ConTEXt

TEXUtil

category: pre- and postprocessing utilities

version: 1997.12.08

date: 1997 december 25

author: Hans Hagen

This is TeXUTIL, a utility program (script) to be used alongside the ConTeXt macro package. This PERL script is derived from the MODULA version and uses slightly better algoritms for sanitizing TeX specific (sub)strings. This implementation is therefore not entirely compatible with the original TeXUTIL, although most users will probably never notice. Now how was this program called?

```
1   $Program = "TeXUtil 6.40 - ConTeXt / PRAGMA 1992-1997" ;
```

By the way, this is my first PERL script, which means that it will be improved as soon as I find new and/or more suitable solutions in the PERL manuals. As can be seen in the definition of `$program`, this program is part of the ConTeXt suite, and therefore can communicate with the users in english as well as some other languages. One can set his favourite language by saying something like:

```
perl texutil.pl --interface=de --figures *.eps *.tif
```

Of course one can also say `--interface=nl`, which happens to be my native language.

I won't go into too much detail on the algoritms used. The next few pages show the functionality as reported by the helpinformation and controled by command line arguments and can serve as additional documentation.

TeXUTIL can handle different tasks; which one is active depends on the command line arguments. Most task are handled by the procedures below. The one exception is the handling of TIFF files when collecting illustration files. When needed, TeXUTIL calls for TIFFINFO or TIFFTAGS, but more alternatives can be added by extending `@TiffPrograms`.

```
2   @TiffPrograms = ("tiffinfo", "tifftags") ;
```

Back to the command line arguments. These are handled by a PERL system module. This means that, at least for the moment, there is no external control as provided by the PRAGMA environment system.

```
3   use Getopt::Long ;
```

We don't want error messages and accept partial switches, which saves users some typing.

```
4   $Getopt::Long::passthrough = 1 ; # no error message
    $Getopt::Long::autoabbrev  = 1 ; # partial switch accepted
```

We also predefine the interface language and set a boolean that keeps track of unknown options.[1]

```
5   $Interface      = "en" ;
    $UnknownOptions = 0 ;
```

Here come the options:

```
6   &GetOptions
      ("references"     => \$ProcessReferences,
          "ij"                => \$ProcessIJ,
          "high"              => \$ProcessHigh,
          "quotes"            => \$ProcessQuotes,
       "documents"      => \$ProcessDocuments,
          "type=s"      => \$ProcessType,
       "sources"        => \$ProcessSources,
       "setups"         => \$ProcessSetups,
       "templates"      => \$ProcessTemplates,
       "infos"          => \$ProcessInfos,
       "figures"        => \$ProcessFigures,
```

[1] This feature is still to be implemented.

```
       "tiff"              =>\$ProcessTiff,
    "logfile"        => \$ProcessLogFile,
       "box"               =>\$ProcessBox,
       "hbox"              =>\$ProcessHBox,
       "vbox"              =>\$ProcessVBox,
       "criterium=f"       =>\$ProcessCriterium,
       "unknown"           =>\$ProcessUnknown,
    "help"           => \$ProcessHelp,
    "interface=s"    => \$Interface) ;
```

By default wildcards are expanded into a list. The subroutine below is therefore only needed when no file or pattern is given.

```
7  $InputFile = "@ARGV" ;
```

```
8  sub InputFiles
     { my ($UserSuppliedPath) = @_ ;
       my (@UserSuppliedFiles) ;
       @UserSuppliedFiles = sort <eval $UserSuppliedPath> ;
       foreach $FileName (@UserSuppliedFiles)
         { $FileName = lc $FileName }
       return @UserSuppliedFiles }
```

In order to support multiple interfaces, we save the messages in a hash table. As a bonus we can get a quick overview of the messages we deal with.

```
9  sub Report
     { foreach $_ (@_)
         { if (! defined $MS{$_})
             { print $_ }
           else
             { print $MS{$_} }
           print " " }
       print "\n" }
```

The messages are saved in a hash table and are called by name. This contents of this table depends on the interface language in use.

```
10  if ($Interface eq "nl")

11    { # begin of dutch section

12      $MS{"ProcessingReferences"}    = "commando's, lijsten en indexen verwerken" ;
        $MS{"GeneratingDocumentation"} = "ConTeXt documentatie file voorbereiden" ;
        $MS{"GeneratingSources"}       = "ConTeXt broncode file genereren" ;
        $MS{"FilteringDefinitions"}    = "ConTeXt definities filteren" ;
        $MS{"CopyingTemplates"}        = "TeXEdit toets templates copieren" ;
        $MS{"CopyingInformation"}      = "TeXEdit help informatie copieren" ;
        $MS{"GeneratingFigures"}       = "figuur file genereren" ;
        $MS{"FilteringLogFile"}        = "log file filteren (poor mans version)" ;

13      $MS{"SortingIJ"}               = "IJ sorteren onder Y" ;
        $MS{"ConvertingHigh"}          = "hoge ASCII waarden converteren" ;
        $MS{"ProcessingQuotes"}        = "characters met accenten afhandelen" ;
        $MS{"ForcingFileType"}         = "filetype instellen" ;
        $MS{"UsingTiff"}               = "TIF files afhandelen" ;
        $MS{"FilteringBoxes"}          = "overfull boxes filteren" ;
        $MS{"ApplyingCriterium"}       = "criterium toepassen" ;
        $MS{"FilteringUnknown"}        = "onbekende ... filteren" ;
```

```
      $MS{"NoInputFile"}             = "geen invoer file opgegeven" ;
      $MS{"EmptyInputFile"}          = "lege invoer file" ;
      $MS{"NotYetImplemented"}       = "nog niet beschikbaar" ;

      $MS{"Action"}                  = "                 actie :" ;
      $MS{"Option"}                  = "                 optie :" ;
      $MS{"Error"}                   = "                  fout :" ;
      $MS{"Remark"}                  = "             opmerking :" ;
      $MS{"SystemCall"}              = "        systeemaanroep :" ;
      $MS{"BadSystemCall"}           = "  foute systeemaanroep :" ;
      $MS{"MissingSubroutine"}       = "  onbekende subroutine :" ;

      $MS{"EmbeddedFiles"}           = "       gebruikte files :" ;
      $MS{"BeginEndError"}           = "           b/e fout in :" ;
      $MS{"SynonymEntries"}          = "     aantal synoniemen :" ;
      $MS{"SynonymErrors"}           = "                fouten :" ;
      $MS{"RegisterEntries"}         = "       aantal ingangen :" ;
      $MS{"RegisterErrors"}          = "                fouten :" ;
      $MS{"PassedCommands"}          = "     aantal commando's :" ;

      $MS{"NOfDocuments"}            = "  documentatie blokken :" ;
      $MS{"NOfDefinitions"}          = "     definitie blokken :" ;
      $MS{"NOfSkips"}                = "  overgeslagen blokken :" ;
      $MS{"NOfSetups"}               = "    gecopieerde setups :" ;
      $MS{"NOfTemplates"}            = " gecopieerde templates :" ;
      $MS{"NOfInfos"}                = " gecopieerde helpinfos :" ;
      $MS{"NOfFigures"}              = "     verwerkte figuren :" ;
      $MS{"NOfBoxes"}                = "        te volle boxen :" ;
      $MS{"NOfUnknown"}              = "         onbekende ... :" ;

      $MS{"InputFile"}               = "           invoer file :" ;
      $MS{"OutputFile"}              = "          outvoer file :" ;
      $MS{"FileType"}                = "             type file :" ;
      $MS{"EpsFile"}                 = "              eps file :" ;
      $MS{"TifFile"}                 = "              tif file :" ;

      $MS{"Overfull"}                = "te vol" ;
      $MS{"Entries"}                 = "ingangen" ;
      $MS{"References"}              = "verwijzingen" ;

    } # end of dutch section

  elsif ($Interface eq "de")

    { # begin of German section

      $MS{"ProcessingReferences"}    = "Verarbeiten der Befehle, Listen und Register" ;
      $MS{"GeneratingDocumentation"} = "Vorbereiten der ConTeXt-Dokumentationsdatei" ;
      $MS{"GeneratingSources"}       = "Erstellen einer nur Quelltext ConTeXt-Datei" ;
      $MS{"FilteringDefinitions"}    = "Filtern der ConTeXt-Definitionen" ;
      $MS{"CopyingTemplates"}        = "Kopieren der TeXEdit-Test-key-templates" ;
      $MS{"CopyingInformation"}      = "Kopieren der TeXEdit-Hilfsinformation" ;
      $MS{"GeneratingFigures"}       = "Erstellen einer Abb-Uebersichtsdatei" ;
      $MS{"FilteringLogFile"}        = "Filtern der log-Datei" ;

      $MS{"SortingIJ"}               = "Sortiere IJ nach Y" ;
      $MS{"ConvertingHigh"}          = "Konvertiere hohe ASCII-Werte" ;
      $MS{"ProcessingQuotes"}        = "Verarbeiten der Akzentzeichen" ;
      $MS{"ForcingFileType"}         = "filetype einstellen" ;      # TOBIAS
      $MS{"UsingTiff"}               = "TIF-Dateien verarbeite" ;
      $MS{"FilteringBoxes"}          = "Filtern der ueberfuellten Boxen" ;
      $MS{"ApplyingCriterium"}       = "Anwenden des uebervoll-Kriteriums" ;
      $MS{"FilteringUnknown"}        = "Filter unbekannt ..." ;

      $MS{"NoInputFile"}             = "Keine Eingabedatei angegeben" ;
      $MS{"EmptyInputFile"}          = "Leere Eingabedatei" ;
      $MS{"NotYetImplemented"}       = "Noch nicht verfuegbar" ;
```

```
26      $MS{"Action"}                  = "                 Aktion :" ;
        $MS{"Option"}                  = "                 Option :" ;
        $MS{"Error"}                   = "                 Fehler :" ;
        $MS{"Remark"}                  = "              Anmerkung :" ;
        $MS{"SystemCall"}              = "            system call :" ; # TOBIAS
        $MS{"BadSystemCall"}           = "        bad system call :" ; # TOBIAS
        $MS{"MissingSubroutine"}       = "     missing subroutine :" ; # TOBIAS

27      $MS{"EmbeddedFiles"}           = "  Eingebettete Dateien :" ;
        $MS{"BeginEndError"}           = "   Beg./Ende-Fehler in :" ;
        $MS{"SynonymEntries"}          = "      Synonymeintraege :" ;
        $MS{"SynonymErrors"}           = " Fehlerhafte Eintraege :" ;
        $MS{"RegisterEntries"}         = "     Registereintraege :" ;
        $MS{"RegisterErrors"}          = " Fehlerhafte Eintraege :" ;
        $MS{"PassedCommands"}          = "    Verarbeite Befehle :" ;

28      $MS{"NOfDocuments"}            = "       Dokumentbloecke :" ;
        $MS{"NOfDefinitions"}          = "    Definitionsbloecke :" ;
        $MS{"NOfSkips"}                = "Uebersprungene Bloecke :" ;
        $MS{"NOfSetups"}               = "       Kopierte setups :" ;
        $MS{"NOfTemplates"}            = "    Kopierte templates :" ;
        $MS{"NOfInfos"}                = "    Kopierte helpinfos :" ;
        $MS{"NOfFigures"}              = "     Verarbeitete Abb. :" ;
        $MS{"NOfBoxes"}                = "        Zu volle Boxen :" ;
        $MS{"NOfUnknown"}              = "         Unbekannt ... :" ;

29      $MS{"InputFile"}               = "          Eingabedatei :" ;
        $MS{"OutputFile"}              = "          Ausgabedatei :" ;
        $MS{"FileType"}                = "             Dateitype :" ; # TOBIAS
        $MS{"EpsFile"}                 = "             eps-Datei :" ;
        $MS{"TifFile"}                 = "             tif-Datei :" ;

30      $MS{"Overfull"}                = "zu voll" ;
        $MS{"Entries"}                 = "Eintraege" ;
        $MS{"References"}              = "Referenzen" ;

31    } # end of German section

32  else

33    { # begin of english section

34      $MS{"ProcessingReferences"}    = "processing commands, lists and registers" ;
        $MS{"GeneratingDocumentation"} = "preparing ConTeXt documentation file" ;
        $MS{"GeneratingSources"}       = "generating ConTeXt source only file" ;
        $MS{"FilteringDefinitions"}    = "filtering formal ConTeXt definitions" ;
        $MS{"CopyingTemplates"}        = "copying TeXEdit quick key templates" ;
        $MS{"CopyingInformation"}      = "copying TeXEdit help information" ;
        $MS{"GeneratingFigures"}       = "generating figure directory file" ;
        $MS{"FilteringLogFile"}        = "filtering log file" ;

35      $MS{"SortingIJ"}               = "sorting IJ under Y" ;
        $MS{"ConvertingHigh"}          = "converting high ASCII values" ;
        $MS{"ProcessingQuotes"}        = "handling accented characters" ;
        $MS{"ForcingFileType"}         = "setting up filetype" ;
        $MS{"UsingTiff"}               = "processing TIF files" ;
        $MS{"FilteringBoxes"}          = "filtering overfull boxes" ;
        $MS{"ApplyingCriterium"}       = "applying overfull criterium" ;
        $MS{"FilteringUnknown"}        = "filtering unknown ..." ;

36      $MS{"NoInputFile"}             = "no input file given" ;
        $MS{"EmptyInputFile"}          = "empty input file" ;
        $MS{"NotYetImplemented"}       = "not yet available" ;

37      $MS{"Action"}                  = "                 action :" ;
        $MS{"Option"}                  = "                 option :" ;
        $MS{"Error"}                   = "                  error :" ;
        $MS{"Remark"}                  = "              remark :" ;
        $MS{"SystemCall"}              = "         system call :" ;
        $MS{"BadSystemCall"}           = "     bad system call :" ;
        $MS{"MissingSubroutine"}       = "  missing subroutine :" ;

38      $MS{"EmbeddedFiles"}           = "      embedded files :" ;
        $MS{"BeginEndError"}           = "        b/e error in :" ;
        $MS{"SynonymEntries"}          = "     synonym entries :" ;
        $MS{"SynonymErrors"}           = "         bad entries :" ;
        $MS{"RegisterEntries"}         = "    register entries :" ;
        $MS{"RegisterErrors"}          = "         bad entries :" ;
        $MS{"PassedCommands"}          = "     passed commands :" ;

39      $MS{"NOfDocuments"}            = "     document blocks :" ;
        $MS{"NOfDefinitions"}          = "   definition blocks :" ;
        $MS{"NOfSkips"}                = "      skipped blocks :" ;
        $MS{"NOfSetups"}               = "       copied setups :" ;
        $MS{"NOfTemplates"}            = "    copied templates :" ;
        $MS{"NOfInfos"}                = "    copied helpinfos :" ;
        $MS{"NOfFigures"}              = "   processed figures :" ;
        $MS{"NOfBoxes"}                = "      overfull boxes :" ;
        $MS{"NOfUnknown"}              = "         unknown ... :" ;

40      $MS{"InputFile"}               = "          input file :" ;
        $MS{"OutputFile"}              = "         output file :" ;
        $MS{"FileType"}                = "           file type :" ;
        $MS{"EpsFile"}                 = "            eps file :" ;
        $MS{"TifFile"}                 = "            tif file :" ;

41      $MS{"Overfull"}                = "overfull" ;
        $MS{"Entries"}                 = "entries" ;
        $MS{"References"}              = "references" ;

42    } # end of english section
```

Showing the banner (name and version of the program) and offering helpinfo is rather straightforward.

```
43   sub ShowBanner
       { Report("\n$Program\n") }

44   sub ShowHelpInfo
       { Report("HelpInfo") }
```

The helpinfo is also saved in the hash table. This looks like a waste of energy and space, but the program gains readability.

```
45   if ($Interface eq "nl")

46     { # begin of dutch section

47       $MS{"HelpInfo"} =

48       "          --references   hulp file verwerken / tui->tuo                \n".
         "                       --ij : IJ als Y sorteren                        \n".
         "                       --high : hoge ASCII waarden converteren         \n".
         "                       --quotes : quotes converteren                   \n".
         "                                                                       \n".
         "           --documents   documentatie file genereren / tex->ted        \n".
         "             --sources   broncode file genereren / tex->tes            \n".
         "              --setups   ConTeXt definities filteren / tex->texutil.tus \n".
         "           --templates   TeXEdit templates filteren / tex->tud         \n".
         "               --infos   TeXEdit helpinfo filteren / tex->tud          \n".
         "                                                                       \n".
         "             --figures   eps figuren lijst genereren / eps->texutil.tuf \n".
         "                       --tiff : ook tif files verwerken                \n".
         "                                                                       \n".
```

```
    "             --logfile   logfile filteren / log->texutil.log            \n".
    "                       --box : overfull boxes controleren               \n".
    "                       --criterium : overfull criterium in pt           \n" .
    "                       --unknown :onbekende ... controleren             \n" ;

  } # end of dutch section

elsif ($Interface eq "de")

  { # begin of German section

    $MS{"HelpInfo"} =

    "          --references   Verarbeiten der Hilfsdatei / tui->tuo          \n" .
    "                       --ij : Sortiere IJ als Y                         \n" .
    "                       --high : Konvertiere hohe ASCII-Werte            \n" .
    "                       --quotes : Konvertiere akzentuierte Buchstaben   \n" .
    "                                                                        \n" .
    "           --documents   Erstelle Dokumentationsdatei / tex->ted        \n" .
    "             --sources   Erstelle reine Quelltextdateien / tex->tes     \n" .
    "              --setups   Filtere ConTeXt-Definitionen / tex->texutil.tus\n" .
    "           --templates   Filtere TeXEdit-templates / tex->tud           \n" .
    "               --infos   Filtere TeXEdit-helpinfo / tex->tud            \n" .
    "                                                                        \n" .
    "             --figures   Erstelle eps-Abbildungsliste / eps->texutil.tuf\n" .
    "                       --tiff : Verarbeite auch tif-Dateien             \n" .
    "                                                                        \n" .
    "             --logfile   Filtere log-Datei / log->texutil.log           \n" .
    "                       --box : Ueberpruefe uebervolle Boxen             \n" .
    "                       --criterium : Uebervoll-Kriterium in pt          \n" .
    "                       --unknown : Ueberpruefe auf unbekannte ...       \n" ;

  } # end of German section

else

  { # begin of english section

    $MS{"HelpInfo"} =

    "          --references   process auxiliary file / tui->tuo              \n" .
    "                       --ij : sort IJ as Y                              \n" .
    "                       --high : convert high ASCII values               \n" .
    "                       --quotes : convert quotes characters             \n" .
    "                                                                        \n" .
    "           --documents   generate documentation file / tex->ted         \n" .
    "             --sources   generate source only file / tex->tes           \n" .
    "              --setups   filter ConTeXt definitions / tex->texutil.tus  \n" .
    "           --templates   filter TeXEdit templates / tex->tud            \n" .
    "               --infos   filter TeXEdit helpinfo / tex->tud             \n" .
    "                                                                        \n" .
    "             --figures   generate eps figure list / eps->texutil.tuf    \n" .
    "                       --tiff : also process tif files                  \n" .
    "                                                                        \n" .
    "             --logfile   filter logfile / log->texutil.log              \n" .
    "                       --box : check overful boxes                      \n" .
    "                       --criterium : overfull criterium in pt           \n" .
    "                       --unknown : check unknown ...                    \n" ;

  } # end of english section
```

In order to sort strings correctly, we have to sanitize them. This is especially needed when we include TEX commands, quotes characters and compound word placeholders.

- `\name`: csnames are stripped
- `{}`: are removed

- `\"e`: and alike are translated into `"e` etc.
- `"e`: is translated into an `e` and `b` etc.
- `||`: becomes `-`
- `\-`: also becomes `-`

Of course other accented characters are handled too. The appended string is responsible for decent sorting.

```
$TargetString = SanitizedString ( $SourceString ) ;
```

The sort order depends on the ordering in array `$ASCII`:

```
60  $ASCII{"^"} = "a" ;  $ASCII{'"'} = "b" ;  $ASCII{"‘"} = "c" ;
    $ASCII{"’"} = "d" ;  $ASCII{"~"} = "e" ;  $ASCII{","} = "f" ;

61  sub SanitizedString
      { my ($string) = $_[0] ;
        if ($ProcessQuotes)
          { $string =~ s/\\([\^\"\‘\’\~\,])/$1/gio ;
            $copied = $string ;
            $copied =~ s/([\^\"\‘\’\~\,])([a-zA-Z])/$ASCII{$1}/gio ;
            $string =~ s/([\^\"\‘\’\~\,])([a-zA-Z])/$2/gio ;
            $string=$string.$copied }
        $string =~ s/\\-|\|\|/\-/gio ;
        $string =~ s/\\[a-zA-Z]*| |\{|\}//gio ;
        return $string }
```

This subroutine looks a bit complicated, which is due to the fact that we want to sort for instance an accented `e` after the plain `e`, so the imaginary words

```
eerste
\"eerste
\"e\"erste
eerst\"e
```

come out in an acceptable order.

We also have to deal with the typical TEX sequences with the double `^`'s, like `^^45`. These hexadecimal coded characters are just converted.

```
$TargetString = HighConverted ( $SourceString ) ;
```

```
62  sub HighConverted
      { my ($string) = $_[0] ;
        $string =~ s/\^\^([a-f0-9][a-f0-9])/chr hex($1)/geo ;
        return $string }
```

references CONTEXT can handle many lists, registers (indexes), tables of whatever and references. This data is collected in one pass and processed in a second one. In between, relevant data is saved in the file `\jobname.tui`. This file also holds some additional information concerning second pass optimizations.

The main task of TEXUTIL is to sort lists and registers (indexes). The results are stored in again one file called `\jobname.tuo`.

Just for debugging purposes the nesting of files loaded during the CONTEXT run is stored. Of course this only applies to files that are handled by the CONTEXT file structuring commands (projects, products, components and environments).

We have to handle the entries:

```
f b {test}
f e {test}
```

and only report some status info at the end of the run.

```
63  sub InitializeFiles
      { $NOfFiles = 0 ;
        $NOfBadFiles = 0 }

64  sub HandleFile
     { $RestOfLine =~ s/.*\{(.*)\}/$1/gio ;
       ++$Files{$RestOfLine} }

65  sub FlushFiles
      { print TUO "%\n" . "% Files\n" . "%\n" ;
        foreach $File (keys %Files)
          { print TUO "% $File ($Files{$File})\n" }
        print TUO "%\n" ;
        $NOfFiles = keys %Files ;
        Report("EmbeddedFiles", $NOfFiles) ;
        foreach $File (keys %Files)
          { unless (($Files{$File} % 2) eq 0)
              { ++$NOfBadFiles ;
                Report("BeginEndError", $File) } } }
```

Commands don't need a special treatment. They are just copied. Such commands are tagged by a `c`, like:

```
c \thisisutilityversion{year.month.day}
c \twopassentry{class}{key}{value}
c \mainreference{prefix}{entry}{pagenumber}{realpage}{tag}
c \listentry{category}{tag}{number}{title}{pagenumber}{realpage}
c \realnumberofpages{number}
```

For historic reasons we check for the presense of the backslash.

```
66  sub InitializeCommands
      { print TUO "%\n" . "% Commands\n" . "%\n" ;
        $NOfCommands = 0 }

67  sub HandleCommand
      { ++$NOfCommands ;
        $RestOfLine =~ s/^\\//go ;
        print TUO  "\\$RestOfLine\n" }

68  sub FlushCommands
      { Report ("PassedCommands", $NOfCommands) }
```

Synonyms are a sort of key–value pairs and are used for ordered lists like abbreviations and units.

```
s e {class}{sanitized key}{key}{associated data}
```

The sorted lists are saved as (surprise):

```
\synonymentry{class}{sanitized key}{key}{associated data}
```

```
69   sub InitializeSynonyms
       { $NOfSynonyms = 0 ;
         $NOfBadSynonyms = 0 }

70   sub HandleSynonym
       { ++$NOfSynonyms ;
         ($SecondTag, $RestOfLine) = split(/ /, $RestOfLine, 2) ;
         ($Class, $Key, $Entry, $Meaning) = split(/} \{/, $RestOfLine) ;
         chop $Meaning ;
         $Class = substr $Class, 1 ;
         if ($Entry eq "")
           { ++$NOfBadSynonyms }
         else
           { $SynonymEntry[$NOfSynonyms] =
               $Class . $JOIN .
               $Key   . $JOIN .
               $Entry . $JOIN .
               $Meaning } }

71   sub FlushSynonyms
       { print TUO "%\n" . "% Synonyms\n" . "%\n" ;
         @SynonymEntry = sort @SynonymEntry ;
         $NOfSaneSynonyms = 0 ;
         for ($n=1; $n<=$NOfSynonyms; ++$n)
           { # check normally not needed
             if ($SynonymEntry[$n] ne $SynonymEntry[$n-1])  # [0]=undef
               { ($Class, $Key, $Entry, $Meaning) =
                     split(/\%\%/, $SynonymEntry[$n]) ;
                 ++$NOfSaneSynonyms ;
                 print TUO "\\synonymentry{$Class}{$Key}{$Entry}{$Meaning}\n" } }
         Report("SynonymEntries", $NOfSynonyms, "->", $NOfSaneSynonyms, "Entries") ;
         if ($NOfBadSynonyms>0)
           { Report("SynonymErrors", $NOfBadSynonyms) } }
```

Register entries need a bit more care, especially when they are nested. In the near future we will also handle page ranges.

```
r e {class}{tag}{sanitized key}{key}{pagenumber}{realpage}
r s {class}{tag}{sanitized key}{key}{string}{pagenumber}
```

The first one is the normal entry, the second one concerns *see this or that* entries. Keys are sanitized, unless the user supplies a sanitized key. To save a lot of programming, all data concerning an entry is stored in one string. Subentries are specified as:

```
first|second|third
first+second+third
```

When these characters are needed for typesetting purposes, we can also use the first character to specify the separator:

```
|$x^2+y^2=r^2$
+compound||word
```

Subentries are first unpacked and next stored in a consistent way, which means that we can use both separators alongside each other. We leave it to the reader to sort out the dirty tricks.

```
72   sub InitializeRegisters
       { $NOfEntries = 0 ;
         $NOfBadEntries = 0 }

73   $SPLIT="&&" ;
     $JOIN ="%%" ;

74   sub HandleRegister
       { ($SecondTag, $RestOfLine) = split(/ /, $RestOfLine, 2) ;
         ++$NOfEntries ;
         if ($SecondTag eq "s")
           { ($Class, $Location, $Key, $Entry, $SeeToo, $Page ) =
                split(/} \{/, $RestOfLine) ;
             chop $Page ;
             $Class = substr $Class, 1 ;
             $RealPage = 0 }
         else
           { ($Class, $Location, $Key, $Entry, $Page, $RealPage ) =
                split(/} \{/, $RestOfLine) ;
             chop $RealPage ;
             $Class = substr $Class, 1 ;
             $SeeToo = "" }
         if ($Key eq "")
           { $Key = SanitizedString($Entry) ;
             if ($ProcessHigh)
               { $Key = HighConverted($Key) } }
         $KeyTag = substr $Key, 0, 1 ;
         if ($KeyTag eq "|")
           { $Key   =~ s/^\|//go ;
             $Key   =~ s/([^\\])\|/$1$SPLIT/go ;
             $Key   =~ s/$SPLIT\|/\|\|/go ;        # repair compound||word
             $Entry =~ s/^\|//go ;
             $Entry =~ s/([^\\])\|/$1$SPLIT/go ;
             $Entry =~ s/$SPLIT\|/\|\|/go }        # repair compound||word
         elsif ($KeyTag eq "+")
           { $Key   =~ s/^\+//go ;
             $Key   =~ s/([^\\])\+/$1$SPLIT/go ;
             $Entry =~ s/^\+//go ;
             $Entry =~ s/([^\\])\+/$1$SPLIT/go }
         else
           { $Key   =~ s/([^\\])\|/$1$SPLIT/go ;
             $Key   =~ s/([^\\])\+/$1$SPLIT/go ;
             $Key   =~ s/$SPLIT\|/\|\|/go ;        # repair compound||word
             $Entry =~ s/([^\\])\|/$1$SPLIT/go ;
             $Entry =~ s/([^\\])\+/$1$SPLIT/go ;
             $Entry =~ s/$SPLIT\|/\|\|/go }        # repair compound||word
         $Key =~ s/^([^a-zA-Z])/ $1/go ;          # special characters
         if ($ProcessIJ)
           { $Key =~ s/ij/yy/go }
         $LCKey = lc $Key ;
         $RegisterEntry[$NOfEntries] =
           $Class     . $JOIN .
           $LCKey     . $JOIN .
```

```
      $Key      . $JOIN .
      $Entry    . $JOIN .
      $RealPage . $JOIN .
      $Location . $JOIN .
      $Page     . $JOIN .
      $SeeToo }

sub FlushRegisters
  { print TUO "%\n" . "% Registers\n" . "%\n" ;
    @RegisterEntry  = sort @RegisterEntry ;
    $NOfSaneEntries = 0 ;
    $NOfSanePages   = 0 ;
    $LastPage       = "" ;
    $LastRealPage   = "" ;
    $AlfaClass      = "" ;
    $Alfa           = "" ;
    $PreviousA      = "" ;
    $PreviousB      = "" ;
    $PreviousC      = "" ;
    for ($n=1 ; $n<=$NOfEntries ; ++$n)
      { ($Class, $LCKey, $Key, $Entry, $RealPage, $Location, $Page, $SeeToo) =
           split(/\%\%/, $RegisterEntry[$n]) ;
        if (((lc substr $Key, 0, 1) ne lc $Alfa) or ($AlfaClass ne $Class))
          { $Alfa= lc substr $Key, 0, 1 ;
            $AlfaClass = $Class ;
            if ($Alfa ne " ")
              { print TUO "\\registerentry{$Class}{$Alfa}\n" } }
        ($ActualA, $ActualB, $ActualC ) =
           split(/$SPLIT/, $Entry, 3) ;
        if ($ActualA eq $PreviousA)
          { $ActualA = "" }
        else
          { $PreviousA = $ActualA ;
            $PreviousB = "" ;
            $PreviousC = "" }
        if ($ActualB eq $PreviousB)
          { $ActualB = "" }
        else
          { $PreviousB = $ActualB ;
            $PreviousC = "" }
        if ($ActualC eq $PreviousC)
          { $ActualC = "" }
        else
          { $PreviousC = $ActualC }
        if ($ActualB eq "")
          { $PreviousC = "" ;
            $ActualC = "" }
        $Copied = 0 ;
        if ($ActualA ne "")
           { print TUO "\\registerentrya{$Class}{$ActualA}\n" ;
             $Copied = 1 }
        if ($ActualB ne "")
           { print TUO "\\registerentryb{$Class}{$ActualB}\n" ;
```

```
              $Copied = 1 }
         if ($ActualC ne "")
            { print TUO "\\registerentryc{$Class}{$ActualC}\n" ;
              $Copied = 1 }
         if ($Copied)
           { $NOfSaneEntries++ }
         if ($RealPage eq 0)
           { print TUO "\\registersee{$Class}{$SeeToo}{$Page}\n" ;
             $LastPage = $Page ;
             $LastRealPage = $RealPage }
         elsif (($Copied) ||
                ! (($LastPage eq $Page) and ($LastRealPage eq $RealPage)))
           { print TUO "\\registerpage{$Class}{$Location}{$Page}{$RealPage}\n" ;
             ++$NOfSanePages ;
             $LastPage = $Page ;
             $LastRealPage = $RealPage }
       }
     Report("RegisterEntries", $NOfEntries, "->", $NOfSaneEntries, "Entries",
                                                  $NOfSanePages,   "References") ;
     if ($NOfBadEntries>0)
       { Report("RegisterErrors", $NOfBadEntries) } }
```

For debugging purposes we flush some status information. The faster machines become, the more important this section will be.

76
```
sub FlushData
  { print TUO
      "% This Session\n" .
      "% \n" .
      "% embedded files    : $NOfFiles ($NOfBadFiles errors)\n" .
      "% passed commands   : $NOfCommands\n" .
      "% synonym entries   : $NOfSynonyms ($NOfBadSynonyms errors)\n" .
      "% register entries : $NOfEntries ($NOfBadEntries errors)" }
```

The functionallity described on the previous few pages is called upon in the main routine:

77
```
sub HandleReferences
  { Report("Action", "ProcessingReferences") ;
    if ($ProcessIJ  )
      { Report("Option", "SortingIJ") }
    if ($ProcessHigh)
      { Report("Option", "ConvertingHigh") }
    if ($ProcessQuotes)
      { Report("Option", "ProcessingQuotes") }
    if ($InputFile eq "")
      { Report("Error", "NoInputFile") }
    else
      { unless (open (TUI, "$InputFile.tui"))
          { Report("Error", "EmptyInputFile", $InputFile) }
        else
          { Report("InputFile", "$InputFile.tui" ) ;
            Report("OutputFile", "$InputFile.tuo" ) ;
            open (TUO, ">$InputFile.tuo") ;
            InitializeCommands ;
```

```
                  InitializeRegisters ;
                  InitializeSynonyms ;
                  InitializeFiles ;
                  while ($SomeLine=<TUI>)
                    { chomp ;
                      chop $SomeLine ;
                      ($FirstTag, $RestOfLine) = split ' ', $SomeLine, 2 ;
                      if    ($FirstTag eq "c")
                        { HandleCommand }
                      elsif ($FirstTag eq "s")
                        { HandleSynonym }
                      elsif ($FirstTag eq "r")
                        { HandleRegister }
                      elsif ($FirstTag eq "f")
                        { HandleFile } }
                  FlushCommands ;
                  FlushRegisters ;
                  FlushSynonyms ;
                  FlushFiles ;
                  FlushData } } }
```

documents Documentation can be woven into a source file. The next routine generates a new, TEX ready file with the documentation and source fragments properly tagged. The documentation is included as comment:

```
%D ......   some kind of documentation
%M ......   macros needed for documenation
%S B        begin skipping
%S E        end skipping
```

The most important tag is `%D`. Both TEX and METAPOST files use `%` as a comment chacacter, while PERL uses `#`. Therefore `#D` is also handled.

The generated file gets the suffix `ted` and is structured as:

```
\startmodule[type=suffix]
\startdocumentation
\stopdocumentation
\startdefinition
\stopdefinition
\stopmodule
```

Macro definitions specific to the documentation are not surrounded by start–stop commands. The suffix specifaction can be overruled at runtime, but defaults to the file extension. This specification can be used for language depended verbatim typesetting.

```
78  sub HandleDocuments
      { Report("Action", "HandlingDocuments") ;
        if ($ProcessType ne "")
          { Report("Option", "ForcingFileType", $ProcessType) }
        if ($InputFile eq "")
          { Report("Error", "NoInputFile") }
        else
          { @UserSuppliedFiles = InputFiles ($InputFile) ;
            foreach $FullName (@UserSuppliedFiles)
```

```
{ ($FileName, $FileSuffix) = split (/\./, $FullName, 2) ;
  if ($FileSuffix eq "")
    { $FileSuffix = "tex" }
  unless (-f "$FileName.$FileSuffix")
    { next }
  unless (open (TEX, "$FileName.$FileSuffix"))
    { Report("Error", "EmptyInputFile", "$FileName.$FileSuffix" ) }
  else
    { Report("InputFile",  "$FileName.$FileSuffix") ;
      Report("OutputFile", "$FileName.ted") ;
      open (TED, ">$FileName.ted") ;
      $NOfDocuments   = 0 ;
      $NOfDefinitions = 0 ;
      $NOfSkips       = 0 ;
      $SkipLevel      = 0 ;
      $InDocument     = 0 ;
      $InDefinition   = 0 ;
      if ($ProcessType eq "")
        { $FileType=lc $FileSuffix }
      else
        { $FileType=lc $ProcessType }
      Report("FileType", $FileType) ;
      print TED "\\startmodule[type=$FileType]\n" ;
      while ($SomeLine=<TEX>)
        { chomp ;
          chop $SomeLine ;
          if ($SomeLine =~ /^[%\#]D/)
            { if ($SkipLevel eq 0)
                { $SomeLine = substr $SomeLine, 3 ;
                  if ($InDocument)
                    { print TED "$SomeLine\n" }
                  else
                    { if ($InDefinition)
                        { print TED "\\stopdefinition\n" ;
                          $InDefinition = 0 }
                      unless ($InDocument)
                        { print TED "\n\\startdocumentation\n" }
                      print TED "$SomeLine\n" ;
                      $InDocument = 1 ;
                      ++$NOfDocuments } } }
          elsif ($SomeLine =~ /^[%\#]M/)
            { if ($SkipLevel eq 0)
                { $SomeLine = substr $SomeLine, 3 ;
                  print TED "$SomeLine\n" } }
         elsif ($SomeLine =~ /^[%\%]S B]/)
           { ++$SkipLevel ;
             ++$NOfSkips }
         elsif ($SomeLine =~ /^[%\%]S E]/)
           { --$SkipLevel }
          elsif ($SomeLine =~ /^[%\#]/)
           { }
          elsif ($SkipLevel eq 0)
            { $InLocalDocument = $InDocument ;
            if ($InDocument)
             { print TED "\\stopdocumentation\n" ;
               $InDocument = 0 }
            if (($SomeLine eq "") && ($InDefinition))
             { print TED "\\stopdefinition\n" ;
               $InDefinition = 0 }
            else
             { if ($InDefinition)
                 { print TED "$SomeLine\n" }
               elsif ($SomeLine ne "")
                 { print TED "\n" . "\\startdefinition\n" ;
                   $InDefinition = 1 ;
                   unless ($InLocalDocument)
                     { ++$NOfDefinitions }
                   print TED "$SomeLine\n" } } } }
        if ($InDocument)
          { print TED "\\stopdocumentation\n" }
        if ($InDefinition)
          { print TED "\\stopdefinition\n" }
        print TED "\\stopmodule\n" ;
        Report ("NOfDocuments", $NOfDocuments) ;
        Report ("NOfDefinitions", $NOfDefinitions) ;
        Report ("NOfSkips", $NOfSkips) } } } }
```

sources Documented sources can be stripped of documentation and comments, although at the current processing speeds the overhead of skipping the documentation at run time is neglectable. Only lines beginning with a % are stripped. The stripped files gets the suffix `tes`.

```
79  sub HandleSources
     { Report("Action", "GeneratingSources") ;
      if ($InputFile eq "")
       { Report("Error", "NoInputFile") }
      else
       { @UserSuppliedFiles = InputFiles ($InputFile) ;
         foreach $FullName (@UserSuppliedFiles)
          { ($FileName, $FileSuffix) = split (/\./, $FullName, 2) ;
            if ($FileSuffix eq "")
             { $FileSuffix = "tex" }
            unless (-f "$FileName.$FileSuffix")
             { next }
            unless (open (TEX, "$FileName.$FileSuffix"))
             { Report("Error", "EmptyInputFile", "$FileName.$FileSuffix" ) }
            else
             { Report("InputFile",  "$FileName.$FileSuffix") ;
               Report("OutputFile", "$FileName.tes") ;
               open (TES, ">$FileName.tes") ;
               $EmptyLineDone = 1 ;
               $FirstCommentDone = 0 ;
               while ($SomeLine=<TEX>)
                { chomp ;
                  chop $SomeLine ;
                  if ($SomeLine eq "")
                   { unless ($FirstCommentDone)
             { $FirstCommentDone = 1 ;
              print TES
               "\n% further documentation is removed\n\n" ;
              $EmptyLineDone = 1 }
            unless ($EmptyLineDone)
             { print TES "\n" ;
              $EmptyLineDone = 1 } }
          elsif ($SomeLine =~ /^%/)
           { unless ($FirstCommentDone)
             { print TES "$SomeLine\n" ;
              $EmptyLineDone = 0 } }
          else
           { print TES "$SomeLine\n" ;
            $EmptyLineDone = 0 } } } } } }
```

setups All CONTEXT commands are specified in a compact format that can be used to generate quick reference tables and cards. Such setups are preceded by %S. The setups are collected in the file `texutil.tus`.

```
sub HandleSetups
 { Report("Action", "FilteringDefinitions" ) ;
  if ($InputFile eq "")
   { Report("Error", "NoInputFile") }
  else
   { Report("OutputFile", "texutil.tus") ;
    open (TUS, ">texutil.tus") ; # this file is always reset!
    $NOfSetups = 0 ;
    @UserSuppliedFiles = InputFiles ($InputFile) ;
    foreach $FullName (@UserSuppliedFiles)
     { ($FileName, $FileSuffix) = split (/\./, $FullName, 2) ;
      if ($FileSuffix eq "")
       { $FileSuffix = "tex" }
      unless (-f "$FileName.$FileSuffix")
       { next }
      unless (open (TEX, "$FileName.$FileSuffix"))
       { Report("Error", "EmptyInputFile", "$FileName.$FileSuffix" ) }
      else
       { Report("InputFile",  "$FileName.$FileSuffix") ;
        print TUS "%\n" . "% File : $FileName.$FileSuffix\n" . "%\n" ;
        while ($SomeLine=<TEX>)
         { chomp ;
          chop $SomeLine ;
          ($Tag, $RestOfLine) = split(/ /, $SomeLine, 2) ;
          if ($Tag eq "%S")
           { ++$NOfSetups ;
            while ($Tag eq "%S")
             { print TUS "$RestOfLine\n" ;
              $SomeLine = <TEX> ;
              chomp ;
              chop $SomeLine ;
              ($Tag, $RestOfLine) = split(/ /, $SomeLine, 2) }
            print TUS "\n" } } } }
    Report("NOfSetups", $NOfSetups) } }
```

templates
infos

From the beginning, the CONTEXT source files contained helpinfo and key–templates for TEXEDIT. In fact, for a long time, this was the only documentation present. More and more typeset (interactive) documentation is replacing this helpinfo, but we still support the traditional method. This information is formatted like:

```
%I n=Struts
%I c=\strut,\setnostrut,\setstrut,\toonstruts
%I
%I text
%I ....
%P
%I text
%I ....
```

Templates look like:

```
%T n=kap
%T m=kap
%T a=k
%T
%T \kap{?}
```

The key–value pairs stand for *name*, *mnemonic*, *key*. This information is copied to files with the extension tud.

81

```
sub HandleEditorCues
  { if ($ProcessTemplates)
      { Report("Action", "CopyingTemplates" ) }
    if ($ProcessInfos)
      {Report("Action", "CopyingInformation" ) }
    if ($InputFile eq "")
      { Report("Error", "NoInputFile") }
    else
      { @UserSuppliedFiles = InputFiles ($InputFile) ;
        foreach $FullName (@UserSuppliedFiles)
          { ($FileName, $FileSuffix) = split (/\./, $FullName, 2) ;
            if ($FileSuffix eq "")
              { $FileSuffix = "tex" }
            unless (-f "$FileName.$FileSuffix")
              { next }
            unless (open (TEX, "$FileName.$FileSuffix"))
              { Report("Error", "EmptyInputFile", "$FileName.$FileSuffix" ) }
            else
              { Report("InputFile",  "$FileName.$FileSuffix") ;
                Report("OutputFile", "$FileName.tud") ;
                open (TUD, ">$FileName.tud") ;
                $NOfTemplates = 0 ;
                $NOfInfos = 0 ;
                while ($SomeLine=<TEX>)
                  { chomp ;
                    chop $SomeLine ;
                    ($Tag, $RestOfLine) = split(/ /, $SomeLine, 2) ;
                    if (($Tag eq "%T") && ($ProcessTemplates))
                      { ++$NOfTemplates ;
```

```
                              while ($Tag eq "%T")
                                { print TUD "$SomeLine\n" ;
                                  $SomeLine = <TEX> ;
                                  chomp ;
                                  chop $SomeLine ;
                                  ($Tag, $RestOfLine) = split(/ /, $SomeLine, 2) }
                                  print TUD "\n" }
                            elsif (($Tag eq "%I") && ($ProcessInfos))
                              { ++$NOfInfos ;
                                while (($Tag eq "%I") || ($Tag eq "%P"))
                                  { print TUD "$SomeLine\n" ;
                                    $SomeLine = <TEX> ;
                                    chomp ;
                                    chop $SomeLine ;
                                    ($Tag, $RestOfLine) = split(/ /, $SomeLine, 2) }
                                    print TUD "\n" } }
                        if ($ProcessTemplates)
                          { Report("NOfTemplates", $NOfTemplates) }
                        if ($ProcessInfos)
                          { Report("NOfInfos", $NOfInfos) } } } } }
```

figures Directories can be scanned for illustrations in EPS or TIFF format. The later type of graphics is prescanned by dedicated programs, whose data is used here. The resulting file `texutil.tuf` contains entries like:

```
\thisisfigureversion{year.month.day}
\presetfigure[file][...specifications...]
```

where the specifications are:

```
[e=suffix,x=xoffset,y=yoffset,w=width,h=height,t=title,c=creator,s=size]
```

This data can be used when determining dimensions (although CONTEXT is able to scan EPS illustrations directly) and to generate directories of illustrations.

82
```
$PTtoCM = 2.54/72.0 ;
$INtoCM = 2.54 ;
```

83
```
sub HandleEpsFigure
  { my ( $SuppliedFileName ) = @_ ;
    ($FileName, $FileSuffix) = split ( /\./, $SuppliedFileName, 2) ;
    if (lc $FileSuffix ne "eps")
     { return }
    $HiResBBOX = "" ;
    $LoResBBOX  = "" ;
    $EpsTitle = "" ;
    $EpsCreator = "" ;
    Report ( "EpsFile", "$SuppliedFileName" ) ;
    open ( EPS , $SuppliedFileName ) ;
    $EpsSize = -s EPS ;
    while ( $SomeLine = <EPS> )
      { chop ($SomeLine) ;
        unless ($HiResBBOX)
          { if ($SomeLine =~ /^%%BoundingBox:/i)
              { ($Tag, $LoResBBOX) = split (/ /, $SomeLine, 2) ;
```

```
                    next }
                  elsif ($SomeLine =~ /^%%HiResBoundingBox:/i)
                    { ($Tag, $HiResBBOX) = split (/ /, $SomeLine, 2) ;
                      next }
                  elsif ($SomeLine =~ /^%%ExactBoundingBox:/i)
                    { ($Tag, $HiResBBOX) = split (/ /, $SomeLine, 2) ;
                      next } }
              if ($SomeLine =~ /^%%Creator:/i)
                { ($Tag, $EpsCreator) = split (/ /, $SomeLine, 2) }
              elsif ($SomeLine =~ /^%%Title:/i)
                { ($Tag, $EpsTitle) = split (/ /, $SomeLine, 2) } }
          if ($HiResBBOX)
            { $EpsBBOX = $HiResBBOX }
          else
            { $EpsBBOX = $LoResBBOX }
          if ($EpsBBOX)
            { ($LLX, $LLY, $URX, $URY, $RestOfLine) = split (/ /, $EpsBBOX, 5 ) ;
              $EpsHeight  = ($URY-$LLY)*$PTtoCM ;
              $EpsWidth   = ($URX-$LLX)*$PTtoCM ;
              $EpsXOffset = $LLX*$PTtoCM ;
              $EpsYOffset = $LLY*$PTtoCM ;
              $Figures[++$NOfFigures] =
                "\\presetfigure[$FileName][e=$FileSuffix" .
                (sprintf  ",x=%5.3fcm,y=%5.3fcm", $EpsXOffset, $EpsYOffset)  .
                (sprintf  ",w=%5.3fcm,h=%5.3fcm", $EpsWidth,   $EpsHeight)  .
                ",t=$EpsTitle,c=$EpsCreator,s=$EpsSize]\n" } }
```

84
```
sub HandleEpsFigures
  { if ($InputFile eq "")
      { $InputFile = "*.eps" }
    @UserSuppliedFiles = InputFiles ($InputFile) ;
    foreach $FileName (@UserSuppliedFiles)
      { HandleEpsFigure ( $FileName ) } }
```

Here we call the programs that generate information on the TIFF files. The names of the programs are defined earlier.

85
```
if ($ProcessTiff)
  { $FindTiffFigure = "" ;
    $UsedTiffProgram = "" ;
    unlink "tiffdata.tmp" ;
    foreach $TiffProgram (@TiffPrograms)
      { if ((system ("$TiffProgram *.tif > tiffdata.tmp") == 0)
            && (-s "tiffdata.tmp"))
          { $UsedTiffProgram = $TiffProgram ;
            $FindTiffFigure = "FindTiffFigure_$UsedTiffProgram" ;
            last } } }
```

The scanning routines use filehandle `TMP` and call for `ReportTifFigure` with the arguments `Name`, `Width` and `Height`.

86
```
sub ReportTifFigure
  { my ($Name, $Width, $Height) = @_ ;
    $Name = lc $Name ;
```

```
#
# if ($InputFile ne "")
#   { $Ok = 0 ;
#     foreach $IFile (@UserSuppliedFiles)
#     { $Ok = ($Ok || ($IFile eq "$Name.tif" )) }
# else
#   { $Ok = 1 }
#
$Ok = 1 ;
#
if ($Ok)
  { $Size = -s "$Name.tif" ;
    Report ( "TifFile", "$Name.tif") ;
    $Figures[++$NOfFigures] =
      "\\presetfigure[$Name][e=tif" .
      (sprintf ",w=%5.3fcm,h=%5.3fcm", $Width, $Height) .
      ",s=$Size]\n" } }
```

87
```
sub HandleTifFigures
  { if ($ProcessTiff)
      { if (-s "tiffdata.tmp")
          { Report ( "SystemCall", "$UsedTiffProgram -> tiffdata.tmp" ) ;
           if ((defined &$FindTiffFigure) && (open(TMP, "tiffdata.tmp")))
              { &$FindTiffFigure }
           else
              { Report ( "MissingSubroutine", $FindTiffFigure ) } }
        else
          { Report ( "BadSystemCall", "@TiffPrograms" ) } } }
```

The next few routines are program specific. Let's cross our fingers on the stability off their output. As one can see, we have to work a bit harder when we use TIFFINFO instead of TIFFTAGS.

88
```
sub FindTiffFigure_tiffinfo
  { while ( $TifName = <TMP> )
      { chomp ;
        chop $TifName ;
        if (($TifName =~ s/^(.*)\.tif\:.*/$1/i) &&
            ($SomeLineA = <TMP>) && ($SomeLineA = <TMP>) &&
            ($SomeLineA = <TMP>) && ($SomeLineB = <TMP>))
          { chop $SomeLineA ;
            $TifWidth  =  $SomeLineA ;
            $TifWidth  =~ s/.*Image : (.*) .*/$1/i ;
            $TifHeight =  $SomeLineA ;
            $TifHeight =~ s/.*Image : (.*).*/$1/i ;
            $TifWRes   =  $SomeLineB ;
            $TifWRes   =~ s/.*: (.*)\,.*/$1/i ;
            $TifHRes   =  $SomeLineB ;
            $TifHRes   =~ s/.*: .*\, (.*).*/$1/i ;
            $TifWidth  = ($TifWidth/$TifWRes)*$INtoCM ;
            $TifHeight = ($TifHeight/$TifHRes)*$INtoCM ;
            ReportTifFigure ($TifName, $TifWidth, $TifHeight) } } }
```

89
```
sub FindTiffFigure_tifftags
  { while ( $TifName = <TMP> )
```

```
              { chomp ;
                chop $TifName ;
                if (($TifName =~ s/.*\`(.*)\.tif\'.*/$1/i) &&
                    ($SomeLine = <TMP>) && ($SomeLine = <TMP>))
                  { chop $SomeLine ;
                    $TifWidth  =  $SomeLine ;
                    $TifWidth  =~ s/.*\((.*) pt.*\((.*) pt.*/$1/ ;
                    $TifHeight =  $SomeLine ;
                    $TifHeight =~ s/.*\((.*) pt.*\((.*) pt.*/$2/ ;
                    $TifWidth  =  $TifWidth*$PTtoCM ;
                    $TifHeight =  $TifHeight*$PTtoCM ;
                    ReportTifFigure ($TifName, $TifWidth, $TifHeight) } } }
```

```
90  sub InitializeFigures
      { $NOfFigures = 0 }
```

```
91  sub FlushFigures
      { $Figures = sort $Figures ;
        open ( TUF, ">texutil.tuf" ) ;
        print TUF "%\n" . "% Figures\n" . "%\n" ;
        print TUF "\\thisisfigureversion\{1996.06.01\}\n" . "%\n" ;
        for ($n=1 ; $n<=$NOfFigures ; ++$n)
          { print TUF $Figures[$n] }
        Report ( "NOfFigures", $NOfFigures ) }
```

```
92  sub HandleFigures
      { Report("Action",  "GeneratingFigures" ) ;
        if ($ProcessTiff)
          { Report("Option", "UsingTiff") }
        InitializeFigures ;
        HandleEpsFigures ;
        HandleTifFigures ;
        FlushFigures }
```

logfiles This (poor man's) log file scanning routine filters overfull box messages from a log file (`\hbox`, `\vbox` or both). The collected problems are saved in `texutil.log`. One can specify a selection criterium.

CONTEXT reports unknown entities. These can also be filtered. When using fast computers, or when processing files in batch, one has to rely on the log files and/or this filter.

```
93  $Unknown = "onbekend|unknown|unbekant" ;
```

```
94  sub FlushLogTopic
      { unless ($TopicFound)
           { $TopicFound = 1 ;
             print ALL "\n% : $FileName.log\n\n" } }
```

```
95  sub HandleLogFile
      { if ($ProcessBox)
          { Report("Option", "FilteringBoxes", "(\\vbox & \\hbox)") ;
            $Key = "[h|v]box" }
        elsif ($ProcessHBox)
          { Report("Option", "FilteringBoxes", "(\\hbox)") ;
            $Key = "hbox" ;
            $ProcessBox = 1 }
        elsif ($ProcessVBox)
          { Report("Option", "FilteringBoxes", "(\\vbox)") ;
            $Key = "vbox" ;
            $ProcessBox = 1 }
        if (($ProcessBox) && ($ProcessCriterium))
          { Report("Option", "ApplyingCriterium") }
        if ($ProcessUnknown)
          { Report("Option", "FilteringUnknown") }
        unless (($ProcessBox) || ($ProcessUnknown))
          { ShowHelpInfo ;
            return }
        Report("Action",  "FilteringLogFile" ) ;
        if ($InputFile eq "")
          { Report("Error", "NoInputFile") }
        else
          { $NOfBoxes = 0 ;
            $NOfMatching = 0 ;
            $NOfUnknown = 0 ;
            Report("OutputFile", "texutil.log") ;
            @UserSuppliedFiles = InputFiles ($InputFile) ;
            open ( ALL, ">texutil.log" ) ;
            foreach $FullName (@UserSuppliedFiles)
              { ($FileName, $FileSuffix) = split (/\./, $FullName, 2) ;
                if (! open (LOG, "$FileName.log"))
                  { Report("Error", "EmptyInputFile", "$FileName.$FileSuffix" ) }
                elsif (-e "$FileName.tex")
                  { $TopicFound = 0 ;
                    Report("InputFile", "$FileName.log") ;
                    while ($SomeLine=<LOG>)
                      { chomp ;
                        if (($ProcessBox) && ($SomeLine =~ /Overfull \\$Key/))
                          { ++$NOfBoxes ;
                            $SomePoints = $SomeLine ;
                            $SomePoints =~ s/.*\((.*)pt.*/$1/ ;
                            if ($SomePoints>=$ProcessCriterium)
                              { ++$NOfMatching ;
                                FlushLogTopic ;
                                print ALL "$SomeLine" ;
                                $SomeLine=<LOG> ;
                                print ALL $SomeLine } }
                        if (($ProcessUnknown) && ($SomeLine =~ /$Unknown/io))
                         { ++$NOfUnknown ;
                           FlushLogTopic ;
                           print ALL "$SomeLine" } } } }
            if ($ProcessBox)
              { Report ( "NOfBoxes" , "$NOfBoxes", "->", $NOfMatching, "Overfull") }
            if ($ProcessUnknown)
              { Report ( "NOfUnknown", "$NOfUnknown") } } }
```

We're done! All this actions and options are organized in one large conditional:

```
96                                ShowBanner       ;
```

```
97  if      ($UnknownOptions   ) { ShowHelpInfo     } # not yet done
    elsif   ($ProcessReferences) { HandleReferences }
    elsif   ($ProcessDocuments ) { HandleDocuments  }
    elsif   ($ProcessSources   ) { HandleSources    }
    elsif   ($ProcessSetups    ) { HandleSetups     }
    elsif   ($ProcessTemplates ) { HandleEditorCues }
    elsif   ($ProcessInfos     ) { HandleEditorCues }
    elsif   ($ProcessFigures   ) { HandleFigures    }
    elsif   ($ProcessLogFile   ) { HandleLogFile    }
    elsif   ($ProcessHelp      ) { ShowHelpInfo     } # redundant
    else                         { ShowHelpInfo     }
```

So far.

TEXUTIL