1 Processing of SGML documents

As has been frequently noted, an SGML document may be processed in a wide variety of ways. In this section, we give a general overview of the principles involved by way of an introduction to a set of specific transformations carried out on SGML documents created for the Memoria project.

An SGML document, although presented as a straightforward ASCII stream, is in fact a representation of a hierarchically organized tree structure in which one node, named for the document type, contains the whole document, and may be hierarchically decomposed into several others, each with a named type, and each optionally containing associated attributes. [1]

An SGML parser, such as the public domain parser SGMLS, checks that documents match the tree structure defined by the SGML grammar, or document type definition. It may also be configured to produce a normalized form of the document, in which the *element structure* is represented unambiguously. (An SGML document need not represent explicitly all of its structural markup: various types of *minimization*, such as the omission of contextually-determined tags, being permitted by the standard). The *Element Structure Information Set* (ESIS) output by such an SGML parser is therefore an essential first step in the creation of an efficient general purpose SGML processing tool.

For example, we give here a tiny example of a valid SGML document, complete with its DTD. The DTD defines a document as a series of P elements, each of which may bear an identifier attribute, and may contain character data, possibly including `<foreign>` elements, each of which must bear a `lang` attribute. The document representation itself uses arbitrary changes of case, and omits various tags, as is permitted by the standard.

```
<!DOCTYPE doc [
<!ELEMENT doc - - (p+)>
<!ELEMENT p - o (#PCDATA | foreign)*>
<!ATTLIST p id ID #implied>
<!ELEMENT foreign - o (#PCDATA)>
<!ATTLIST foreign  lang cdata #required>
<!ENTITY ccedil "&#135"> ]>
<Doc>
<p>This is the first paragraph.
<p Id=p2>This is
the second paragraph, which contains <foreign lang=FRA>
des mots fran&ccedil;ais.
</doC>
```

The document itself contains two paragraphs, of which the second bears an identifier P2, and includes the non-ASCII character c-cedilla. This file, when passed through the public domain

[1]For an introduction to the basic concepts of SGML, see any one of a number of text books now available; for a simple humanities-oriented introduction, see the Gentle Introduction to SGML published as part of the TEI Guidelines

SGMLS parser will produce the following output:

```
(DOC
AID IMPLIED
(P
-This is the first paragraph.
)P
AID TOKEN P2
(P
-This is \nthe second paragraph, which contains
ALANG CDATA FRA
(FOREIGN
-des mots franĞais.
)FOREIGN
)P
)DOC
```

The output in this instance consists of a series of lines, the first character of each of which indicates its function. The left parenthesis (indicates the start of an element; the right, the end of one. Lines beginning with the letter A indicate an attribute value for the next element to begin in sequence; lines beginning with a hyphen contain character data, the content of the element.

It should be evident that output in this format is far more easily processed than the original SGML input. Note, for example, that the line breaks have been made explicit; attributes and end-tags not specified are explicitly supplied; and the entity reference (the c-cedilla) has been replaced by an appropriate numeric code.

Several varyingly flexible and powerful programming tools are now available which operate directly on the ESIS output by an SGML parser. In some cases, the parser is embedded within the tool (for example, the commercial products Balise or Omnimark); in others the tool is used as a part of a UNIX pipeline (for example the public domain tools SGMLSpm, CoST and tf). The basic principles remain the same, and may be summarized as follows:

- The SGML document instance is processed left to right, and an internal tree representation is created.
- Each node of the document tree carries its name and any associated attribute values, or (if it is a terminal node) a fragment of parsed character data, representing the document contents.
- The programmer supplies a set of declarations which specify the actions to be performed, usually in terms of *event*s, such as “start of element type x”, “end of element type x” etc. and often using a particular programming language.
- Processing may be made conditional on the position of elements in the tree, in terms of attribute values found by traversing the tree in an arbitrary way, etc.

SGML tools vary greatly in the sophistication and generality with which these basic principles are implemented, as might be expected. Some tools will only operate on a single dtd and provide only a simple set of mappings (for example the public domain LinuxDOC system, which generates LaTeX output for documentation of the public domain Linux system); by contrast, commercial products such as Balise or Omnimark will operate on any DTD and provide all the features of a high level programming language. In between, there are several useful public domain utilities which allow the user to define the actions to be performed by the tool using a particular programming language. We have experimented with two such: tf, which uses the string processing language Spitbol; and SGMLSpm, which uses the Unix object-oriented language Perl5.

2 SGML conversions

For the Memoria project, we carried out the following experiments in SGML conversion:

- conversion of non-SGML encoded texts to SGML (import)
- conversion of SGML-encoded texts to HTML (export)
- conversion of SGML-encoded texts from one dtd (TEI Lite) to another (that used by the PLAO prototype)

The following sections discuss various aspects of each experiment, and include examples of the data and procedures used.

2.1 Conversion of non-SGML encoded texts

Any form of encoded text makes explicit a range of textual features, one way or another. Even a so-called plain ASCII text makes explicit such features as headings or paragraph divisions by means of its use of white space. Comnversion of such texts is often problematic because of their reliance on features of the text which are only evident to a human reader, and cannot therefore be automatically performed.

In texts prepared for electronic processing during the seventies, a wide range of features were typically made explicit, generally for specific purposes such as the production of concordances or indices verborum. As an example of this type of material, we worked on a text of Proust's *A la recherche du temps perdu*, in the form in which it was originally deposited with the Oxford Text Archive by the InAlf in 1978. The first few lines of this text in its original form are reproduced below:

```
K428 PROUST ALRDTP DU COTE SWANN
h1$RE PARTIE - COMBRAY (I)
    3   00010001LONGTEMPS, JE ME SUIS COUCH] DE BONNE HEURE. PARFOIS,
    3   00010002[ PEINE MA BOUGIE ]TEINTE, MES YEUX SE FERMAIENT SI
    3   00010003VITE QUE JE N'AVAIS PAS LE TEMPS DE ME DIRE : "JE
    3   00010004M'ENDORS." ET, UNE DEMI-HEURE APR$S, LA PENS]E QU'IL
    3   00010005]TAIT TEMPS DE CHERCHER LE SOMMEIL M']VEILLAIT ; JE
    3   00010006VOULAIS POSER LE VOLUME QUE JE CROYAIS AVOIR ENCORE
    3   00010007DANS LES MAINS ET SOUFFLER MA LUMI$RE ; JE N'AVAIS
    3   00010008PAS CESS] EN DORMANT DE FAIRE DES R]FLEXIONS SUR CE
    3   00020001QUE JE VENAIS DE LIRE, MAIS CES R]FLEXIONS AVAIENT
    3   00020002PRIS UN TOUR UN PEU PARTICULIER ; IL ME SEMBLAIT
    3   00020003QUE J']TAIS MOI-M^ME CE DONT PARLAIT L'OUVRAGE :
    3   00020004UNE ]GLISE, UN QUATUOR, LA RIVALIT] DE *FRAN+OIS IER
    3   00020005ET DE *CHARLES-QUINT. CETTE CROYANCE SURVIVAIT
```

In this representation, we note that the limitations of the available technology (no upper case letters, no accents) have been overcome in a variety of ingenious, but inescapably ad hoc, ways.

Uppercase letters are used to stand for their lower case equivalents, and the * symbol used to indicate uppercase letters where these precede proper names. Accents are indicated by a fairly arbitrary selection of characters not otherwise used (e.g. the left square bracket represents a lower case a-grave; the right square bracket an e-acute, the "greater than" symbol stands for an o-circumflex). Considerable importance is attached to the serialization appliued to each record: this serves both as an important security measure in the days when such files were stored as thousands of physically discrete pieces of cardboard, and as a means of providing a reference for each line of text. A program operating on this text could treat each line independently, since the only contextual information available is repeated in the series of numbers at the start of the line. Finally, we note the complete absence of any more detailed contextual information: the edition of the text used, even the title of the work and its originators, are determinable only by consulting contemporary printed records, or by internal examination.

In converting this text to a TEI Lite representation, therefore, a great deal of additional material had to be obtained from other sources. The header attached to the text is given in full below, partly also in order to demonstrate the level of detailed information possible with the TEI scheme.

```
<teiHeader>
<fileDesc><titleStmt>
   <title>Marcel Proust's A la recherche du temps perdu: TEI Edition</title>
   <author>Proust, Marcel</author>
   <respStmt><resp>Original data captured by</resp>
     <name>Institut de la langue francaise</name>
   </respStmt>
   <respStmt><resp>TEI Conversion by</resp>
     <name>Oxford Text Archive</name>
   </respStmt>
</titleStmt>
<extent>23 files totalling 8.4 Mb </extent>
<publicationStmt>
  <distributor>Oxford text Archive, for Memoria Project</distributor>
  <pubPlace>Oxford</pubPlace><date>1995</date>
  <availability><p>Not available outside the Memoria Project</availability>
  <idno type=OTA>xxx</idno>
</publicationStmt>
<sourceDesc>
<p>The text has been automatically converted from a text originally obtained
on magnetic tape from the InaLF in 1978. The pagination and apparently
the text itself are derived from the 1954 Pleiade edition published by
Gallimard, but excludes the dedications, notes, and variant readings.
</sourceDesc>
</fileDesc>
<encodingDesc>
<p>To be supplied
</encodingDesc>
<revisionDesc>
<change><date>17 Oct 95</date>
  <respStmt><name>Glynis Baguley</name><resp>ed.</resp></respStmt>
```

```
  <item>Automatic conversion completed</item>
</change>
<change><date>23 Oct 95</date>
  <respStmt><name>Lou Burnard</name><resp>ed.</resp></respStmt>
  <item>Created TEI Header and steer file</item>
</change>
</revisionDesc>
</teiHeader>
```

For completeness, we show below the start of the text itself in its converted form. Note how referencing information is now implicit in the SGML structure, permitting an SGML aware processor now to infer more about the context in which words appear than would be possible with the "fixed field" approach of the original. Conversion to this format was carried out automatically using a special purpose program: it is worth noting that no information was required to produce this level of encoding additional to that made explicit in the original mark up.

```
<head>gK428 PROUST ALRDTP DU COTE SWANN</head>
<div2 type='chapter'>
<head>1&egrave;re partie &mdash; combray (i)</head>
<pb n=3>
<p>Longtemps, je me suis couch&eacute; de bonne heure. Parfois,
&agrave; peine ma bougie &eacute;teinte, mes yeux se fermaient si
vite que je n'avais pas le temps de me dire : &quot;Je
m'endors.&quot; Et, une demi-heure apr&egrave;s, la pens&eacute;e qu'il
&eacute;tait temps de chercher le sommeil m'&eacute;veillait ; je
voulais poser le volume que je croyais avoir encore
dans les mains et souffler ma lumi&egrave;re ; je n'avais
pas cess&eacute; en dormant de faire des r&eacute;flexions sur ce
que je venais de lire, mais ces r&eacute;flexions avaient
pris un tour un peu particulier ; il me semblait
que j'&eacute;tais moi-m&ecirc;me ce dont parlait l'ouvrage :
une &eacute;glise, un quatuor, la rivalit&eacute; de Fran&ccedil;ois ier
et de Charles-quint. Cette croyance survivait
...
```

As a further example, we carried out a small experiment on a sample of material in a very different non-SGML format, supplied by the Istituto di Linguistuica Compuzionale di Pisa. This format is that used at the ILC for a wide variety of its textual materials, and was designed to facilitate automatic lexico-morphological analyses of various kinds. Only a few details of the format are given here, to indicate the kind of structure likely to be encountered when dealing with such materials.

The file converted was a sample provided by Dr Andrea Bozzi. Its first few lines are as follows:

```
0002015         GLK#001'04&B
R002016         \
 002017         IIII
 002018         de
```

```
 002019         syllabis/
R002020         \\
R002021         R
 002022         syllaba
 002023         est
 002024         littera
 002025         uocalis
 002026         aut
 002027         litterarum
 002028         coitus
 002029         per
R002030         R
 002031         aliquam
 002032         uocalem
 002033         conprehensus
1002034        ’
 002035         syllabae
 002036         dicuntur
 002037         a
R002038         R
 002039         Graecis
D002040         EH
 002041  paraf
 002042  tof
 002043  sullambanein
 002044  taf
 002045  grammata
D002046        KS
1002047        ,
 002048         Latine
R002049         R
 002050         conexiones
 002051         uel
 002052         conceptiones
1002053        ,
 002054         quod
 002055         litteras
 002056         concipiunt
R002057         R
 002058         atque
 002059         conectunt
1002060        ;
 002061         uel
 002062         mprehensio
1002063        ,
 002064         hoc
 002065         est
 002066         litterarum
R002067         R
 002068         iuncta
```

```
 002069          enuntiatio
1002070         '
 002071          syllabae
 002072          aut
 002073          breues
 002074          sunt
 002075          aut
 002076          longae
1002077         '
R002078          R
 002079          breues
 002080          correpta
 002081          uocalis
 002082          efficit
1002083         ,
 002084          aut
 002085          cum
 002086          antecedente
R002087          R
 002088          consonante
 002089          uocalis
 002090          in
 002091          fine
 002092          syllabae
 002093          corripitur
1002094         ;
 002095          longas
R002096          R
 002097          producta
 002098          uocalis
 002099          facit
1002100         '
```

In this "vertical" format, each word of the file is given a record of its own. Each record of the file contains a prefix, indicating its type, a record number, and other data. Each word may be given twice, both in its textual form, and as a lemma. The lemmatized forms were ignored by our conversion, but could have been integrated using the `<reg>` element defined in TEI Lite. Words are always right filled with spaces, and delimited by a space and a forward slash character. As with the French text above, a number of non-standard character mappings must be translated within the text.

The first character of each record specifies its type, as follows:

[blank] A text record, containing one word (or other token) of the text.

D Toggle the text type, e.g. in Latin sample, switch to or from Greek, depending on content of text field.

P New page.

R End of record in source. These also frame headings.

0 Beginning of a new structural division of the text.

1 Non-lexical item. Used for punctuation and (in this sample)for illustrative phrases to be printed in bold type.

Here is the start of the same text, as output by the conversion program:

```
<text>
<body>
<div1 n=001.04>
<div2>
<head>IIII de syllabis</head>
<p>
syllaba est littera uocalis aut litterarum coitus per
aliquam uocalem conprehensus. syllabae dicuntur a
Graecis <foreign lang=greek>paraf tof sullambanein
taf grammata</foreign>, Latine
conexiones uel conceptiones, quod litteras concipiunt
atque conectunt; uel comprehensio, hoc est litterarum
iuncta enuntiatio. syllabae aut breues sunt aut longae.
breues correpta uocalis efficit, aut cum antecedente
consonante uocalis in fine syllabae corripitur; longas
producta uocalis facit.
...
```

2.2 Conversion of SGML to HTML

HTML, the de facto standard language used for documents distributed via the World Wide Web, is itself a simple application of SGML. Converting SGML documents to HTML however is not generally a simple matter of converting one tag into another, because the requirements of easy legibility and efficient hyperlinking frequently necessitate a reorganization of the source document into a series of discrete fragments, with an associated index.

In converting a range of technical documentation for publication on the World Wide Web therefore, we chose to create a suite of special purpose perl programs which operate together in sequence, using as input the ESIS output from an SGML parser. The action of these programs may be summarized as follows

- generate an index file and a separate file of footnotes
- translate cross references in TEI Lite format into HTML-style links
- translate TEI Lite tags into the corresponding HTML formatting tags
- split the resulting single HTML file into a series of hyper-linked HTML files

These functions could, of course, all be integrated into a single production system, but have been kept separate for ease of development. An integrated version, using the SGMLSpm package referred to above, is currently in preparation.

2.2.1 Creation of index and notes files

This simple program illustrates the ease with which the ESIS output referred to above may be processed. The program looks only at the structural elements marking subdivisions within the text, and calculates a hierarchical numbering system for them, which is written out to the index file together with the associated `<head>` element, which gives each subdivision's title. It also seeks out any `<note>` elements, numbers them, and writes them out to a file.

The full text of the program follows:

```
# divgen.prl
# make HTML toc and notes file from SGML doc
#  run with [n]sgmls [sgmlfile] | perl divgen.prl
#  creates files notes.htm and toc.htm
# LB 27 jun 94, revised 2 sept 95

@ref = (0,0,0,0,0,0);  # array to hold reference numbers

open(NOTES, ">notes.htm")
  || die "Couldnt open notes.htm $!\n";
open(TOC, ">index.htm")
  || die "Couldnt open toc.htm $!\n";
print TOC
  "<html><head><title>\nGenerated by DIVGEN.PRL</title></head><body>\n";
print TOC "<h1>Title</h1>\n";
print TOC
  "<hr><i>This HTML version was derived automagically from
    <a href=http:??> the version prepared in TEI Lite format</a> </i><hr>\n";

print TOC "<h2>Table of contents</h2><a name=TOC>\n<ul>\n";
print NOTES "<h1>Notes</h1>\n";

while (<>){
chop;
if    (/^AID\sTOKEN\s(.+)/) {$id = $1;}     # save any ID value
elsif (/^\(DIV(\d)/)                         # at start of any divn
      { $idval = $id;
        $divlev=$1-1;           # set level
        if ($divlev >= 0)
          { $ref[$divlev]++;        #calculate new number at this level
            $ref[$divlev+1]=0;      #initialize next higher level
            if ($divlev > $last_lev )
               { print TOC "<ul>\n"; }
            elsif ($divlev < $last_lev)
               { for ($i = $divlev; $i < $last_lev; $i++)
                 { print TOC "</ul>\n"; }}
          }
     $last_lev = $divlev;
#     print "lastlev=$last_lev, divlev=$divlev\n";
      }
```

```
elsif (/^\(HEAD/) {$heading = 1; } # set flag at start of head
elsif (/^\(NOTE/) {$in_note = 1; } # set flag at start of note
elsif (/^\)HEAD/) {$heading = 0;
    $head =~ s#\\n# #g;        # turn fake newlines into spaces
    $str = @ref[0];
    foreach $i (@ref[1..5]) {             # print the reference
        if ($i > 0) { $str .= ".$i";}
         else {last;}}
      if ($divlev == 0){
  $bf = $idval . ".htm";
          print TOC "<li><a href=$bf>$str $head</a></li>\n";  }
      else {
      print TOC "<li><a href=$bf#$idval>$str $head</a></li>\n"; }
      $head=""; }
elsif (/^\)NOTE/) {$in_note = 0;
     $note =~ s#\\n# #g;
     $note_no ++;
     print NOTES "<p><a name=NOTE$note_no>[ $note_no ] $note \n";
     $note =""; }
elsif (/^-(.+)/) {
    if ($heading)          # in content and flag set
       { $head .= $1;}
    elsif ($in_note)
       { $note .= $1;}
   }
}

print TOC "</ul>\n";
```

2.2.2 Translation of cross references

Cross references in the TEI scheme may take a variety of forms. In HTML however, only one form is permitted, involving the explicit statement of a URL ("Universal Resource Locator"). Whereas in the TEI scheme internal cross references are handled using the standard ID/IDREF mechanism of SGML alone, in HTML it is desirable to include within the body of the cross reference some descriptive text such as the title of the section being referred to. This is handled in the following simple program, which collects the additional information needed from the TOC file written in the preceding step.

```
# ptr to ref
# constructs ID to section number and name map from toc.html
# turns all <ptr> elements to <ref>s
# CAUTION: loops if <ptr> elements are not just right
#          iinput must be fully normalised!

open(TOC, "index.htm") ||  die "Cant find a TOC file: $!\n";

while (<TOC>) {
  if (/<a href=([^>]+)>([^<]+)/) {
```

```
    $r ++;
   $id_tab{$1} = $2; }
}

while (<>) {
  $safety=0;
  while (/<PTR/)  {
  $safety ++;
  s/<PTR[^T]+TARGET="([^"]+)">/zxw/;
  $id = $1;
  $id =~ tr/[a-z]/[A-Z]/;
  $ref = $id_tab{$id};
  s/zxw/<ref target=$id>$ref<\/ref>/;
  if ($safety > 10) { die "Stuck in a rut at $_\n"; } }
  print;
}
```

2.2.3 Translation of SGML tags

The bulk of the conversion of the TEI Lite document is carried out using the tf program. TF is a transducing filter designed for use with the public domain sgml parser sgmls. It operates on the minimal SGML document output by that utility to produce a new version in which user-specified transductions have been carried out. The program builds a representation of the document being parsed as it processes it, so that transduction can take account of the existing SGML structure.

The current version of the program is written in Macro Spitbol for the Catspaw Sparc or 386 compilers, but can easily be ported to other environments. One particular benefit of using Spitbol is that arbitrary Spitbol expressions may be use to specify the transduction required, which gives the program much of its expressive power

The transduction model is very simple. A pair of string-valued expressions is defined for each element type, one to be evaluated when the start of such elements is found in a document, and one to be evaluated at its end. In addition, element content may be suppressed or (by default) retained in the output stream. Context-sensitive transduction depends on the fact that the start- and end-tag replacement expressions are evaluated afresh each time an element occurrence is found. Any legal Spitbol expression may be used (the string is passed to the built-in Spitbol function EVAL), which means that statements can be quite complex, including for example conditionals and assignments: some examples are given below. Expressions may also include program-defined functions and constants, specifically those relating to the SGML document tree which is built during document processing. When element content is suppressed, previously defined replacements for any elements contained by the suppressed element are also suppressed.

The TF program requires as input a file referred to below as the dictionary file. This is itself a simple SGML document, conforming to the following dtd:

```
<!ELEMENT tfmap - o (map+)>
<!ELEMENT map - o (start?,end?) >
<!ATTLIST map gi name #required
          skip (y|n) n >
<!ELEMENT (start|end) - o (#pcdata)>
```

A MAP specifies the transduction to be carried out on each occurrence of a given element. The element concerned is specified by the GI attribute. If the START element is not null, it is evaluated as a Spitbol expression, and the result passed through the filter when each start-tag for the element concerned is found in the document. If the SKIP attribute has the value Y, then the content of the element is suppressed, otherwise, its content is passed through the filter. Finally, if the END element is not null, it is evaluated, and the result passed through when the end of the element concerned is found.

The expressions stored in START and END elements may contain anything legal in Spitbol, including parenthesized assignments, predicate or other function calls as well as literals. The following functions are provided:

ATT_VAL(NAME, NODE) Returns the value of the attribute NAME on the element identified by NODE, if it exists. If NODE is null, the CURRENT_NODE is used. Example: ATT_VAL('REND') returns the value of the REND attribute on the current element

PARENT(NAME, NODE) Returns the first node found with gi NAME which is an ancestor of NODE. If no NAME is specified, the immediate parent is returned. If NODE is null, the CURRENT_NODE is used. If no node of the correct NAME is found, the call fails. Examples: PARENT('DIV1') returns the parent DIV1 node for the current element, if there is one. PARENT() returns the parent node for the current element. PARENT(,PARENT()) returns the grandparent of the current element.

LEFT(NAME, NODE) Returns the first node found with gi NAME which is a left sibling of NODE (i.e. has the same parent and precedes it in the document). If no NAME is specified, the first node found is returned. If NODE is null, the CURRENT_NODE is used. Example: LEFT('HEAD') returns the last HEAD element found with the same parent as the current node.

RIGHT(NAME, NODE) Returns the first node found with gi NAME which is a right sibling of NODE (i.e. has the same parent and follows it in the document). If no NAME is specified, the first node found is returned. If NODE is null, the CURRENT_NODE is used.

GI(NODE) Returns the GI of the NODE specified.

These functions can be combined as necessary. For example, ATT_VAL('ID',PARENT()) returns the value of the ID attribute on the parent of the current node. GI(PARENT()) returns the GI of the parent of the current node.

An extended version of this program has also been written in the language TCL. The public domain SGMLSpm package written by David Megginson of the University of Ottawa provides a more robust and powerful version of the same basic algorithm.

The dictionary file used for conversion of TEI Lite to HTML is given below: note that not all elements are mapped.

```
<tfmap>
<map gi=ABBR><start>"<sc>"<end>"</sc>"
<map gi=BODY><start>?(rf = table())  ?(pf = cf = 'xxxx.htm') ?(dp = "XX")
  <end>    "<hr><a href=index.html#toc.htm>Back to table of contents</a><br>"
```

```
    "<a href="  pf ">Back to previous section</a><br>"
<map gi=CODE><start>"<TT>" <end>"</TT>"
<map gi=DIV><start>?(dl = dl + 1)
(differ(att_val("ID")) "<A NAME=" att_val("ID") ">", "")
"<h1>" att_val("TYPE") "</h1>"
<end>?(dl = dl - 1)
<map gi=DIV1><start>?(rf<(dl = 1)> = rf<1> + 1)
    "<hr><a href=index.html#toc.htm>Back to table of contents</a><br>"
   "<a href=" (nf = (ident(att_val("ID")) dp (dn = dn + 1),
          att_val("ID")) ".htm")  ">On to next section</a><br>"
    "<a href="  pf ">Back to previous section</a><br>" r_start
<end>?(dl = 0) ?(rf<2> = 0) ?(cf = nf) r_start
<map gi=DIV2><start>?(rf<(dl = 2)> = rf<2> + 1) (differ(att_val("ID"))
    "<A NAME=" att_val("ID") ">", "")
<end>?(dl = 1) ?(rf<3> = 0)
<map gi=DIV3><start>?(rf<(dl = 3)> = rf<3> + 1) (differ(att_val("ID"))
              "<A NAME=" att_val("ID") ">", "")
<end>?(dl = 2) ?(rf<4> = 0)
<map gi=DIV4><start>?(rf<(dl = 4)> = rf<4> + 1) (differ(att_val("ID"))
               "<A NAME=" att_val("ID") ">", "")
<end>?(dl = 3)
<map gi=DOCAUTHOR><start>"<p>"
<map gi=DOCIMPRINT><start>"<p>"
<map gi=DOCDATE><start>"<p>"
<map gi=DOCTITLE><start>(leq(att_val("TYPE"), "main") "<h1>", "<h2>")
 <end>(leq(att_val("TYPE"), "main") "</h1>", "</h2>")
<map gi=EG><start>"<PRE>" r_start  <end>r_start "</PRE>" r_start
<map gi=EMPH><start>"<B>"<end>"</B>"
<map gi=FOREIGN><start>"<I>" <end>"</I>"
<map gi=GI><start>"<B>&lt;" <end>"&gt;</B>"
<map gi=HEAD><start>
  ( eq(dl,1) ?(pf = cf) "<title>" att_val("ID",parent()) "</title>" r_start,  '')
    r_start "<H" dl ">" rf<1>
   (gt(rf<2>,0) "." rf<2>, "  ")
   (gt(rf<3>,0) "." rf<3>, "  ")
   (gt(rf<4>,0) "." rf<4> "  ", "  ") <end>"</H" dl ">" r_start
<map gi=HEADER skip=Y><start>
<map gi=HI><start>"<" (ident(att_val("REND")) "I", att_val("REND")) ">"
           <end> "</" (ident(att_val("REND")) "I", att_val("REND")) ">"
<map gi=IDENT><start>"<TT>" <end>"</TT>"
<map gi=ITEM><start>
(lne(att_val("TYPE",parent()), "gloss") r_start "<LI>", "")
<end>"</LI>" r_start
<map gi=LABEL><start>r_start "<LI><B>" <end>"</B>    "
<map gi=LIST><start>r_start "<UL>"
        <end> "</UL>"
<map gi=LB><start>r_start<end>"<BR>"
<map gi=MENTIONED><start>"`"<end>"'"
<map gi=NAME><start>"<B>"<end>"</B>"
<map gi=NOTE skip=Y><start>"<a href=notes.htm#NOTE" (nn = nn + 1) "> [See note "
```

```
<end>nn "]</a>"
<map gi=P><start>r_start<end>"<P>"
<map gi=PTR><start>'<A HREF="' att_val("TARGET") '"> ??? </A>'
<map gi=REF><start>'<A HREF="'
          substr(att_val("TARGET"),1,2)  ".htm" (gt(size(att_val("TARGET")),2)
         "#" att_val("TARGET"), "")  '">' <end>'</A>'
<map gi=Q><start>"``" <end>"''"
<map gi=SOCALLED><start>"`"<end>"'"
<map gi=TEIHEADER skip=Y><start>
<map gi=TERM><start> "<I>"<end>"</I>"
<map gi=TITLE><start>"<I>"<end>"</I>"
<map gi=XREF><start>
  (leq(att_val("TYPE"),"URL") "<A HREF=" att_val("FROM") ">","")
<end>
  (leq(att_val("TYPE"),"URL") "</A>","")
</tfmap>
```

2.2.4 Splitting and tidying the output file

The output from the previous stage in the process is a single large HTML document, containing the whole input document. The final step is then simply to split it up into a series of files, named for convenience by their original SGML identifiers. In addition, any SGML tags other than the HTML markup must be represented by entity reference, since HTML browsers will otherwise suppress them. These functions are carried out by the following simple perl program:

```
# splitter.prl
# splits output from tf\html.dic into one file per div1
# files are named using the ID of the DIV1
# and titled according to the TOC.htm generated by divgen.prl
#
open(TOC, "index.htm") ||  die "Cant find a TOC file: $!\n";

while (<TOC>) {
  if (/<a href=([^>]+)>[\s\d]*([^<]+)/) {
#    print "$1 = $2\n";
    $r ++;
   $id_tab{$1} = $2; }
}
print "Loaded $r IDs...\n";

open(out, ">foo.htm") || die "Cannot open foo.htm";

while (<>) {
# use <title> lines to make up a heading for the new file
if (/^\<title>([^\<]+)/) {
  $tit = "$1.htm";
  print OUT "</body></html>\n";
  close out;
  open(OUT, ">$tit") || die "Cannot open $tit\n";
```

```
  $tit = $id_tab{$tit};
  print OUT "<html><head><title>$tit</title></head><body>\n";
  next;
  }
# sanitize any tags inside <pre> elements

elsif (/\<PRE/) { $protect = 1; }
elsif (/\<\/PRE/) { $protect = 0;}
elsif ($protect) {
  s/\</\&lt;/g; }
print OUT;
}
print OUT "</body></html>\n";
```

Example technical documents processed using this procedure may be found at the following URLs:

- http://info.ox.ac.uk/ archive/teilite/
- http://info.ox.ac.uk/ archive/teij13/
- http://info.ox.ac.uk/ archive/teip3sg/

2.3 Conversion of TEI Lite to PLAO format

In this section, we describe an Import/Export module enabling the PLAO (Poste de Lecture Assistee par Ordinateur) of the BNF to operate on texts conforming to the Memoria DTD.

Deliverables 8 and 9 defined respectively a MEMORIA DTD (derived from the TEI Lite DTD elements matching requirements identified in Deliverable 1) and a corpus of texts encoded following this DTD. For deliverable 10, the WP3.2 team decided that the added value of the proposed MEMORIA encoding scheme could most effectively be evaluated by loading a document using it into the existing PLAO prototype.

We therefore built a translation module to convert from MEMORIA encoding into PLAO encoding. This was done with Balise2, a software tool published by AIS - Berger-Levrault. Using this tool, we constructed a program called "TEI_to_PLAO" which is described below.

2.3.1 The TEI_to_PLAO program

Input for the program consists of just three file names:

- the name of the MEMORIA DTD
- the name of the MEMORIA file to be converted
- the name of the PLAO file to be generated (possibly including a path)

The command to launch the translator is thus: `TEI_to_PLAO DTDname MEMORIAFilename PLAOFilename`

Output from the program consists of:

- a file which can be imported into the PLAO, named PLAOFilename
- an updated "types" file in the "plao" user environment (/usr/local/BDF/runtime/users/plao/types) and a "types.bak" backup file in the same directory

The actions of the program may be summarized as follows:

- the translation of the MEMORIAfile into a PLAOfile
- the listing and storing of the existing TYPES in current "plao" ueser's environment
- the creation of a new types file for the "plao" user which includes the new types present in the created PLAO file and not in the current environment

The PLAO currently operates only on documents confirming to a specific and very simple dtd, which is as follows:

```
<!DOCTYPE ioplaoT [
<!ELEMENT ioplaoT - - (entete , corps ) +(m, dz, fz) >
<!ELEMENT entete - - (mot*) -(m, dz, fz) >
<!ATTLIST entete
        nom             CDATA           #REQUIRED
        date-cr         CDATA           #REQUIRED
        date-mo         CDATA           #REQUIRED
        util            CDATA           #REQUIRED
        mc              NUTOKENS        #IMPLIED
>
<!ELEMENT                mot - O EMPTY>
<!ATTLIST mot
        id              NUTOKEN         #REQUIRED
        lib             CDATA           #REQUIRED
>
<!ELEMENT corps - - (#PCDATA) >
<!ELEMENT m - - (#PCDATA)>
<!ATTLIST m
        type            CDATA           #REQUIRED
        comm            CDATA           #IMPLIED
        mc              NUTOKENS        #IMPLIED
        id              NAME            #REQUIRED
>
<!ELEMENT dz - O EMPTY>
<!ATTLIST dz
type            CDATA           #REQUIRED
        comm            CDATA           #IMPLIED
        mc              NUTOKENS        #IMPLIED
        id              NAME            #REQUIRED
>
<!ELEMENT fz - O EMPTY>
```

```
<!ATTLIST fz
        id              ID              #REQUIRED
>
<!ENTITY #DEFAULT SDATA "[UNDEFINED!]">
```

Simply translating the tags of the MEMORIA file into a PLAO file corresponding to the above DTD is not sufficient. As a prototype system, the PLAO has some weaknesses. One of them is that it is not able to import zone types that were not defined previously in the user environment. Any unknown imported zone is discarded by the import module. The first thing the translator must do is therefore to define any new zone types in the user's "types environment file". This is done in two stages:

- first the "listfile" Balise program parses the user's "types" file and outputs the maximum number of zones as well as the set containing the zones currently available in PLAO environment.
- secondly, by comparing this list with a list of the zones defined in the MEMORIA file, the "proctype" program creates a new "types" file which includes every zone. The former "types" file is backed up as "types.bak".

After this initial process, importing the new document into the PLAO is simply a matter of translating the tags according to a defined mapping. An extra command in the TEI_to_PLAO script is necessary to ignore any blank line in the created "types" file. It is absolutely necessary for the PLAO will not understand any file containing blank lines. In this case, it will simply refuse to start.

Once the "types" file has been changed, the user just has to enter the PLAO, and import the converted MEMORIA document using the "Document/import" command.

The first time the user opens the new document in the PLAO, any new zones will not appear on screen, as no appearance or formatting has been specified for them by TEI_to_PLAO. To change this appearance, the user must use the the "liste des types" menu and specify a format to be used for each new zone.

2.3.2 The TEI_to_PLAO script

The script used to run the program is as follows:

```
balise -src main
balise -src trad $1 $2
balise -src listtype types.dtd BDF/runtime/users/plao/types
balise -src proctype types.dtd BDF/runtime/users/plao/types
ex BDF/runtime/users/plao/types <<eof
g/^\$/d
w
q
eof
mv PLAOFILE $3
rm TMP
```

2.3.3 Balise source code used by TEI_to_PLAO

The main program

```
// The aim of this program is only to output a "welcome" Message to the
// standard output.
// Treatments follow in "TEI\_to\_PLAO" script

before {
  cout << "\n";
  cout << "*********************************************************\n";
  cout << "*                MEMORIA Translator                     *\n";
  cout << "*                                                       *\n";
  cout << "*                                                       *\n";
  cout << "*          MEMORIA Files  ------> PLAO Files            *\n";
  cout << "*                                                       *\n";
  cout << "*********************************************************\n\n\n";

  cout << "Starting translation\n";
}
```

The trad program

```
// TRAD
// Balise2 Program

// Author: Gwendal Auffret
//         AIS BERGER-LEVRAULT

// This program takes as input a MEMORIA File and creates the corresponding
// PLAO File (the temp name of which is PLAOFILE).
// The program consist essentially in establishing a correspondance between
// MEMORIA and PLAO elements.

// VARIABLES

// CorrespMap         : Maps the correspondances between MEMORIA elements and
//                      some currently existing Zone or Markers in "plao"
//                      user environment. The idea is to avoid creating new
//                      zone types if these exist with another name. If someone
// wants to avoid this mapping, he/she, just has to empty
// the map.
// ComptMap       : This Map stores during runtime the couple ZoneName/
// IDName. This is used for recovering the ID of each zone
// when the closing element appears (FZ).
// As in MEMORIA DTD, it is not possible for two equivalen
// elements to overlap, the program has just to store the
// Name/ID couples. When the end of zone appears, it just
// has to look for the Name Key, get the ID and remove the
//  couple.
// MonEntityMap       : Customized Text Map ( containing quot)
```

```
// MonNewMap       : Customized copy of Latin1Entities (N.B: The PLAO Import
// Module does not deal with entities, character changes
// have to be done before importation)
// ComptZ       : Zone counter
// ComptM       : Marker counter
// ZoneSet        : Set storing the zones contained in the MEMORIA
// file. This is stored in TMP file and
// processed afterward by the "proctype" program
// MarqSet       :  Set storing the Markers contained in the MEMORIA
//                      file. This is stored in TMP file and
//                      processed afterward by the "proctype" program
// OutStream       : PLAOFILE Created from the current MEMORIA file
// TempStream         : TMP file used to store data for further Balise
// processings
// Today       : Today's date


var CorrespMap = Map ();

var ComptMap = Map ();
var MonEntityMap = Map ();
var MonNewMap = Map ();

var ComptZ = 0;
var ComptM = 0;

var ZoneSet = Set ();
var MarqSet = Set ();

var OutStream = FileStream("PLAOFILE","w");
var TmpStream = FileStream("TMP", "w");

var Today = shell("date").replace(0,"\n","");


// FUNCTIONS
// debut_zone      : Takes in Input the MEMORIE element Elt (string) and
//        stores in PLAOFILE the corresponding DZ (Debut de Zone)
//        element
// fin_zone       : When the program meets a "end of element" event, it
//        stores in PLAOFILE the corresponding FZ (Fin de Zone)
//        tag.

function debut_zone(Elt) {
   ComptZ = ComptZ + 1;
   ComptMap << Map(Elt,ComptZ);
   if CorrespMap.knows(Elt) then {
if not ZoneSet.knows(CorrespMap[Elt]) then ZoneSet << CorrespMap[Elt];
OutStream << format("<DZ TYPE = \"ULD/%1\" ID = \"Z%2\">", CorrespMap[Elt], String(ComptZ));
}
```

```
       else {
if not ZoneSet.knows(Elt) then ZoneSet << Elt;
OutStream << format("<DZ TYPE = \"ULD/%1\" ID = \"Z%2\">", Elt,String(ComptZ));
}
}


function fin_zone(Elt) {
OutStream << format("<FZ ID = \"Z%1\">", String(ComptMap[Elt]));
ComptMap.remove(Elt);
}


// PROGRAMME


before {
 MonEntityMap = Map("quot", 34);
 MonNewMap = Latin1Entities + MonEntityMap;
 installTextMap(MonNewMap);
}



element "tei.2" {
    on start {
OutStream << "<IOPLAOT>\n";
    }
    on end {
OutStream << "</IOPLAOT>";
    }
}


element teiheader {
    on start {
OutStream << "<ENTETE \n";
    }
    on end {
OutStream << "</ENTETE>\n";
    }
}



element text {
   on start {
      OutStream << "<CORPS>\n";
   }
   on end {
```

```
      OutStream << "</CORPS>\n";
   }
}


element (div|div0|div1|div2|div3|div4|div5|div6|div7|div8|div9|div10) {
   on start {
      if [attr TYPE] then
debut_zone(attr["TYPE"]);
      else
debut_zone(elemName());
   }
   on end {
      if [attr TYPE] then
fin_zone(attr["TYPE"]);
      else
fin_zone(elemName());
   }
}


default
   when [within text]
    on start {
       debut_zone(elemName());
    }
    on end {
       fin_zone(elemName());
    }
end


// RULES FOR MARKERS
element PB {
    on start {
       if not MarqSet.knows("Page") then MarqSet << "Page";
       ComptM=ComptM+1;
       OutStream << format("<M TYPE = \"Jalon/Page\" ID = \"M%1\"></M>",String(ComptM));
    }
}


element LB {
    on start {
       if not MarqSet.knows("ligne") then MarqSet << "ligne";
       ComptM=ComptM+1;
       OutStream << format("<M TYPE = \"Jalon/Ligne\" ID = \"M%1\"></M>",String
(ComptM));
    }
```

```
}



// CONTENT RULE

content TEXT
    when [within title] {
       OutStream << format("\tNOM = \"%1\"\n",echo);
       OutStream << format("\tDATE-CR = \"%1\"\n",Today);
       OutStream << format("\tDATE-MO = \"%1\"\n",Today);
       OutStream << format("\tUTIL = \"plao\"\n");
       OutStream << ">\n";
       }

    when [within text] {
  OutStream << echo;
  }

end



content SDATA {
      OutStream << echo.replace(0,"&mdash;", "-");
}



after {
  TmpStream << String(ZoneSet) + "\n";
  TmpStream << String(MarqSet) + "\n";
  close(TmpStream);

  close(OutStream);

  cout << "Translation OK\n";
}
```

The listtype program

```
// ListType
// Balise2 Program

// Author: Gwendal Auffret
//         AIS - Berger-Levrault

// This program takes as input the "types" file of the "plao" user
```

```
// and its DTD. It parses the file and stores into TMP file:
// - The current Max number of Zones in PLAO env
// - The current Max Number of Marqueurs in PLAO env
// - The current set of Zone types in PLAO env.
// - The current set of Marqueurs in PLAO env.
// These data are then processed in proctype for creating the updated "types"
// file.

// VARIABLES
var TempFile = FileStream("TMP", "u");
var ZoneUtilSet = Set ();
var MarqUtilSet = Set ();


element TZ {
 on start {
  if [attr NATURE] then
    if attr["NATURE"] == "ULD" then
      if [attr LIBELLE] then
        if not ZoneUtilSet.knows(attr["LIBELLE"])
        then ZoneUtilSet << attr["LIBELLE"];
  }
}

element TM {
 on start {
  if [attr NATURE] then
    if attr["NATURE"] == "Jalon" then
      if [attr LIBELLE] then
       if not MarqUtilSet.knows(attr["LIBELLE"])
       then MarqUtilSet << attr["LIBELLE"];
  }
}


element LTZ {
    on start {
      if [attr IDMAX] then TempFile << attr["IDMAX"] +"\n";
    }
}

element LTM {
    on start {
      if [attr IDMAX] then TempFile << attr["IDMAX"] +"\n";
 }
}

after {
   TempFile << String(ZoneUtilSet) + "\n";
   TempFile << String(MarqUtilSet) + "\n";
```

```
   close(TempFile);
   }

</div4>
<div4><head>The proctype program</head>
<p><eg><![ CDATA [
// Proctype (process types file)
// Balise2 program

// Author: Gwendal Auffret
//         AIS -  BERGER-LEVRAULT

// This program takes the "plao" user "types" environment file and process
// it. This means that it creates a new "types" file containing the former
// defined Zones and Markers and the new zones and markers defined in the
// MEMORIA file to be imported.


// VARIABLES

// ZoneUtilSet       : ULD types currently available in "plao" environnement.
// ZoneFichSet       : ULD types defined in MEMORIA file
// MUtilSet          : Marker types currently available in "plao" environnement.
// MFichSet          : Marker types defined in MEMORIA file
// ZoneDiffSet       : Set of ULDtypes defined in MEMORIAfile but NOT in "plao"
//                     environment
// MDiffSet          : Set of Marker types defined in MEMORIA file but NOT
//                     in "plao" environment
// AncZoneIDMax      : Ancient Zone Max ID
// AncMIDMax         : Ancient Marker Max ID
// NovZoneIDMax      : New Zone Max ID
// NovMIDMax         : New Marker Max ID
// TempIDMax         : Temp Variable used during processing New Zones and
//                     Markers
// TempKey           : Temp Variable used during processing New Zones and
//                     Markers
// outStream         : Output file pointing on "types.tmp" were the program
//                     writes the new Types file
// TempFile          : "TMP" file in which the translator had stored
//                      - AncZoneIDMax
//                      - AncMIDMax
//                      - ZoneUtilSet
//                      - MUtilFichSet
//                      - ZoneFichSet
//                      - MFichSet


var ZoneUtilSet = Set ();
var ZoneFichSet = Set ();
var MUtilSet = Set();
```

```
var MFichSet = Set();
var ZoneDiffSet = Set ();
var MDiffSet = Set ();

var AncZoneIDMax =0;
var AncMIDMax =0;
var NovZoneIDMax =0;
var NovMIDMax =0;
var TempIDMax = 0;

var TempKey ="";

var outStream = FileStream("/usr/local/BDF/runtime/users/plao/types.tmp","w");
var TempFile = FileStream("TMP","r");


// FUNCTIONS

// GET-INFO        : Opens the TempFile and reads global variables
// Write_diff_zone : Writes in "types.tmp" file the elements corresponding
//                   to ZoneDiffSet.
// Write_diff_Marq : Writes in "types.tmp" file the elements corresponding
//                   to MDiffSet


function get_info ()
{
  ZoneFichSet  = Object(readLine(TempFile));
  MFichSet     = Object(readLine(TempFile));
  AncZoneIDMax = dec(readLine(TempFile));
  AncMIDMax    = dec(readLine(TempFile));
  ZoneUtilSet  = Object(readLine(TempFile));
  MUtilSet     = Object(readLine(TempFile));

  close(TempFile);

  ZoneDiffSet  = ZoneFichSet - ZoneUtilSet;
  MDiffSet     = MFichSet - MUtilSet;

  NovZoneIDMax = AncZoneIDMax + ZoneDiffSet.length();
  NovMIDMax    = AncMIDMax + MDiffSet.length();
}



function write_diff_zone() {
   TempIDMax = AncZoneIDMax;

   while ZoneDiffSet.length() > 0 do {
     TempKey = ZoneDiffSet.element();
```

```
     TempIDMax = TempIDMax + 1;

     outStream << format("<TZ ID=\"%1\" LIBELLE=\"%2\" NATURE=\"ULD\"
                ESPACE=\"\" ACTIF=\"OUI\">\n", TempIDMax, TempKey);
     outStream << "<PRZ>\n";
     outStream << "<PRZ-T>\n";
     outStream << "<PRZ-AV RC-AV=\"NON\" RC-AP=\"NON\"
                ESP-AV=\"0\" ESP-AP=\"0\" CHAINE=\"\">\n";
     outStream << "<PRT FAMILLE=\"\" GRAISSE=\"\" INCLIN=\"\" SOULIGNE=\"\"
                TAILLE=\"\" VISIBLE=\"NON\" COULEUR=\"\" FOND=\"\">\n";
     outStream << "<\/PRZ-AV>\n";
     outStream << "<PRT FAMILLE=\"\" GRAISSE=\"\" INCLIN=\"\" SOULIGNE=\"\"
                 TAILLE=\"\" VISIBLE=\"OUI\" COULEUR=\"\" FOND=\"\">\n";
     outStream << "<PRZ-AP RC-AV=\"NON\" RC-AP=\"NON\"
                ESP-AV=\"0\" ESP-AP=\"0\" CHAINE=\"\">\n";
     outStream << "<PRT FAMILLE=\"\" GRAISSE=\"\" INCLIN=\"\" SOULIGNE=\"\"
                 TAILLE=\"\" VISIBLE=\"NON\" COULEUR=\"\" FOND=\"\">\n";
     outStream << "<\/PRZ-AP>\n";
     outStream << "<\/PRZ-T>\n";
     outStream << "<PRZ-I COULEUR=\"\" FOND=\"\" TRAIT=\"0\" EPAISS=\"0\">\n";
     outStream << "<\/PRZ>\n";
     outStream << "<\/TZ>\n";

    ZoneDiffSet.remove(TempKey);
    }
}

function write_diff_Marq() {
   TempIDMax = AncMIDMax;

   while MDiffSet.length() > 0 do {
      TempKey = MDiffSet.element();
      TempIDMax = TempIDMax + 1;

      outStream << format("<TM ID=\"%1\" LIBELLE=\"%2\" NATURE=\"Jalon\" ESPACE=\"\" ACTIF=\"OUI\">\n", TempIDMax, TempKey);
      outStream << "<PRM COULEUR=\"Bleu Dodger\" FOND=\"Rosee\" ICONE=\"jalon\" ESP-AV=\"0\" ESP-AP=\"0\" MARGE=\"NON\">\n";
      outStream << "<PRT FAMILLE=\"Courier\" GRAISSE=\"normal\" INCLIN=\"normal\" SOULIGNE=\"non\" TAILLE=\"tres petite\" VISIBLE=\"OUI\" COULEUR=\"noir\" FOND=\"noir\">\n";
       outStream << "<\/TM>\n";

    MDiffSet.remove(TempKey);
    }
}


// PROGRAMME


before {
   get_info();
}
```

```
element LTZ {
   on start {
    outStream << format("<LTZ IDMAX=\"%1\">\n",NovZoneIDMax);
   }
   on end {
    if ZoneDiffSet.length() != 0 then write_diff_zone();
    outStream << echo  + "\n";
    }
}


element LTM {
  on start {
     outStream << format("<LTM IDMAX=\"%1\">\n",NovMIDMax);
     }

  on end {
   if MDiffSet.length() != 0 then write_diff_Marq();
   outStream << echo + "\n";
   }
}

default {
 on start {outStream << echo + "\n";}
 on end {outStream << echo + "\n";}
}


after {

close(outStream);

shell("mv /usr/local/BDF/runtime/users/" + "plao" + "/types "+
      "/usr/local/BDF/runtime/users/" + "plao" + "/types.bak");

shell("mv /usr/local/BDF/runtime/users/" + "plao" + "/types.tmp "+
       "/usr/local/BDF/runtime/users/" + "plao" + "/types");
}
```