The aeb_mlink Package
A member of the AeB Pro family

D. P. Story
Email: dpstory@uakron.edu

processed January 9, 2020

Contents

1 Introduction

This package creates multiline-links. The package hyperref does create links, but generally these links cannot be broken across lines, unless pdflatex is used to create a PDF.

This package uses the QuadPoints entry in the link annotation to create a bounding region; consequently, this package requires **Acrobat Distiller** to create a PDF. QuadPoints is a PDF 1.6 feature, so these multiline links will work in Adobe Reader 7.0 or later. If viewed in a version of Adobe Reader previous to 7.0, the viewer will use the underlying bounding box.

LaTeX package requirements are the eForms and hyperref. Only the use of dvips and dvipsone is supported.

The key to creating a multi-line is contained in Table 8.24 of the PDF Reference. The description of QuadPoints in the PDF Reference is as follows:

(Optional; PDF 1.6) An array of $8 \times n$ numbers specifying the coordinates of n quadrilaterals in default user space that comprise the region in which the link should be activated. The coordinates for each quadrilateral are given in the order

$$x_1 \ y_1 \ x_2 \ y_2 \ x_3 \ y_3 \ x_4 \ y_4$$

specifying the four vertices of the quadrilateral in counterclockwise order. For orientation purposes, such as when applying an underline border style, the bottom of a quadrilateral is the line formed by (x_1, y_1) and (x_2, y_2). If this entry is not present or the viewer application does not recognize it, the region specified by the `Rect` entry should be used. `QuadPoints` should be ignored if any coordinate in the array lies outside the region specified by `Rect`.

2 The aeb-mlink Package

The aeb_mlink package is listed on CTAN as aeb-mlink; there was, in fact, no aeb-mlink. Here, we provide a ‘dummy’ package by that name which passes everything on to aeb_mlink.

```
% Begin Alt pkg
⟨*altpkgname⟩
\NeedsTeXFormat{LaTeX2e}
\RequirePackage{xkeyval}
\ProvidesPackage{aeb-mlink}
 [2018/04/26 v1.0 AeB MLink Alt-name (dps)]
\DeclareOptionX*{\PassOptionsToPackage{\CurrentOption}{aeb_mlink}}
\ProcessOptionsX
\RequirePackage{aeb_mlink}[2018/08/18]
⟨/altpkgname⟩
```

3 Package Requirements and Options

After having established the alternate of this package, we now work on the package itself.

```
% Begin Package
⟨*package⟩
\RequirePackage{xkeyval}
\RequirePackage{ifpdf}[2006/02/20]
\RequirePackage{ifxetex}[2006/08/21]
\newif\if@ml@dvips \@ml@dvipstrue
\def\mlcsarg#1#2{\expandafter#1\csname#2\endcsname}
```

`dvipsone` Set the driver for `dvipsone`

```
\DeclareOptionX{dvipsone}{\def\eq@drivernum{0}\@ml@dvipsfalse
  \PassOptionsToPackage{dvipsone}{eforms}
  \PassOptionsToPackage{dvipsone}{hyperref}
}
```

dvips Set the driver for dvips

```
\DeclareOptionX{dvips}{\def\eq@drivernum{0}\@ml@dvipstrue
  \PassOptionsToPackage{dvips}{eforms}
  \PassOptionsToPackage{dvips}{hyperref}
}
```

urlOpts The options of the url package may be passed through the value of this key; for example, `urlOpts={hyphens}`.

```
\define@key{aeb_mlink.sty}{urlOpts}[]{\def\url@Opts{[#1]}}
\let\url@Opts\@empty
```

Undefined options are passed to eforms.

```
\DeclareOptionX*{\PassOptionsToPackage{\CurrentOption}{eforms}}
```

dblevel Sets the debug level.

```
\@ifundefined{mldblevel}{\newcount\mldblevel\mldblevel=0 }{}
\define@key{aeb_mlink.sty}{dblevel}[0]{\mldblevel=#1 }
```

Package error message when not dvips (or dvispone)

```
\def\ml@err@msg{This package requires the driver dvips and\MessageBreak
  Adobe Distiller as the PDF creator}
```

(2020/01/06) Conform to the new `web.cfg` format.

```
\let\bWebCustomize\endinput
\let\eWebCustomize\relax
\ifpdf\PackageError{aeb_mlink}{\ml@err@msg}\else
  \ifxetex\PackageError{aeb_mlink}{\ml@err@msg}\else
    \let\ExecuteOptions@SAVE\ExecuteOptions
    \let\ExecuteOptions\ExecuteOptionsX
    \InputIfFileExists{web.cfg}{}
      {\@ifundefined{l@tex@@@@driver}{\ExecuteOptionsX{dvips}}
        {\ExecuteOptionsX{dvipsone}}}%
    \let\ExecuteOptions\ExecuteOptions@SAVE
\fi\fi
```

(2020/01/06) Now require url package and pass options to url through `urlOpts`.

```
\ProcessOptionsX
\expandafter\RequirePackage\url@Opts{url}
```

We require hyperref, eforms and soul. For eforms, a recent version is needed, 2008/03/14 or later.

```
\RequirePackage{hyperref}
\RequirePackage{refcount}
```

Beginning with the version of eforms dated 2018/03/22 or later, there are several link options that are defined. These are `\mlfix`, `\mlstrut`, `\mlcrackat` and `\mlhyph`.

```
\RequirePackage{eforms}[2018/08/16]
\RequirePackage{soul}
```

4 Driver Dependent Code

Driver dependent definitions for dvipsone and dvips.

```
\def\ps@mark{[\space}
\ifnum\mldblevel>0
  \def\mlpgMsg{(\string\n Beginning of page: ) pf
    PhysicalPage 20 string cvs
    pf(\string\n)pf}\else
  \def\mlpgMsg{}\fi
\def\pgmonitoring{\if@ml@dvips
    dup /PhysicalPage exch 1 add def
    /PhysicalPage PhysicalPage def^^J\fi
    \mlpgMsg
}
```

Redefining `\mllnkcontainer`, defined in eforms dated 2018/03/14 or later. If `\ifoldstylequads` is true, we do not set `/Rect` to the minimal rectangle, rather it is set to the region defining the whole page. This was the default rectangle in the past.

```
\def\smallRectTF{\ifoldstylequads false\else
  \iffixmlinks true\else false\fi\fi\space
  \ifSmallRect true\else false\fi\space and}
\def\ml@@nnotName{mLink} % dps
\def\mllnkcontainer#1{bCreateLink { xoMsgB {
  \smallRectTF\space mlRectFix^^J%
  #1}if}{(\ml@@nnotName\the\aeb@mLinkCnt) mlIsBldMsg}ifelse}
```

`\pboxRect` is defined in eforms

```
\def\pboxRect{mlRect }
\if@ml@dvips
```

dvips driver: Code for the dvips driver This next `\special` defines some standard conversion formulas, TEX to PDF and PDF to TEX for dvips.

```
  \def\mlDict{SDict}
  \special{!userdict begin
   /TeXtoPDF {65536 div DVImag mul} def          % sp to pts
   /PDFtoDvips {72.27 div Resolution mul} def    % points to dots
   /PDFtoVDvips {72.27 div VResolution mul} def  % points to dots
   /DvipstoPDF {72.27 mul Resolution div} def    % dots to points
   /HTeXtoDvips {TeXtoPDF PDFtoDvips} def        % sp to dots
   /VTeXtoDvips {TeXtoPDF PDFtoVDvips} def^^J%   % sp to dots
   /PhysicalPage 0 def^^J%
   /PageHeight {vsize} def^^J%
   /PDFtoTeX {PDFtoDvips} def^^J%
   /pf{print flush}def^^J%
   /bop-hook{ \pgmonitoring\space } def
  end}
```

This command calculates the `\QuadPoints` array when we are using the dvips driver.

```
  \def\setQuadBox{%
  currentpoint DvipstoPDF \aeb@bbox@dp\space TeXtoPDF add
  neg vsize add 72 sub                                % y1
  exch DvipstoPDF 72 add exch                         % x1
  2 copy exch \aeb@bbox@wd\space TeXtoPDF add exch    % x2
  2 copy \aeb@bbox@ht\space TeXtoPDF add              % y3
  2 copy exch \aeb@bbox@wd\space TeXtoPDF sub exch    % x4
}
```

For the bounding rectangle, we just enclose the entire page. This simplifies things greatly.

```
\def\par@@Rect
{%
  72 neg PDFtoDvips vsize 72 sub PDFtoVDvips
  hsize 72 sub PDFtoDvips 72 neg PDFtoVDvips
}
```

dvipsone driver: Code for the dvipsone driver This next `\special` defines some standard conversion formulas, TeX to PDF and PDF to TeX in the YandY TeX System.

```
\else
  \def\mlDict{dvidict}
  \special{!/TeXtoPDF {65536 div mag 1000 div mul} def
    /PDFtoTeX {65536 mul mag 1000 div div} def^^J%
    /pf{print flush}def^^J%
    /bphook{ \pgmonitoring\space } def^^J%
  }
```

This command calculates the `\QuadPoints` array when we are using the dvipsone driver.

```
  \def\setQuadBox{%
     currentpoint \aeb@bbox@dp\space add TeXtoPDF
     neg PageHeight add 72 sub                            % y1
     exch TeXtoPDF 72 add  exch                           % x1
     2 copy exch \aeb@bbox@wd\space TeXtoPDF add exch     % x2
     2 copy \aeb@bbox@ht\space TeXtoPDF add               % y3
     2 copy exch \aeb@bbox@wd\space TeXtoPDF sub exch     % x4
  }
```

For the bounding rectangle, we just enclose the entire page. This simplifies things greatly.

```
  \def\par@@Rect
  {%
      72 neg PDFtoTeX PageHeight 72 sub PDFtoTeX
      PageWidth 72 sub PDFtoTeX 72 neg PDFtoTeX
  }
\fi
```

5 The Multi-line Linking Commands

We use a box, and two counters for this package.

```
\newbox\aeb@bbox
\newcount\aeb@arrayIndx \aeb@arrayIndx=0
\newcount\aeb@mLinkCnt \aeb@mLinkCnt=0
\newcount\syllableCnt \syllableCnt=0
\newif\ifmllinktotalchanged\mllinktotalchangedfalse
```

When `\ifSmallRect` is true (the default), the smallest possible `Rect` is constructed to enclose the text; obeyed only when `\iffixmlinks` (defined in eforms is true.

```
\newif\ifSmallRect \SmallRecttrue
\AtEndDocument{\wrtmlinktot@l\ckchngmlinktot@l\wrt@linksnotformed}
\def\wrt@linksnotformed{\iflinknotformed
  \PackageWarningNoLine{aeb_mlink}{Some link calculations are not
  complete.\MessageBreak
  DO NOT CONVERT TO PDF at this time. Compile at \MessageBreak
  least twice more}\fi}
\def\wrtmlinktot@l{\immediate\write\@auxout{\string\gdef
  \string\mlinkstotal{\the\aeb@mLinkCnt}}}
\def\ckchngmlinktot@l{\@ifundefined{mlinkstotal}{}
    {\ml@mlinktot@l@changed}}
\def\ml@mlinktot@l@changed{%
  \ifnum\mlinkstotal=\the\aeb@mLinkCnt\relax\else
  \PackageWarningNoLine{aeb_mlink}{The number of links has
  changed. Compile again\MessageBreak until this message
  does not appear}\immediate
  \write\@auxout{\string\mllinktotalchangedtrue}\fi
}
\def\ml@mllinktotalchanged{\ifmllinktotalchanged
  \PackageWarningNoLine{aeb_mlink}
  {The number of links has changed, continue\MessageBreak
   to compile}\fi}
\AtBeginDocument{\ml@mllinktotalchanged}
\def\CurrentBorderColor{\@linkbordercolor}
\def\ml@nocolorHighlight{I}
\def\ml@nocolorLineStyle{S}
\def\ml@nocolorLineWidth{1}
\def\ml@setnocolorDefaults{%
\def\ml@nocolor@defaults{\H{\ml@nocolorHighlight}%
  \S{\ml@nocolorLineStyle}\W{\ml@nocolorLineWidth}%
  \Color{\CurrentBorderColor}}%
}
\ifHy@colorlinks
  \let\ml@nocolor@defaults\@empty
\else
  \ml@setnocolorDefaults
\fi
\def\ml@earlyExecProps#1{%
    \eq@setWidgetProps\relax{#1}%
}
```

The new scheme of fixing up the quad points write information to the AUX file, which then requires multiple compilations to bring that information up to date in the document. When working on document development, you can declare \OldStyleBoxesOn in the preamble to revert to the old style boxes that do not require AUX info.

```
\newif\ifoldstylequads \oldstylequadsfalse
\def\OldStyleBoxesOn{\mlfixOff\oldstylequadstrue}
\def\OldStyleBoxesOff{\oldstylequadsfalse}
\@onlypreamble\OldStyleBoxesOn
\@onlypreamble\OldStyleBoxesOff
\let\mlh@preambleCmdInsert\relax
\def\mlcs#1{\texttt{\@backslashchar#1}}
\bgroup\@makeother\%
\gdef\CMT#1{ %\space #1}\egroup
\def\mldbModeOn{\def\mldb##1##2{##2}}
\def\mldbModeOff{\def\mldb##1##2{}}
\def\mldb#1#2{\ifnum#1<\mldblevel#2\fi}
\def\ml@adj@x{2}\def\ml@adj@y{2}
\def\mlMaxNSylls{30}
% usage \mlcrackinsat{\removelastspace}
\def\removelastspace{\hskip-\fontdimen2\font}
\AtBeginDvi{\special{!%
```

Begin Postscript code This code is executed as the PDF is created by either Adobe Distiller (preferred) or ps2pdf. Information is written to the distiller (ps2pdf) log.

A switch to determine if the link should be created or now; it may not be fully formed.

```
/bCreateLink true def
```

The length of the arrays we deal with are multiples of eight (8), we use a value of 17 as a way of testing whether an array size has been updated. All arrays are set to 17 in length initially.

```
/mlIsBld 17 def^^J%
/mlIsBldMsg {^^J%
  /sName exch def^^J%
  (\string\n!! )pf
  sName 20 string cvs pf
  ( is not completely formed,
   compile again!!\string\n)pf^^J%
} def^^J%
/xoMsgB true def^^J%
/xoMsg {^^J%
  /Indx exch def^^J%
  /sName exch def^^J%
  /nSyllable Indx 8 div def^^J% dpsa08
  (!!------------------------------------------------------------------%
  \string\n Warning:\string\n
  The text of )pf
  sName 20 string cvs pf
  ( has crossed a page boundary from page )pf
  PhysicalPage 1 sub 10 string cvs pf
  ( to ) pf PhysicalPage 10 string cvs pf
  sName 0 1 getinterval (m) eq {
    (.\string\n Cross page links are not supported by the
    PDF Specification)pf
    (.\string\n This link is not constructed,
    please fix it.\string\n)pf
    (Break point is after syllable number )pf
    nSyllable cvi 20 string cvs pf (.\string\n)pf
    (Use the \string\\mlcrackat{)pf
    nSyllable cvi 20 string cvs pf
    (} option with this link.\string\n)pf
  }{
    (.\string\n Cross page annotations are not supported by the
    PDF Specification)pf
    (.\string\n This annotation is not constructed,
    please fix it.\string\n)pf
    (Break point is after syllable number )pf
    nSyllable cvi 20 string cvs pf (.\string\n)pf
    (Use the mlcrackat=)pf
    nSyllable cvi 20 string cvs pf
    ( option with this annotation.\string\n)pf
  } ifelse
  (!!------------------------------------------------------------------%
  \string\n)pf^^J%
} def^^J%
```

quadpointsfixup is the major procedure for combining all ‘rectangles’ that are on the same line.

```
/quadpointsfixup {^^J%
  /ary exch def^^J%
  /quadL exch def^^J%
  /sName exch def^^J%
\mldb0{(Processing )pf sName pf (: OK\string\n)pf^^J}%
%\mldb0{lnkCnt 20 string cvs pf (: OK\string\n) pf^^J}%
quadL 0 eq {
(Problems with this link, length=0,
  will skip the creation of this link)pf^^J%
}{
```

Begin by defining some variables needed for this operation.

```
/gOffset 0 def^^J%
```

gY holds the y-coordinate of the lower-left corner of the first entry of a quad. We use it and others like it to determine at which syllable the line is broken.

```
/gY ary\space 1 gOffset add get def^^J%
\mldb1{(gY is ) pf gY 20 string cvs print^^J}%
\mldb1{flush (\string\n) pf^^J}%
/gN 0 def^^J%
```

gMrk is an array that will load the offsets into mLinkFxup⟨*num*⟩. Each offset is the beginning of a line. We initially set the length of the array to 10, though this may not be correct. That is, we assume the hypertext will not exceed 10 lines in length.

```
/gMrk 10 array def^^J% limitation
```

The first entry is always 0, because marks the location of the quad corresponding to the beginning of the first syllable.

```
gMrk 0 0 put^^J%
```

gMrkL is the length of the array, we set it to 1, as we've already inserted the first entry into gMrkL.

```
/gMrkL 1 gOffset add def^^J%
\mldb2{(Begin first for\string\n) pf^^J}%
```

for loop

Begin a for loop. The purpose of this loop is to search through the quad points of mLinkFxup⟨*num*⟩ and find the offsets into the structure where the line breaks occur.

```
0 8 quadL 8 sub {^^J%
```

gIndx is the loop counter, which we use below. Since we are dealing with quads, we increment by 8 each time.

```
  /gIndx exch def^^J%
\mldb2{(Outside gt if with gIndx=) pf^^J}%
\mldb2{gIndx 20 string cvs pf^^J}%
\mldb2{(\string\n) pf^^J}%
```

getEntry is the y-coordinate of the lower-left corner of the quad we are currently examining. We will compare its value to that of \gY.

```
  /getEntry ary\space 1 gOffset add get def^^J%
\mldb2{(getEntry=) pf getEntry 20 string cvs pf^^J}%
\mldb2{(\string\n) pf^^J}%
```

if comparison

Here's where we compare gY with getEntry. If they differ by more than 2 points then the line has changed. (We do it this way since these are decimal numbers and they may have been rounded in unpredictable ways.

```
  gY getEntry sub abs 2 gt  {^^J%
\mldb2{(Inside gt if\string\n) pf^^J}%
```

The two entries differ, so a line break must have occurred. We inert value of gIndx into gMrk. gIndx is essentially the offset into mLinkFxup⟨*num*⟩ where the line break occurs.

```
    gMrk gMrkL gIndx put^^J%
```

Increment gMrkL accordingly.

```
    /gMrkL gMrkL 1 add def^^J%
```

Place the new y-value in gY before we look back and look for another line break.

```
    /gY  getEntry def^^J%
\mldb2{(Updating gY to )pf gY 20 string cvs pf^^J}%
\mldb2{(\string\n) pf^^J}%
\mldb2{(gMrkL=)pf gMrkL 20 string cvs pf^^J}%
\mldb2{(\string\n) pf^^J}%
```

end if End of the if

```
  } if^^J%
  /gOffset gOffset 8 add def^^J%
```

end for End of the for loop

```
} for^^J%
\mldb2{(end first for\string\n) pf^^J}%
```

We finished searching for line breaks and are now going to work on combining contiguous quads. Contiguous quads are the ones between the offsets recorded in the gMrk array. Now, if there are now line breaks, the length of gMrk is one

gAry is a temporary array that holds all the quads *corresponding to one line.* Each syllable generates an quad of length 8. Here, we assume any given line has at most \mlMaxNSylls syllables, (currently set to 30, but may be revised). The array gAry is declared inside the next loop, so it is redeclared at each iteration of the loop. If Distiller of ps2pdf fails, it may be due to \mlMaxNSylls being too small for some of your sentences; in this case, redefine \mlMaxNSylls to a larger value.

```
   /gAryL 8 \mlMaxNSylls\space mul def^^J% limitation
\mldb2{(gAryL=) pf gAryL 20 string cvs pf^^J}%
\mldb2{(\string\n)pf^^J}%
```

gFixup is an array that will hold the final combined quad points, this array is the one that will be referenced by the QuadPoints entry of the link annotation. It's length is set to 8 times the number of lines over which the hypertext is broken (gMrkL) One quad for each line, rather than many quads, one for each syllable.

```
  /gFixup 8 gMrkL mul array def^^J% links
  /aFixup 8 gMrkL mul array def^^J% text markup annotations
  /gOffset 0 def^^J%
\mldb2{(for loop: gMrkL=)pf gMrkL 20 string cvs pf^^J}%
\mldb2{(\string\n) pf^^J}%
```

begin for

The last loop. In this loop, for each line of hypertext, we build its contiguous quad and load it into gFixup.

```
0 1 gMrkL 1 sub {^^J%
\mldb2{(After gAry\string\n) pf^^J}%
\mldb2{(Top of for loop\string\n) pf^^J}%
    /gIndx exch def^^J%
\mldb2{(gIndx=)pf gIndx 20 string cvs pf^^J}%
\mldb2{(\string\n)pf^^J}%
\mldb2{(gMrk=)pf gMrk gIndx get 20 string cvs pf^^J}%
\mldb2{(\string\n) pf^^J}%
\mldb2{(mLinkFxup<num> length = )^^J}%
\mldb2{pf ary\space length 20 string cvs^^J}%
\mldb2{pf (\string\n) pf^^J}%
```

We need to determine if we are examining the quads for the last line, or not. The calculation of gCount is dependent on this.

```
gIndx 1 add gMrkL eq {^^J%
  /gCount ary\space length gMrk gIndx get sub def^^J%
}{^^J%
  /gCount gMrk gIndx 1 add get gMrk gIndx get sub def^^J%
} ifelse^^J%
```

Declare the gAry array

```
/gAry gAryL array def^^J%
\mldb2{(gCount=)pf gCount 20 string cvs pf^^J}%
\mldb2{(\string\n)pf^^J}%
```

We want to copy a slice of mLinkFxup⟨*num*⟩, we declare the next array of length gCount just computed.

```
/sliceOfLinkfxup gCount array def^^J%
```

Populate this array with a slice of mLinkFxup⟨*num*⟩ beginning at gMrk[gIndx] and including the subsequent gCount entries. array of length gCount just computed.

```
sliceOfLinkfxup 0 ary gMrk gIndx get^^J%
  gCount getinterval putinterval^^J%
    gAry 0 sliceOfLinkfxup putinterval^^J%
\mldb1{(Listing elements of gFixup\string\n) pf^^J}%
\mldb1{gAry 0 get 20 string cvs pf (\string\n)pf^^J}%
\mldb1{gAry 1 get 20 string cvs pf (\string\n)pf^^J}%
\mldb1{gAry gCount 1 sub 5 sub get
  20 string cvs pf (\string\n)pf^^J}%
\mldb1{gAry gCount 1 sub 4 sub get
  20 string cvs pf (\string\n)pf^^J}%
\mldb1{gAry gCount 1 sub 3 sub get
  20 string cvs pf (\string\n)pf^^J}%
\mldb1{gAry gCount 1 sub 2 sub get
  20 string cvs pf (\string\n)pf^^J}%
\mldb1{gAry 6 get 20 string cvs pf (\string\n)pf^^J}%
\mldb1{gAry 7 get 20 string cvs pf (\string\n)pf^^J}%
    gFixup gOffset [^^J%
      gAry 0 get ^^J%                         x1 ll 1
      gAry 1 get^^J%                          y1 ll 1
      gAry gCount 1 sub 5 sub get^^J%         x2 lr 2
      gAry gCount 1 sub 4 sub get^^J%         y2 lr 2
      gAry gCount 1 sub 3 sub get^^J%         x3 ur 3
      gAry gCount 1 sub 2 sub get^^J%         y3 ur 3
      gAry 6 get^^J%                          x4 ul 4
      gAry 7 get^^J%                          y4 ul 4
    ] putinterval^^J%
```

When forming quad points for text markup annotations, the PDF reference is not followed. We have to reorder the array gFixup to conform to how Acrobat/Reader expect it. Entries 0 and 4 are increased by the same amount that was subtracted out earlier in

```
    aFixup gOffset [^^J%
      gAry 6 get gOffset 0 eq {\ml@adj@x\space add}if^^J% x4 ul 4
      gAry 7 get^^J%                                    y4 ul 4
      gAry gCount 1 sub 3 sub get^^J%                   x3 ur 3
      gAry gCount 1 sub 2 sub get^^J%                   y3 ur 3
      gAry 0 get gOffset 0 eq {\ml@adj@x\space add}if^^J% x1 ll 1
      gAry 1 get 1 sub^^J%                               y1 ll 1
      gAry gCount 1 sub 5 sub get^^J%                    x2 lr 2
      gAry gCount 1 sub 4 sub get 1 sub^^J%              y2 lr 2
    ] putinterval^^J%
    /gOffset gOffset 8 add def^^J%
  } for^^J%
\mldb2{(End of second for\string\n) pf^^J}%
  } ifelse
} def^^J%
```

`smallquadpointsfixup` fixes up the bounding boxes. Raises their heights by `\ml@adj@y` (removed), and moves the left boundary `\ml@adj@x` to the left.

```
/smallquadpointsfixup {^^J%
/gIndx exch def^^J%
/ary exch def^^J%
/lnkCnt exch def^^J%
/quadL exch def^^J%
/gSFup 8 array def^^J%
  gSFup 0 ary 0 gIndx add get
  gIndx 0 eq {\ml@adj@x\space sub} if put^^J% x1
  gSFup 1 ary 1 gIndx add get put^^J%          y1
  gSFup 2 ary 2 gIndx add get put^^J%          x2
  gSFup 3 ary 3 gIndx add get put^^J%          y2
  gSFup 4 ary 4 gIndx add get put^^J%          x3
  gSFup 5 ary 5 gIndx add get put^^J%
  gSFup 6 ary 6 gIndx add get
  gIndx 0 eq {\ml@adj@x\space sub} if put^^J% x4
  gSFup 7 ary 7 gIndx add get put^^J%
  ary gIndx gSFup putinterval^^J%
} def^^J%
```

The `mlRectFix` procedure creates the minimum sized rectangle that encloses the text, and we use this as the dimensions of `/Rect`

```
/mlRectFix {^^J%
/ifRectFix exch def^^J%
ifRectFix {
  /nL gFixup length 8 sub def^^J% number of lines
  /xMin gFixup 0 get def^^J%
  0 8 nL {^^J%
  /Indx exch def^^J%
  gFixup Indx get xMin lt {/xMin gFixup Indx get def}if } for^^J%
  /xMin xMin 2 sub def^^J%
  /xMax gFixup 2 get def^^J%
  2 8 nL 2 add {^^J%
  /Indx exch def^^J%
  gFixup Indx get xMax gt {/xMax gFixup Indx get def}if } for^^J%
  /xMax xMax 2 add def^^J%
  /yMin gFixup 1 get def^^J%
  1 8 nL 1 add {^^J%
  /Indx exch def^^J%
  gFixup Indx get yMin lt {/yMin gFixup Indx get def}if } for^^J%
  /yMin yMin 4 sub def^^J%
  /yMax gFixup 5 get def^^J%
  5 8 nL 5 add{^^J%
  /Indx exch def^^J%
  gFixup Indx get yMax gt {/yMax gFixup Indx get def}if } for^^J%
  /yMax yMax 2 add def^^J%
  /mlRect {/Rect [^^J%
    xMin 72 sub PDFtoTeX^^J%
    PageHeight 72 sub yMax sub PDFtoTeX^^J%
    xMax 72 sub PDFtoTeX^^J%
    PageHeight yMin sub 72 sub PDFtoTeX ]^^J%
  }def^^J%
}{^^J%
  /mlRect{/Rect [ \par@@Rect ] }def^^J%
}ifelse^^J%
ifRectFix {
\mldb1{(/Rect [)pf^^J%
xMin 20 string cvs pf( )pf^^J%
yMax 20 string cvs pf( )pf^^J%
xMax 20 string cvs pf( )pf^^J%
yMin 20 string cvs pf(]\string\n)pf^^J}%
} if^^J%
} def
}}
```

\mlMarksOn \mlMarksOff Added tracking marks. Turn them on with `\mlMarksOn` and off again with `\mlMarksOff`.

```
\newif\ifmlmarks\mlmarksfalse
\def\mlMarksOn{\mlmarkstrue}
\def\mlMarksOff{\mlmarksfalse}
```

\ml@MrkLnk{⟨*num*⟩} The internal command `\ml@MrkLnk` typesets the tracking mark; it may be redefined. When the link is within a `tabular` environment (and perhaps others), in this case, `\baselineskip=0pt`. We raise instead the height of the capital letter 'T', plus a little.

```
\def\MrkLnkLtr{L}
\def\ml@MrkLnk#1{\ifmlmarks\bgroup\ifdim\baselineskip=0pt
  \setbox\z@\hbox{T}\gdef\ml@raiseamt{\ht\z@+.4pt}\else
  \gdef\ml@raiseamt{.6\baselineskip}\fi\smash{\rlap{\normalfont
    \normalcolor\bfseries
    \raisebox{\ml@raiseamt}{\tiny\strut{\MrkLnkLtr#1}}}}\egroup\fi}
\newif\iflinknotformed \linknotformedfalse
\newif\ifcr@ckit \cr@ckitfalse
\def\ml@underlinded{U}
```

\mlhypertext[⟨*opts*⟩]{⟨*text*⟩} This is a general purpose hypertext link. Not only is it a fine standalone linking command, but it also serves as a building block to some convenience commends that follow.

The commands takes two arguments, the first an optional one the second one requires.

⟨*opts*⟩ (Optional) A standard optional argument for eforms to change the appearance of the link and/or to include actions.

⟨*text*⟩ The text to be enclosed by the link.

The most recently, eforms defines `\mlhypertext` to a warning message. So if `\mlhypertext` is already defined, we `\renewcommand` else we `\newcommand`.

```
\@ifundefined{mlhypertext}{\newcommand}{\renewcommand}%
{\mlhypertext}[2][]{\hglue0pt\begingroup
  \global\ml@displaytrue
  \toks@=\expandafter{#2}%
  \edef\ml@HytextArg{{\the\toks@}}%
  \global\aeb@arrayIndx=\z@
  \def\ml@setlink##1{\setLinkPbox{%
      \QuadPoints{mLink##1}#1}}%
    \expandafter\processAppArgs\set@LinkPboxDefaults
    \presets{\ml@nocolor@defaults}\S{S}\W{0}#1\end\@nil
    \ifx\eq@S@value\ml@underlinded
      \let\itsunderline\ef@YES\else\let\itsunderline\ef@NO\fi
```

Now we test whether `\eq@mlcrackat` is empty or not. If not-empty we break this link to two parts.

```
    \ifx\eq@mlignore\ef@YES
      \global\advance\aeb@mLinkCnt\@ne\relax
      \def\ml@next{\mlhypertext@i{#1}}\else
    \ifx\eq@mlcrackat\@empty
```

A ‘normal’ link, continue

```
      \global\advance\aeb@mLinkCnt\@ne\relax
      \def\ml@next{\mlhypertext@i{#1}}%
    \else % \eq@mlcrackat not \@empty
```

Crack it up. We define `\ml@next` to call `\mlhypertext` consecutively; the first time we specify `\mlignore{0}` to indicate this link is the first part, and then `mlignore{1}` to indicate the second part. For the first part, we proceed as usual until we get the the syllable number specified by `\mlcrackAt`, then we continue with the next `\mlhypertext` command, we remove all content up to the syllable number `\mlcrackAt`, and typeset the rest; for example,

> This is a test sentence that we want to break across pages. (1)
> This is a test sentence that we want to break across pages. (2)

The first link produces the typeset material in (1), where grayed-out text are removed; while the second link produces line (2), again, with the grayed-out material removed.

```
      \def\ml@next{\global\ml@displaytrue\let\ml@space\space
      \mlhypertext[#1\mlignore{0}]{#2}\eq@mlcrackinsat
      \penalty-100 \cr@ckittrue
```

```
     \global\ml@displayfalse\let\ml@space\space
     \mlhypertext[#1\mlignore{1}]{#2}\aftergroup
     \normalcolor\endgroup}%
    \fi
  \fi
  \ml@next
}
\def\mlhypertext@i#1{%
  \@ifundefined{mLinkLngth\the\aeb@mLinkCnt}{\global
    \linknotformedtrue\def\ml@lngth{17}}
    {\edef\ml@lngth{\@nameuse{mLinkLngth\the\aeb@mLinkCnt}}}%
    \ml@start@link{\the\aeb@mLinkCnt}{\ml@lngth}% Step 1
    \def\mlh@preambleCmdInsert{%
      \ml@MrkLnk{\the\aeb@mLinkCnt}\ml@earlyExecProps{#1}}%
    \def\mlh@postambleCmd{\endgroup}%
```

Start soul on \mlhypertext (\aeb@mlh)

```
    \expandafter\aeb@mlh\ml@HytextArg
    \ml@finish@link{\the\aeb@mLinkCnt}{\ml@lngth}%
    \ml@setlink{\the\aeb@mLinkCnt}%
    \ifoldstylequads\else
      \iffixmlinks\literalps@out{restore}\fi\fi
      \@ifundefined{mLinkLngth\the\aeb@mLinkCnt}{%
      \immediate\write\@auxout{\string\mlcsarg
        \string\gdef{mLinkLngth\the\aeb@mLinkCnt}{17}}%
      }{\immediate\write\@auxout{\string\mlcsarg
        \string\gdef{mLinkLngth\the\aeb@mLinkCnt}%
        {\the\aeb@arrayIndx}}}\endgroup
}
\def\mlh@setQuadSyllable#1#2#3#4{%
% #1 = current array Index
% #2 = quad total
% #3 = link cnt
% #4 = content
  \setbox\aeb@bbox=\hbox{\ml@strut#4}%
  {%
    \setbox\@tempboxa\hbox{\ml@strut}%
    \ifx\itsunderline\ef@YES\@tempdima1bp\relax\else
      \@tempdima\dp\@tempboxa \ifdim\@tempdima>2bp
        \advance\@tempdima-2bp\fi
    \fi
    \ifx\isstrikeout\ef@YES\advance\@tempdima-2bp\fi
    \dp\aeb@bbox\@tempdima
    \@tempdima\ht\@tempboxa \advance\@tempdima\dp\aeb@bbox
    \advance\@tempdima1bp
    \ht\aeb@bbox\@tempdima\def\x{\the\count\z@}%
    \count\z@=\ht\aeb@bbox\xdef\aeb@bbox@ht{\x}%
    \count\z@=\wd\aeb@bbox\xdef\aeb@bbox@wd{\x}%
    \count\z@=\dp\aeb@bbox\xdef\aeb@bbox@dp{\x}%
  }%
  \ifoldstylequads
    \literalps@out{%
      bCreateLink {^^J%
        \mlDict\space/mLinkFxup#3\space known {^^J%
      \ps@mark{mLink#3}
      \the\aeb@arrayIndx\space [\setQuadBox]
          \space /PUTINTERVAL pdfmark}{^^J%
      xoMsgB {
      /xoMsgB false def
      (\ml@@nnotName\the\aeb@arrayIndx)
      #3\space
      xoMsg % dpsa08
      } if^^J% xoMsgB
      } ifelse } if
    }%
  \else
    \literalps@out{%
      bCreateLink  {^^J%
        \mlDict\space/mLinkFxup#3\space known {^^J%
        mLinkFxup#3\space
        #1\space[\setQuadBox] putinterval^^J%
```

Initiate fix up of little rectangles

```
        #2\space %
        #3\space
        mLinkFxup#3\space
        #1\space
        smallquadpointsfixup }{^^J%
        xoMsgB {
          /xoMsgB false def
          (\ml@@nnotName#3)
          #1\space
          xoMsg % dpsa08
        } if^^J% xoMsgB
      } ifelse } if
    }%
    \fi
    \global\advance\aeb@arrayIndx8\relax
}
```

The next four commands are used internally, though `\aebnameref`, `\labelRef` and `\atPage` are public, and can be used.

```
\def\aeb@exiii{\expandafter\expandafter\expandafter}
\def\aebnameref#1{\@ifundefined{r@#1}{??}
    {\aeb@exiii\@thirdoffive\csname r@#1\endcsname}}
\def\labelRef#1{\@ifundefined{r@#1}{Doc-Start}
   {\aeb@exiii\@fourthoffive\csname r@#1\endcsname}}
\def\atPage#1{\getrefbykeydefault{#1}{page}{??}}
```

`\mlhyperlink`
`\mlhyperref`
`\mlnameref`
`\mlNameref`

These four commands mimic the hyperref commands of the same root name. The commands `\mlhyperlink` and `\mlhyperref` take three parameters (the first one optional). The optional parameter modifies the appearance of the link, the second is the target/destination of the link, the third is the text the link is wrapped around. In the case of `\mlhyperlink` that target is a defined by `\hypertarget`; for `\mlhyperref` the target is a latex label.

The commands `\mlnameref` and `\mlNameref` take two parameters (the first is optional). As before, the first modifies the appearance of the link, the second is the target, a latex label.

Syntax: `\mlhyperlink[`⟨*opts*⟩`]{`⟨*named-dest*⟩`}{`⟨*text*⟩`}`

```
\newcommand\mlhyperlink[3][]{%
  \mlhypertext[#1\A{/S/GoTo/D (#2)}]{#3}}
```

Syntax: `\mlhyperref[`⟨*label*⟩`]{`⟨*opts*⟩`}{`⟨*text*⟩`}`

```
\newcommand\mlhyperref[3][]{%
  \mlhypertext[#1\A{/S/GoTo/D (\labelRef{#2})}]{#3}}
```

Syntax: `\mlnameref[`⟨*opts*⟩`]{`⟨*label*⟩`}`

```
\newcommand\mlnameref[2][]{\protected@edef\ml@temp{\aebnameref{#2}}%
  \def\ml@tempi{\mlhypertext[#1\A{/S/GoTo/D (\labelRef{#2})}]}%
  \expandafter\ml@tempi\expandafter{\ml@temp}}
```

We use a work around to a `\relax` problem encountered in the `\mlNameRef` command. LaTeX inserts a `relax` at the end of label titles, which stops soul. We insert `\let\SOUL@stop\ml@SOUL@stop`, this seems to work, no guarantees.

Syntax: `\mlNameref[`⟨*opts*⟩`]{`⟨*label*⟩`}`

```
\newcommand\mlNameref[2][]{\let\SOUL@stop\ml@SOUL@stop
  \protected@edef\ml@temp{`\aebnameref{#2}' on page~\atPage{#2}}%
  \def\ml@tempi{\mlhypertext[#1\A{/S/GoTo/D (\labelRef{#2})}]}%
  \expandafter\ml@tempi\expandafter{\ml@temp}}
```

The next three commands are modifications some low hyperref commands found in the `pdfmark.def` file. Depending on the parsing, `\href` calls one of these three; we intercept them, and insert our own command `\mlhypertext` so the link string gets wrapped around if needed. These are in preparation for the definition of `\mlhref`.

```
\def\ml@hyper@linkurl#1#2{\hyper@chars
  \let\ef@thislinkcolor\@urlcolor
  \let\CurrentBorderColor\@urlbordercolor
  \mlhypertext[\presets{\mlhref@args}\A{/S/URI/URI(#2)}]{#1}%
  \endgroup
}
\def\ml@hyper@linkfile#1#2#3{%
  \let\ef@thislinkcolor\@filecolor
  \let\CurrentBorderColor\@filebordercolor
  \ifx\@pdfstartview\@empty
    \def\theView{[0 /Fit]}\else
    \def\theView{[0 \@pdfstartview]}\fi
  \@ifundefined{ifHy@pdfnewwindow}
  {\ifHy@newwindow}{\ifHy@pdfnewwindow}%
    \def\isWindow{/NewWindow true}\else
    \let\isWindow\@empty\fi
  \mlhypertext[\presets{\mlhref@args}\A{/S/GoToR \isWindow
    /F (#2) /D \ifx\\#3\\\theView\else(#3)\fi}]{#1}%
  \endgroup
}
\def\ml@hyper@launch run:#1\\#2#3{%
  \let\ef@thislinkcolor\@filecolor
  \let\CurrentBorderColor\@runbordercolor
  \@ifundefined{ifHy@pdfnewwindow}
    {\ifHy@newwindow}{\ifHy@pdfnewwindow}%
  \def\isWindow{/NewWindow true}\else
    \let\isWindow\@empty\fi
  \mlhypertext[\presets{\mlhref@args}\A{/S/Launch\isWindow
     /F (#1) \ifx\\#3\\\else /Win << /P (#3) /F (#1) >> \fi}]{#2}%
  \endgroup
}
```

Below is the code for `\mlhref`. We first let the old commands found in `pdfmark.def` equal to the new versions, then we call `\href` to do all the parsing. Things eventually comes back to the above three commands.

```
\let\aeb@saved@href\href
```

`\mlhref` This command is similar to `\href`. This command also takes three arguments, one optional. The first is usual optional argument that allow one to modify the appearance of the link, the second one is the URL that we are linking to, the third is the text that we are wrapping this link around.

Syntax: `\mlhref[`⟨*opts*⟩`]{`⟨*url*⟩`}{`⟨*text*⟩`}`

```
\newcommand{\mlhref}[1][]{%
  \begingroup
    \def\mlhref@args{#1}%
    \let\hyper@linkurl\ml@hyper@linkurl
    \let\hyper@linkfile\ml@hyper@linkfile
    \let\@hyper@launch\ml@hyper@launch
    \aeb@saved@href
}
```

`\mlurl` The multi-line version of Donald Arseneau's url package. There are problems with this one, will continue to work on it.

The problem is not as "easy" as the previous cases. Arseneau places the URL in math mode and it does not reconstruct (`soul` terminology) as it should. Our solution is to hijack three commands of `soul`, these are `\SOUL@doword`, `\SOUL@analyze`, and `\SOUL@dosyllable`, and modify them to do the work on an URL.

```
\newbox\ml@urlbuildi
```

```
\newbox\ml@urlbuildii
```

We modify `\SOUL@doword` and name it `\ml@SOUL@doword`.

```
\def\ml@SOUL@doword{%
  \global\setbox\ml@urlbuildi\hbox{}%
  \global\setbox\ml@urlbuildii\hbox{}%
  \edef\x{\the\SOUL@word}%
  \ifx\x\empty
  \else
    \SOUL@buffer={}%
    \setbox\z@\vbox{%
      \SOUL@tt
      \hyphenchar\font`\-
      \hfuzz\maxdimen
      \hbadness\@M
      \pretolerance\m@ne
      \tolerance\@M
      \leftskip\z@
      \rightskip\z@
      \hsize1sp
      \everypar{}%
      \parfillskip\z@\@plus1fil
      \hyphenpenalty-\@M
      \noindent
      \hskip\z@
      \relax
      \the\SOUL@word}%
```

We don’t do the reconstruction, so no need for the message.

```
    \let\SOUL@errmsg\relax
%   \let\SOUL@errmsg\SOUL@error
    \let\-\relax
    \count@\m@ne
```

Here is the first major change, rather than splitting off to `\SOUL@analyze`, we go to our modified version, `\ml@SOUL@analyze`.

```
    \ml@SOUL@analyze
    \SOUL@word={}%
  \fi
}
```

We modify `\SOUL@analyze` and name it `\ml@SOUL@analyze`.

```
\def\ml@SOUL@analyze{{%
  \setbox\z@\vbox{%
    \unvcopy\z@
    \unskip
    \unpenalty
    \global\setbox\@ne=\lastbox}%
  \ifvoid\@ne
  \else
```

We encapsulate our changes at this point in `\ml@interface@analyze`

```
  \ml@interface@analyze
  \SOUL@syllgoal=\wd\@ne
  \advance\count@\@ne
```

We get the tokens recursively, but we jump back to `\ml@SOUL@analyze` not `\SOUL@analyze`.

```
  \ml@SOUL@analyze
  \SOUL@syllwidth\z@
  \SOUL@syllable={}%
  \ifnum\count@>\z@
   \advance\SOUL@syllgoal-\SOUL@ttwidth
```

At this point, we jump to `\ml@SOUL@dosyllable` rather than `\SOUL@dosyllable`.

```
   \ml@SOUL@dosyllable
   \SOUL@getkern{\the\SOUL@lasttoken}{\SOUL@hyphkern}%
     {\SOUL@sethyphenchar}%
   \SOUL@everyhyphen
  \else
```

Use `\ml@SOUL@dosyllable` not `\SOUL@dosyllable`.

```
    \ml@SOUL@dosyllable
  \fi
 \fi
}}
```

```
\newif\ifml@display \ml@displaytrue
```

We modify `\SOUL@dosyllable` and name it `\ml@SOUL@dosyllable`.

```
\def\ml@SOUL@dosyllable{%
 \SOUL@gettoken
 \SOUL@eventuallyexhyphen{\the\SOUL@token}%
 \edef\x{\the\SOUL@token}%
 \ifx\x\SOUL@hyphenhintM
  \let\SOUL@n\ml@SOUL@dosyllable
 \else\ifx\x\SOUL@lowerthanM
  \SOUL@gettoken
  \SOUL@getkern{\the\SOUL@lasttoken}{\SOUL@charkern}
    {\the\SOUL@token}%
  \SOUL@everylowerthan
  \SOUL@puttoken
  \let\SOUL@n\ml@SOUL@dosyllable
 \else\ifdim\SOUL@syllwidth=\SOUL@syllgoal
  \SOUL@everysyllable
  \SOUL@puttoken
  \let\SOUL@n\relax
 \else
  \ifx\x\SOUL@stopM
   \SOUL@errmsg
   \global\let\SOUL@errmsg\relax
   \let\SOUL@n\relax
  \else
   \setbox\tw@\hbox{\SOUL@tt\the\SOUL@token}%
```

```
      \advance\SOUL@syllwidth\wd\tw@
      \global\SOUL@lasttoken=\SOUL@token
      \SOUL@gettoken
      \SOUL@getkern{\the\SOUL@lasttoken}{\SOUL@charkern}
          {\the\SOUL@token}%
      \SOUL@puttoken
      \global\SOUL@token=\SOUL@lasttoken
      \SOUL@everytoken
      \edef\x{\SOUL@syllable={\the\SOUL@syllable\the\SOUL@token}}\x
```

Here is the only change, we direct flow back to `\ml@SOUL@dosyllable`

```
      \let\SOUL@n\ml@SOUL@dosyllable
  \fi\fi\fi\fi
  \SOUL@n
}
```

`\ml@interface@analyze` We put most of our changes to `\ml@SOUL@analyze` in `\ml@interface@analyze`

```
\def\ml@interface@analyze{%
  \global\advance\syllableCnt\@ne % dpsa11
  \setbox\@ne\hbox{\unhbox\@ne}%
```

If we say `\mlurl{http://www.math.uakron.edu/~dpstory}`, then the `\box\z@` above contains the following tokens, listed at their breakpoints:

```
http:
\\
www.
math.
uakron.
edu/
~dpstory
```

The idea is to get each of these using `\global\setbox\@ne=\lastbox` (bottom to top) and to build the URL with the quad points calculated. Each new token is added in front of the URL as we build it. Results are held in `\ml@urlbuild`. We insert `\penalty0` to promote a break point between components, as each component is enclosed in an `\hbox` now.

We'll try cracking the url here, if requested.

```
  \ifx\eq@mlcrackat\@empty
```

Ordinary link, we don't crack it apart, continue with the old code.

```
    \ml@bld@quadchunks{\the\aeb@mLinkCnt}
      {\ml@qlngthchunki}{\ml@urlbuildi}%
  \else
    \ifnum\syllableCnt=\revCrackAt\relax
```

Everything is in reverse order, this is the last syllable of the first chunk

```
      \aeb@arrayIndx=\z@
```

start first link

```
      \ml@start@link{\the\aeb@mLinkCnt}{\ml@qlngthchunki}%
      \ml@bld@quadchunks{\the\aeb@mLinkCnt}
        {\ml@qlngthchunki}{\ml@urlbuildi}%
    \else
```

Continue with first link

```
      \ifnum\syllableCnt>\revCrackAt\relax
        \ml@bld@quadchunks{\the\aeb@mLinkCnt}
          {\ml@qlngthchunki}{\ml@urlbuildi}%
      \else
        \ifnum\syllableCnt=\@ne
```

start second link

```
          \aeb@arrayIndx=\z@
```

Everything is in reverse order, this is the last syllable of the second chunk

```
          \ml@start@link{\aeb@mLinkCnt@}{\ml@qlngthchunkii}%
          \ml@bld@quadchunks{\aeb@mLinkCnt@}
            {\ml@qlngthchunkii}{\ml@urlbuildii}%
        \else
```

continue with second link

```
          \ml@bld@quadchunks{\aeb@mLinkCnt@}
            {\ml@qlngthchunkii}{\ml@urlbuildii}%
        \fi
      \fi
    \fi
  \fi
}
\def\ml@bld@quadchunks#1#2#3{{%
% #1 = link cnt
% #2 = chunk size
% #3 = \ml@urlbuild to i or ii
  \@ifundefined{mLinkLngth\the\aeb@mLinkCnt}
    {\edef\@indx{\the\aeb@arrayIndx}}
    {\@tempcnta#2\relax
     \advance\@tempcnta-8\relax
     \advance\@tempcnta-\aeb@arrayIndx
     \def\@indx{\the\@tempcnta}%
  }%
  \@tempcntb=\@indx \divide\@tempcntb by 8
  \advance\@tempcntb by 1
\ifx#3\ml@urlbuildii \advance\@tempcntb by \eq@mlcrackat\relax\fi
  \global\setbox#3=\hbox{%
    \mlh@setQuadSyllable{\@indx}{#2}{#1}{\unhcopy\@ne}%
    \hbox{\unhcopy\@ne}\relax
    \ml@typeset@@syl{\@tempcntb}\penalty0\unhcopy#3}%hbox
}}
```

\mlurl After the above preliminaries, we finally define \mlurl.

Syntax: \mlurl[⟨*opts*⟩]{⟨*url*⟩}

```
\newcommand{\mlurl}{\begingroup\@makeother\~\relax% \def~{\string~}%
    \ef@sanitize@toks\mlurl@}
```

After sanitizing, we save the URL (#2) an a macro \ml@url using the \urldef command, defined in the url package.

```
\newcommand{\mlurl@}[2][]{\@ifundefined{ef@thislinkcolor}
  {\let\ef@thislinkcolor\normalcolor}{}\expandafter
  \def\expandafter\ef@thislinkcolor@SAVE
    \expandafter{\ef@thislinkcolor}%
    \def\ml@setlink##1{\setLinkPbox{\A{\URI{#2}}%
    \QuadPoints{mLink##1}#1}}%
```

We get the link options early to determine if this link is to be underlined.

```
    \expandafter\processAppArgs\set@LinkPboxDefaults
    \presets{\ml@nocolor@defaults}\S{S}\W{0}#1\end\@nil
    \ifx\eq@S@value\ml@underlinded
      \let\itsunderline\ef@YES\else\let\itsunderline\ef@NO\fi
```

\ml@url is the specified url to create a link around

```
  \global\aeb@arrayIndx=0\relax
  \global\syllableCnt=0\relax
  \global\advance\aeb@mLinkCnt\@ne\relax
  \@ifundefined{mLinkLngth\the\aeb@mLinkCnt}{%
    \global\linknotformedtrue\def\ml@lngth{17}%
    \def\ml@qlngthchunki{17}}
    {\@tempcnta\@nameuse{mLinkLngth\the\aeb@mLinkCnt}\relax
    \edef\ml@lngth{\the\@tempcnta}%\multiply\@tempcnta8\relax
    \edef\ml@qlngthchunki{\the\@tempcnta}}%
```

We make some calculations in preparation to a link that will be cracked apart. \mlM@XCNT is the number of syllables of the un-cracked URL.

```
  \@ifundefined{mLinkSyCnt\the\aeb@mLinkCnt}
    {\def\mlM@XCNT{0}\def\ml@lngthchunki{0}\def\ml@lngthchunkii{0}%
    \def\ml@qlngthchunki{0}\def\ml@qlngthchunkii{0}\def\revCrackAt{0}}
    {\edef\mlM@XCNT{\@nameuse{mLinkSyCnt\the\aeb@mLinkCnt}}%
```

When \mlM@XCNT is known, and \eq@mlcrackat is known, we can calculate the number of syllables of each chunk of the URL, for the first chunk, it is \eq@mlcrackat, for the second it is \mlM@XCNT − \eq@mlcrackat.

```
    \ifx\eq@mlcrackat\@empty\else
      \@tempcnta\mlM@XCNT\relax\advance\@tempcnta-\eq@mlcrackat\relax
      \edef\ml@lngthchunki{\eq@mlcrackat}%
      \edef\ml@lngthchunkii{\the\@tempcnta}%
      \@tempcnta\ml@lngthchunki\relax\multiply\@tempcnta8\relax
      \edef\ml@qlngthchunki{\the\@tempcnta}%
      \@tempcnta\ml@lngthchunkii\relax\multiply\@tempcnta8\relax
      \edef\ml@qlngthchunkii{\the\@tempcnta}%
```

We take these chunks off in reverse order so we measure each from the end of the url. This calculation can be move to earlier code where it is not executed for each syllable (chunk). `\revCrackAt` is the value of `\eq@mlcrackat` as measure from the end of the url.

```
    \@tempcnta\mlM@XCNT\relax
    \advance\@tempcnta-\eq@mlcrackat\relax
    \advance\@tempcnta\@ne
    \edef\revCrackAt{\the\@tempcnta}%
  \fi}%
  \@tempcnta\aeb@mLinkCnt\advance\@tempcnta\@ne
    \edef\aeb@mLinkCnt@{\the\@tempcnta}%
\urldef\ml@url\nolinkurl{#2}%
\def\SOUL@mlhpreamble{%\begingroup
  \mlh@preambleCmdInsert\ef@colorthislink}\hyper@chars
  \let\ef@thislinkcolor\@urlcolor
  \let\CurrentBorderColor\@urlbordercolor
```

Within this group, we direct the soul package to our customized versions of the commands.

```
  \let\ml@SOUL@doword@SAVE\SOUL@doword
  \let\SOUL@doword\ml@SOUL@doword
```

The next several lines are taken from the definition of `\mlhypertext`, the basic command for construction many of the '`\ml`' commands of this package.

```
  \ifx\eq@mlcrackat\@empty
    \ml@start@link{\the\aeb@mLinkCnt}{\ml@lngth}\fi
  \def\mlh@preambleCmdInsert{\ml@MrkLnk{\the\aeb@mLinkCnt}%
    \ml@earlyExecProps{#1}}%
  \def\mlh@postambleCmd{%
      \expandafter
```

The coloring of the hypertext does not work unless we make the definition global, so we do so and hope this does not mess other things up. Save the incoming link color so we can globally change it. After the link is formed, change it back again.

```
      \def\expandafter\ef@thislinkcolor
           \expandafter{\ef@thislinkcolor}}%
```

Finally, we call `\aeb@mlh` which starts soul with `\SOUL@`. This does this analysis, the custom command build the url in `\ml@urlbuild`, which we then unbox.

```
  \aeb@mlh\ml@url\ef@colorthislink\unhcopy\ml@urlbuildi
  \expandafter\gdef\expandafter\ef@thislinkcolor
  \expandafter{\ef@thislinkcolor@SAVE}%
  \immediate\write\@auxout{\string\mlcsarg\string
    \gdef{mLinkSyCnt\the\aeb@mLinkCnt}{\the\syllableCnt}}% dpsa11
  \immediate\write\@auxout{\string\mlcsarg
    \string\gdef{mLinkLngth\the\aeb@mLinkCnt}{\the\aeb@arrayIndx}}%
  \ifx\eq@mlcrackat\@empty
    \ml@finish@link{\the\aeb@mLinkCnt}{\ml@lngth}%
    \ml@setlink{\the\aeb@mLinkCnt}%
    \iffixmlinks\literalps@out{restore}\fi
    \else
      \ml@finish@link{\the\aeb@mLinkCnt}{\ml@qlngthchunki}%
      \ml@setlink{\the\aeb@mLinkCnt}%
      \iffixmlinks\literalps@out{restore}\fi
      \eq@mlcrackinsat\penalty-100
      \ml@start@link{\aeb@mLinkCnt@}{\ml@qlngthchunkii}%
      \penalty0\@ifundefined{mLinkLngth\the\aeb@mLinkCnt}{}
      {\ml@MrkLnk{\aeb@mLinkCnt@}}\unhcopy\ml@urlbuildii
      \ml@finish@link{\aeb@mLinkCnt@}{\ml@qlngthchunkii}%
      \ml@setlink{\aeb@mLinkCnt@}\iffixmlinks
        \literalps@out{restore}\fi
      \global\advance\aeb@mLinkCnt\@ne
    \fi\aftergroup\normalcolor\endgroup
}
\def\ml@start@link#1#2{% Step 1
% #1=link number
% #2 = final quad length
  \literalps@out{%
    /xoMsgB true def^^J%
    \ps@mark/_objdef {mLink#1}%
      /type /array /OBJ pdfmark^^J%
    \ifoldstylequads
    /mLinkFxup#1\space0 array def
    \else
      /bCreateLink mlIsBld #2\space eq not def^^J%
      bCreateLink{ /mLinkFxup#1\space
        #2\space array def }if
    \fi
  }%
}
```

Make gFixup into a macro, so other packages (aeb_mlink can intercept and insert its own procedure here.

```
\def\FixupProc{gFixup}
\def\ml@finish@link#1#2{% Step 4 and 5
% #1=link number
% #2 = final quad length
  \ifoldstylequads\else
    \ifnum\aeb@arrayIndx=0\relax
      \PackageWarning{aeb_mlink}{%
        Problem with mLink\the\aeb@mLinkCnt, Will skip the \MessageBreak
        creation of this link}\fi
  \literalps@out{%
```

If \iffixmlinks is true, we fix the links, otherwise, we no not.

```
  \iffixmlinks
    \ifnum\aeb@arrayIndx>0
      save^^J%
      bCreateLink {^^J%
      \mlDict\space/mLinkFxup#1\space known {^^J%
      (\ml@@nnotName#1)
      #2\space
      mLinkFxup#1\space
      quadpointsfixup^^J%
      \ps@mark{mLink#1} 0 \FixupProc\space
      /PUTINTERVAL pdfmark^^J%
      }if }if
    \fi
  \else
```

(2018/04/20) Added `bCreateLink`, etc., to protect against distiller/ps2pdf from crashing when `\mlfix{n}`.

```
    bCreateLink {^^J%
    \mlDict\space/mLinkFxup#1\space known {^^J%
    \ps@mark{mLink#1} 0 mLinkFxup#1
      /PUTINTERVAL pdfmark^^J%
     }if }if
  \fi}%
\fi}
```

6 Macros used by the SOUL Interface

```
\ifHy@colorlinks
  \def\ef@colorthislink{\color{\ef@thislinkcolor}}
\else
  \let\ef@colorthislink\relax
\fi
```

I've inserted `\let\protect\@empty` to make `mlnameref` and `mlNameref` work.

```
\def\ml@SOUL@stop{\relax}
\def\SOUL@mlhpreamble{\begingroup
```

(2011/12/27) Originally, I had `\let\protect\@empty` here, but removing this seems to do no harm, so, we'll go for it.

```
  \mlh@preambleCmdInsert\ef@colorthislink}
\def\SOUL@mlheverysyllable{% dpsaug16
  \global\advance\syllableCnt\@ne
  \ifx\eq@mlcrackat\@empty
    \expandafter\SOUL@mlheverysyllable@i
  \else
    \expandafter\SOUL@mleverysyllable@ii
  \fi
}
\let\eq@@mlhyph\@empty
\def\ml@typset@syl#1{\raisebox{\ml@raiseamt}
  {\smash{\normalfont\normalcolor\tiny\strut\llap{\the#1}}}}
\def\turnSyllbCntOn{\mlMarksOn\let\ml@typeset@@syl\ml@typset@syl}
\def\turnSyllbCntOff{\let\ml@typeset@@syl\@gobble}
\turnSyllbCntOff
\def\SOUL@mlheverysyllable@i{%
  \mlh@setQuadSyllable{\the\aeb@arrayIndx}{\ml@lngth}%
```

```
    {\the\aeb@mLinkCnt}{\the\SOUL@syllable\eq@@mlhyph}%
  \the\SOUL@syllable %\SOUL@setkern\SOUL@charkern
  \ml@typeset@@syl{\syllableCnt}\eq@@mlhyph
}%
\def\SOUL@mleverysyllable@ii{%
  \ifnum\eq@mlchunk=0\relax
    \ifnum\syllableCnt>\eq@mlcrackat\relax
      \global\ml@displayfalse
    \else
      \global\ml@displaytrue
      \ifnum\syllableCnt=\eq@mlcrackat\relax
        \let\eq@@mlhyph\eq@mlhyph
        \global\let\ml@space\relax
      \else
        \let\eq@@mlhyph\@empty
        \global\let\ml@space\space
      \fi
      \SOUL@mlheverysyllable@i
    \fi
  \else
    \ifnum\syllableCnt>\eq@mlcrackat\relax
      \global\ml@displaytrue
      \SOUL@mlheverysyllable@i
    \else
      \global\ml@displayfalse
    \fi
  \fi
  \ml@dynamicsetup
}
%\def\SOUL@mlheveryspace#1{#1\space\hskip\spaceskip}
\let\ml@space\space
\def\SOUL@mlheveryspace#1{%
  #1\ml@space\global\let\ml@space\space
}
\def\SOUL@mlheveryhyphen{%
  \discretionary{%
   \unkern
    \SOUL@setkern\SOUL@hyphkern
    \SOUL@sethyphenchar
  }{}{}%
}
\def\SOUL@mlheveryexhyphen#1{\global\advance\syllableCnt\@ne
  \mlh@setQuadSyllable{\the\aeb@arrayIndx}{\ml@lngth}%
    {\the\aeb@mLinkCnt}{\SOUL@setkern\SOUL@hyphkern#1}%
  \SOUL@setkern\SOUL@hyphkern\hbox{#1}%
  \discretionary{}{}{%
      \SOUL@setkern\SOUL@charkern
  }%
}
\def\mlh@postambleCmd{\relax}
```

```
\def\ml@dynamicsetup{%dpsaug16
  \ifml@display
    \global\let\SOUL@everyspace\SOUL@mlheveryspace
    \global\let\SOUL@everyexhyphen\SOUL@mlheveryexhyphen
  \else
    \gdef\SOUL@everyspace##1{}%
    \gdef\SOUL@everyexhyphen##1{}%
  \fi
}
\def\SOUL@mlhpostamble{\mlh@postambleCmd}
\def\SOUL@mlhsetup{\SOUL@setup
  \let\SOUL@preamble\SOUL@mlhpreamble
  \let\SOUL@everysyllable\SOUL@mlheverysyllable
  \ml@dynamicsetup
%  \let\SOUL@everyspace\SOUL@mlheveryspace
  \let\SOUL@everyhyphen\SOUL@mlheveryhyphen
%  \let\SOUL@everyexhyphen\SOUL@mlheveryexhyphen
  \def\SOUL@postamble{\SOUL@mlhpostamble}%
}
\DeclareRobustCommand*\aeb@mlh{\syllableCnt=0
  \SOUL@mlhsetup\SOUL@}
% End of Package
⟨/package⟩
```

7 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

8 Change History

v2.0 (2016/02/16)
`\mlurl`: Added support for the `\url` command. . 18

v2.0.1 (2017/09/19)
`\mlurl`: Save the incoming link color 24

v2.1.10 (2018/03/25)
General: Set `/Rect` to minimal rectangle containing text 12

v2.1.17 (2018/04/19)
`\mlhypertext`: added conditional command definition for `\mlhypertext` 14

v2.1.18 (2018/04/20)
`\mlurl`: Added `bCreateLink`, etc., to protect against distiller/ps2pdf from crashing 26

v2.1.2 (2018/03/09)
General: Added PS proc `smallquadpointsfixup` 12

v2.1.3 (2018/03/10)
General: Reinstate old style link 7

v2.1.8 (2018/03/19)
General: Added tracking marks 13

v2.1.9 (2018/03/22)
General: Changed the definition of `\ml@MrkLnk` . 13

v2.3 (2018/04/26)
General: Added aeb-mlink as an alternate name for this package 2

v2.3.1 (2018/07/18)
`\mlhypertext`: added `\hglueOpt` to add some hard glue just before the beginning of `\mlhypertext` 14

v2.3.2 (2018/08/09)
General: `\mlMaxNSylls` now sets the array size of gAryL 10
Form the array `aFixup` for aeb_mlink 11
`\mlurl`: We make `gFixup` into a macro name `\FixupProc` 25

v2.3.5 (2020/01/06)
General: Added `urlOpts` 3
Conform to the new `web.cfg` format 3
Now require url package and pass options to url through `urlOpts` 3