenty to seven. Les"
===== It's just after twenty to seven. Les
=====    137 *-*       say
=====        >>>         ""
=====
=====    138 *-*   end
=====    140 *-* exit
F»1Add  2Next  3Quit   4Save  5Curl  6?   7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att    ?MN  Listf  PkgA  V   RetSub   Set.a   /.a
THE 3.3RC2 Files=4  ■Les Koehler 26x80■                  ' '=20/032  CO Ins
```

captured by THE.

If we had been doing intensive debugging, at some point we might have inserted this to execute **dumpvars.rex**

```
Interpret dumpvars(,'edit')
Interpret edit
```

to see what the variables were set to:



```
THE 3.3RC2 - C:\Program Files (x86)\ooRexx\Samples\C_qtime.rex.0.SDV
C:\Program File<>_qtime.rex.0.SDV Size=23     Line=12     Col=1   Alt=0,0
====>
===== Name=MN, Value='45'
===== Name=HR, Value='7'
===== Name=RESULT, Value='2'
===== Name=H.1, Value='one'
===== Name=H.3, Value='three'
===== Name=H.2, Value='two'
===== Name=H.5, Value='five'
===== Name=H.7, Value='seven'
===== Name=H.6, Value='six'
===== Name=H.4, Value='four'
===== Name=H.9, Value='nine'
===== Name=H.12, Value='twelve'
===== Name=H.11, Value='eleven'
===== Name=H.10, Value='ten'
===== Name=H.8, Value='eight'
===== Name=ARG, Value=''
===== Name=SC, Value='36'
===== Name=REST, Value=''
===== Name=MOD, Value='1'
===== Name=OT, Value='It's just gone a quarter to seven.'
===== Name=C8, Value='18:45:36'
F»1Add  2Next  3Quit   4Save  5Curl  6?   7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att    ?MN  Listf  PkgA  V   RetSub   Set.a   /.a
THE 3.3RC2 Files=4  ■Les Koehler 26x80■                  ' '=20/032  CO Ins
```

The file created by **dumpvars** remains behind for later perusal and ultimately should be deleted, either manually or using **dumpvars** itself.

When we're finished, we simply QQUIT from both files.

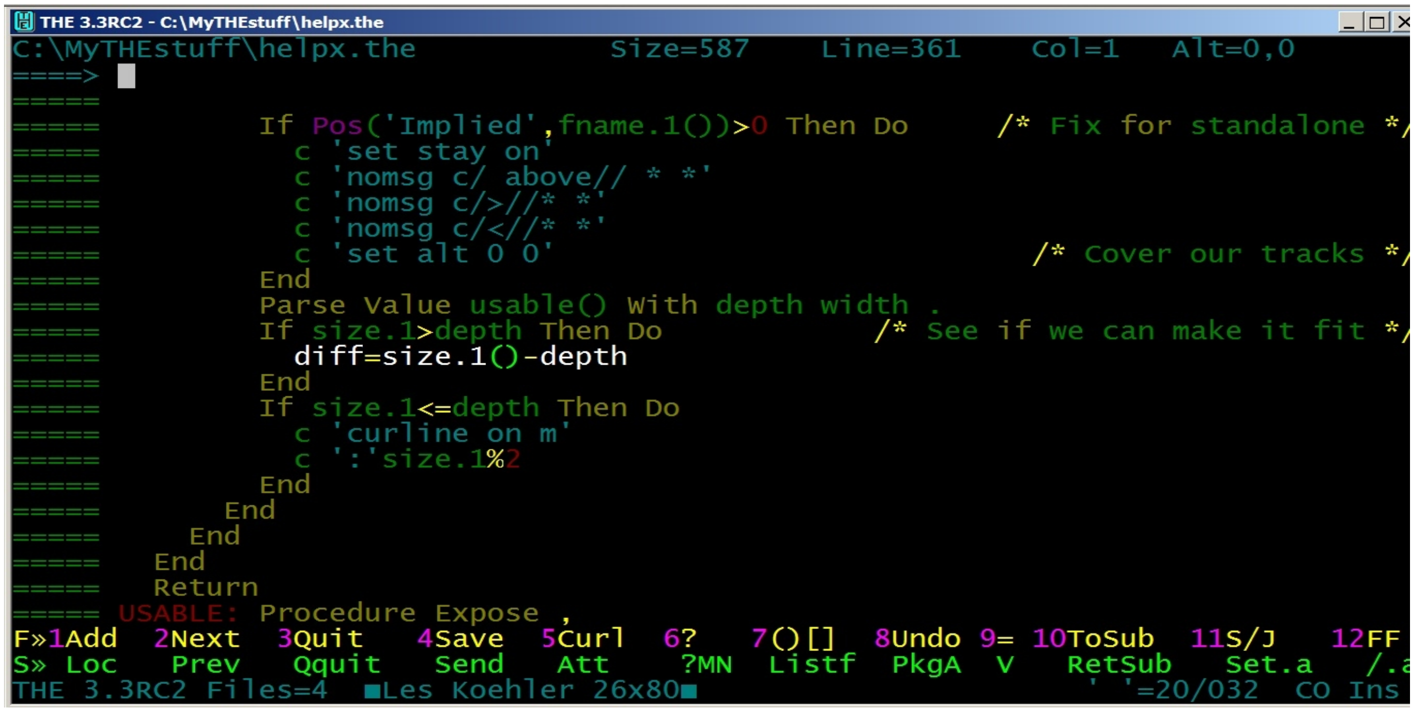
Coding assistance for Classic Rexx and HTML

The **smart_enter** macro provides this support and is triggered by this definition in the **defkeys** macro, which is run as part of the **profile**:

```
define enter macro smart_enter
```

Classic Rexx

The code generated is quite dense. That's because of my poor vision and the limited size of the screen (26x80), as shown at the bottom of the screen.
Here's an example:



```
THE 3.3RC2 - C:\MyTHEstuff\helpx.the
C:\MyTHEstuff\helpx.the           Size=587    Line=361    Col=1   Alt=0,0
====> 
=====
=====         If Pos('Implied',fname.1())>0 Then Do     /* Fix for standalone */
=====           c 'set stay on'
=====           c 'nomsg c/ above// * *'
=====           c 'nomsg c/>//* *'
=====           c 'nomsg c/<//* *'
=====           c 'set alt 0 0'                           /* Cover our tracks */
=====         End
=====         Parse Value usable() With depth width .
=====         If size.1>depth Then Do            /* See if we can make it fit */
=====           diff=size.1()-depth
=====         End
=====         If size.1<=depth Then Do
=====           c 'curline on m'
=====           c ':'size.1%2
=====         End
=====       End
=====     End
=====   End
=====   Return
===== USABLE: Procedure Expose ,
F»1Add  2Next  3Quit   4Save  5Curl  6?   7()[]  8Undo 9= 10ToSub   11S/J   12FF
S» Loc   Prev   Qquit   Send   Att    ?MN  Listf  PkgA  V   RetSub   Set.a   /.
THE 3.3RC2 Files=4  ■Les Koehler 26x80■                      ' '=20/032  CO Ins
```

Notice that **Do** is on the same line to save vertical screen space and to allow the

```
End
```

statements to undent evenly without any gaps. This provides visual confirmation that all the matching

```
End
```

statements are present.

Since this is a subroutine, the macro **rexxfmt.the** automatically indents it two characters.

smart_enter behaves much like you would expect it to, but it does not keep a history of what you've done and try to anticipate what you want next. What it does next is based on the line where the cursor is when the ENTER key is pressed.

Here's a summary:

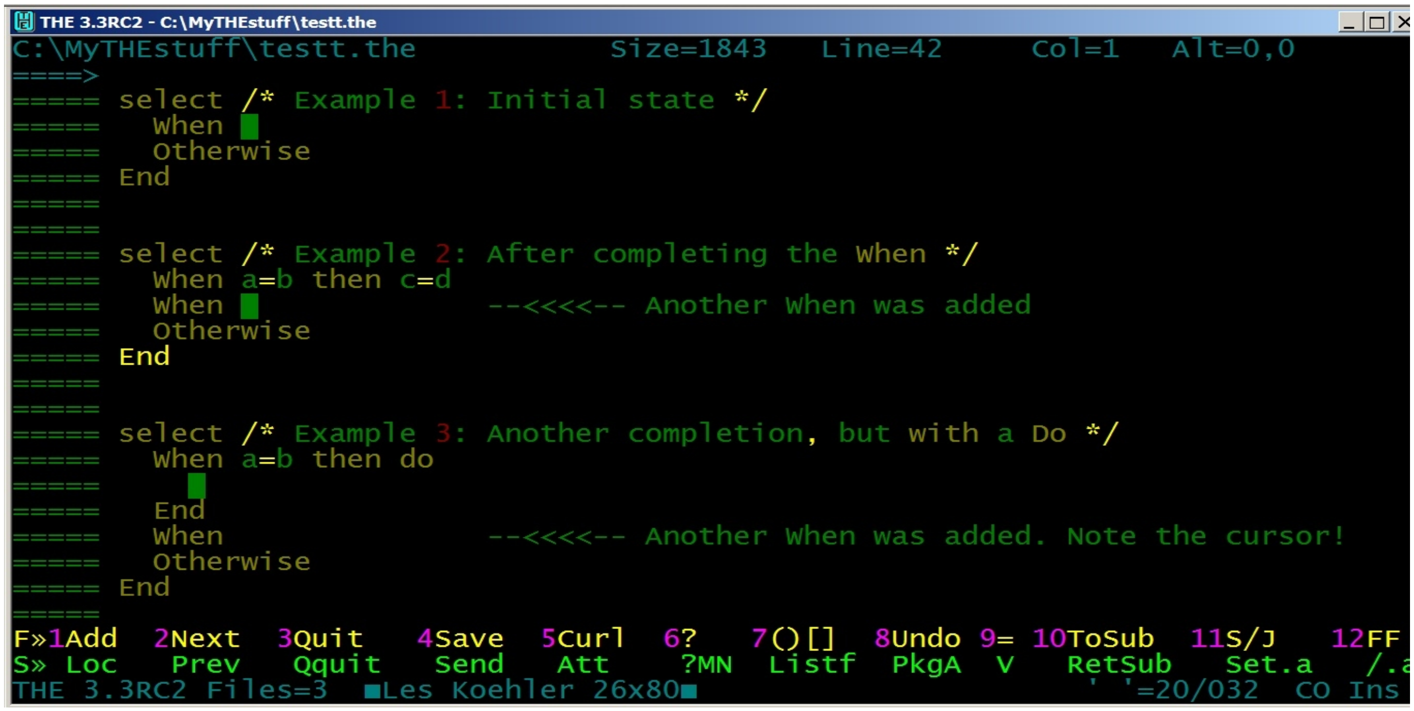
- IF without a trailing DO adds a THEN if needed. If the line is continued, a line is added with the THEN.
- IF with a THEN and no DO adds an ELSE.
- IF with a trailing DO adds a blank line and an END line, as does a leading DO.
- SELECT adds a WHEN line, an OTHERWISE and an END.
- WHEN adds another WHEN.

Of course the cursor is appropriately positioned.

How it handles a

```
Select
```

is, perhaps, somewhat different from what you might ordinarily expect. Here are three examples to help clarify how it behaves:

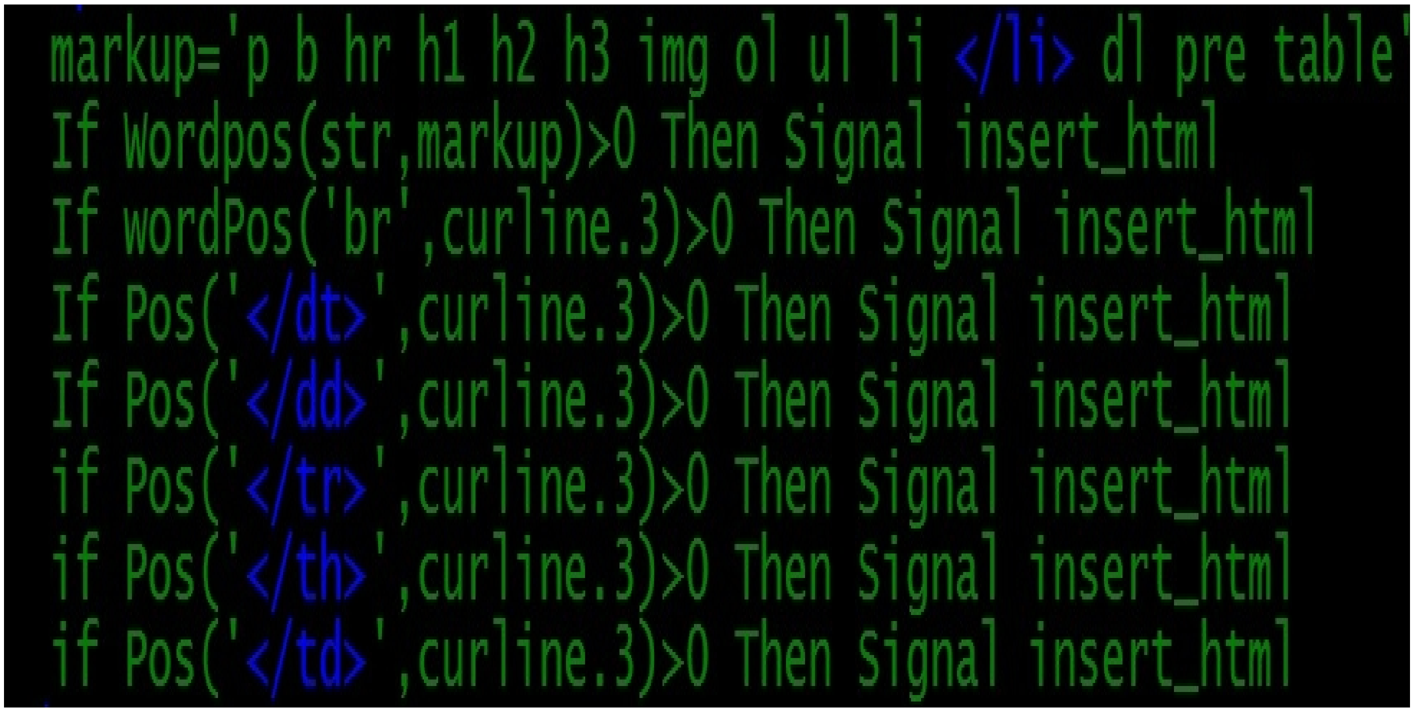


```
THE 3.3RC2 - C:\MyTHEstuff\testt.the
C:\MyTHEstuff\testt.the              Size=1843   Line=42     Col=1   Alt=0,0
====>
===== select /* Example 1: Initial state */
=====   When █
=====   Otherwise
===== End
=====
=====
===== select /* Example 2: After completing the When */
=====   When a=b then c=d
=====   When █                 --<<<<-- Another When was added
=====   Otherwise
===== End
=====
=====
===== select /* Example 3: Another completion, but with a Do */
=====   When a=b then do
=====     █
=====   End
=====   When                   --<<<<-- Another When was added. Note the cursor!
=====   Otherwise
===== End
=====
F»1Add  2Next  3Quit   4Save  5Curl  6?   7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att    ?MN  Listf  PkgA  V   RetSub   Set.a   /.
THE 3.3RC2 Files=3  ■Les Koehler 26x80■              ' '=20/032  CO Ins
```

The intent, in all cases, is to reduce the chance for keying errors and to make writing the code as easy as possible.

HTML

Currently, HTML is limited to the markup used most often, as this snippet of the actual code shows:



```
markup='p b hr h1 h2 h3 img ol ul li </li> dl pre table'
If Wordpos(str,markup)>0 Then Signal insert_html
If wordPos('br',curline.3)>0 Then Signal insert_html
If Pos('</dt>',curline.3)>0 Then Signal insert_html
If Pos('</dd>',curline.3)>0 Then Signal insert_html
if Pos('</tr>',curline.3)>0 Then Signal insert_html
if Pos('</th>',curline.3)>0 Then Signal insert_html
if Pos('</td>',curline.3)>0 Then Signal insert_html
```

markup

and all that is required is to enter the character string by itself in a blank line, or to position the cursor on the line with the

```
"</whatever>"
```

end markup to get another set of that markup.

```
<br />
```

gets special handling, since it is often repeated on the same line. As long as the sequence *br* appears to be a word, it will be expanded to the correct markup.

Some examples:

table

```
<!-- Use VIEW to follow this link for help with html tables:
http://www.w3schools.com/tags/tag_table.asp
-->
<table width="90%" align="center">
  <tr>
    <th>

    </th>
    <th>

    </th>
  </tr>
  <tr>
    <td>

    </td>
    <td>
    </td>
  </tr>
</table>
```

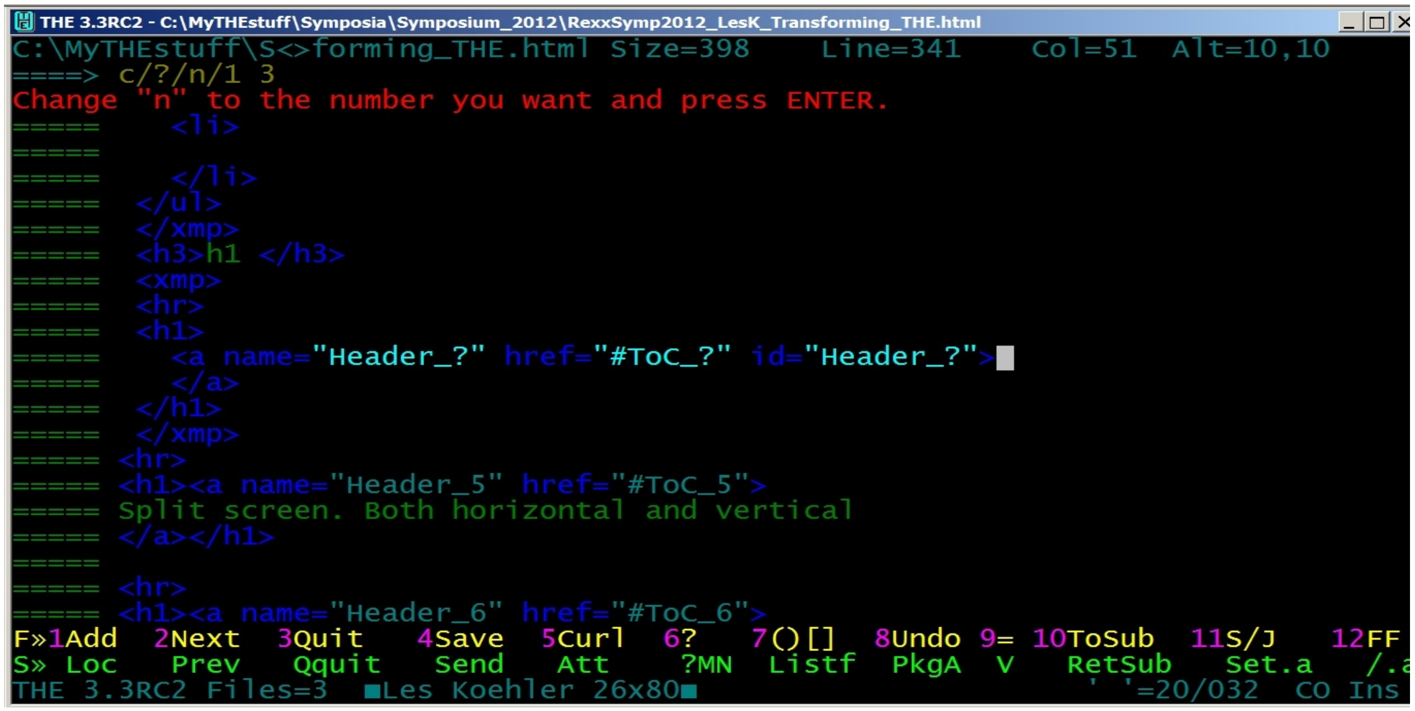
dl

```
 <dl>
   <dt> ? </dt>
   <dd> ? </dd>

 </dl>
```

ul

```
 <ul>
   <li>

   </li>
 </ul>
```

h1



```
THE 3.3RC2 - C:\MyTHEstuff\Symposia\Symposium_2012\RexxSymp2012_LesK_Transforming_THE.html
C:\MyTHEstuff\S<>forming_THE.html Size=398    Line=341    Col=51  Alt=10,10
====> c/?/n/1 3
Change "n" to the number you want and press ENTER.
=====     <li>
=====
=====     </li>
=====   </ul>
=====   </xmp>
=====   <h3>h1 </h3>
=====   <xmp>
=====   <hr>
=====   <h1>
=====     <a name="Header_?" href="#ToC_?" id="Header_?">
=====     </a>
=====   </h1>
=====   </xmp>
===== <hr>
===== <h1><a name="Header_5" href="#ToC_5">
===== Split screen. Both horizontal and vertical
===== </a></h1>
=====
===== <hr>
===== <h1><a name="Header_6" href="#ToC_6">
F»1Add  2Next  3Quit   4Save  5Curl  6?   7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att    ?MN  Listf  PkgA  V   RetSub   Set.a   /.
THE 3.3RC2 Files=3  ■Les Koehler 26x80■                 ' '=20/032  CO Ins
```

Although the above examples show the markup starting at the leftmost column, it will be put wherever the triggering string is found.

Split screen. Both horizontal and vertical

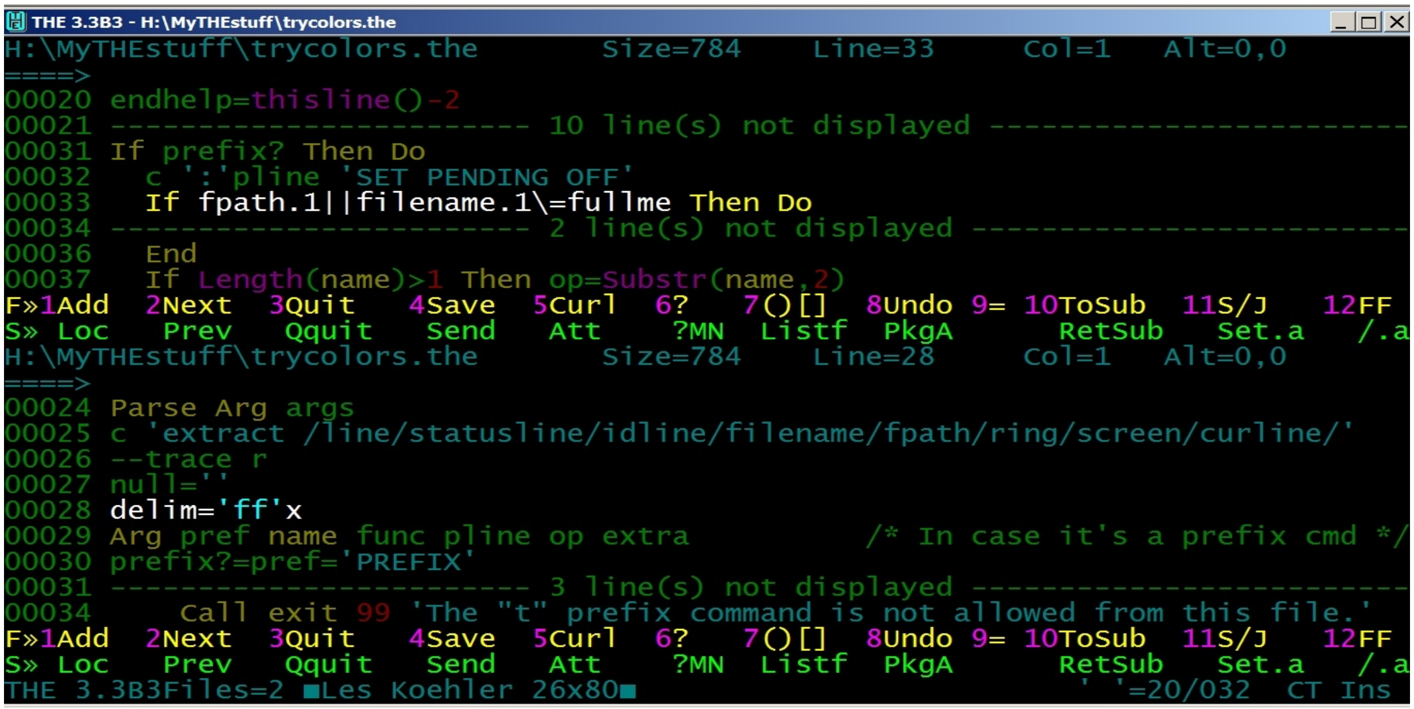
The *screen 2* command will split the screen into two views. By default, the arrangement is Horizontal, but Vertical can be passed if needed.

If there are multiple files in the ring, the second (bottom/right) view will be the next file in the ring.

When there is only one file, you get two views of it that can be managed individually. You might, for instance, explore the main line code in the top view and each subroutine (perhaps using**ToSub** and **RetSub**) in the bottom view.

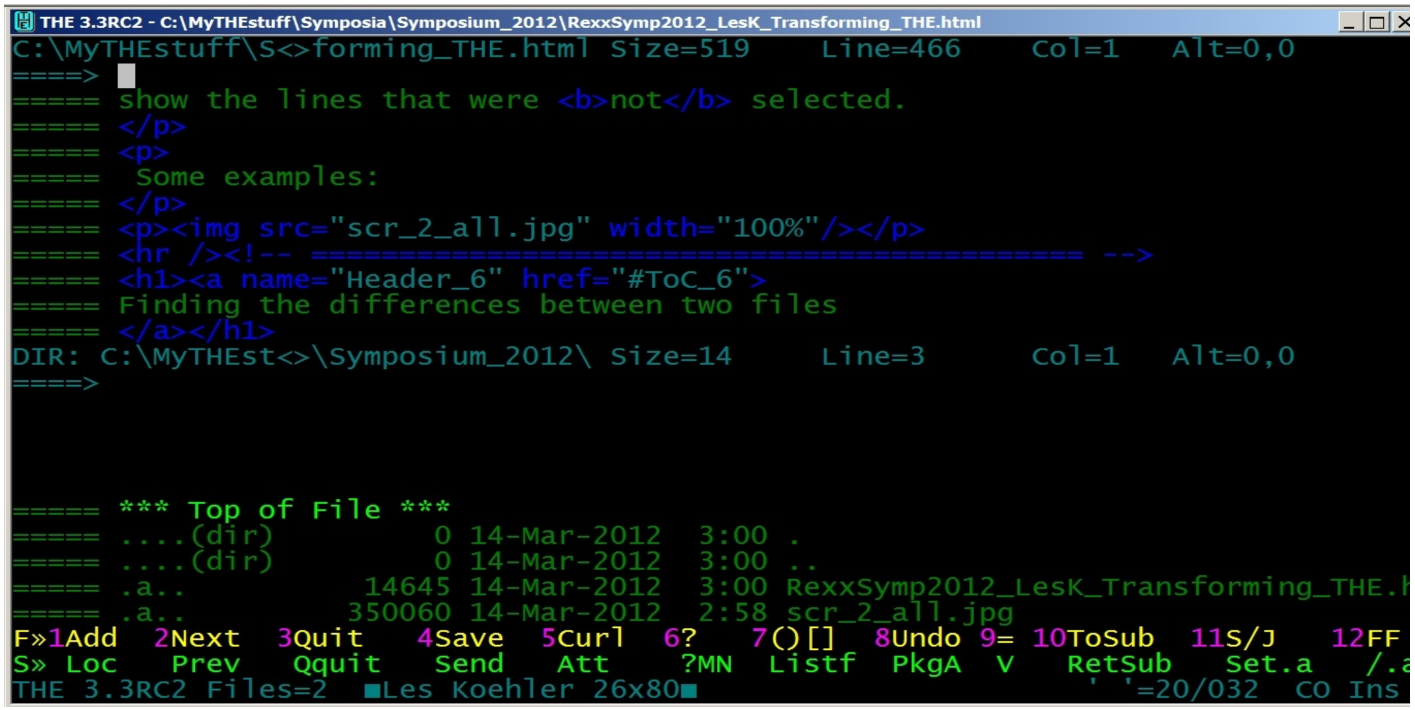
One interesting side-effect of having two views of the same file: when the *all* command is used in one view, the other view will show the lines that were **not** selected.

Some examples:



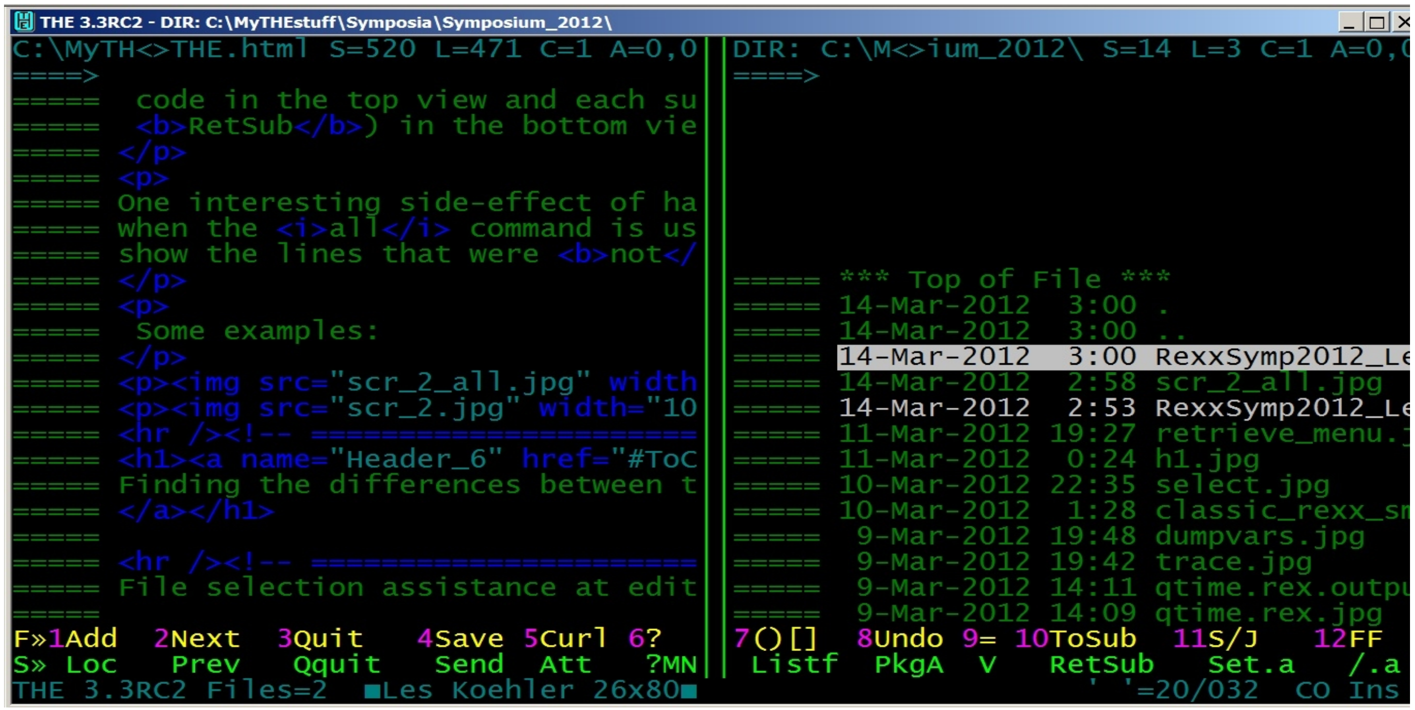
```
THE 3.3B3 - H:\MyTHEstuff\trycolors.the
H:\MyTHEstuff\trycolors.the        Size=784    Line=33     Col=1   Alt=0,0
====>
00020 endhelp=thisline()-2
00021 ------------------------ 10 line(s) not displayed ------------------------
00031 If prefix? Then Do
00032   c ':'pline 'SET PENDING OFF'
00033   If fpath.1||filename.1\=fullme Then Do
00034 ------------------------ 2 line(s) not displayed -------------------------
00036   End
00037   If Length(name)>1 Then op=Substr(name,2)
F»1Add  2Next  3Quit   4Save  5Curl  6?   7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att    ?MN  Listf  PkgA      RetSub   Set.a   /.a
H:\MyTHEstuff\trycolors.the        Size=784    Line=28     Col=1   Alt=0,0
====>
00024 Parse Arg args
00025 c 'extract /line/statusline/idline/filename/fpath/ring/screen/curline/'
00026 --trace r
00027 null=''
00028 delim='ff'x
00029 Arg pref name func pline op extra          /* In case it's a prefix cmd */
00030 prefix?=pref='PREFIX'
00031 ------------------------ 3 line(s) not displayed ------------------------
00034     Call exit 99 'The "t" prefix command is not allowed from this file.'
F»1Add  2Next  3Quit   4Save  5Curl  6?   7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att    ?MN  Listf  PkgA      RetSub   Set.a   /.a
THE 3.3B3Files=2 ■Les Koehler 26x80■                          ' '=20/032  CT Ins
```

Horizontal split of a file. The *all /if/* and *more /end/* commands have been used, as can be seen in the top view. The bottom view shows the other lines.



```
THE 3.3RC2 - C:\MyTHEstuff\Symposia\Symposium_2012\RexxSymp2012_LesK_Transforming_THE.html
C:\MyTHEstuff\S<>forming_THE.html Size=519    Line=466    Col=1   Alt=0,0
====>
===== show the lines that were <b>not</b> selected.
===== </p>
===== <p>
=====  Some examples:
===== </p>
===== <p><img src="scr_2_all.jpg" width="100%"/></p>
===== <hr /><!-- ========================================== -->
===== <h1><a name="Header_6" href="#ToC_6">
===== Finding the differences between two files
===== </a></h1>
DIR: C:\MyTHEst<>\Symposium_2012\ Size=14     Line=3      Col=1   Alt=0,0
====>




===== *** Top of File ***
===== ....(dir)         0 14-Mar-2012  3:00 .
===== ....(dir)         0 14-Mar-2012  3:00 ..
===== .a..          14645 14-Mar-2012  3:00 RexxSymp2012_LesK_Transforming_THE.h
===== .a..         350060 14-Mar-2012  2:58 scr_2_all.jpg
F»1Add  2Next  3Quit   4Save  5Curl  6?  7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att   ?MN  Listf  PkgA  V  RetSub   Set.a   /.a
THE 3.3RC2 Files=2  ■Les Koehler 26x80■              ' '=20/032  CO Ins
```

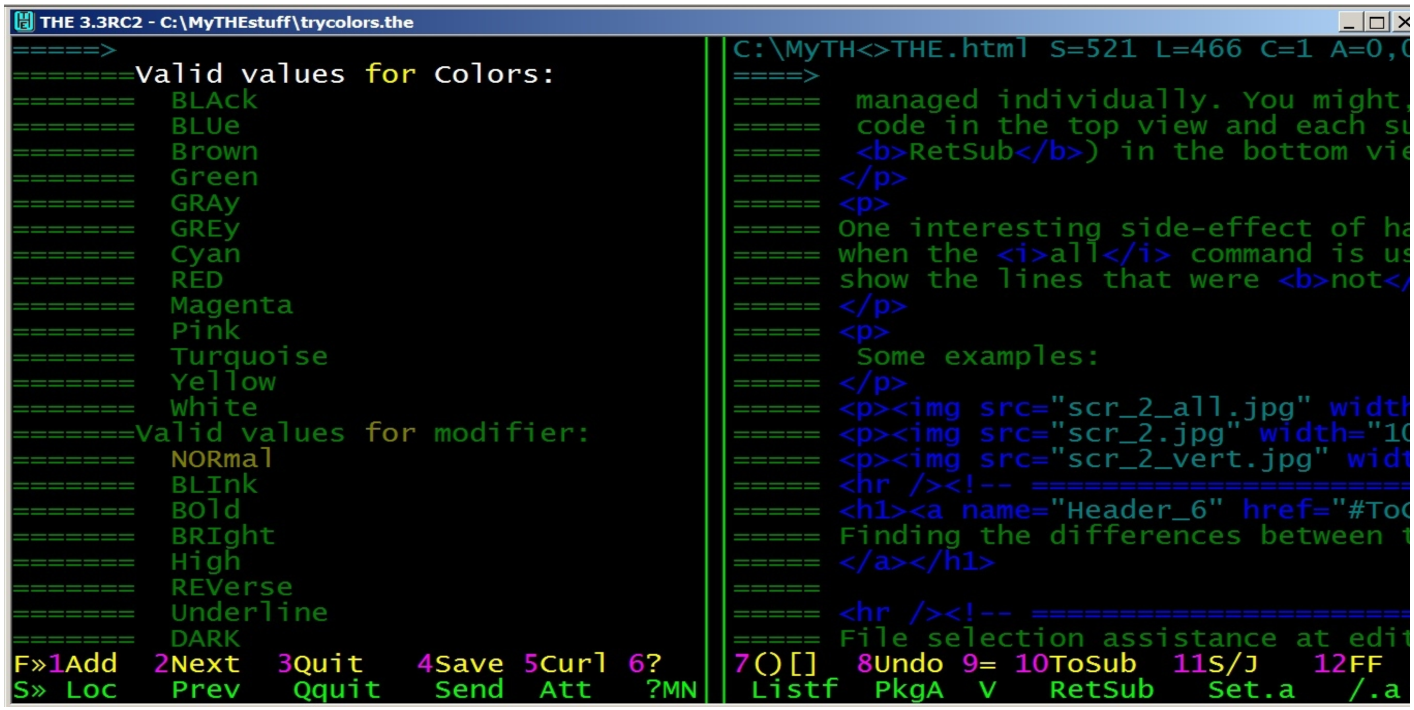
Horizontal split showing two files: The source for this document and the directory for it. Notice that the top view does not show the reserved function key lines.



```
THE 3.3RC2 - DIR: C:\MyTHEstuff\Symposia\Symposium_2012\
C:\MyTH<>THE.html S=520 L=471 C=1 A=0,0 ||DIR: C:\M<>ium_2012\ S=14 L=3 C=1 A=0,0
====>                                   ||====>
=====  code in the top view and each su ||
=====  <b>RetSub</b>) in the bottom vie ||
===== </p>                              ||
===== <p>                               ||
===== One interesting side-effect of ha ||
===== when the <i>all</i> command is us ||
===== show the lines that were <b>not</ ||
===== </p>                              ||===== *** Top of File ***
===== <p>                               ||===== 14-Mar-2012  3:00 .
=====  Some examples:                   ||===== 14-Mar-2012  3:00 ..
===== </p>                              ||===== 14-Mar-2012  3:00 RexxSymp2012_Le
===== <p><img src="scr_2_all.jpg" width ||===== 14-Mar-2012  2:58 scr_2_all.jpg
===== <p><img src="scr_2.jpg" width="10 ||===== 14-Mar-2012  2:53 RexxSymp2012_Le
===== <hr /><!-- ====================== ||===== 11-Mar-2012 19:27 retrieve_menu.j
===== <h1><a name="Header_6" href="#ToC ||===== 11-Mar-2012  0:24 h1.jpg
===== Finding the differences between t ||===== 10-Mar-2012 22:35 select.jpg
===== </a></h1>                         ||===== 10-Mar-2012  1:28 classic_rexx_sm
=====                                   ||=====  9-Mar-2012 19:48 dumpvars.jpg
===== <hr /><!-- ====================== ||=====  9-Mar-2012 19:42 trace.jpg
===== File selection assistance at edit ||=====  9-Mar-2012 14:11 qtime.rex.outpu
=====                                   ||=====  9-Mar-2012 14:09 qtime.rex.jpg
F»1Add  2Next  3Quit   4Save 5Curl 6?   ||7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send  Att   ?MN ||  Listf  PkgA  V  RetSub   Set.a   /.a
THE 3.3RC2 Files=2  ■Les Koehler 26x80■              ' '=20/032  CO Ins
```

Vertical split of the same two files. Notice the reserved function key lines.

Even though THE only handles two views, you can do some interesting things. Here my **trycolors** macro is showing a list of colors and elements to select on the left side. The right side will be used to show the result.



THE 3.3RC2 - C:\MyTHEstuff\trycolors.the

```
=====>
=======Valid values for Colors:
=======  BLAck
=======  BLUe
=======  Brown
=======  Green
=======  GRAy
=======  GREy
=======  Cyan
=======  RED
=======  Magenta
=======  Pink
=======  Turquoise
=======  Yellow
=======  White
=======Valid values for modifier:
=======  NORmal
=======  BLInk
=======  BOld
=======  BRIght
=======  High
=======  REVerse
=======  Underline
=======  DARK
```

```
C:\MyTH<>THE.html S=521 L=466 C=1 A=0,(
====>
=====  managed individually. You might,
=====  code in the top view and each su
=====  <b>RetSub</b>) in the bottom vie
===== </p>
===== <p>
===== One interesting side-effect of ha
===== when the <i>all</i> command is us
===== show the lines that were <b>not</
===== </p>
===== <p>
=====  Some examples:
===== </p>
===== <p><img src="scr_2_all.jpg" width
===== <p><img src="scr_2.jpg" width="10
===== <p><img src="scr_2_vert.jpg" widt
===== <hr /><!-- =====================
===== <h1><a name="Header_6" href="#ToC
===== Finding the differences between t
===== </a></h1>
=====
===== <hr /><!-- =====================
===== File selection assistance at edit
```

```
F»1Add  2Next   3Quit   4Save 5Curl 6?   7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc    Prev    Qquit   Send  Att   ?MN   Listf  PkgA  V   RetSub   Set.a   /.a
```

trycolors

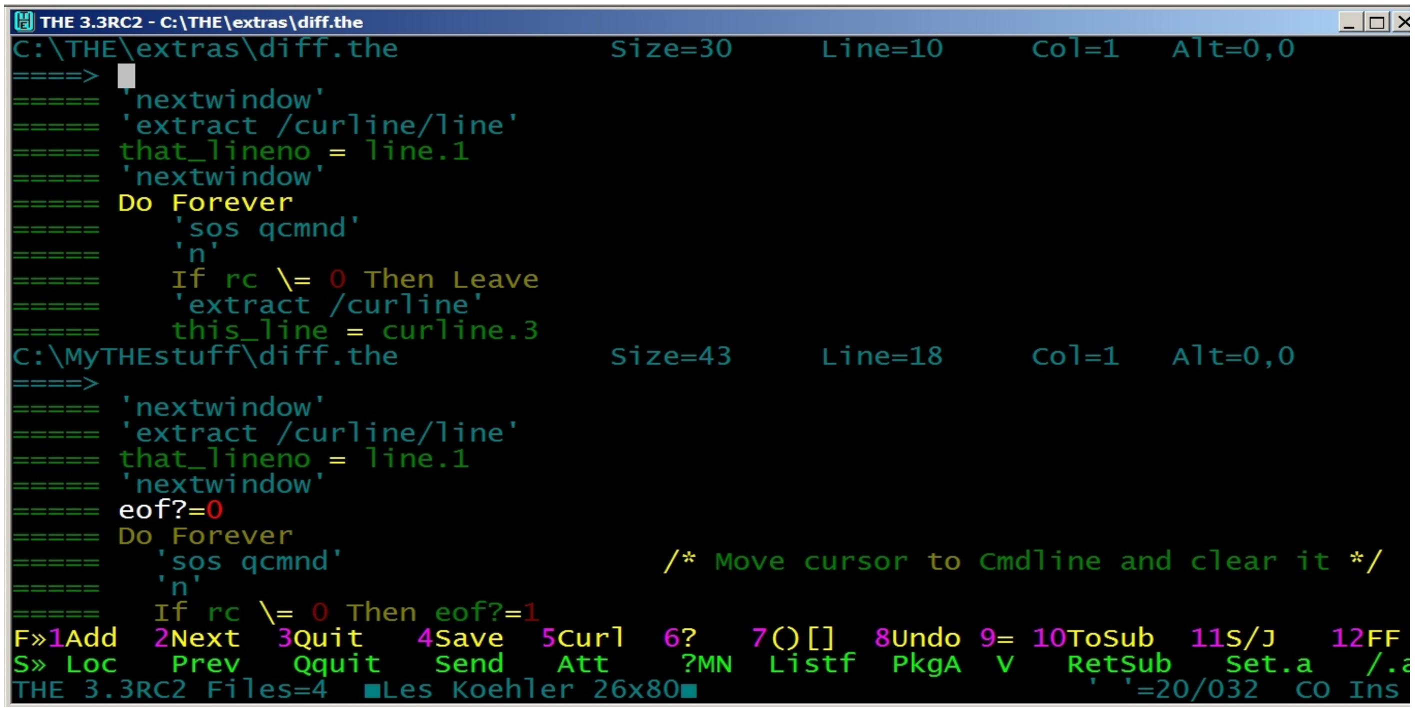
Finding the differences between two files

THE provides the **diff** macro for finding the differences (ignoring spaces) between two files in the ring. When it finds a difference, it exits with the current line set to the offending line. F2 can be used to switch to the other file.

My enhanced version requires a split screen and allows two views of a single file. This would allow you to compare two subroutines in the same file.

The macros **both** and **other** can be handy for resynchronizing the files to a common point before running **diff** again.

Below I've used **both** to skip over the first change I made to **diff** and then found the second change.



```
THE 3.3RC2 - C:\THE\extras\diff.the
C:\THE\extras\diff.the            Size=30     Line=10     Col=1   Alt=0,0
====> 
===== 'nextwindow'
===== 'extract /curline/line'
===== that_lineno = line.1
===== 'nextwindow'
===== Do Forever
=====    'sos qcmnd'
=====    'n'
=====    If rc \= 0 Then Leave
=====    'extract /curline'
=====    this_line = curline.3
C:\MyTHEstuff\diff.the            Size=43     Line=18     Col=1   Alt=0,0
====>
===== 'nextwindow'
===== 'extract /curline/line'
===== that_lineno = line.1
===== 'nextwindow'
===== eof?=0
===== Do Forever
=====   'sos qcmnd'                 /* Move cursor to Cmdline and clear it */
=====   'n'
=====   If rc \= 0 Then eof?=1
F»1Add  2Next  3Quit   4Save  5Curl   6?    7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att     ?MN  Listf  PkgA  V   RetSub   Set.a   /.a
THE 3.3RC2 Files=4  ■Les Koehler 26x80■                     ' '=20/032  CO Ins
```

Diff of two versions of diff

File selection assistance at edit time

THE expects a fully qualified fileid when you edit a file with any of: *THE, EDIT, KEDIT or XEDIT* commands. If the fileid is not found, THE will create it in the directory that THE was launched from.

To make things easier, I wrote the **newfile** macro, which does two things when THE thinks a file is new:

1. Searches the list of "favorite" folders listed in **getvar** and edits the file if it is found.
2. Displays a popup of the "closest fits" from all the files in the favorites list.

For example, a subset of my favorites looks like this:

```
C:\MyTHEstuff
C:\MyTHEstuff\Symposia\Symposium_2012
C:\MyTHEstuff\Symposia\Symposium_2011
C:\MyRexxStuff
C:\MyTHEstuff\Email
C:\RexxLA2011
```

Instead of using the **dirs** macro to first navigate to the folder and make my selection from there, I could just enter:

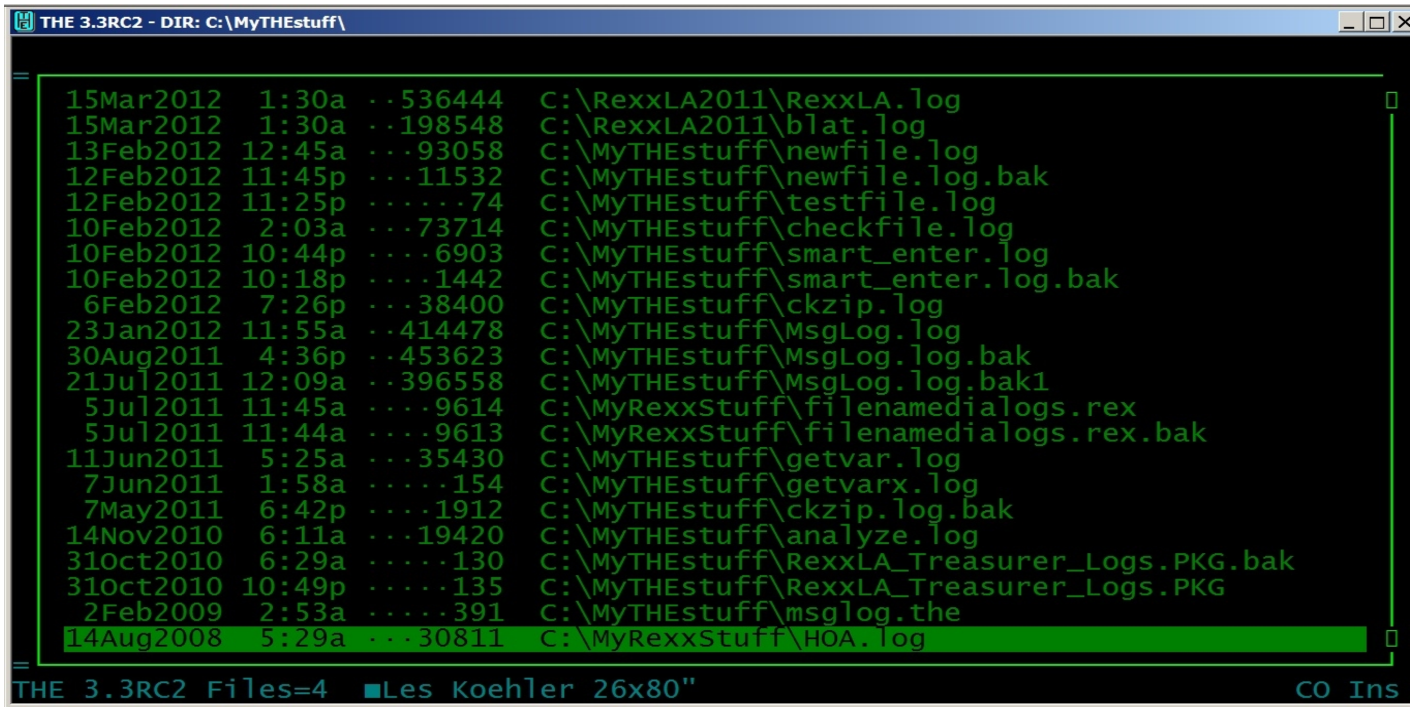
x rexxla_members

which is in the

```
C:\RexxLA2011
```

folder and it would automatically be found.

When an exact match isn't found, **newfile** does a wildcard search, sorts the result by descending date and presents a popup to select from:



```
THE 3.3RC2 - DIR: C:\MyTHEstuff\

 15Mar2012  1:30a ··536444  C:\RexxLA2011\RexxLA.log
 15Mar2012  1:30a ··198548  C:\RexxLA2011\blat.log
 13Feb2012 12:45a ···93058  C:\MyTHEstuff\newfile.log
 12Feb2012 11:45p ···11532  C:\MyTHEstuff\newfile.log.bak
 12Feb2012 11:25p ······74  C:\MyTHEstuff\testfile.log
 10Feb2012  2:03a ···73714  C:\MyTHEstuff\checkfile.log
 10Feb2012 10:44p ····6903  C:\MyTHEstuff\smart_enter.log
 10Feb2012 10:18p ····1442  C:\MyTHEstuff\smart_enter.log.bak
  6Feb2012  7:26p ···38400  C:\MyTHEstuff\ckzip.log
 23Jan2012 11:55a ··414478  C:\MyTHEstuff\MsgLog.log
 30Aug2011  4:36p ··453623  C:\MyTHEstuff\MsgLog.log.bak
 21Jul2011 12:09a ··396558  C:\MyTHEstuff\MsgLog.log.bak1
  5Jul2011 11:45a ····9614  C:\MyRexxStuff\filenamedialogs.rex
  5Jul2011 11:44a ····9613  C:\MyRexxStuff\filenamedialogs.rex.bak
 11Jun2011  5:25a ···35430  C:\MyTHEstuff\getvar.log
  7Jun2011  1:58a ·····154  C:\MyTHEstuff\getvarx.log
  7May2011  6:42p ····1912  C:\MyTHEstuff\ckzip.log.bak
 14Nov2010  6:11a ···19420  C:\MyTHEstuff\analyze.log
 31Oct2010  6:29a ·····130  C:\MyTHEstuff\RexxLA_Treasurer_Logs.PKG.bak
 31Oct2010 10:49p ·····135  C:\MyTHEstuff\RexxLA_Treasurer_Logs.PKG
  2Feb2009  2:53a ·····391  C:\MyTHEstuff\msglog.the
 14Aug2008  5:29a ···30811  C:\MyRexxStuff\HOA.log

THE 3.3RC2 Files=4  ■Les Koehler 26x80"                                CO Ins
```

newfile: Wildcard search result when *x log* was entered

Selecting a file from the ring of active files being edited

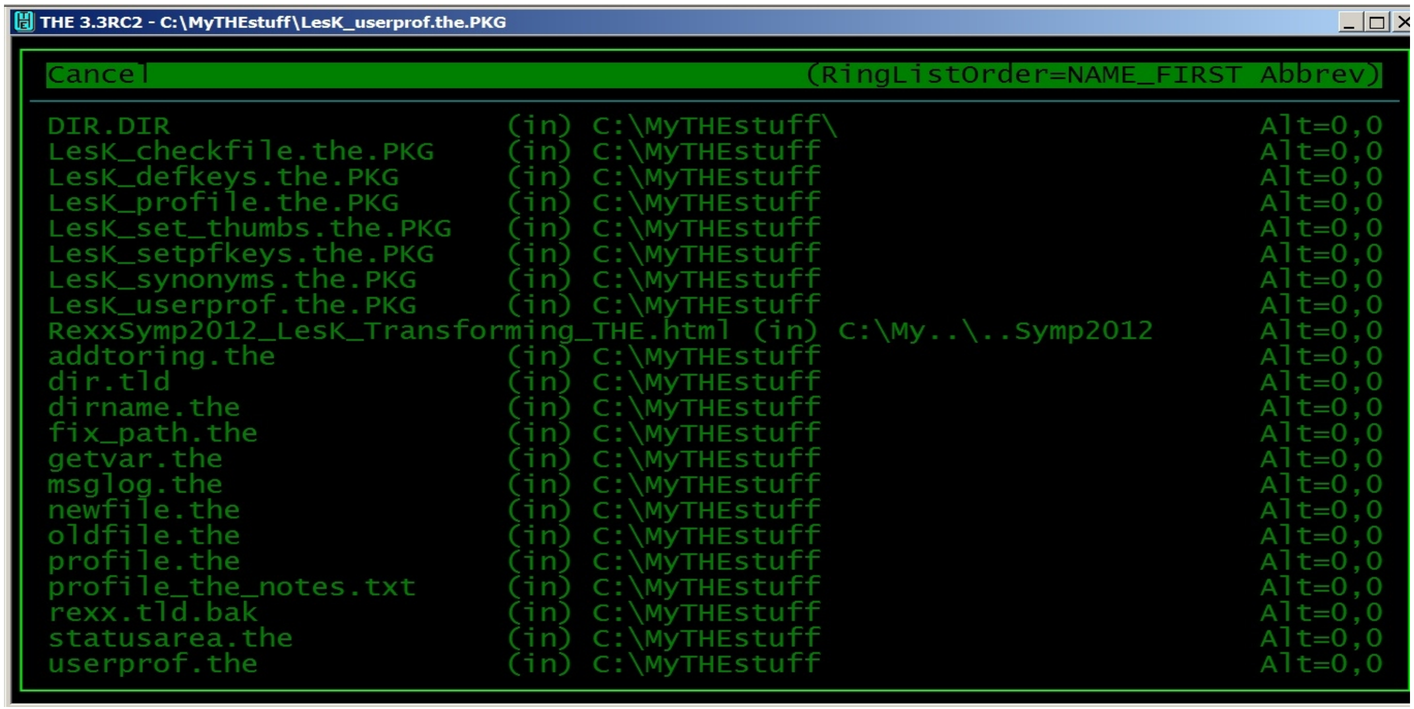
When there are a lot of files in the ring, pressing F2 to "walk the ring" to get to the one of interest doesn't allow you to see all the files in the ring and conceptualize why you have them there.

The **ringlist** macro displays a popup of all the files in the ring for you to select from, navigating with the up/down arrow keys. The format is controlled by an *EDITV* setting of your choice in your **profile**:

```
"EDITV SETL RINGLISTORDER NAME_FIRST"
"EDITV SETL RINGLISTORDER NAME"
"EDITV SETL RINGLISTORDER FILEID"
"EDITV SETL RINGLISTORDER NONE"
```

This is a reworked version of the original Kedit code, making it compatible with THE and adding:

- NONE parameter
- abbrev? switch to abbreviate long path names if needed to reduce horizontal scrolling
- Walk around the ring to get to DIR.DIR to avoid rebuilding
- ALT count in popup so you can see if a file has been changed



```
THE 3.3RC2 - C:\MyTHEstuff\LesK_userprof.the.PKG
Cancel                                               (RingListOrder=NAME_FIRST Abbrev)

DIR.DIR                    (in) C:\MyTHEstuff\                          Alt=0,0
LesK_checkfile.the.PKG     (in) C:\MyTHEstuff                           Alt=0,0
LesK_defkeys.the.PKG       (in) C:\MyTHEstuff                           Alt=0,0
LesK_profile.the.PKG       (in) C:\MyTHEstuff                           Alt=0,0
LesK_set_thumbs.the.PKG    (in) C:\MyTHEstuff                           Alt=0,0
LesK_setpfkeys.the.PKG     (in) C:\MyTHEstuff                           Alt=0,0
LesK_synonyms.the.PKG      (in) C:\MyTHEstuff                           Alt=0,0
LesK_userprof.the.PKG      (in) C:\MyTHEstuff                           Alt=0,0
RexxSymp2012_LesK_Transforming_THE.html (in) C:\My..\..Symp2012         Alt=0,0
addtoring.the              (in) C:\MyTHEstuff                           Alt=0,0
dir.tld                    (in) C:\MyTHEstuff                           Alt=0,0
dirname.the                (in) C:\MyTHEstuff                           Alt=0,0
fix_path.the               (in) C:\MyTHEstuff                           Alt=0,0
getvar.the                 (in) C:\MyTHEstuff                           Alt=0,0
msglog.the                 (in) C:\MyTHEstuff                           Alt=0,0
newfile.the                (in) C:\MyTHEstuff                           Alt=0,0
oldfile.the                (in) C:\MyTHEstuff                           Alt=0,0
profile.the                (in) C:\MyTHEstuff                           Alt=0,0
profile_the_notes.txt      (in) C:\MyTHEstuff                           Alt=0,0
rexx.tld.bak               (in) C:\MyTHEstuff                           Alt=0,0
statusarea.the             (in) C:\MyTHEstuff                           Alt=0,0
userprof.the               (in) C:\MyTHEstuff                           Alt=0,0
```

ringlist with abbreviation

Switching between a limited subset of all the files in the ring

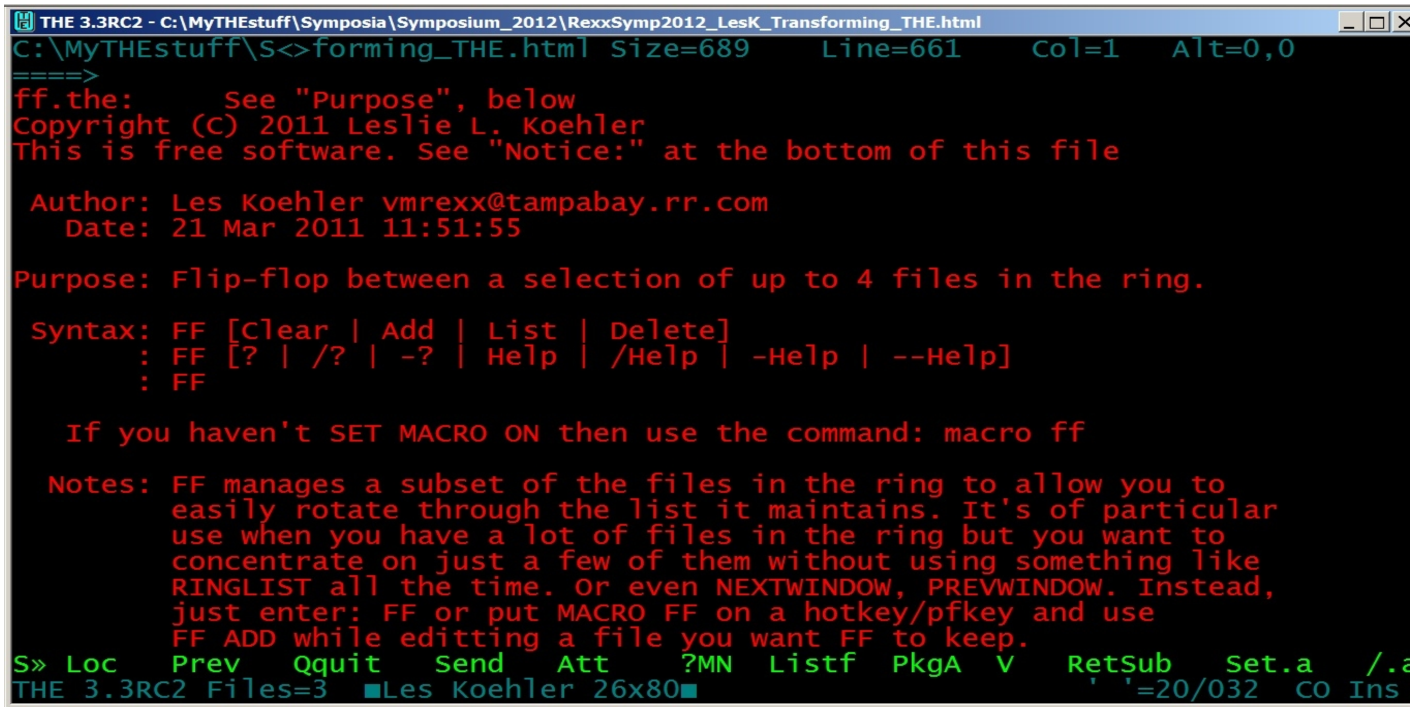
When you have a lot of files in the ring, as seen above, and you want to concentrate your efforts on just one or two, but perhaps another is needed for reference, the **FF** macro can help.

Originally written to flip-flop between two files (thus its name), it wasn't long before I realized it should be able to do more, so now the limit is four files.

while editing a file you want it to remember, just enter
ff add
in the cmdline.

After you've done this for the files of interest, use the **FF** function key to flip between them.

When you enter *ff ?* you see:


```
THE 3.3RC2 - C:\MyTHEstuff\Symposia\Symposium_2012\RexxSymp2012_LesK_Transforming_THE.html
C:\MyTHEstuff\S<>forming_THE.html Size=689    Line=661    Col=1    Alt=0,0
====>
ff.the:     See "Purpose", below
Copyright (C) 2011 Leslie L. Koehler
This is free software. See "Notice:" at the bottom of this file

 Author: Les Koehler vmrexx@tampabay.rr.com
   Date: 21 Mar 2011 11:51:55

Purpose: Flip-flop between a selection of up to 4 files in the ring.

 Syntax: FF [Clear | Add | List | Delete]
       : FF [? | /? | -? | Help | /Help | -Help | --Help]
       : FF

   If you haven't SET MACRO ON then use the command: macro ff

  Notes: FF manages a subset of the files in the ring to allow you to
         easily rotate through the list it maintains. It's of particular
         use when you have a lot of files in the ring but you want to
         concentrate on just a few of them without using something like
         RINGLIST all the time. Or even NEXTWINDOW, PREVWINDOW. Instead,
         just enter: FF or put MACRO FF on a hotkey/pfkey and use
         FF ADD while editting a file you want FF to keep.
S» Loc   Prev   Qquit   Send   Att   ?MN   Listf   PkgA  V   RetSub   Set.a   /.a
THE 3.3RC2 Files=3  ■Les Koehler 26x80■                ' '=20/032  CO Ins
```

ff help

Carrying skeleton code within a macro, instead of in a separate file

The problem with a skeleton file is: where is it? In THE you would have to code, or issue a command like:
 get c:\some_path\skeleton_file.rex
which means that the fileid must be configured somewhere, which means that is must be tailored, which means instructions must be written to point the user at the configuration file so it can be tailored.

It is often a lot easier to just imbed the skeleton in the macro that needs it. A simple example is within *smart_enter:*

```
TABLE_HTML:
  first=thisline()+2
/*
<!-- Use VIEW to follow this link for help with html tables:
http://www.w3schools.com/tags/tag_table.asp
-->
<table width="90%" align="center">
  <tr>
    <th>

    </th>
    <th>

    </th>
  </tr>
  <tr>
    <td>
    </td>
    <td>

    </td>
  </tr>
</table>
*/
  last=thisline()-2
  'replace' sourceline(first)
  'extract /line/curline/'
  do s=first+1 to last
    'input' sourceline(s)
  end
  ':'line.1
  'cursor file' line.1+6 6
  'sos makecurr'
  call prompt ,
  'Position cursor to the "</" line and press ENTER for another pair'
return
```

The results of this technique were shown earlier.

The same technique can be used to initialize Rexx/THE code, thus ensuring consistency of a set of labels, variables and subroutines that are commonly needed.

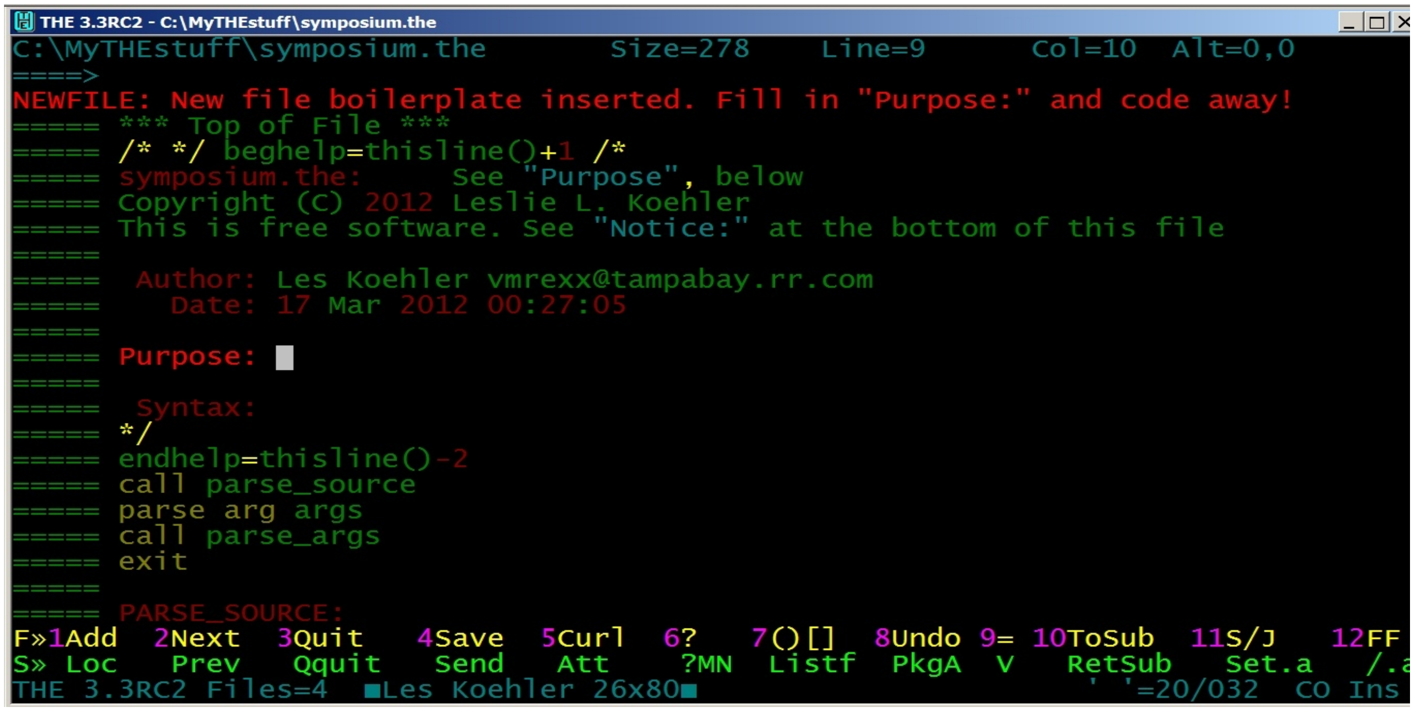
The macro *newfile*, which you saw earlier, does this for Rexx and THE code, initializing the header block (which is used for Help) and including subroutines to handle parsing keyword arguments, instead of obscure, meaningless 'switches' based on the limited capability of early pc chips and memory limitations.

Combining this with the *getvar* function I presented back in December at the Rexx Symposium in Aruba, yields the following THE code:

```
INIT_REXX:
  copyright_name=getvar('copyright_name')
  author=getvar('author')
  hdr=fname.1()'.'ftype.1()':     See "Purpose", below'
  cpyr='Copyright (C)' Word(Date(),3) copyright_name
  notic='This is free software. See "Notice:" at the bottom of this file'
  sc='/'||'*'
  ec='*'||'/'
  'input' sc   ec 'beghelp=thisline()+1' sc
  'input' hdr
  'input' cpyr
  'input' notic
  'input  '
  'input  Author:' author
  'input    Date:' Date() Time()
  'input  '
  'input Purpose:'
  'input  '
  'input  Syntax:'
  'input' ec
  'input endhelp=thisline()-2'
  Call skeleton
  '-/Purpose:'
  'extract /curline/'
  'cursor escreen' curline.2 Length(curline.3)+2
  'set alt 0 0'
  exists?=1
  Call next 'msg','New file boilerplate inserted. Fill in "Purpose:"',
   'and code away!'

  Return
```

SKELETON: is where all the subroutines are extracted from the macro and added to the new file, using the same technique as TABLE_HTML: above.

The initial result might look like this:



```
THE 3.3RC2 - C:\MyTHEstuff\symposium.the
C:\MyTHEstuff\symposium.the        Size=278    Line=9      Col=10  Alt=0,0
====>
NEWFILE: New file boilerplate inserted. Fill in "Purpose:" and code away!
===== *** Top of File ***
===== /* */ beghelp=thisline()+1 /*
===== symposium.the:     See "Purpose", below
===== Copyright (C) 2012 Leslie L. Koehler
===== This is free software. See "Notice:" at the bottom of this file
=====
=====  Author: Les Koehler vmrexx@tampabay.rr.com
=====    Date: 17 Mar 2012 00:27:05
=====
===== Purpose: █
=====
=====  Syntax:
===== */
===== endhelp=thisline()-2
===== call parse_source
===== parse arg args
===== call parse_args
===== exit
=====
===== PARSE_SOURCE:
F»1Add  2Next  3Quit   4Save  5Curl  6?   7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att    ?MN  Listf  PkgA  V   RetSub   Set.a   /.
THE 3.3RC2 Files=4  ■Les Koehler 26x80■                     ' '=20/032  CO Ins
```

newfile symposium.the

Notice that:

- The new file was placed in the folder I want it in.
- The header block has been appropriately tailored.
- The cursor is placed properly for you to enter the Purpose.
- You can see the start of the skeleton code, so all you have to do is code the details, starting with *INIT_VARS:* A piece of it is shown below:

```
INIT_VARS:
  valids='?  /? -? Help /Help -Help --Help' /* Keywords                */
  abbrev='1  2  2  1    2     2     3     ' /* Minimum abbreviation    */
  flags=copies('Help? ',words(valids))      /* Flag to set for keyword */
  helps=valids
  valids=valids '   ' --< Your keywords
  abbrev=abbrev '   ' --< Your abbreviations
  flags=flags   '   ' --< Your flagnames
  flags=flags 'Unknown? Keyword_parms?'        /* Always the last ones */
```

A little later in the code is where you define keywords that have parameters.

About 278 lines of commentary and code have been provided to get you started.

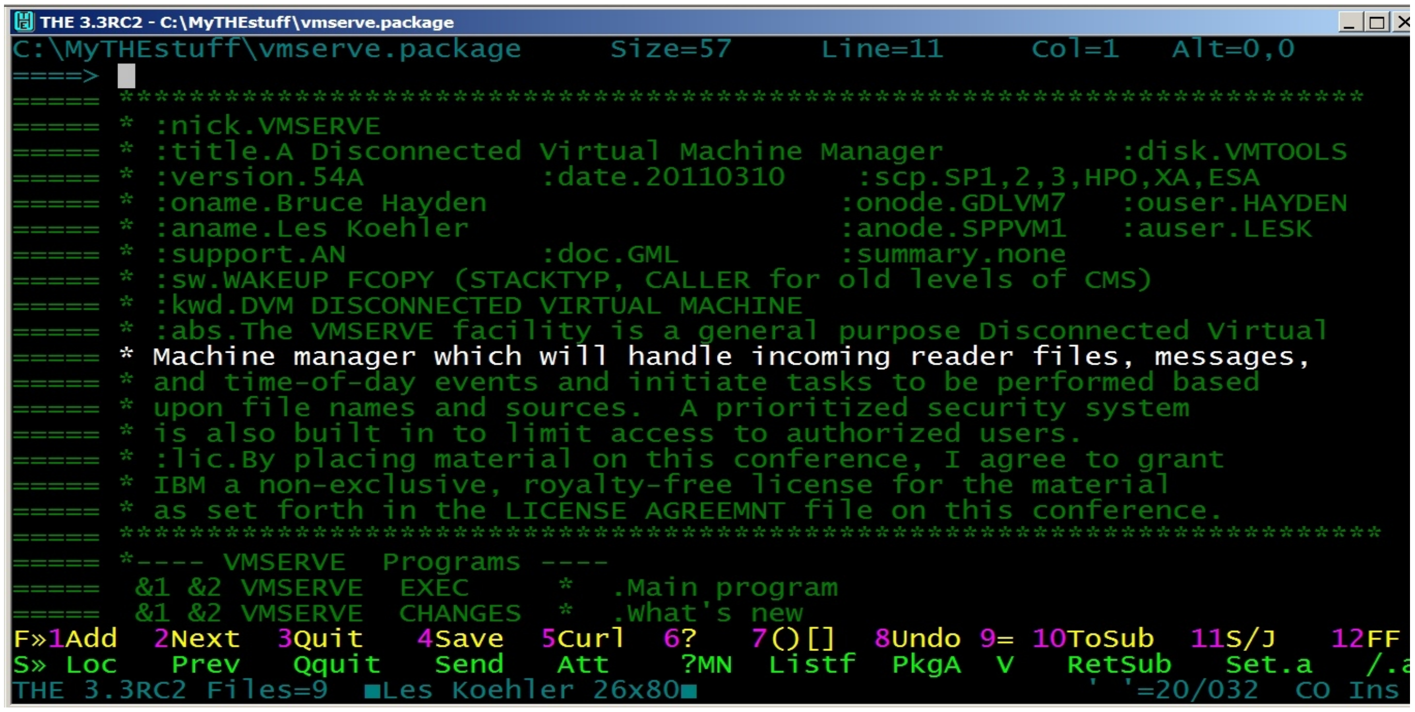
You saw earlier the Help for the *ff* macro, but I didn't show the interesting part of the source:

```
Purpose: Flip-flop between a selection of up to &maxsaves files in the ring.

 Syntax: &sme [Clear | Add | List | Delete]
       : &sme [&helps]
       : &sme

   If you haven't SET MACRO ON then use the command: macro &sme

  Notes: &sme manages a subset of the files in the ring to allow you to
         easily rotate through the list it maintains. It's of particular
         use when you have a lot of files in the ring but you want to
         concentrate on just a few of them without using something like
         RINGLIST all the time. Or even NEXTWINDOW, PREVWINDOW. Instead,
         just enter: &sme or put MACRO &sme on a hotkey/pfkey and use
         &sme ADD while editing a file you want &sme to keep.
```

The *HELP* subroutine will resolve all the ampersand variables. So if the macro gets renamed, the only change necessary is the fileid near the top of the header. Also notice that the number of fileids that can be saved is set by the variable *maxsaves*. If the variable is changed, the Help will still be correct!

Packaging code to include a Table of Contents and origin, using 7zip

The idea of "packaging" comes from my mainframe VM/CMS days. We had a PACKAGE tool that provided an agreed-to syntax for a package header and a list of files that made up the package:



```
THE 3.3RC2 - C:\MyTHEstuff\vmserve.package
C:\MyTHEstuff\vmserve.package     Size=57     Line=11     Col=1   Alt=0,0
====>
===== *****************************************************************
===== * :nick.VMSERVE
===== * :title.A Disconnected Virtual Machine Manager          :disk.VMTOOLS
===== * :version.54A          :date.20110310    :scp.SP1,2,3,HPO,XA,ESA
===== * :oname.Bruce Hayden                    :onode.GDLVM7    :ouser.HAYDEN
===== * :aname.Les Koehler                     :anode.SPPVM1    :auser.LESK
===== * :support.AN           :doc.GML          :summary.none
===== * :sw.WAKEUP FCOPY (STACKTYP, CALLER for old levels of CMS)
===== * :kwd.DVM DISCONNECTED VIRTUAL MACHINE
===== * :abs.The VMSERVE facility is a general purpose Disconnected Virtual
===== * Machine manager which will handle incoming reader files, messages,
===== * and time-of-day events and initiate tasks to be performed based
===== * upon file names and sources.  A prioritized security system
===== * is also built in to limit access to authorized users.
===== * :lic.By placing material on this conference, I agree to grant
===== * IBM a non-exclusive, royalty-free license for the material
===== * as set forth in the LICENSE AGREEMNT file on this conference.
===== ******************************************************************
===== *---- VMSERVE  Programs ----
=====  &1 &2 VMSERVE  EXEC      *  .Main program
=====  &1 &2 VMSERVE  CHANGES   *  .What's new
F»1Add  2Next  3Quit   4Save  5Curl  6?   7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att    ?MN  Listf  PkgA  V   RetSub   Set.a   /.a
THE 3.3RC2 Files=9  ■Les Koehler 26x80■                   ' '=20/032  CO Ins
```

vmserve_package

Another tool, FCOPY, could compress files into a single PACKLIB file, much like any zipping tool does on the pc.

Of course there were several tools for handling lists of files.

One of the goals I had was to make it easier for THE users to freely exchange tools amongst themselves, limiting the need to put them in separate folders and having to update the MACRO_PATH setting to include another folder.

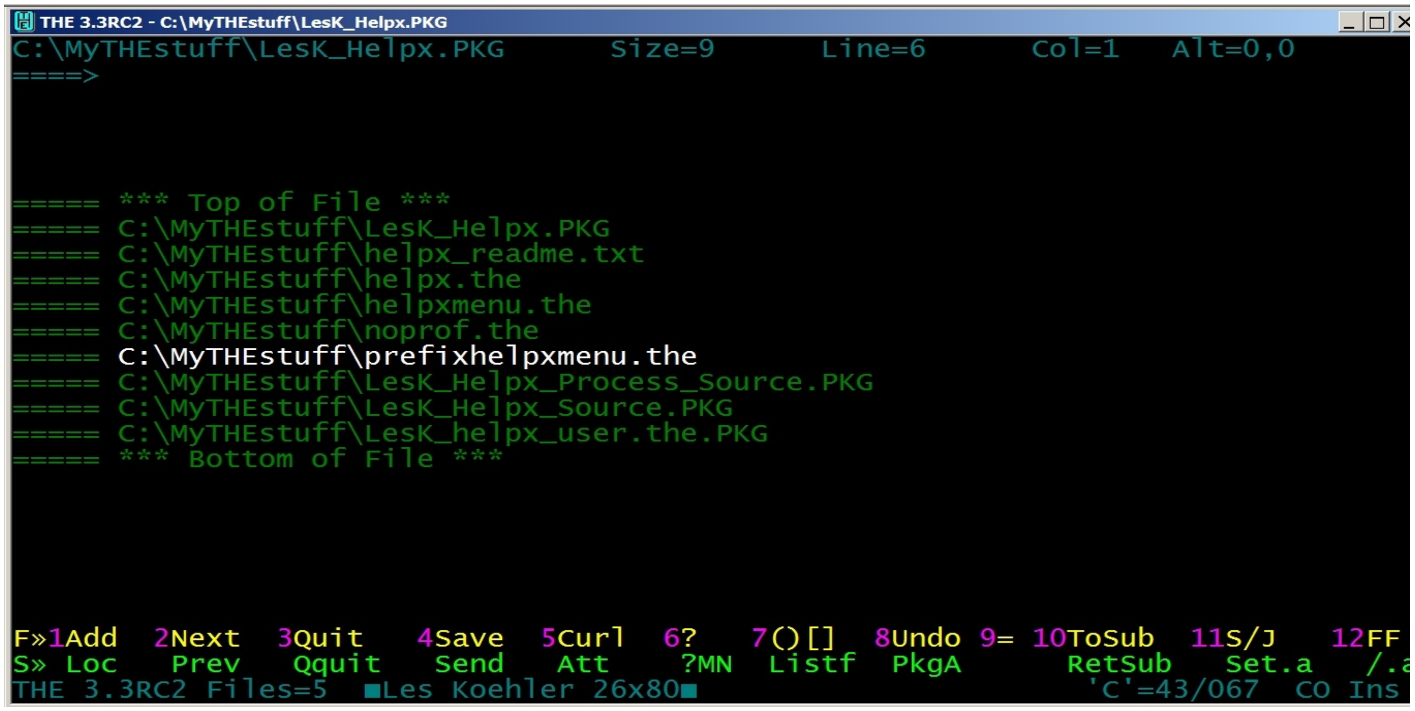
An outgrowth of this idea was what I now call "Virtual Directories", which I presented in Part One in Aruba.

Briefly, a Virtual Directory is a list of fully qualified fileids and macros that recognize it and treat it just like it was a DIR.DIR that THE shows when you issue the *dir* command. For my purposes, I've chosen a file extension (file type, as it's known in THE) of PKG, upper cased, as shown, so that it stands out. See last year's presentation for more details.

Creating a package is as simple as:

1. Edit the file that is the main file for the package
2. Issue the
   pkg =
   macro to define the package fileid, including a prefix that you've already tailored. Mine, for instance, is "LesK".
3. Navigate to the DIR of choice (or another PKG) and use
   pkg add to add files to the list.

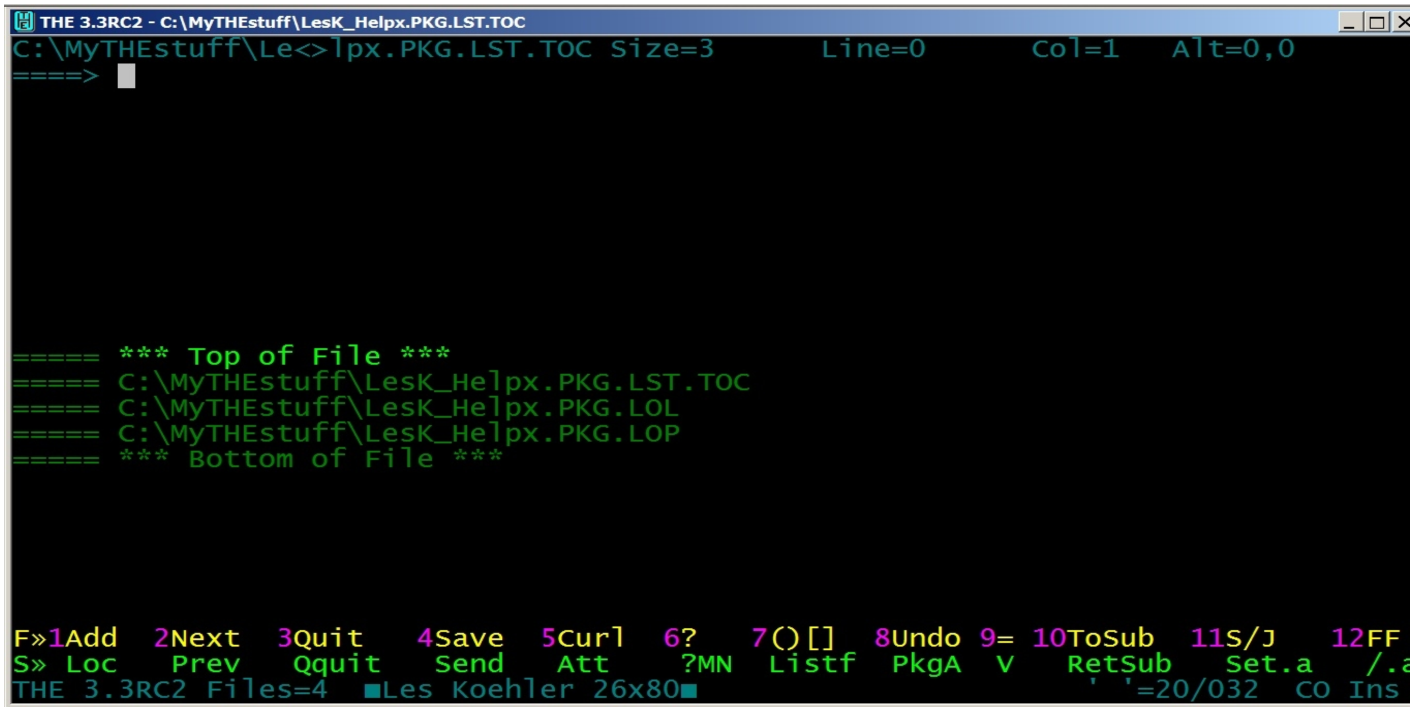
You might have created something like this:



```
THE 3.3RC2 - C:\MyTHEstuff\LesK_Helpx.PKG
C:\MyTHEstuff\LesK_Helpx.PKG       Size=9       Line=6       Col=1    Alt=0,0
====>



===== *** Top of File ***
===== C:\MyTHEstuff\LesK_Helpx.PKG
===== C:\MyTHEstuff\helpx_readme.txt
===== C:\MyTHEstuff\helpx.the
===== C:\MyTHEstuff\helpxmenu.the
===== C:\MyTHEstuff\noprof.the
===== C:\MyTHEstuff\prefixhelpxmenu.the
===== C:\MyTHEstuff\LesK_Helpx_Process_Source.PKG
===== C:\MyTHEstuff\LesK_Helpx_Source.PKG
===== C:\MyTHEstuff\LesK_helpx_user.the.PKG
===== *** Bottom of File ***






F»1Add   2Next   3Quit   4Save   5Curl   6?    7()[]   8Undo 9= 10ToSub  11S/J   12FF
S» Loc    Prev   Qquit    Send    Att     ?MN   Listf   PkgA      RetSub   Set.a   /.
THE 3.3RC2 Files=5  ■Les Koehler 26x80■                        'C'=43/067  CO Ins
```

helpx_pkg

The next thing to do is to run the
explode
macro to resolve everything. At completion, it will show you the TOC file that it created:



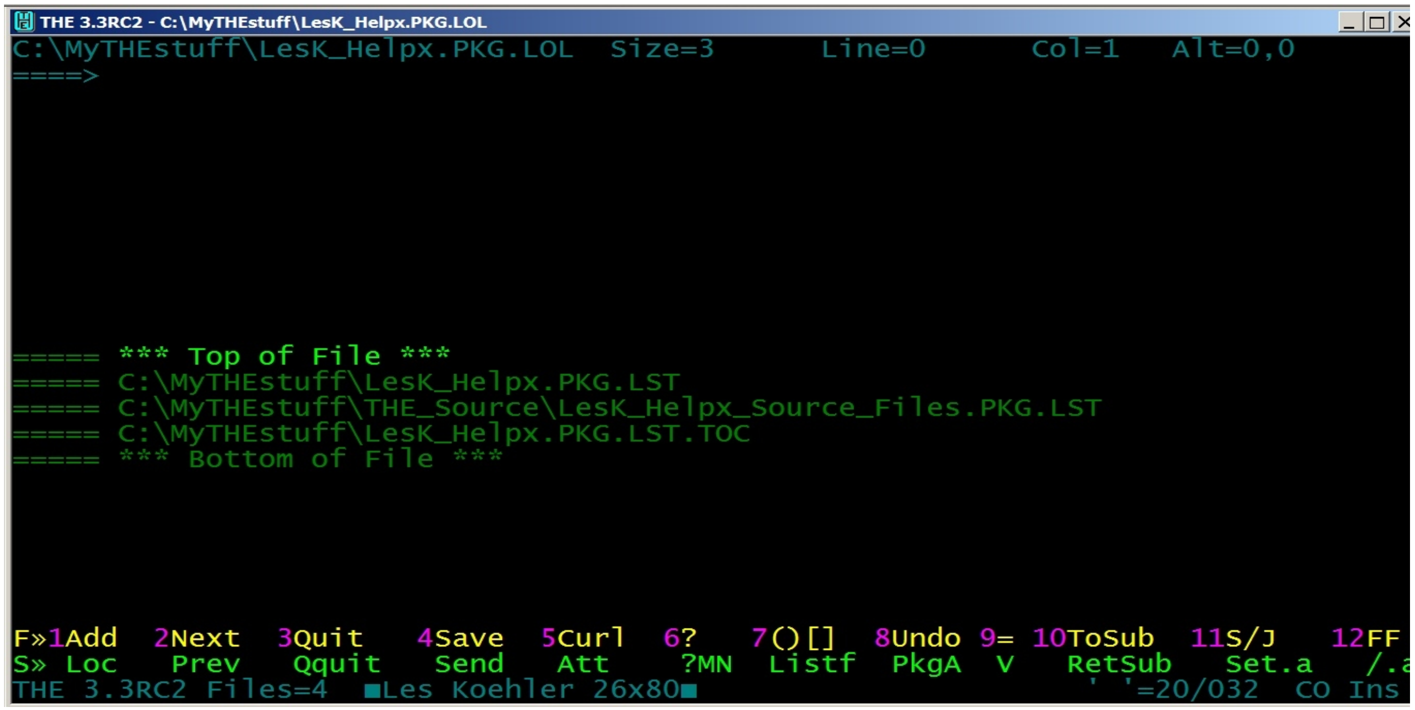
```
THE 3.3RC2 - C:\MyTHEstuff\LesK_Helpx.PKG.LST.TOC
C:\MyTHEstuff\Le<>lpx.PKG.LST.TOC Size=3      Line=0      Col=1   Alt=0,0
====> 

===== *** Top of File ***
===== C:\MyTHEstuff\LesK_Helpx.PKG.LST.TOC
===== C:\MyTHEstuff\LesK_Helpx.PKG.LOL
===== C:\MyTHEstuff\LesK_Helpx.PKG.LOP
===== *** Bottom of File ***

F»1Add  2Next  3Quit   4Save  5Curl  6?   7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att    ?MN  Listf  PkgA  V   RetSub   Set.a   /.a
THE 3.3RC2 Files=4  ■Les Koehler 26x80■                   ' '=20/032  CO Ins
```

helpx_TOC

which contains:

- LOL - List of Lists
- LOP - List of PKG files

The LOL looks like this:



```
THE 3.3RC2 - C:\MyTHEstuff\LesK_Helpx.PKG.LOL
C:\MyTHEstuff\LesK_Helpx.PKG.LOL  Size=3      Line=0      Col=1   Alt=0,0
====>

===== *** Top of File ***
===== C:\MyTHEstuff\LesK_Helpx.PKG.LST
===== C:\MyTHEstuff\THE_Source\LesK_Helpx_Source_Files.PKG.LST
===== C:\MyTHEstuff\LesK_Helpx.PKG.LST.TOC
===== *** Bottom of File ***

F»1Add  2Next  3Quit   4Save  5Curl  6?   7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att    ?MN  Listf  PkgA  V   RetSub   Set.a   /.a
THE 3.3RC2 Files=4  ■Les Koehler 26x80■                   ' '=20/032  CO Ins
```

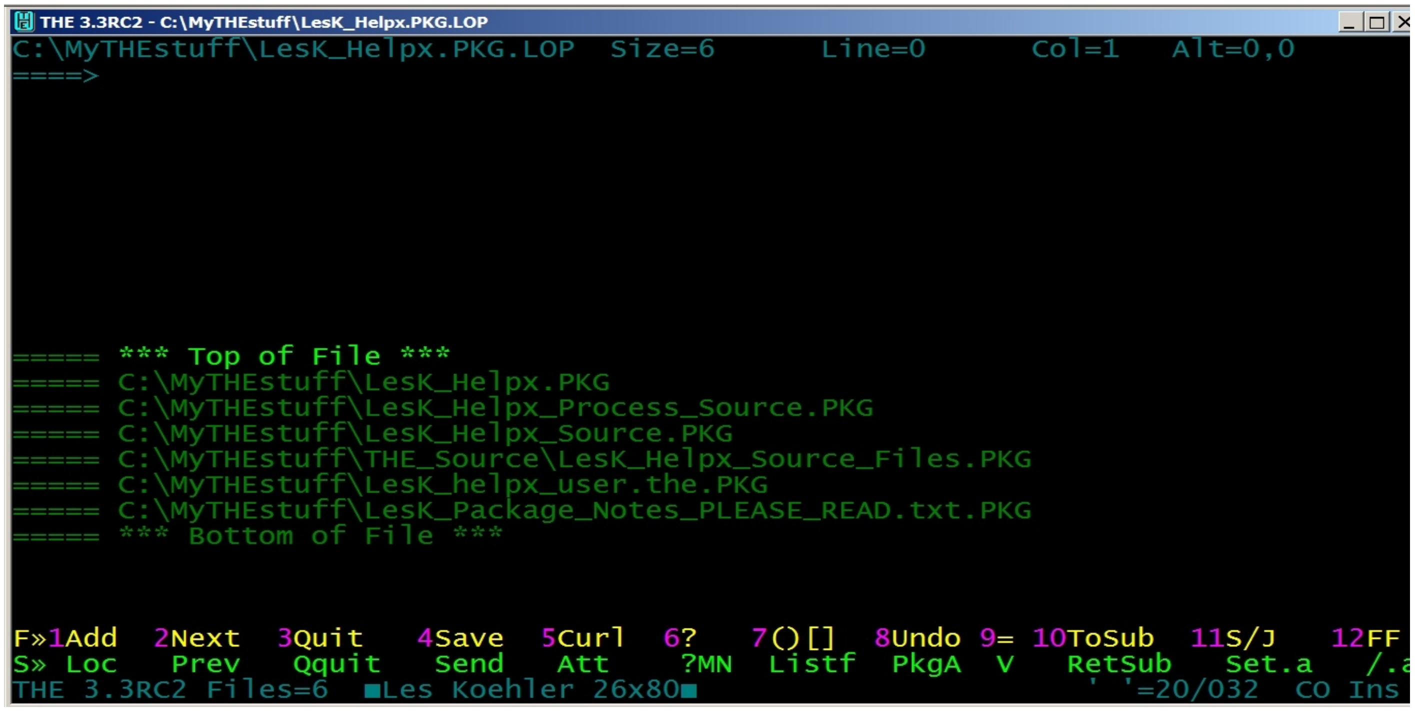
helpx_LOL

It contains a LST entry for each unique path, since 7zip uses relative paths when it builds an archive.

The LST files themselves contain the fully qualified fileids of the files to put in the archive.

There is also a pointer back to the parent TOC file.

Here is the LOP:



```
THE 3.3RC2 - C:\MyTHEstuff\LesK_Helpx.PKG.LOP
C:\MyTHEstuff\LesK_Helpx.PKG.LOP  Size=6      Line=0      Col=1   Alt=0,0
====>






===== *** Top of File ***
===== C:\MyTHEstuff\LesK_Helpx.PKG
===== C:\MyTHEstuff\LesK_Helpx_Process_Source.PKG
===== C:\MyTHEstuff\LesK_Helpx_Source.PKG
===== C:\MyTHEstuff\THE_Source\LesK_Helpx_Source_Files.PKG
===== C:\MyTHEstuff\LesK_helpx_user.the.PKG
===== C:\MyTHEstuff\LesK_Package_Notes_PLEASE_READ.txt.PKG
===== *** Bottom of File ***


F»1Add  2Next  3Quit   4Save  5Curl  6?   7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att    ?MN  Listf  PkgA  V   RetSub   Set.a   /.a
THE 3.3RC2 Files=6  ■Les Koehler 26x80■              ' '=20/032  CO Ins
```

helpx_LOP

which provides documentation about what PKG files are present.

You might have noticed that there is a PKG that did not appear in the original PKG file:

```
C:\MyTHEstuff\LesK_Package_Notes_PLEASE_READ.txt.PKG
```

It contains information about the packaging technique and how to take full advantage of it, including the macros to do so. One of the macros, **fix_pkg**, can be tailored to fix the contents of the PKG, TOC, LOL, LOP and LST files to match where you unzipped the package to. If a line in such a file matches its own fileid, and some do, that line is skipped. This leaves behind a pointer to where the file originally lived.

Another macro, **addtoring**, can be used to add files from a list to the ring of files being edited.

That completes gathering and resolving all the information associated with a PKG.

The next step is to build the zip file(s) with the **package** macro.

If the package is to be sent to another THE user, then simply enter:

package

while still in the TOC file.

If it is going to someone who doesn't need all the additional THE information, then enter:

package no

package will build the zip files and put them in the originating folders.

It will also build a file like this, for the example we've been using:

C:\MyTHEstuff\LesK_Helpx.PKG.LST.zips
====================================

```
C:\MyTHEstuff\LesK_Helpx.PKG.zip
C:\MyTHEstuff\THE_Source\LesK_Helpx_Source_Files.PKG.zip
```

This makes it easier to send the zip files to someone else, perhaps with the **att** macro, which recognizes the *.zips* file type and attaches the files to the **note** when the note is sent.

Extended Help for THE, built from the C source code. Includes a User Exit to integrate your own macros.

Mark Hessling, author of THE, keeps the text for creating the PDF for THE in the C source code. This makes it (relatively) easy to create an Extended Help Facility beyond what is already available:

1. Built-in Quick Help file
2. The PDF file for THE
3. The web site provided by Franz-Josef Wirtz THE Help (3.0) http://www.gut-wirtz.de/THE/rearranged/index.htm

All of them are helpful and each has its own advantages, but none can readily be extended to encompass user macros or other environments. Plus, they just don't work like the flexible CMS and Xedit Help that I'm used to.

The source code for THE is kept in individual files, which I created a PKG file for:

```
C:\MyTHEstuff\THE_Source\LesK_Helpx_Source_Files.PKG
C:\MyTHEstuff\THE_Source\comm1.c
C:\MyTHEstuff\THE_Source\comm2.c
C:\MyTHEstuff\THE_Source\comm3.c
C:\MyTHEstuff\THE_Source\comm4.c
C:\MyTHEstuff\THE_Source\comm5.c
C:\MyTHEstuff\THE_Source\commset1.c
C:\MyTHEstuff\THE_Source\commset2.c
C:\MyTHEstuff\THE_Source\commsos.c
C:\MyTHEstuff\THE_Source\query.c
C:\MyTHEstuff\THE_Source\appendix.1
C:\MyTHEstuff\THE_Source\appendix.2
C:\MyTHEstuff\THE_Source\appendix.3
C:\MyTHEstuff\THE_Source\appendix.4
C:\MyTHEstuff\THE_Source\appendix.7
```

```
C:\MyTHEstuff\THE_Source\overview
C:\MyTHEstuff\THE_Source\glossary
```

For my purposes, the ones I needed were grouped into:

- Commands
- SET
- SOS
- Query/Status/Extract
- Boolean
- Other

and a matching macro was written to extract the position of the help information by searching on the marker lines:

```
/*man-start**********************************
**man-end***********************************/
```

and writing the data to
.idx
and
.txt
files, to improve interactive performance.

The *.txt* files are preformatted to be used as "List" files, with one line for each command, like this:

```
add - add blank line
alert - display a user configurable dialog box with notification
all - select and display restricted set of lines
backward - scroll backwards [n] screens
bottom - move to the bottom of the file
cancel - quit from all unaltered files in the ring
cappend - append text after column pointer
```

The *.idx* files are used as index files to the actual help prose. For example:

```
C:\MyTHEstuff\THE_Source\comm1.c (GET  - :47 3 )
C:\MyTHEstuff\THE_Source\comm1.c (GET add - :53 30 )
C:\MyTHEstuff\THE_Source\comm1.c (GET alert - :133 28 )
C:\MyTHEstuff\THE_Source\comm1.c (GET all - :185 36 )
C:\MyTHEstuff\THE_Source\comm1.c (GET backward - :370 29 )
C:\MyTHEstuff\THE_Source\comm1.c (GET bottom - :482 19 )
C:\MyTHEstuff\THE_Source\comm1.c (GET cancel - :550 20 )
```

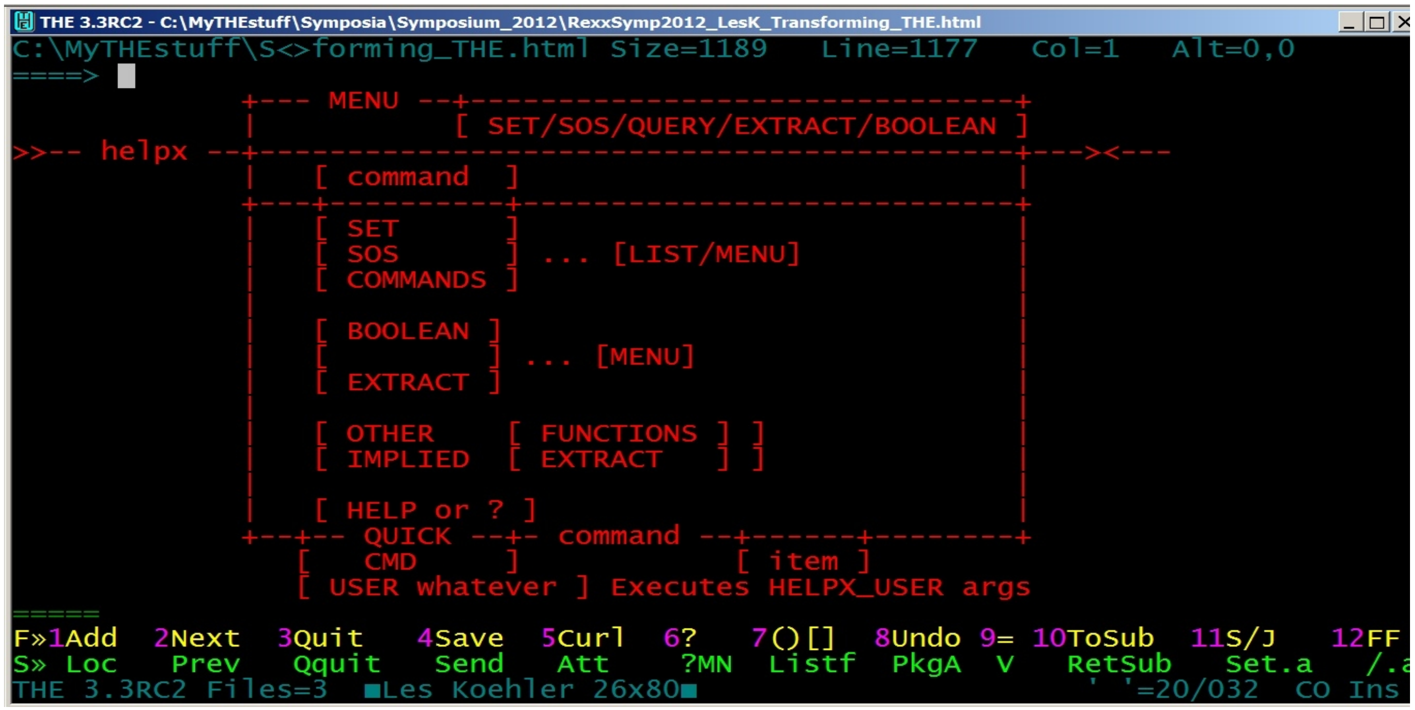
These files are the basis for the Extended Help Facility. The same methodologies can be applied to provide additional information, all integrated into the *help* command provided by THE, using the

synonym help macro helpx

command.

This tells THE to run the **helpx** macro when you enter the *help* command.

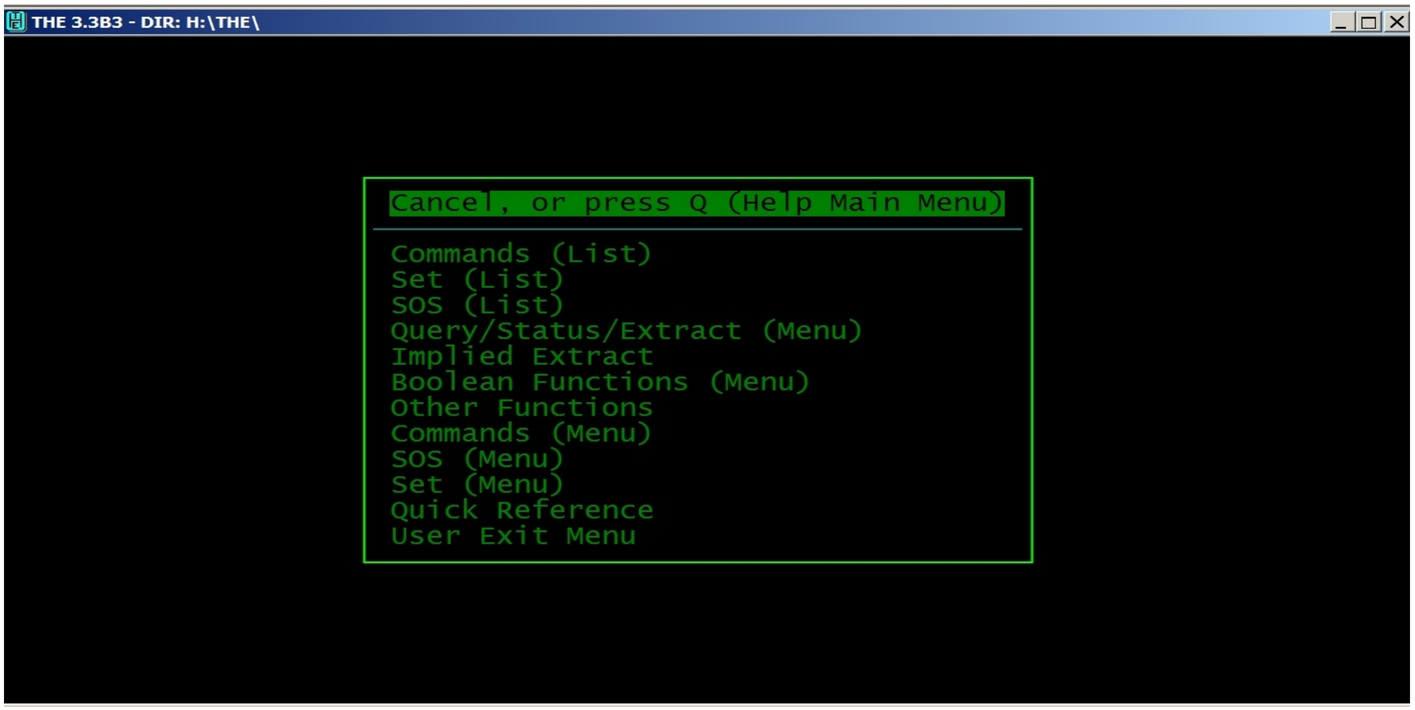
The syntax looks like this:



```
THE 3.3RC2 - C:\MyTHEstuff\Symposia\Symposium_2012\RexxSymp2012_LesK_Transforming_THE.html
C:\MyTHEstuff\S<>forming_THE.html Size=1189   Line=1177   Col=1   Alt=0,0
====> 
                +--- MENU --+-----------------------------------+
                |           [ SET/SOS/QUERY/EXTRACT/BOOLEAN ]
>>-- helpx --+--------------------------------------------+--->
                |    [ command   ]                              |
                +---+----------+----------------------------+
                |   [ SET      ]                              |
                |   [ SOS      ] ... [LIST/MENU]              |
                |   [ COMMANDS ]                              |
                |                                             |
                |   [ BOOLEAN ]                               |
                |   [         ] ... [MENU]                    |
                |   [ EXTRACT ]                               |
                |                                             |
                |   [ OTHER     [ FUNCTIONS ] ]               |
                |   [ IMPLIED   [ EXTRACT   ] ]               |
                |                                             |
                |   [ HELP or ? ]                             |
                +--+-- QUICK --+-- command --+------+--------+
                   [    CMD    ]             [ item ]
                   [ USER whatever ] Executes HELPX_USER args
=====
F»1Add  2Next  3Quit   4Save  5Curl  6?   7()[]  8Undo 9= 10ToSub  11S/J   12FF
S» Loc   Prev   Qquit   Send   Att    ?MN  Listf  PkgA  V   RetSub   Set.a   /.a
THE 3.3RC2 Files=3  ■Les Koehler 26x80■                   ' '=20/032  CO Ins
```

helpx_help

Its default Menu mode with no arguments yields:



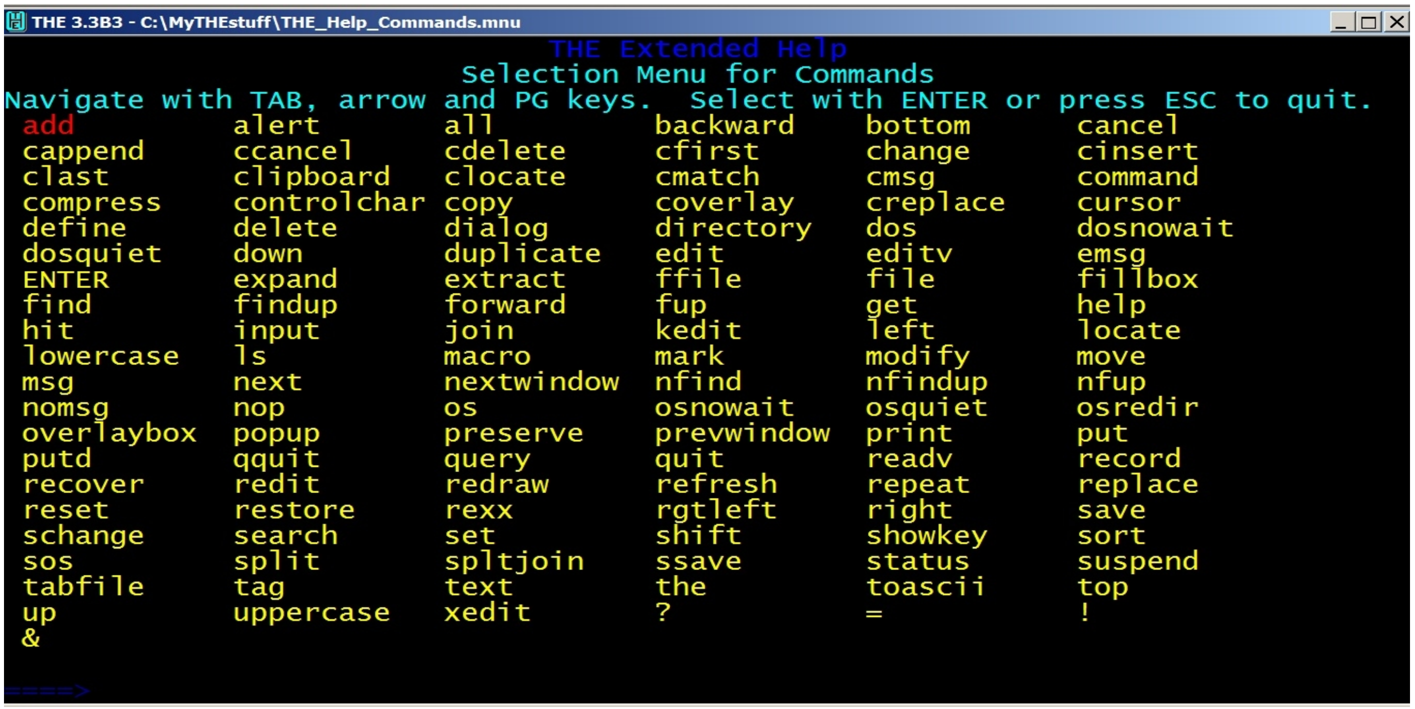
```
THE 3.3B3 - DIR: H:\THE\

    Cancel, or press Q (Help Main Menu)
    -----------------------------------
    Commands (List)
    Set (List)
    SOS (List)
    Query/Status/Extract (Menu)
    Implied Extract
    Boolean Functions (Menu)
    Other Functions
    Commands (Menu)
    SOS (Menu)
    Set (Menu)
    Quick Reference
    User Exit Menu
```

helpx_menu

If you pass an argument, for instance:

help commands

you get:



```
THE 3.3B3 - C:\MyTHEstuff\THE_Help_Commands.mnu
                                THE Extended Help
                            Selection Menu for Commands
Navigate with TAB, arrow and PG keys.  Select with ENTER or press ESC to quit.
 add          alert        all          backward     bottom       cancel
 cappend      ccancel      cdelete      cfirst       change       cinsert
 clast        clipboard    clocate      cmatch       cmsg         command
 compress     controlchar  copy         coverlay     creplace     cursor
 define       delete       dialog       directory    dos          dosnowait
 dosquiet     down         duplicate    edit         editv        emsg
 ENTER        expand       extract      ffile        file         fillbox
 find         findup       forward      fup          get          help
 hit          input        join         kedit        left         locate
 lowercase    ls           macro        mark         modify       move
 msg          next         nextwindow   nfind        nfindup      nfup
 nomsg        nop          os           osnowait     osquiet      osredir
 overlaybox   popup        preserve     prevwindow   print        put
 putd         qquit        query        quit         readv        record
 recover      redit        redraw       refresh      repeat       replace
 reset        restore      rexx         rgtleft      right        save
 schange      search       set          shift        showkey      sort
 sos          split        spltjoin     ssave        status       suspend
 tabfile      tag          text         the          toascii      top
 up           uppercase    xedit        ?            =            !
 &

====>
```

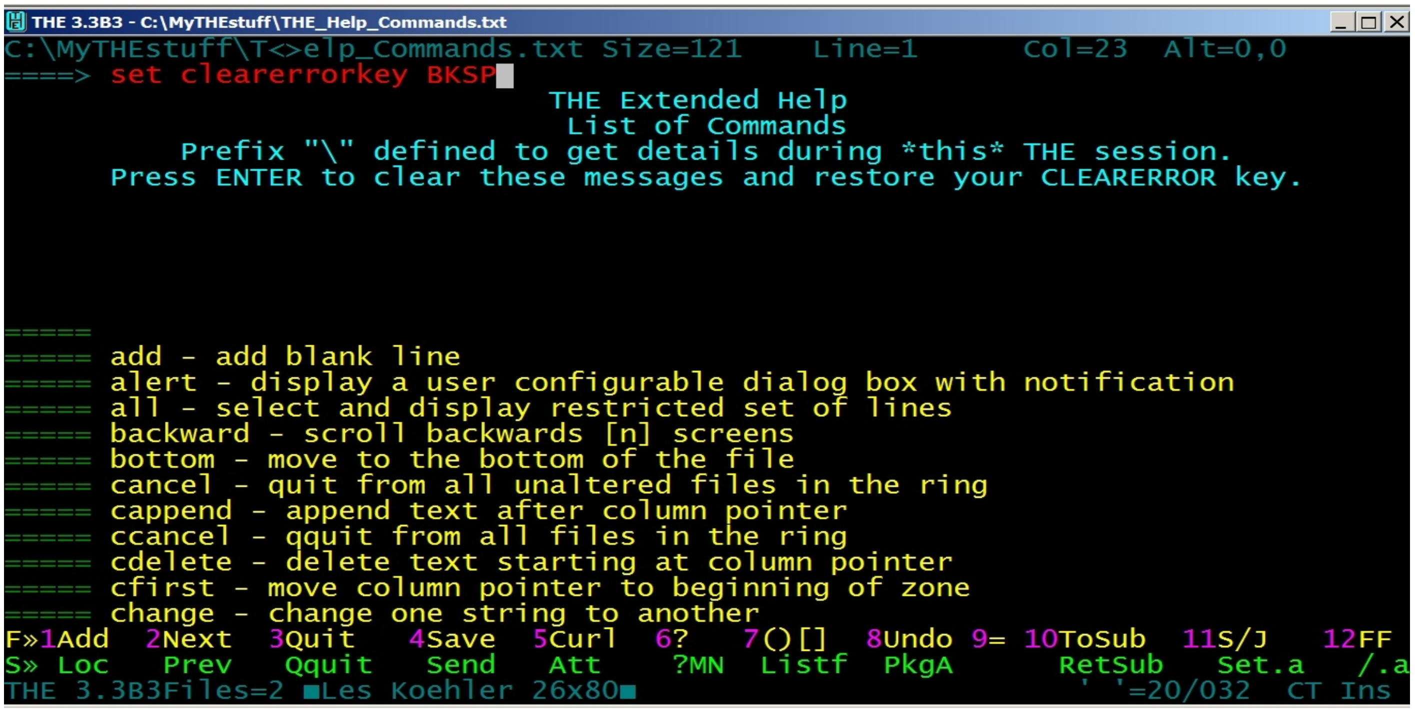
help_commands

As you navigate the menu, your selection is shown in red. The colors, and many other settings are, of course, configurable by editing **helpx_config.rex** to suit your own tastes.

The *list* argument, like this:

help commands list

overrides the default Menu mode to give you:



```
THE 3.3B3 - C:\MyTHEstuff\THE_Help_Commands.txt
C:\MyTHEstuff\T<>elp_Commands.txt Size=121    Line=1      Col=23   Alt=0,0
====> set clearerrorkey BKSP
                              THE Extended Help
                               List of Commands
          Prefix "\" defined to get details during *this* THE session.
      Press ENTER to clear these messages and restore your CLEARERROR key.





=====
===== add - add blank line
===== alert - display a user configurable dialog box with notification
===== all - select and display restricted set of lines
===== backward - scroll backwards [n] screens
===== bottom - move to the bottom of the file
===== cancel - quit from all unaltered files in the ring
===== cappend - append text after column pointer
===== ccancel - qquit from all files in the ring
===== cdelete - delete text starting at column pointer
===== cfirst - move column pointer to beginning of zone
===== change - change one string to another
F»1Add   2Next   3Quit    4Save   5Curl   6?    7()[]   8Undo 9= 10ToSub   11S/J    12FF
S» Loc     Prev    Qquit    Send     Att     ?MN   Listf   PkgA      RetSub    Set.a    /.a
THE 3.3B3Files=2 ■Les Koehler 26x80■                      ' '=20/032   CT Ins
```

help_commands_list

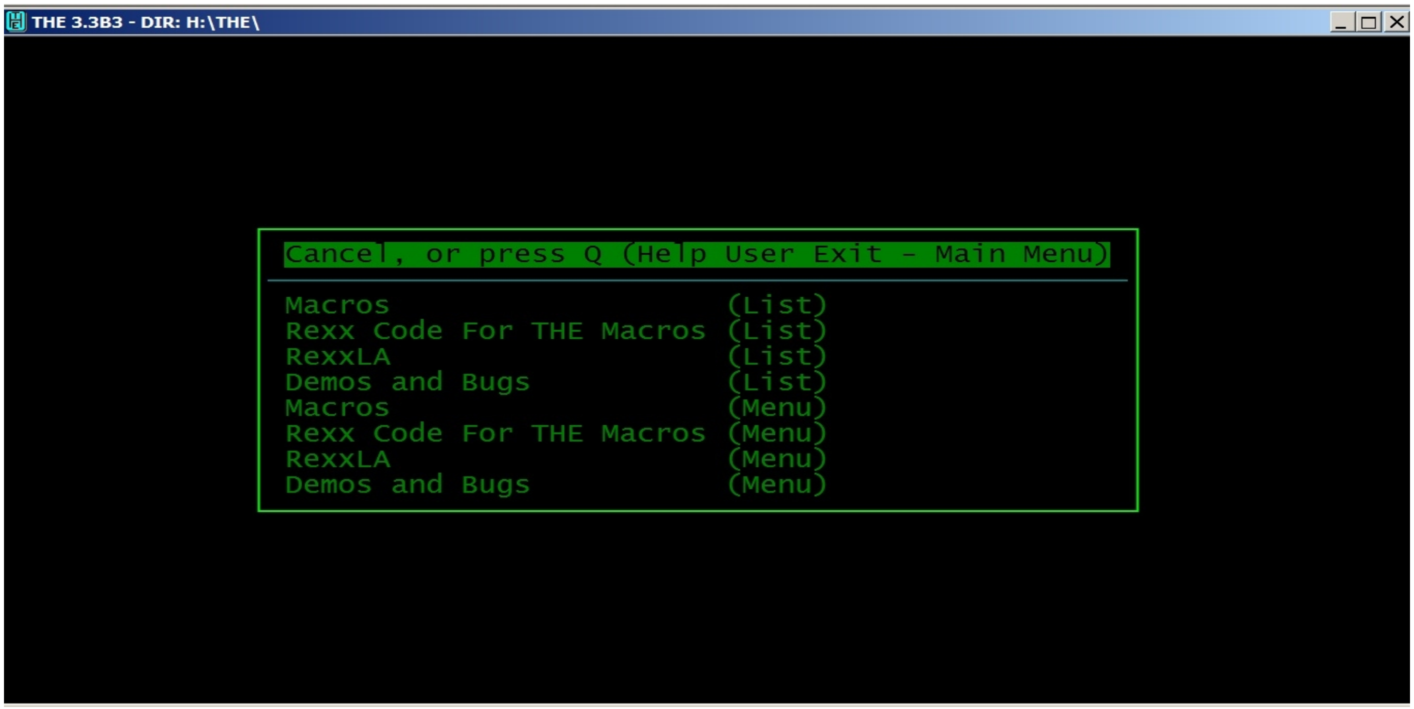
After you press *Enter* you have a regular file that you can search, selecting a command by putting

"\" in the prefix area.

You can leave this file in the ring and always return to it using F2, **ff** or **ringlist**.

User Exit

The User Exit capability can be used to extend the Help to whatever you need. As an example, I created this to meet my own needs:



```
Cancel, or press Q (Help User Exit - Main Menu)

Macros                    (List)
Rexx Code For THE Macros  (List)
RexxLA                    (List)
Demos and Bugs            (List)
Macros                    (Menu)
Rexx Code For THE Macros  (Menu)
RexxLA                    (Menu)
Demos and Bugs            (Menu)
```

help_user

Summary

Last year we explored these macros or features in some detail:

- SHOWS
- Favorite DIRS
- Virtual DIR
- VIEW
- SCANFILE

Now we've seen a few more of the tools that demonstrate elevating THE from the realm of "Just an editor" to that of a "Productivity tool":

- Command recall
- Executing modified Rexx code without you having to SAVE it
- Coding assistance for Classic Rexx and HTML
- Split screen. Both horizontal and vertical

- Finding the differences between two files
- File selection assistance at edit time
- Selecting a file from the ring of active files being edited
- Switching between a limited subset of all the files in the ring
- Carrying skeleton code within a macro, instead of in a separate file
- Packaging code to include a Table of Contents and origin, using 7zip
- Extended Help for THE, built from the C source code. Includes a User Exit to integrate your own macros.

And these are just a few of the 140 general purpose macros that I use to enhance my own productivity!