ANNEXE V.3

Programme d'interprétation de fichiers et de traduction dans la DTD Corpus

Table des matières de l'Annexe V.3

A. EXEMPLE DE TRAITEMENT

1. Comportement du programme : quelques points particuliers

Signalons en quelques mots des propriétés du programme de passage de HTML à Corpus :

- le programme n'achoppe pas sur une balise fermante sans balise ouvrante correspondante, ou vice versa (bien sûr, y compris hors des cas normaux de balises implicites, prévus par l'option `OMITTAG`) ;
- il rétablit des balises manifestement manquantes, lors de transitions trop abruptes (par exemple, si la page commence par une balise `H1` ou `P`, oubliant les éléments délimitant le document comme `HTML` ou `BODY`) ;
- un élément introduisant une image peut avoir, dans un attribut appelé `ALT`, une valeur textuelle destinée à remplacer l'affichage de l'image si celui-ci ne peut être effectué : le programme recueille ces indications textuelles, au fil de la page ; elles peuvent ainsi être prises en compte pour une caractérisation de la page via son texte ;
- une page vide, munie uniquement d'un titre, est interprétée et traduite comme telle. Ceci est un cas limite, car le titre (au sens HTML) n'est pas ce qui est affiché en haut d'une page, il est au contraire peu visible. Il apparaît par exemple sur l'encadrement de la fenêtre du navigateur, ou comme désignation de la page quand on en enregistre l'adresse par un signet (*bookmark*).

Ces cas ont été prévus dans l'implémentation, et le comportement du programme a été testé et validé notamment sur eux.

2. Une page du site du W3C

a) Commentaires

Cette page est conforme à la DTD HTML 3.2 (validation par un parseur). Le programme serait également capable de traiter des pages avec des irrégularités par rapport à cette DTD, –c'est même sa motivation initiale.

Il est intéressant de voir comment le traitement se comporte sur cette page car elle présente une certaine variété de dispositions classiques de texte : intitulés, parties rédigées, listes, etc.

b) La page, telle qu'elle est visualisée

cf. copies d'écran ci-après.

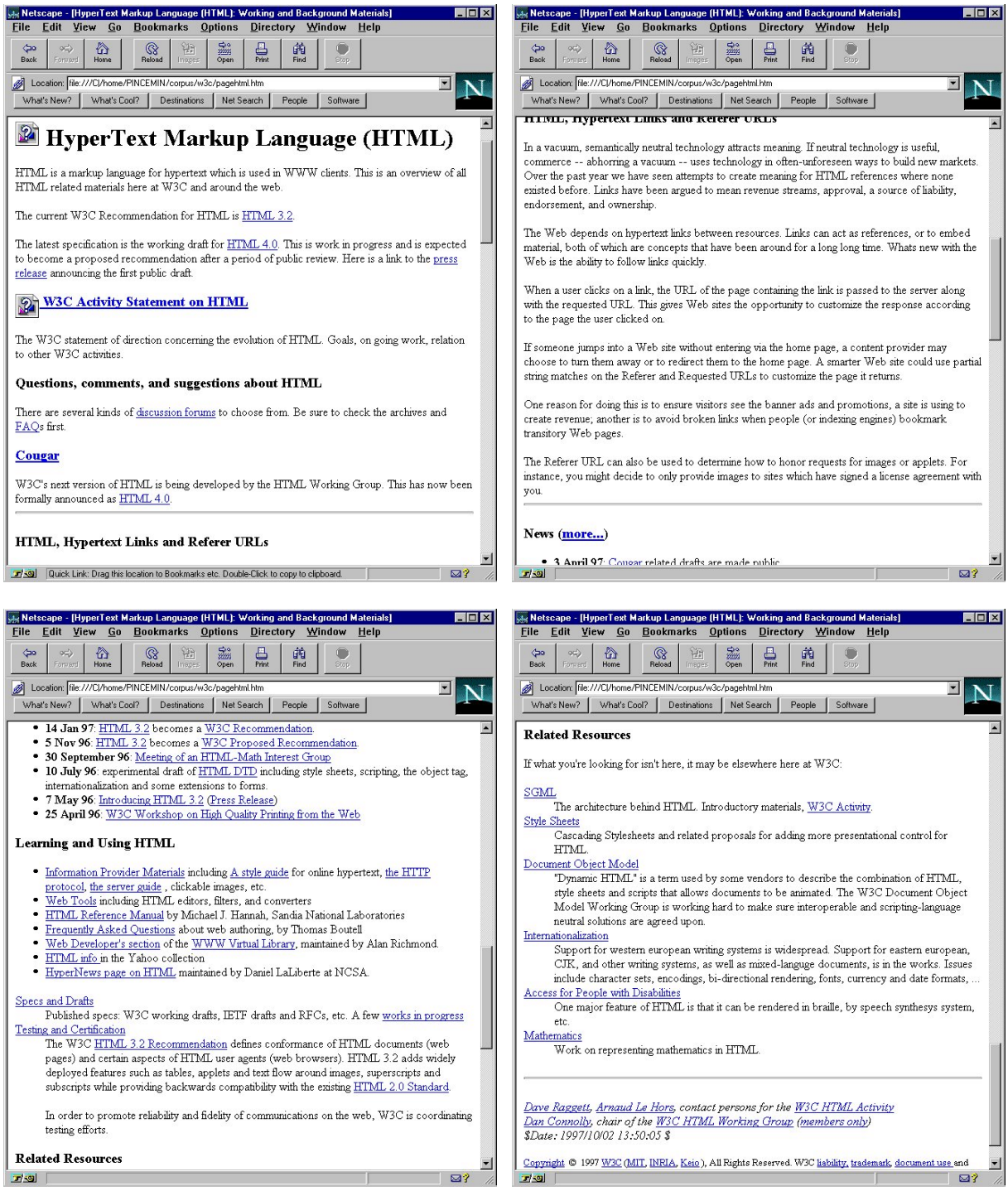


Figure 1 : La page choisie comme exemple, telle qu'elle s'affiche dans un navigateur Web.

c) Le codage HTML de la page

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN" >
 <HTML>
 <HEAD>
   <TITLE>HyperText Markup Language (HTML): Working and Background
Materials</TITLE>
 </HEAD>

<BODY BGCOLOR="#FFFFFF" TEXT="#000000">

 <P>
 <A HREF="../"><IMG BORDER="0" ALT="W3C" SRC="../Icons/WWW/w3c_home"></A>
 <H1>
   <IMG SRC="../Icons/WWW/html_48x48"> HyperText Markup Language (HTML)
 </H1>

 <P>HTML is a markup language for hypertext which is used in WWW
 clients. This is an overview of all HTML related materials here at W3C and
 around the web.


 <p>The current W3C Recommendation for HTML is <a href="Wilbur/">HTML
3.2</A>.

 <p>The latest specification is the working draft  for
 <a href="/TR/WD-html40/">HTML 4.0</a>. This is work in progress and
 is expected to become a proposed recommendation after a period of
 public review. Here is a link to the <a href="/Press/HTML4">press
 release</a> announcing the first public draft.


 <H3><A HREF="Activity"><IMG BORDER="0" SRC="../Icons/WWW/activity_48x48"
 WIDTH="48" HEIGHT="48" align=middle> W3C Activity Statement on
HTML</A></H3>

 <P>The W3C statement of direction concerning the evolution of HTML. Goals,
 on going work, relation to other W3C activities.


 <H3>Questions, comments, and suggestions about HTML</H3>

 <P>There are several kinds of <A HREF="Forums"
NAME="discussion">discussion
 forums</A> to choose from. Be sure to check the archives and <A
 HREF="http://www.boutell.com/faq/#authoring">FAQ</A>s first.

 <H3><a href="Cougar/">Cougar</a></H3>


 <p>W3C's next version of HTML is being developed by the HTML Working
Group.
 This has now been formally announced as <a href="/TR/WD-html40/">HTML
4.0</a>.

 <hr>

 <h3><a name="links">HTML, Hypertext Links and Referer URLs</a></h3>

 <p>In a vacuum, semantically neutral technology attracts meaning.
 If neutral technology is useful, commerce -- abhorring a vacuum --
 uses technology in often-unforeseen ways to build new markets.
 Over the past year we have seen attempts to create meaning for
 HTML references where none existed before. Links have been argued
 to mean revenue streams, approval, a source of liability,
 endorsement, and ownership.

 <p>The Web depends on hypertext links between resources. Links can
 act as references, or to embed material, both of which are concepts
 that have been around for a long long time. Whats new with the Web
 is the ability to follow links quickly.

 <p>When a user clicks on a link, the URL of the page containing
 the link is passed to the server along with the requested URL.
 This gives Web sites the opportunity to customize the response
 according to the page the user clicked on.

 <p>If someone jumps into a Web site without entering via
 the home page, a content provider may choose to turn them
 away or to redirect them to the home page. A smarter Web
 site could use partial string matches on the Referer and
 Requested URLs to customize the page it returns.

 <p>One reason for doing this is to ensure visitors see the
 banner ads and promotions, a site is using to create revenue;
 another is to avoid broken links when people (or indexing
 engines) bookmark transitory Web pages.

 <p>The Referer URL can also be used to determine how to
 honor requests for images or applets. For instance,
 you might decide to only provide images to sites which
 have signed a license agreement with you.
 <hr>

 <H3>News (<A HREF="../News/">more...</A>)</H3>
 <UL>
   <li><strong>3 April 97</strong>: <a href="Cougar/">Cougar</a> related
   drafts are made public.

   <li><strong>14 Jan 97</strong>: <a href="Wilbur/">HTML 3.2</A> becomes a
   <a href="../Consortium/Process/Recommendation">W3C Recommendation</A>.

   <li><strong>5 Nov 96</strong>: <a href="Wilbur/">HTML 3.2</A> becomes a
```

```
   <a href="../Consortium/Process/Recommendation">W3C Proposed
   Recommendation</A>.

   <li><strong>30 September 96</strong>: <a
   href="http://www.ams.org/html-math/meeting-960930-notes.html">Meeting of
   an HTML-Math Interest Group</a>

   <li><strong>10 July 96</strong>: experimental draft of <a
   href="Cougar/HTML.dtd">HTML DTD</A> including style sheets, scripting,
   the object tag, internationalization and some extensions to forms.

   <li><strong>7 May 96</STRONG>: <A href="Wilbur/">Introducing HTML
3.2</A>
   (<A href="Wilbur/pr7may96">Press Release</A>)

   <li><strong>25 April 96</STRONG>: <A
   HREF="../Printing/Workshop_960425">W3C Workshop on High Quality Printing
   from the Web</A>
 </UL>

 <H3>Learning and Using HTML</H3>
 <UL>

   <LI><A href="../Provider/">Information Provider Materials</A> including
   <A HREF="../Provider/Style/"> A style guide</A> for online hypertext, <A
   HREF="../Protocols/"> the HTTP protocol</A>, <A HREF="../Daemon/User/">
   the server guide</A> , clickable images, etc.

   <LI><A href="../Tools/">Web Tools</A> including HTML editors, filters,
   and converters

   <LI><A HREF="http://www.sandia.gov/sci_compute/html_ref.html">HTML
   Reference Manual</A> by Michael J. Hannah, Sandia National Laboratories
   <!-- &lt;mjhanna@sandia.gov&gt; -->

   <LI><A HREF="http://www.boutell.com/faq/#authoring">Frequently Asked
   Questions</A> about web authoring, by Thomas Boutell

   <LI><A href="http://WWW.Stars.com/Vlib/">Web Developer's section</A> of
   the <A HREF="../../DataSources/"> WWW Virtual Library</A>, maintained by
   Alan Richmond.

   <LI><A href="http://www.yahoo.com/Computers/World_Wide_Web/HTML/">HTML
   info </A>in the Yahoo collection

   <LI><A
   href="http://union.ncsa.uiuc.edu/HyperNews/get/www/html.html">HyperNews
   page on HTML</A> maintained by Daniel LaLiberte at NCSA.

 </UL>


 <DL>

   <DT>
     <A NAME="specs" HREF="Bibliography">Specs and Drafts</A>
   <DD>
     Published specs: W3C working drafts, IETF drafts and RFCs, etc. A few
     <A HREF="Bibliography#in-progress">works in progress</A>
   <DT>
     <A href="html-test/">Testing and Certification</A>
   <DD>
     The W3C <A href="../TR/REC-html32.html">HTML 3.2 Recommendation</a>
     defines conformance of HTML documents (web pages) and certain aspects
     of HTML user agents (web browsers). HTML 3.2 adds widely deployed
     features such as tables, applets and text flow around images,
superscripts
     and subscripts while providing backwards compatibility with the
existing
     <A href="html-spec/">HTML 2.0 Standard</a>.

     <P>In order to promote reliability
     and fidelity of communications on the web, W3C is coordinating testing
     efforts.</P>
 </DL>

 <H3><A name="related">Related Resources</A></H3>
 <P>
 If what you're looking for isn't here, it may be elsewhere here at W3C:
 <DL>
   <DT>
     <A href="SGML/">SGML</A>
   <DD>
     The architecture behind HTML. Introductory materials,
     <A HREF="SGML/Activity">W3C Activity</A>.
   <DT>
     <A HREF="../Style/">Style Sheets</A>
   <DD>
     Cascading Stylesheets and related proposals for adding more
     presentational control for HTML.
   <DT>
     <A HREF="DOM/">Document Object Model</A>
   <DD>"Dynamic HTML" is a term used by some vendors to describe the
combination of HTML, style sheets and scripts that allows documents to be
animated.
 The W3C Document Object Model Working Group is working hard to make sure
 interoperable and scripting-language neutral solutions are agreed upon.
   <DT>
     <A href="../International/">Internationalization</A>
   <DD>
     Support for western european writing systems is widespread. Support
for
     eastern european, CJK, and other writing systems, as well as
     mixed-languge documents, is in the works. Issues include character
     sets, encodings, bi-directional rendering, fonts, currency and date
     formats, ...
   <DT>
```

```
    <A HREF="../Disabilities/">Access for People with Disabilities</A>
  <DD>
    One major feature of HTML is that it can be rendered in braille, by
    speech synthesys system, etc.
  <DT>
    <A HREF="Math/">Mathematics</A>
  <DD>
    Work on representing mathematics in HTML.

</DL>
<P>
  <HR>

<ADDRESS>
<p><A href="../People/Raggett/">Dave Raggett</A>, <a
href="../People/Arnaud/">Arnaud Le Hors</a>, contact persons for the <a
href="Activity">W3C HTML Activity</a><br> <a href="../People/Connolly">Dan
Connolly</a>, chair of the <A href="Group/">W3C HTML Working Group</a> (<a
href="../Help/AccessForm">members only</a>)<br>
$Date: 1997/10/02 13:50:05 $
</ADDRESS>

<p>
<small><a href="../Consortium/Legal/ipr-
notice.html#Copyright">Copyright</a>
 &copy; 
1997 <a href="http://www.w3.org">W3C</a> (<a
href="http://www.lcs.mit.edu">MIT</a>,
<a href="http://www.inria.fr/">INRIA</a>, <a
href="http://www.keio.ac.jp/">Keio</a> ),
All Rights Reserved.
W3C <a href="../Consortium/Legal/ipr-notice.html#Legal
Disclaimer">liability,</a>
<a href="../Consortium/Legal/ipr-notice.html#W3C Trademarks">trademark</a>,
<a href="../Consortium/Legal/copyright-documents.html">document use </a>and
<a href="../Consortium/Legal/copyright-software.html">software licensing
</a>rules apply.
Your interactions with this site are in accordance with our
<a href="../Consortium/Legal/privacy-statement.html#Public">public</a> and
<a href="../Consortium/Legal/privacy-statement.html#Members">Member</a>
privacy statements.</small>
</p>


</BODY>
</HTML>
```

d) Traduction obtenue dans la DTD Corpus

(Pour une meilleure lisibilité, a été mis en gras ce qui a été repéré par le programme comme ayant une fonction d'intitulé, et qui a été transcrit par des éléments `HEAD` *et* `TIT` *de la DTD Corpus)*

```
<CORPUS><DOC><TEXT><TIT>HyperText Markup Language (HTML): Working and
Background Materials</TIT>
<NLS> W3C </NLS>
<HEAD> HyperText Markup Language (HTML) </HEAD>
<NLS>HTML is a markup language for hypertext which is used in WWW clients.
This is an overview of all HTML related materials here at W3C and around
the web. </NLS>
<NLS>The current W3C Recommendation for HTML is HTML 3.2. </NLS>
<NLS>The latest specification is the working draft for HTML 4.0. This is
work in progress and is expected to become a proposed recommendation after
a period of public review. Here is a link to the press release announcing
the first public draft. </NLS>
<HEAD> W3C Activity Statement on HTML</HEAD>
<NLS>The W3C statement of direction concerning the evolution of HTML.
Goals, on going work, relation to other W3C activities. </NLS>
<HEAD>Questions, comments, and suggestions about HTML</HEAD>
<NLS>There are several kinds of discussion forums to choose from. Be sure
to check the archives and FAQs first. </NLS>
<HEAD>Cougar</HEAD>
<NLS>W3C's next version of HTML is being developed by the HTML Working
Group. This has now been formally announced as HTML 4.0. </NLS>
<HEAD>HTML, Hypertext Links and Referer URLs</HEAD>
<NLS>In a vacuum, semantically neutral technology attracts meaning. If
neutral technology is useful, commerce -- abhorring a vacuum -- uses
technology in often-unforeseen ways to build new markets. Over the past
year we have seen attempts to create meaning for HTML references where none
existed before. Links have been argued to mean revenue streams, approval, a
source of liability, endorsement, and ownership. </NLS>
<NLS>The Web depends on hypertext links between resources. Links can act as
references, or to embed material, both of which are concepts that have been
around for a long long time. Whats new with the Web is the ability to
follow links quickly. </NLS>
<NLS>When a user clicks on a link, the URL of the page containing the link
is passed to the server along with the requested URL. This gives Web sites
the opportunity to customize the response according to the page the user
clicked on. </NLS>
<NLS>If someone jumps into a Web site without entering via the home page, a
content provider may choose to turn them away or to redirect them to the
home page. A smarter Web site could use partial string matches on the
Referer and Requested URLs to customize the page it returns. </NLS>
<NLS>One reason for doing this is to ensure visitors see the banner ads and
promotions, a site is using to create revenue; another is to avoid broken
```

links when people (or indexing engines) bookmark transitory Web pages.
</NLS>
<NLS>The Referer URL can also be used to determine how to honor requests for images or applets. For instance, you might decide to only provide images to sites which have signed a license agreement with you. </NLS>
<HEAD>News (more...)</HEAD>
<BLCK><ITEM><NLS>3 April 97: Cougar related drafts are made public. </NLS>
</ITEM>
<ITEM><NLS>14 Jan 97: HTML 3.2 becomes a W3C Recommendation. </NLS>
</ITEM>
<ITEM><NLS>5 Nov 96: HTML 3.2 becomes a W3C Proposed Recommendation. </NLS>
</ITEM>
<ITEM><NLS>30 September 96: Meeting of an HTML-Math Interest Group</NLS>
</ITEM>
<ITEM><NLS>10 July 96: experimental draft of HTML DTD including style sheets, scripting, the object tag, internationalization and some extensions to forms. </NLS>
</ITEM>
<ITEM><NLS>7 May 96: Introducing HTML 3.2 (Press Release) </NLS>
</ITEM>
<ITEM><NLS>25 April 96: W3C Workshop on High Quality Printing from the Web</NLS>
</ITEM>
</BLCK>
<HEAD>Learning and Using HTML</HEAD>
<BLCK><ITEM><NLS>Information Provider Materials including A style guide for online hypertext, the HTTP protocol, the server guide , clickable images, etc. </NLS>
</ITEM>
<ITEM><NLS>Web Tools including HTML editors, filters, and converters </NLS>
</ITEM>
<ITEM><NLS>HTML Reference Manual by Michael J. Hannah, Sandia National Laboratories </NLS>
</ITEM>
<ITEM><NLS>Frequently Asked Questions about web authoring, by Thomas Boutell </NLS>
</ITEM>
<ITEM><NLS>Web Developer's section of the WWW Virtual Library, maintained by Alan Richmond. </NLS>
</ITEM>
<ITEM><NLS>HTML info in the Yahoo collection </NLS>
</ITEM>
<ITEM><NLS>HyperNews page on HTML maintained by Daniel LaLiberte at NCSA.
</NLS>
</ITEM>
</BLCK>
<BLCK><ITEM>**<HEAD>Specs and Drafts</HEAD>**
<NLS> Published specs: W3C working drafts, IETF drafts and RFCs, etc. A few works in progress</NLS>
</ITEM>
<ITEM>**<HEAD>Testing and Certification</HEAD>**
<NLS> The W3C HTML 3.2 Recommendation defines conformance of HTML documents (web pages) and certain aspects of HTML user agents (web browsers). HTML 3.2 adds widely deployed features such as tables, applets and text flow around images, superscripts and subscripts while providing backwards compatibility with the existing HTML 2.0 Standard. </NLS>
<NLS>In order to promote reliability and fidelity of communications on the web, W3C is coordinating testing efforts.</NLS>
</ITEM>
</BLCK>
<HEAD>Related Resources</HEAD>
<NLS> If what you're looking for isn't here, it may be elsewhere here at W3C: </NLS>
<BLCK><ITEM>**<HEAD>SGML</HEAD>**
<NLS> The architecture behind HTML. Introductory materials, W3C Activity.
</NLS>
</ITEM>
<ITEM>**<HEAD>Style Sheets</HEAD>**
<NLS> Cascading Stylesheets and related proposals for adding more presentational control for HTML. </NLS>
</ITEM>
<ITEM>**<HEAD>Document Object Model</HEAD>**
<NLS>"Dynamic HTML" is a term used by some vendors to describe the combination of HTML, style sheets and scripts that allows documents to be animated. The W3C Document Object Model Working Group is working hard to make sure interoperable and scripting-language neutral solutions are agreed upon. </NLS>
</ITEM>
<ITEM>**<HEAD>Internationalization</HEAD>**
<NLS> Support for western european writing systems is widespread. Support for eastern european, CJK, and other writing systems, as well as mixed-languge documents, is in the works. Issues include character sets, encodings, bi-directional rendering, fonts, currency and date formats, ...
</NLS>
</ITEM>
<ITEM>**<HEAD>Access for People with Disabilities</HEAD>**
<NLS> One major feature of HTML is that it can be rendered in braille, by speech synthesys system, etc. </NLS>
</ITEM>
<ITEM>**<HEAD>Mathematics</HEAD>**
<NLS> Work on representing mathematics in HTML. </NLS>
</ITEM>
</BLCK>
<NLS>Dave Raggett, Arnaud Le Hors, contact persons for the W3C HTML Activity Dan Connolly, chair of the W3C HTML Working Group (members only)
$Date: 1997/10/02 13:50:05 $ </NLS>
<NLS>Copyright © 1997 W3C (MIT, INRIA, Keio), All Rights Reserved. W3C liability,trademark, document use and software licensing rules apply. Your interactions with this site are in accordance with our public and Member privacy statements.</NLS>
</TEXT>
</DOC>
</CORPUS>

B. ARCHITECTURE DU PROGRAMME ET DESCRIPTIF DES MODULES

Le développement a été fait en Ada, en raison de la puissance et de la solidité de ce langage, et de la disponibilité d'un compilateur fiable, performant, développé et entretenu par les meilleurs spécialistes (le compilateur GNAT). Ada est puissant, car il permet de réaliser de façon élégante tout ce qui a jamais été conçu en termes de programmation : sémantique maîtrisée par un typage fort, réutilisabilité (génériques, héritages, surcharges,...), modularité (dépendance, visibilités...), fiabilité (traitement et remontée d'exceptions, initialisation et finalisation systématiques qui peuvent être utilisées pour les pointeurs), parallélisme, temps réel,... Tous les éléments pour une programmation objet sont là. Ada est un langage de haut niveau, d'une grande lisibilité (facilitant l'entretien et l'évolution des programmes) ; il est clair et précis (pas d'implicite confus). Normé, il est parfaitement portable, et n'est pas dépendant d'une version de compilateur ou d'un choix de plate-forme. Pour en savoir plus sur Ada, on peut lire l'ouvrage très complet de Norman H. COHEN (Cohen 1996), suivre l'excellente formation dispensée par Jean-Pierre ROSEN (société Adalog, Vanves, `http://perso.wanadoo.fr/adalog`), ou/et consulter le *Manuel de référence* d'Ada 95, disponible sous forme hypertexte sur le site général Ada (`http://www.adahome.com/`).

Les suffixes `.ads` désignent des fichiers contenant des spécifications, les `.adb` explicitent la mise en œuvre des traitements (*b* comme *body*, *i.e. corps* du programme).

1. Programmes de traitement

a) De texte simple vers Corpus

Le fichier `asc2dcd.adb` contient le programme qu'il faut lancer pour réaliser la transcription d'un fichier ASCII simple en un fichier selon la DTD Corpus.

b) De texte ASCII mis en forme vers Corpus

Le fichier `csr2dcd.adb` contient le programme qu'il faut lancer pour réaliser la transcription d'un fichier ASCII mis en page (notamment les césures de lignes sont codées par des retours chariot) en un fichier selon la DTD Corpus.

c) De HTML vers Corpus

Le fichier `htm2dcd.adb` contient le programme qu'il faut lancer pour réaliser la transcription d'un fichier ASCII HTML « mou » (c'est-à-dire appliquant de façon approximative les DTD HTML 2.0 ou 3.2) en un fichier selon la DTD Corpus.

Le principe général utilisé pour contourner les blocages liés à une syntaxe rigide (conformité stricte à la grammaire que représente la DTD) est la répartition des balises en niveaux : typiquement, il y a des balises dont la portée est de l'ordre du document (mettons `BODY` en HTML), d'autres dont la portée est de l'ordre d'un découpage en paragraphes ou en parties, d'autres intervenant sur un ou quelques mots. Cette approche est expliquée au chapitre V, §C.3.

2. Les chaînes de caractères

a) Définition générique des chaînes de caractères pour DECID

Les spécifications `decid_strings_g.ads` fixent précisément les opérations dont on souhaite doter (en puissance) tous les objets s'apparentant à des chaînes de caractères (ce seront des 'chaînes de caractères DECID'). Le fichier `decid_strings_g.adb` décrit une manière de les effectuer : le module est opérationnel.

Ce module est un générique (ce que nous signalons par le suffixe `_g`, mnémonique), c'est-à-dire qu'il ne définit aucun objet en lui-même, mais donne un patron, un moule, pour créer

instantanément et de façon unifiée les objets dont on a besoin. En l'occurrence, les trois modules suivants sont des instanciations de ce générique.

b) Ligne

Les lignes d'un fichier sont des chaînes de caractères dont la longueur est de l'ordre d'un paragraphe. Le module `lines.ads` les définit donc comme des 'chaînes de caractères DECID' de grande taille. Les lignes bénéficient de fait de toutes les possibilités de manipulations explicitées dans le générique.

c) Chemin (nom complet d'un fichier)

Le module `paths.ads` correspond aux noms de fichiers, généralement donnés avec leur chemin, à savoir la succession de répertoires emboîtés qui y conduit. Ce sont des chaînes de caractères DECID de longueur moyenne.

d) Identifieur générique (en SGML)

Les *generic identifiers* de la norme SGML (abrégés GI) sont par exemple les noms des entités SGML, ou les noms des balises. Ils s'apparentent à des chaînes de caractères DECID courtes, sauf qu'ils ne sont pas sensibles aux distinctions de casse (majuscules / minuscules) ; ils ne sont par ailleurs susceptibles que de peu de manipulations et transformations. Le module `sgml_gis.ads` (rendu opératoire par `sgml_gis.adb`) détermine tout cela.

3. Les segments

Un segment est une unité, un élément, d'un fichier. L'idée est que cela doive permettre de « lire intelligemment » les fichiers, sans les découper n'importe comment, à l'aveuglette, sur des critères arbitraires comme par exemple la longueur d'une chaîne de caractères ou les retours-chariots (non significatifs en SGML).

a) Eléments généraux de définition

Opérations de nettoyage des fichiers (effacement ou remplacement de caractères de contrôle).

b) Les segments dans les fichiers texte (.txt)

Base de définition des segments

Segment de type data : le contenu d'un alinéa

Segment correspondant au contenu d'un paragraphe (ce peut être le regroupement de plusieurs lignes si le fichier a été mis en page avec des retours chariot).

Segment de type méta : séparation structurante

Séparation entre 2 paragraphes : saut de ligne(s), retrait...

Parcours des segments d'un fichier texte simple

Opérations sur fichier ASCII simple (itérateur sur les segments).

Parcours des segments d'un fichier texte mis en forme

Opérations sur fichier ASCII mis en page (itérateur sur les segments).

c) Les segments dans les fichiers SGML

Base de définition des segments

Les segments de type data : contenu textuel

C'est grosso modo le `#PCDATA` de la norme SGML.

Les segments de type méta : les balises

Il s'agit des balises (par exemple `<P>` ou `</P>`) ; ce module est générique (et par conséquent tous ses descendants aussi) car chaque DTD a sa famille de balises (cf. le type énumératif `Tag_Category`).

Balises ouvrantes

Balise ouvrante (par exemple `<P>`), dite « start tag » dans la norme SGML. L'abréviation « stag » est directement inspirée de la norme, qui note « stago » et « stagc » les délimiteurs de la balise (*start tag open, start tag close*).

Balises fermantes

Balise fermante (par exemple `</P>` ou `</>`), dite « end tag » dans la norme SGML.

Segments de type indéfini (data ou méta)

Segment polymorphe, lorsqu'il faut considérer indifféremment les segments « data » (textuels) et « méta » (balises) d'un fichier.

Le contexte (emboîtement des éléments ouverts)

Définit et gère la pile des balises ouvertes, au fur et à mesure du parcours d'un fichier.

Les traitements contextuels

Le module `segments-sgml-context_g.process_g` permet de définir des traitements associés aux opérations d'ouverture et de fermeture de balises.

d) Une instance de segments SGML : les segments HTML

Définition du jeu de balises et de son comportement

C'est dans le module `segments-sgml-html` que sont regroupées toutes les informations propres à la DTD HTML.

Modules qui en découlent

En dérive ensuite tout ce qui a trait aux balises, pour le cas de HTML :

Balises

Balises ouvrantes

Balises fermantes

Balises indéfinies

Contexte structurel

e) Une autre instance de segments SGML : les segments Corpus

Définition du jeu de balises et de son comportement

C'est dans le module `segments-sgml-decid` que sont regroupées toutes les informations propres à la DTD Corpus.

Modules qui en découlent

De même que pour HTML, tout ce qui est lié aux balises est actualisé pour cette autre DTD :

Balises

Le nom de ce module se termine en `meta_pkg` au lieu de `meta`, car il y a une balise nommée `meta` dans le jeu de balises de la DTD Corpus.

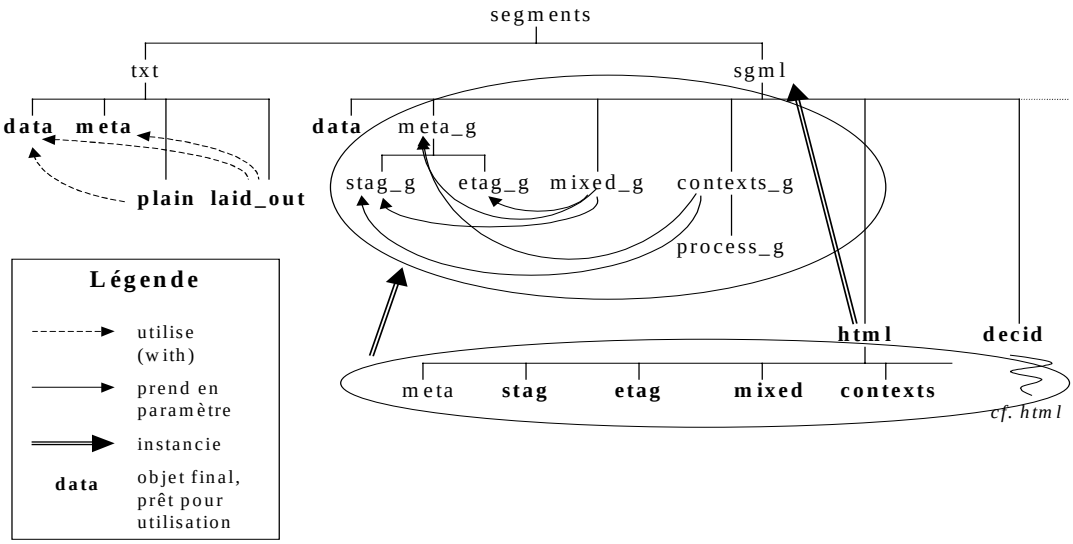
Balises ouvrantes

Balises fermantes

Balises indéfinies

Contexte structurel

f) Schéma récapitulatif de l'organisation de la famille « segments »



4. Utilitaires

Ne sont pas présentés ici certains modules très généraux et utilisés dans tous nos programmes, comme :

- celui qui gère l'ouverture et la fermeture de fichiers passés en paramètres (il s'agit pour nous d'un module que nous avons appelé `files`) ;
- celui qui s'occupe des messages d'erreur, de signalement ou d'information diverses sur le déroulement du traitement (notre module `logs`) ;

- celui qui concerne les paramètres d'installation du programme (localisation des fichiers de données, de l'espace de travail, des fichiers générés, valeurs par défaut de paramètres, etc.) (module `parameters`).

a) Tailles : la gestion des ordres de grandeur

Le module `sizes` effectue . Nous n'en donnons que les spécifications (`.ads`), le corps (`.adb`) étant facile à (re)constituer et ne présentant pas de particularités significatives.

b) Lecture de fichiers tabulés

Nous utilisons le module `read_files`, conçu et implémenté par Pascal OBRY. Nous n'en donnons ici que les spécifications (`.ads`), qui sont ce que nos modules « voient », ce à quoi ils ont accès.

c) Un générique pour construire et manipuler des objets polymorphes

Les principes de base de ce générique nous ont été donnés par Pascal OBRY. Il s'agit de disposer d'un modèle (un générique, `polymorphic_objects_g`) à partir duquel dériver simplement des objets subsumant plusieurs types, ainsi qu'une série d'opérations associées.

5. Données auxiliaires

a) Entités SGML

Les entités SGML servent notamment à coder tous les caractères diacritiques et caractères spéciaux. Nous donnons un court extrait de notre table d'association pour montrer comment elle se présente.

C. CODE SOURCE

1. Programmes de traitement

a) De texte simple vers Corpus

```
with Ada.Text_IO;
with Files;
with Segments.Sgml.Data;
with Segments.Sgml.Decid;
with Segments.Sgml.Decid.Etag;
with Segments.Sgml.Decid.Stag;
with Segments.Txt.Data;
with Segments.Txt.Plain_Ascii;

procedure Asc2dcd is

   use Segments;

   Ascii_File, Decid_File : Ada.Text_IO.File_Type ;

   procedure Translate (Object : in Segments.Txt.Data.Segment) is
   begin
      Sgml.Decid.Stag.Write (Tag => Sgml.Decid.Nls, File => Decid_File);
      Sgml.Data.Write (Object =>
                          Sgml.Data.Create (Content =>
                               Txt.Data.Get_Content (Object => Object)),
                       File => Decid_File);
      Sgml.Decid.Etag.Write (Tag => Sgml.Decid.Nls, File => Decid_File);
   end Translate;

   procedure Translate_Each_Segment is
     new Txt.Plain_Ascii.For_Each_Segment_G (Action => Translate);

begin

--   Files.Interactive_Open (Which => "ASCII",
--                           Logical_Name => Ascii_File,
--                           Mode => Ada.Text_IO.In_File);

--   Files.Interactive_Create (Which => "DECID",
--                             Logical_Name => Decid_File,
--                             Mode => Ada.Text_IO.Out_File);

   Files.Command_Line_Open_create (Program_Name   => "asc2dcd.exe",
                                   Logical_Src    => Ascii_file,
                                   Logical_dest   => Decid_File,
                                   Dest_Extension => "dcd");

   Sgml.Decid.Stag.Write (Tag => Sgml.Decid.Corpus, File => Decid_File);
   Sgml.Decid.Stag.Write (Tag => Sgml.Decid.Doc, File => Decid_File);
   Sgml.Decid.Stag.Write (Tag => Sgml.Decid.Text, File => Decid_File);

   Translate_Each_Segment (File => Ascii_File);

   Sgml.Decid.Etag.Write (Tag => Sgml.Decid.Text, File => Decid_File);
   Sgml.Decid.Etag.Write (Tag => Sgml.Decid.Doc, File => Decid_File);
   Sgml.Decid.Etag.Write (Tag => Sgml.Decid.Corpus, File => Decid_File);

   Ada.Text_IO.Close (File => Ascii_File);
   Ada.Text_IO.Close (File => Decid_File);

end Asc2dcd;
```

b) De texte ASCII mis en forme vers Corpus

```
with Ada.Text_IO;
with Files;
with Segments.Sgml.Data;
with Segments.Sgml.Decid;
with Segments.Sgml.Decid.Etag;
with Segments.Sgml.Decid.Stag;
with Segments.Txt.Data;
with Segments.Txt.Laid_Out_Ascii;
with Segments.Txt.Meta;

procedure Csr2dcd is

   use Segments;

   Caesure_File, Decid_File : Ada.Text_IO.File_Type ;

   procedure Translate_And_Write (Object : in Txt.Data.Segment) is
   begin
      Sgml.Decid.Stag.Write (Tag => Sgml.Decid.Nls, File => Decid_File);
      Sgml.Data.Write (Object =>
                          Sgml.Data.Create (Content =>
                               Txt.Data.Get_Content (Object => Object)),
                       File => Decid_File);
      Sgml.Decid.Etag.Write (Tag => Sgml.Decid.Nls, File => Decid_File);
   end Translate_And_Write;

   procedure Nothing_Done (Object : in Txt.Meta.Segment) is
   begin
      null;
   end Nothing_Done;
```

```
   procedure Translate_Each_Segment is
     new Txt.Laid_Out_Ascii.For_Each_Segment_G
     (Action_Data => Translate_And_Write,
      Action_Meta => Nothing_Done);

begin

   Files.Interactive_Open (Which => "ASCII with caesure",
                           Logical_Name => Caesure_File,
                           Mode => Ada.Text_IO.In_File);
   Files.Interactive_Create (Which => "DECID",
                             Logical_Name => Decid_File,
                             Mode => Ada.Text_IO.Out_File);

   Sgml.Decid.Stag.Write (Tag => Sgml.Decid.Corpus, File => Decid_File);
   Sgml.Decid.Stag.Write (Tag => Sgml.Decid.Doc, File => Decid_File);
   Sgml.Decid.Stag.Write (Tag => Sgml.Decid.Text, File => Decid_File);

   Translate_Each_Segment (File => Caesure_File);

   Sgml.Decid.Etag.Write (Tag => Sgml.Decid.Text, File => Decid_File);
   Sgml.Decid.Etag.Write (Tag => Sgml.Decid.Doc, File => Decid_File);
   Sgml.Decid.Etag.Write (Tag => Sgml.Decid.Corpus, File => Decid_File);

   Ada.Text_IO.Close (File => Caesure_File);
   Ada.Text_IO.Close (File => Decid_File);

end Csr2dcd;
```

c) De HTML vers Corpus

```
with Ada.Characters.Latin_1;
with Ada.Exceptions;
with Ada.Text_IO;
with Files;
with Lines;
with Logs;
with Paths;

with Segments.Sgml.Contexts_G.Process_G;

with Segments.Sgml.Data;

with Segments.Sgml.Decid;
with Segments.Sgml.Decid.Contexts;
with Segments.Sgml.Decid.Etag;
with Segments.Sgml.Decid.Mixed;
with Segments.Sgml.Decid.Stag;

with Segments.Sgml.Html;
with Segments.Sgml.Html.Contexts;
with Segments.Sgml.Html.Etag;
with Segments.Sgml.Html.Mixed;
with Segments.Sgml.Html.Stag;

with Sizes;

procedure Htm2dcd is

   use Segments.Sgml;

-------------------------------------------------------------------------------

   Html_File, Decid_File : Ada.Text_IO.File_Type;

   Html_Context : aliased Html.Contexts.Tag_Context;
   Decid_Context  : aliased Decid.Contexts.Tag_Context;

   Title : Lines.Line;
   Url   : Paths.Path;
   Alt   : Lines.Line;
   Space : Sizes.M_Natural := 0;
   Test  : Boolean;

-------------------------------------------------------------------------------

   procedure Write_Decid_Stag (Object : in Decid.Stag.Segment) is
   begin
      Decid.Stag.Write (Object => Object, File => Decid_File);
   end Write_Decid_Stag;

   procedure Write_Decid_Echo (Object : in Decid.Stag.Segment) is
   begin
      case Decid.Stag.Get_Tag (Object) is
         when Decid.Arr | Decid.Idea | Decid.Expr =>
            Decid.Etag.Write_Short
              (Object => Decid.Mixed.Stag_To_Etag (Object),
               File   => Decid_File);
         when others =>
            Decid.Etag.Write (Object => Decid.Mixed.Stag_To_Etag (Object),
                              File   => Decid_File);
      end case;  -- Decid.Stag.Get_Tag (Object)
   end Write_Decid_Echo;

   procedure Write_Decid_Data (Object : in Data.Segment) is
   begin
      Data.Write (Object => Object,
                  Entities_Resolved => True,
                  File => Decid_File);
   end Write_Decid_Data;

   procedure Do_Nothing (Object : in Decid.Stag.Segment) is
   begin
```

```
      null;
   end Do_Nothing;

   package Decid_Process is new Decid.Contexts.Process_G
     (Action_On_Virtual_Opening => Do_Nothing,
      Action_On_Real_Opening    => Write_Decid_Stag,
      Action_On_Virtual_Closing => Do_Nothing,
      Action_On_Real_Closing    => Write_Decid_Echo,
      Action_On_Data            => Write_Decid_Data);

--------------------------------------------------------------------------

   procedure Actualize_Title is
   begin
      Decid_Process.Opening (Decid.Text);
      if not Lines.Is_Empty (Title) then
         Decid_Process.Opening (Decid.Tit);
         Decid_Process.Actualize (Title);
         Decid_Process.Closing (Decid.Tit);
      end if;  -- not Lines.Is_Empty (Title)
   end Actualize_Title;

   procedure Translate_Stag (Object : Html.Stag.Segment) is
   begin
      case Html.Stag.Get_Tag (Object) is

         when Html.Html =>
            Decid_Process.Opening (Decid.Doc);
            Lines.Reset (Title);
            Paths.Reset (Url);
            Space := 0;

         when Html.Body_Side =>
            Decid.Contexts.Reset_Base (Decid_Context);
            Space := 0;

         when Html.Heading =>
            Decid_Process.Opening (Decid.Head);
            Space := 0;

         when Html.High_Division | Html.Low_Division  =>
            Decid_Process.Opening (Object     => Decid.Nls,
                                   Maybe_Done => True);
            Space := 0;

         when Html.Caesura =>
            if Space > 1 then  --2 is a threshold
               Decid_Process.Opening (Object =>
                                      Decid.Contexts.Get_Base (Decid_Context));
               Space := 0;
            end if;  -- Space > 1

         when Html.List =>
            Decid_Process.Opening (Decid.Blck);
            Space := 0;

         when Html.Item =>
            Decid_Process.Opening (Decid.Item);
            Space := 0;

         when Html.Definiendum =>
            Decid_Process.Opening (Decid.Item);
            Decid_Process.Opening (Decid.Head);
            Space := 0;

         when Html.Definiens =>
            Decid_Process.Opening (Decid.Nls);
            Space := 0;

         when Html.Table =>
            Decid_Process.Opening (Decid.Scop);
            Space := 0;

         when Html.Caption =>
            Decid_Process.Opening (Decid.Meta);
            Space := 0;

         when Html.Header_Side | Html.Title |  -- not used but borders
           Html.Row | Html.Header_Cell | Html.Data_Cell =>  -- (table
parts)
            Space := 0;

         when Html.Dividing | Html.Base_Url | Html.Alt_Media =>
            -- empty tags and mark tags, have been processed before
            -- (as virtual)
            null;
         when Html.Code =>
            null;

         when Html.Other =>
            null;

         when Html.Null_Tag =>  -- is not considered in procedure Opening
            null;
      end case;  -- Html.Stag.Get_Tag (Object)
   end Translate_Stag;

   procedure Translate_Etag (Object : Html.Stag.Segment) is
   begin
      case Html.Stag.Get_Tag (Object) is

         when Html.Html =>
            Decid_Process.Closing (Decid.Doc);

         when Html.Body_Side =>
            Decid_Process.Closing (Decid.Text);
```

```
         when Html.Heading | Html.Definiendum =>
            Decid_Process.Closing (Decid.Head);

         when Html.High_Division | Html.Low_Division =>
            Decid_Process.Closing_Base_If_Not_Done;

         when Html.List =>
            Decid_Process.Closing (Decid.Blck);

         when Html.Item =>
            Decid_Process.Closing (Decid.Item);

         when Html.Definiens =>
            Decid_Process.Closing_Base_If_Not_Done;

         when Html.Table =>
            Decid_Process.Closing (Decid.Scop);

         when Html.Caption =>
            Decid_Process.Closing (Decid.Meta);

         when Html.Header_Side | Html.Title | Html.Other |  -- not used
           Html.Row | Html.Header_Cell | Html.Data_Cell =>  -- (table
parts)
            null;

         when Html.Dividing | Html.Caesura |  -- empty tags
           Html.Base_Url | Html.Alt_Media =>  -- (used for attributes)
            null;

         when Html.Code =>
            null;

         when Html.Null_Tag =>  -- case processed by the basic
opening/closing
            null;

      end case;  --  Html.Stag.Get_Tag (Object)
   end Translate_Etag;

   procedure Translate (Object : in Data.Segment) is
   begin
      if Space = 1 then
         Decid_Process.Actualize (Lines.To_Decid_String
                                  (String'(1 =>
                                           Ada.Characters.Latin_1.Space)));
      end if;  -- Space = 1
      Space := 0;
      case Html.Contexts.Get_Present_Context (Html_Context) is
         when Html.Title =>
            Title :=
              Lines.Append (Left => Title,
                            Right =>
                              Data.Get_Content (Object => Object,
                                                Entities_Resolved =>
                                                  True));
         when Html.Code =>
            null;
         when others =>
            Decid_Process.Actualize (Object);
      end case;  -- Html.Contexts.Get_Present_Context (Html_Context)
   end Translate;

   procedure Mark_Tags_Echo (Object : in Html.Stag.Segment) is
   begin
      case Html.Stag.Get_Tag (Object) is
         when Html.Caesura =>
            Space := Space +1;

         when Html.Dividing =>
            Decid_Process.Opening (Decid.Contexts.Get_Base
(Decid_Context));
            Space := 0;

         when Html.Base_Url =>
            Html.Stag.Get_Attribute (Object => Object,
                                     Name => "HREF",
                                     Value => Url,
                                     Is_Present => Test);
            if Test then
               Decid_Process.Opening (Decid.Arr);
               Decid_Process.Actualize (Lines.To_Decid_String ("URL"));
               Decid_Process.Closing (Decid.Arr);
               Decid_Process.Opening (Decid.Box);
               Decid_Process.Actualize (Lines.To_Decid_String
                                        (Paths.To_String (Url)));
               Decid_Process.Closing (Decid.Box);
            end if;  -- Test

         when Html.Alt_Media =>
            Html.Stag.Get_Attribute (Object => Object,
                                     Name => "ALT",
                                     Value => Alt,
                                     Is_Present => Test);
            if Test then
               Alt := Lines.Append (Ada.Characters.Latin_1.Space, Alt);
               Alt := Lines.Append (Alt, Ada.Characters.Latin_1.Space);
               Decid_Process.Actualize (Alt);
            end if;  -- Test

         when Html.Body_Side =>
            Actualize_Title;

         when others =>
            null;
```

```
      end case; -- Html.Stag.Get_Tag (Object)
   end Mark_Tags_Echo;

   procedure Do_Not_Forget_Stored_Data (Object : in Html.Stag.Segment) is
   begin
      if Html."=" (Html.Stag.Get_Tag (Object), Html.Html) then
-- if the tag <HTML> is virtually closed, that means that no SGML data
-- segment was found
            Actualize_Title;
      end if;  -- Html."=" (Html.Stag.Get_Tag (Object), Html.Html)
   end Do_Not_Forget_Stored_Data;

------------------------------------------------------------------------

   package Html_Process is new Html.Contexts.Process_G
     (Action_On_Virtual_Opening => Mark_Tags_Echo,
      Action_On_Real_Opening    => Translate_Stag,
      Action_On_Virtual_Closing => Do_Not_Forget_Stored_Data,
      Action_On_Real_Closing    => Translate_Etag,
      Action_On_Data            => Translate);


------------------------------------------------------------------------

   procedure Html_Process_Closing (Object : Html.Etag.Segment) is
   begin
      Html_Process.Closing (Object => Object,
                            Maybe_Done => False);
   end Html_Process_Closing;

   procedure Html_Process_Opening (Object : Html.Stag.Segment) is
   begin
      Html_Process.Opening (Object => Object,
                            Maybe_Done => False);
   end Html_Process_Opening;

   procedure Read_And_Do is
     new Html.Mixed.Action_G (Action_Data => Html_Process.Actualize,
                              Action_Stag => Html_Process_Opening,
                              Action_Etag => Html_Process_Closing);

   procedure Interpret_Each_Segment is
     new Html.Mixed.For_Each_Segment_G (Action => Read_And_Do);

------------------------------------------------------------------------

begin

   Files.Command_Line_Open_create (Program_Name   => "htm2dcd.exe",
                                   Logical_Src    => Html_File,
                                   Logical_dest   => Decid_File,
                                   Dest_Extension => "dcd");

   Ada.Text_IO.Set_Input (File  => Html_File);
   Ada.Text_IO.Set_Output (File => Decid_File);

   Html_Process.Init_Context (Html_Context'Access);
   Decid_Process.Init_Context (Decid_Context'Access);

   Decid_Process.Opening (Decid.Corpus);
   Html_Process.Opening (Html.Html);

   Interpret_Each_Segment (File => Html_File);

   Html_Process.Closing (Object => Html.Html, Maybe_Done => True);
   Decid_Process.Closing (Decid.Corpus);

   Logs.Write (Message => "End of processing : if no other message, " &
               "everything is OK.",
               Ends_The_Line => True);

   Ada.Text_IO.Close (File => Html_File);
   Ada.Text_IO.Close (File => Decid_File);

exception

   when E : others =>
      Ada.Text_IO.Put_Line (File => Ada.Text_IO.Standard_Output,
                            Item => "htm2dcd  Unhandled exception : " &
                            Ada.Exceptions.Exception_Name (E));


end Htm2dcd;
```

2. Les chaînes de caractères

a) Définition générique des chaînes de caractères pour DECID

```
with Ada.Characters.Latin_1;
with Ada.Strings;
with Ada.Strings.Bounded;
with Ada.Strings.Maps;
with Ada.Text_IO;
with Ada.Streams;
with Sizes;

generic
   Size_Group : Sizes.Size;

package Decid_Strings_G is

   use Ada.Streams;

   type Decid_String is private;

   Nasty_Look : exception;  -- = mal fichu !

   Out_Of_Bounds_Index : exception;
-- returned when dealing with character index
-- outside the real length of the Decid_String
-- if the parameter Control_Index is set to True,
-- and except for the case when upper_bound < lower_bound

   Index_Should_Be_Controlled : constant Boolean;
-- fix the default value for parameter Control_Index
-- cf. explanation of Out_Of_Bounds_Index exception in Decid_Strings.ads

   Padding_Character_Default : Character
     renames Ada.Characters.Latin_1.Space;

   function Standard_Max_Size return Positive;

   function Are_The_Same (Left, Right : in Decid_String)
                          return Boolean;
   function Are_The_Same (Left : in Decid_String; Right : in String)
                          return Boolean;
   function Are_The_Same (Left : in String; Right : in Decid_String)
                          return Boolean;

   Null_Constant : constant Decid_String;
   function Null_Decid_String return Decid_String;
-- the constant is usefull to give value to other constants ;
-- the function has the advantage to be automatically redefined
-- for every derived type.

   function Is_Empty (Object : in Decid_String) return Boolean;
   procedure Reset (Object : in out Decid_String);
   function Size (Object : in Decid_String) return Natural;

   function To_Decid_String (Object : in String) return Decid_String;
   function To_Decid_String (Object : in Character) return Decid_String;
   function To_String (Object : in Decid_String) return String;
-- the first character of the Decid_String gets the index 1.

   function Append (Left, Right : in Decid_String)
                    return Decid_String;
   function Append (Left : in Decid_String; Right : in String)
                    return Decid_String;
   function Append (Left : in String; Right : in Decid_String)
                    return Decid_String;
   function Append (Left : in Decid_String; Right : in Character)
                    return Decid_String;
   function Append (Left : in Character; Right : in Decid_String)
                    return Decid_String;

   function "&" (Left, Right : in Decid_String)
                 return Decid_String
     renames Append;

   function "&" (Left : in Decid_String; Right : in String)
                 return Decid_String
   renames Append;

   function "&" (Left : in String; Right : in Decid_String)
                 return Decid_String
   renames Append;

   function "&" (Left : in Decid_String; Right : in Character)
                 return Decid_String
   renames Append;

   function "&" (Left : in Character; Right : in Decid_String)
                 return Decid_String
   renames Append;

   function Get_Character (Source : in Decid_String;
                           Index  : in Positive;
                           Control_Index : in Boolean :=
                             Index_Should_Be_Controlled) return Character;
   -- the first character of the Decid_String has always the position 1.
   -- if Control_Index is set to False and index out of bounds,
   -- then the function returns the Null Character
   -- (cf. constant Null_Character in Decid_Strings.adb).
```

```
   function Get_String (Source : in Decid_String;
                        From   : in Positive;
                        To     : in Natural;
                        Control_Index : in Boolean :=
                          Index_Should_Be_Controlled) return String;
-- if From > To : return an empty string
-- else return the part of the Decid_String between from and to
-- (its length can be different from To-From+1 ;
-- it can also be the empty string)

   function Chunk (Source : in Decid_String;
                   From   : in Positive;
                   To     : in Natural;
                   Control_Index : in Boolean :=
                     Index_Should_Be_Controlled) return Decid_String;
-- if From > To : return an empty Decid_String
-- else return the part of the Decid_String between from and to
-- (its length can be different from To-From+1 ;
-- it can also be the empty Decid_String)

   function Replace (Source : in Decid_String;
                     From   : in Positive;
                     To     : in Natural;
                     By     : in Decid_String;
                     Control_Index : in Boolean :=
                       Index_Should_Be_Controlled;
                     Padding_Character : in Character :=
                       Padding_Character_Default) return Decid_String;

   procedure Replace (Source : in out Decid_String;
                      From : in Positive;
                      To : in Natural;
                      By : in Decid_String;
                      Control_Index : in Boolean :=
                        Index_Should_Be_Controlled;
                      Padding_Character : in Character :=
                        Padding_Character_Default);
-- for function and procedure Replace_Slice,
-- nothing is done if From > To. Else :
-- if they exist, characters of source between from and to are erased,
-- first character of By gets the position From,
-- if From > length of source, padding characters are inserted.

   function Insert (Source : in Decid_String;
                    At_Index : in Positive;
                    Object : in Decid_String;
                    Control_Index : in Boolean :=
                      Index_Should_Be_Controlled;
                    Padding_Character : in Character :=
                      Padding_Character_Default) return Decid_String;

   procedure Insert (Source : in out Decid_String;
                     At_Index : in Positive;
                     Object : in Decid_String;
                     Control_Index : in Boolean :=
                       Index_Should_Be_Controlled;
                     Padding_Character : in Character :=
                       Padding_Character_Default);
-- first character of Object gets position At_Index.
-- If At_Index is greater than the length of Source
-- and Control_Index set to False, then
-- padding characters are inserted.

   function Delete (Source : in Decid_String;
                    From : in Positive;
                    To : in Natural;
                    Control_Index : in Boolean :=
                      Index_Should_Be_Controlled) return Decid_String;

   procedure Delete (Source : in out Decid_String;
                     From : in Positive;
                     To : in Natural;
                     Control_Index : in Boolean :=
                       Index_Should_Be_Controlled);
-- for function and procedure Delete, nothing is done if From > To.
-- Else : characters between positions From and To are suppressed.

   function Transliterate (Source : in Decid_String;
                           Mapping : in Ada.Strings.Maps.Character_Mapping)
                           return Decid_String;

   procedure Transliterate (Source : in out Decid_String;
                            Mapping : in
Ada.Strings.Maps.Character_Mapping);

   function Has_Characters (Object : in Decid_String;
                            In_Set : in Ada.Strings.Maps.Character_Set)
                            return Boolean;

   function Find_String (Source : in Decid_String;
                         Object : in String;
                         Going : in Ada.Strings.Direction :=
                           Ada.Strings.Forward;
                         Mapping : in Ada.Strings.Maps.Character_Mapping
                           := Ada.Strings.Maps.Identity) return Natural;

   function Find_String (Source : in Decid_String;
                         Low_Bound : in Positive;
                         High_Bound : in Natural;
                         Object : in String;
                         Going : in Ada.Strings.Direction :=
                           Ada.Strings.Forward;
                         Mapping : in Ada.Strings.Maps.Character_Mapping
                           := Ada.Strings.Maps.Identity;
                         Control_Index : in Boolean :=
```

```
                          Index_Should_Be_Controlled) return Natural;

   function Find_Chunk (Source : in Decid_String;
                        Object : in Decid_String;
                        Going : in Ada.Strings.Direction :=
                          Ada.Strings.Forward;
                        Mapping : in Ada.Strings.Maps.Character_Mapping :=
                          Ada.Strings.Maps.Identity) return Natural;

   function Find_Chunk (Source : in Decid_String;
                        Low_Bound : in Positive;
                        High_Bound : in Natural;
                        Object : in Decid_String;
                        Going : in Ada.Strings.Direction :=
                          Ada.Strings.Forward;
                        Mapping : in Ada.Strings.Maps.Character_Mapping :=
                          Ada.Strings.Maps.Identity;
                        Control_Index : in Boolean :=
                          Index_Should_Be_Controlled) return Natural;
-- if Object is not found : returns 0.

   procedure Find_Pattern (Source : in Decid_String;
                           Set : in Ada.Strings.Maps.Character_Set;
                           Test : in Ada.Strings.Membership :=
                             Ada.Strings.Inside;
                           Going : in Ada.Strings.Direction :=
                             Ada.Strings.Forward;
                           From : out Positive;
                           To : out Natural);

   procedure Find_Pattern (Source : in Decid_String;
                           Low_Bound : in Positive;
                           High_Bound : in Natural;
                           Set : in Ada.Strings.Maps.Character_Set;
                           Test : in Ada.Strings.Membership :=
                             Ada.Strings.Inside;
                           Going : in Ada.Strings.Direction :=
                             Ada.Strings.Forward;
                           From : out Positive;
                           To : out Natural;
                           Control_Index : in Boolean :=
                             Index_Should_Be_Controlled);
-- if no token found : From is set to 1 and To is set to 0.

   procedure Read (Object : out Decid_String;
                   File : in Ada.Text_IO.File_Type :=
                     Ada.Text_IO.Current_Input);

private


   package Base is new
     Ada.Strings.Bounded.Generic_Bounded_Length (Sizes.To_Number
(Size_Group));

   type Decid_String is new Base.Bounded_String;

   procedure Write_Stream_Decid (Stream : access Root_Stream_type'Class;
                                 Object : Decid_string);

   procedure Read_Stream_Decid (Stream : access Root_Stream_type'Class;
                                Object : out Decid_string);

   for Decid_String'Write use Write_Stream_Decid;
   for Decid_String'read  use Read_Stream_Decid;

   Index_Should_Be_Controlled : constant Boolean := True;

   Null_Constant : constant Decid_String :=
     Decid_String (Base.Null_Bounded_String);


end Decid_Strings_G;

================================================================================
================================================================================

with Logs;

package body Decid_Strings_G is

   subtype Natural_Range is Base.Length_Range;
   subtype Positive_Range is Positive range 1..Base.Max_Length;

   Null_Character : Character renames Ada.Characters.Latin_1.NUL;

   function Standard_Max_Size return Positive is
   begin
      return Sizes.To_Number (Notation => Size_Group);
   end Standard_Max_Size;


   function Are_The_Same (Left, Right : in Decid_String)
                          return Boolean is
   begin
      return Left = Right;
   end Are_The_Same;


   function Are_The_Same (Left : in Decid_String; Right : in String)
                          return Boolean is
   begin
      return Left = Right;
   end Are_The_Same;
```

```
function Are_The_Same (Left : in String; Right : in Decid_String)
                       return Boolean is
begin
   return Left = Right;
end Are_The_Same;


function Null_Decid_String return Decid_String is
begin
   return Null_Constant;
end Null_Decid_String;


function Is_Empty (Object : in Decid_String) return Boolean is
begin
   return Object = Null_Decid_String;
end Is_Empty;


procedure Reset (Object : in out Decid_String) is
begin
   Object := Null_Decid_String;
end Reset;


function Size (Object : in Decid_String) return Natural is
begin
   return Length (Object);
end Size;


function To_Decid_String (Object : in String) return Decid_String is
begin
   return To_Bounded_String (Source=>Object, Drop=>Ada.Strings.Error);
exception
   when Ada.Strings.Length_Error =>
      Logs.Write(Kind=>Logs.Warning,
                         Localization=>
                           "function Decid_Strings.To_Decid_String",
                         Exception_Name=>"Ada.Strings.Length_Error",
                         Propagated=>True,
                         Propagation_Name=>"Decid_Strings.Nasty_Look");
      Logs.Write(Data=>Object);
      raise Nasty_Look;
end To_Decid_String;


function To_Decid_String (Object : in Character) return Decid_String is
begin
   return To_Decid_String (String'(1 => Object));
end To_Decid_String;
```

```
function To_String (Object : in Decid_String) return String is
begin
   return Base.To_String(Base.Bounded_String(Object));
end To_String;


function Append (Left, Right : in Decid_String)
                 return Decid_String is
begin
   return Append(Left=>Left, Right=>Right, Drop=>Ada.Strings.Error);
exception
   when Ada.Strings.Length_Error =>
      Logs.Write(Kind=>Logs.Warning,
                         Localization=>"function Decid_Strings.Append",
                         Exception_Name=>"Ada.Strings.Length_Error",
                         Propagated=>True,
                         Propagation_Name=>"Decid_Strings.Nasty_Look");
      Logs.Write(Data=>To_String(Left));
      Logs.Write(Data=>To_String(Right));
      raise Nasty_Look;
end Append;


function Append (Left : in Decid_String; Right : in String)
                 return Decid_String is
begin
   return Append(Left=>Left, Right=>Right, Drop=>Ada.Strings.Error);
exception
   when Ada.Strings.Length_Error =>
      Logs.Write(Kind=>Logs.Warning,
                         Localization=>"function Decid_Strings.Append",
                         Exception_Name=>"Ada.Strings.Length_Error",
                         Propagated=>True,
                         Propagation_Name=>"Decid_Strings.Nasty_Look");
      Logs.Write(Data=>To_String(Left));
      Logs.Write(Data=>Right);
      raise Nasty_Look;
end Append;


function Append (Left : in String; Right : in Decid_String)
                 return Decid_String is
begin
   return Append(Left=>Left, Right=>Right, Drop=>Ada.Strings.Error);
exception
   when Ada.Strings.Length_Error =>
      Logs.Write(Kind=>Logs.Warning,
                         Localization=>"function Decid_Strings.Append",
                         Exception_Name=>"Ada.Strings.Length_Error",
                         Propagated=>True,
                         Propagation_Name=>"Decid_Strings.Nasty_Look");
      Logs.Write(Data=>Left);
```

```
         Logs.Write(Data=>To_String(Right));
         raise Nasty_Look;
   end Append;


   function Append (Left : in Decid_String; Right : in Character)
                    return Decid_String is
   begin
      return Append(Left=>Left, Right=>Right, Drop=>Ada.Strings.Error);
   exception
      when Ada.Strings.Length_Error =>
         Logs.Write(Kind=>Logs.Warning,
                          Localization=>"function Decid_Strings.Append",
                          Exception_Name=>"Ada.Strings.Length_Error",
                          Propagated=>True,
                          Propagation_Name=>"Decid_Strings.Nasty_Look");
         Logs.Write(Data=>To_String(Left));
         raise Nasty_Look;
   end Append;


   function Append (Left : in Character; Right : in Decid_String)
                    return Decid_String is
   begin
      return Append(Left=>Left, Right=>Right, Drop=>Ada.Strings.Error);
   exception
      when Ada.Strings.Length_Error =>
         Logs.Write(Kind=>Logs.Warning,
                    Localization=>"function Decid_Strings.Append",
                    Exception_Name=>"Ada.Strings.Length_Error",
                    Propagated=>True,
                    Propagation_Name=>"Decid_Strings.Nasty_Look");
         Logs.Write(Data=>To_String(Right));
         raise Nasty_Look;
   end Append;


   function Get_Character (Source : in Decid_String;
                           Index : in Positive;
                           Control_Index : in Boolean :=
                             Index_Should_Be_Controlled) return Character
is
   begin
      return Element(Source=>Source, Index=>Index);
   exception
      when Constraint_Error =>  -- index < 1
         case Control_Index is
            when True =>
               pragma Debug (Logs.Write(Kind=>Logs.Trace,
                                        Localization=>
                                          "function
Decid_Strings.Get_Character",
                                        Exception_Name=>"Constraint_Error",
                                        Propagated=>True,
                                        Propagation_Name=>

"Decid_Strings.Out_Of_Bounds_Index"));
               pragma Debug (Logs.Write(Message=>"index = ",
                                        Ends_The_Line=>False));
               pragma Debug (Logs.Write(Data=>Index));
               pragma Debug (Logs.Write(Message=>".",
                                        Ends_The_Line=>True));
               raise Out_Of_Bounds_Index;
            when False =>
               return Null_Character;
         end case;
      when Ada.Strings.Index_Error =>  -- index > lenth of the Decid_String
         case Control_Index is
            when True =>
               pragma Debug (Logs.Write(Kind=>Logs.Trace,
                                        Localization=>
                                          "function
Decid_Strings.Get_Character",
                                        Exception_Name=>
                                          "Ada.Strings.Index_Error",
                                        Propagated=>True,
                                        Propagation_Name=>

"Decid_Strings.Out_Of_Bounds_Index"));
               pragma Debug (Logs.Write(Message=>"index = ",
                                        Ends_The_Line=>False));
               pragma Debug (Logs.Write(Data=>Index));
               pragma Debug (Logs.Write(Message=>".",
                                        Ends_The_Line=>True));
               pragma Debug (Logs.Write(Data=>To_String(Source)));
               raise Out_Of_Bounds_Index;
            when False =>
               return Null_Character;
         end case;
   end Get_Character;


   function Get_String (Source : in Decid_String;
                        From : in Positive;
                        To : in Natural;
                        Control_Index : in Boolean :=
                          Index_Should_Be_Controlled) return String is
      Source_Size : Natural_Range;
   begin
      case From > To is
         when True =>
            return "";
         when False =>
            Source_Size := Length(Source);
            case To > Source_Size is
               when True =>
```

```
               case Control_Index is
                  when True =>
                     raise Out_Of_Bounds_Index;
                  when False =>
                     case From > Source_Size is
                        when True =>
                           return "";
                        when False =>
                           return Slice(Source=>Source,
                                        Low=>From, High=>Source_Size);
                     end case;  -- From > Source_Size
               end case;  -- Control_Index
            when False =>
               return Slice(Source=>Source, Low=>From, High=>To);
         end case;  -- To > Source_Size
   end case;  -- From > To
exception
   when Out_Of_Bounds_Index =>
      pragma Debug (Logs.Write(Kind=>Logs.Trace,
                               Localization=>
                                 "function Decid_Strings.Get_String",
                               Exception_Name=>
                                 "Decid_Strings.Out_Of_Bounds_Index",
                               Propagated=>True));
      pragma Debug (Logs.Write(Data=>To_String(Source)));
      raise;
end Get_String;


function Chunk (Source : in Decid_String;
                From : in Positive;
                To : in Natural;
                Control_Index : in Boolean :=
                  Index_Should_Be_Controlled) return Decid_String is
begin
   return To_Decid_String(Get_String(Source=>Source,
                                     From=>From,
                                     To=>To,
                                     Control_Index=>Control_Index));
exception
   when Out_Of_Bounds_Index =>
      pragma Debug (Logs.Write(Kind=>Logs.Trace,
                               Localization=>
                                 "function Decid_Strings.Chunk",
                               Exception_Name=>
                                 "Decid_Strings.Out_Of_Bounds_Index",
                               Propagated=>True));
      pragma Debug (Logs.Write(Data=>To_String(Source)));
      raise;
end Chunk;


function Mirror (Object : in Decid_String) return Decid_String is
   Result : Decid_String := Null_Decid_String;

   function Local_Append(Left : in Decid_String; Right : in Character)
                         return Decid_String
     renames Append;

begin
   for Position in reverse 1..Size(Object) loop
      Result := Local_Append(Left=>Result,
                             Right=>Get_Character(Source=>Object,
                                                  Index=>Position));
   end loop;  -- Position
   return Result;
end Mirror;


procedure Mirror (Size : in Natural; Position : in out Natural) is
begin
   if Position /= 0 then
      if Position > Size then
         raise Out_Of_Bounds_Index;
      end if;
      Position := Size + 1 - Position;
   end if;
exception
   when Out_Of_Bounds_Index =>
      pragma Debug (Logs.Write(Kind=>Logs.Trace,
                               Localization=>
                                 "procedure Decid_Strings.Mirror",
                               Exception_Name=>
                                 "Decid_Strings.Out_Of_Bounds_Index",
                               Propagated=>True));
      pragma Debug (Logs.Write(Message=>"index = ",
                               Ends_The_Line=>False));
      pragma Debug (Logs.Write(Data=>Position));
      pragma Debug (Logs.Write(Message=>".",
                               Ends_The_Line=>True));
      pragma Debug (Logs.Write(Data=>To_String(Source)));
      raise Out_Of_Bounds_Index;
end Mirror;


procedure Mirror (Size : in Natural;
                  One_Position, Another_Position : in out Positive) is
   Position_Memory : Positive;
begin
   if One_Position > Size or Another_Position > Size then
      raise Out_Of_Bounds_Index;
   end if;
   Position_Memory := One_Position;
   One_Position := Size + 1 - Another_Position;
   Another_Position := Size + 1 - Position_Memory;
exception
```

```
   when Out_Of_Bounds_Index =>
      pragma Debug (Logs.Write(Kind=>Logs.Trace,
                               Localization=>
                                 "procedure Decid_Strings.Mirror",
                               Exception_Name=>
                                 "Decid_Strings.Out_Of_Bounds_Index",
                               Propagated=>True));
      pragma Debug (Logs.Write(Message=>"Size of the string = ",
                               Ends_The_Line=>False));
      pragma Debug (Logs.Write(Data=>Size));
      pragma Debug (Logs.Write(Message=>" ; ",
                               Ends_The_Line=>False));
      pragma Debug (Logs.Write(Message=>"index = ",
                               Ends_The_Line=>False));
      pragma Debug (Logs.Write(Data=>One_Position));
      pragma Debug (Logs.Write(Message=>" and ",
                               Ends_The_Line=>False));
      pragma Debug (Logs.Write(Data=>Another_Position));
      pragma Debug (Logs.Write(Message=>".",
                               Ends_The_Line=>True));
      raise Out_Of_Bounds_Index;
end Mirror;


function Replace (Source : in Decid_String;
                  From : in Positive;
                  To : in Natural;
                  By : in Decid_String;
                  Control_Index : in Boolean :=
                    Index_Should_Be_Controlled;
                  Padding_Character : in Character :=
                    Padding_Character_Default) return Decid_String is
   Source_Size : Natural_Range;
begin
   case From > To is
      when True =>
         return Source;
      when False =>
         Source_Size := Length(Source);
         case Control_Index is
            when True =>
               case To > Source_Size is
                  when True =>
                     raise Out_Of_Bounds_Index;
                  when False =>
                     return Chunk(Source=>Source, From=>1, To=>From-1) &
                       By & Chunk(Source=>Source,
                                  From=>To+1, To=>Source_Size);
               end case;  -- To > Source_Size
            when False =>
               case From > Source_Size is
                  when True =>
                     return Source &
                            (From -1 - Source_Size)*Padding_Character &
                            By;
                  when False =>
                     return Chunk(Source=>Source, From=>1, To=>From-1) &
                       By &
                       Chunk(Source=>Source,
                             From=>To+1, To=>Source_Size,
                             Control_Index=>False);
               end case;  -- From > Source_Size
         end case;  -- Control_Index
   end case;  -- From > To
exception
   when Out_Of_Bounds_Index =>
      pragma Debug (Logs.Write(Kind=>Logs.Trace,
                               Localization=>
                                 "subpgrm Decid_Strings.Replace",
                               Exception_Name=>
                                 "Decid_Strings.Out_Of_Bounds_Index",
                               Propagated=>True));
      pragma Debug (Logs.Write(Data=>To_String(Source)));
      raise;
end Replace;


procedure Replace (Source : in out Decid_String;
                   From : in Positive;
                   To : in Natural;
                   By : in Decid_String;
                   Control_Index : in Boolean :=
                     Index_Should_Be_Controlled;
                   Padding_Character : in Character :=
                     Padding_Character_Default) is
begin
   Source := Replace(Source=>Source,
                     From=>From, To=>To,
                     By=>By,
                     Control_Index=>Control_Index,
                     Padding_Character=>Padding_Character);
end Replace;


function Insert (Source : in Decid_String;
                 At_Index : in Positive;
                 Object : in Decid_String;
                 Control_Index : in Boolean :=
                   Index_Should_Be_Controlled;
                 Padding_Character : in Character :=
                   Padding_Character_Default) return Decid_String is
   Source_Size : Natural_Range := Length(Source);
begin
   case At_Index > Source_Size is
      when True =>
         case Control_Index is
```

```
            when True =>
               raise Out_Of_Bounds_Index;
            when False =>
               return Source &
                 (At_Index -1 - Source_Size)*Padding_Character &
                 Object;
         end case;
      when False =>
         return Chunk(Source=>Source, From=>1, To=>At_Index-1) &
           Object &
           Chunk(Source=>Source, From=>At_Index, To=>Source_Size);
   end case;
exception
   when Out_Of_Bounds_Index =>
      pragma Debug (Logs.Write(Kind=>Logs.Trace,
                               Localization=>
                                 "subpgrm Decid_Strings.Insert",
                               Exception_Name=>
                                 "Decid_Strings.Out_Of_Bounds_Index",
                               Propagated=>True));
      pragma Debug (Logs.Write(Message=>"index = ",
                               Ends_The_Line=>False));
      pragma Debug (Logs.Write(Data=>At_Index));
      pragma Debug (Logs.Write(Message=>".",
                               Ends_The_Line=>True));
      pragma Debug (Logs.Write(Data=>To_String(Source)));
      raise;
end Insert;


procedure Insert (Source : in out Decid_String;
                  At_Index : in Positive;
                  Object : in Decid_String;
                  Control_Index : in Boolean :=
                    Index_Should_Be_Controlled;
                  Padding_Character : in Character :=
                    Padding_Character_Default) is
begin
   Source := Insert(Source=>Source, At_Index=>At_Index, Object=>Object,
                    Control_Index=>Control_Index,
                    Padding_Character=>Padding_Character);
end Insert;


function Delete (Source : in Decid_String;
                 From : in Positive;
                 To : in Natural;
                 Control_Index : in Boolean :=
                   Index_Should_Be_Controlled) return Decid_String is
   Source_Size : Natural_Range;
begin
   case To < From is
      when True =>
         return Source;
      when False =>
         Source_Size := Length(Source);
         case Control_Index is
            when True =>
               case To > Source_Size is
                  when True =>
                     raise Out_Of_Bounds_Index;
                  when False =>
                     return Chunk(Source=>Source, From=>1, To=>From-1) &
                       Chunk(Source=>Source, From=>To+1,
To=>Source_Size);
               end case; --  To > Source_Size
            when False =>
               case From > Source_Size is
                  when True =>
                     return Source;
                  when False =>
                     return Chunk(Source=>Source, From=>1, To=>From-1) &
                       Chunk(Source=>Source, From=>To+1,
To=>Source_Size,
                             Control_Index=>False);
               end case;  -- From > Source_Size
         end case;  -- Control_Index
      end case;  -- To < From
   exception
      when Out_Of_Bounds_Index =>
         pragma Debug (Logs.Write(Kind=>Logs.Trace,
                                  Localization=>
                                    "subpgrm Decid_Strings.Delete",
                                  Exception_Name=>

"Decid_Strings.Out_Of_Bounds_Index",
                                  Propagated=>True));
         pragma Debug (Logs.Write(Message=>"upper index = ",
                                  Ends_The_Line=>False));
         pragma Debug (Logs.Write(Data=>To));
         pragma Debug (Logs.Write(Message=>".",
                                  Ends_The_Line=>True));
         pragma Debug (Logs.Write(Data=>To_String(Source)));
         raise;
   end Delete;

   procedure Delete (Source : in out Decid_String;
                     From : in Positive;
                     To : in Natural;
                     Control_Index : in Boolean :=
                       Index_Should_Be_Controlled) is
   begin
      Source := Delete(Source=>Source, From=>From, To=>To,
                       Control_Index=>Control_Index);
   end Delete;
```

```
   function Transliterate (Source : in Decid_String;
                           Mapping : in Ada.Strings.Maps.Character_Mapping)
                           return Decid_String is
   begin
      return Translate(Source=>Source, Mapping=>Mapping);
   end Transliterate;


   procedure Transliterate (Source : in out Decid_String;
                            Mapping : in
                              Ada.Strings.Maps.Character_Mapping) is
   begin
      Source := Transliterate(Source=>Source, Mapping=>Mapping);
   end Transliterate;


   function Has_Characters (Object : in Decid_String;
                            In_Set : in Ada.Strings.Maps.Character_Set)
                 return Boolean is

      function Is_Subset (Elements : in Ada.Strings.Maps.Character_Set;
                          Set : in Ada.Strings.Maps.Character_Set)
                          return Boolean
        renames Ada.Strings.Maps.Is_Subset;

      function To_Set (Sequence : in Ada.Strings.Maps.Character_Sequence)
-- RM95 : subtype Character_Sequence is String
                       return Ada.Strings.Maps.Character_Set
        renames Ada.Strings.Maps.To_Set;

   begin
      return Is_Subset (Elements => To_Set (Sequence => To_String
(Object)),
                        Set => In_Set);
   end Has_Characters;

   function Find_String (Source : in Decid_String;
                         Object : in String;
                         Going  : in Ada.Strings.Direction :=
                           Ada.Strings.Forward;
                         Mapping : in Ada.Strings.Maps.Character_Mapping
                           := Ada.Strings.Maps.Identity) return Natural is
   begin
      return Index(Source=>Source, Pattern=>Object,
                   Going=>Going, Mapping=>Mapping);
   end Find_String;


   function Find_String (Source : in Decid_String;
                         Low_Bound : in Positive;
                         High_Bound : in Natural;
                         Object : in String;
                         Going : in Ada.Strings.Direction :=
                           Ada.Strings.Forward;
                         Mapping : in Ada.Strings.Maps.Character_Mapping
                           := Ada.Strings.Maps.Identity;
                         Control_Index : in Boolean :=
                           Index_Should_Be_Controlled) return Natural is
      Result : Natural;
   begin
      Result := Find_String(Source=>Chunk(Source=>Source,
                                          From=>Low_Bound,
                                          To=>High_Bound,
                                          Control_Index=>Control_Index),
                            Object=>Object,
                            Going=>Going,
                            Mapping=>Mapping);
      case Result is
         when 0 =>
            return 0;
         when others =>
            return Result + Low_Bound -1;
      end case;
   end Find_String;


   function Find_Chunk (Source : in Decid_String;
                        Object : in Decid_String;
                        Going : in Ada.Strings.Direction :=
                          Ada.Strings.Forward;
                        Mapping : in Ada.Strings.Maps.Character_Mapping
                          := Ada.Strings.Maps.Identity) return Natural is
   begin
      return Find_String(Source=>Source, Object=>To_String(Object),
                         Going=>Going, Mapping=>Mapping);
   end Find_Chunk;


   function Find_Chunk (Source : in Decid_String;
                        Low_Bound : in Positive;
                        High_Bound : in Natural;
                        Object : in Decid_String;
                        Going : in Ada.Strings.Direction :=
                          Ada.Strings.Forward;
                        Mapping : in Ada.Strings.Maps.Character_Mapping
                          := Ada.Strings.Maps.Identity;
                        Control_Index : in Boolean :=
                          Index_Should_Be_Controlled) return Natural is
   begin
      return Find_String(Source=>Source,
                         Low_Bound=>Low_Bound, High_Bound=>High_Bound,
                         Object=>To_String(Object),
                         Going=>Going, Mapping=>Mapping,
                         Control_Index=>Control_Index);
   end Find_Chunk;
```

```
procedure Find_Pattern (Source : in Decid_String;
                        Set : in Ada.Strings.Maps.Character_Set;
                        Test : in Ada.Strings.Membership :=
                          Ada.Strings.Inside;
                        Going : in Ada.Strings.Direction :=
                          Ada.Strings.Forward;
                        From : out Positive;
                        To : out Natural) is
begin
   case Going is
      when Ada.Strings.Forward =>
         Find_Token(Source=>Source, Set=>Set, Test=>Test,
                    First=>From, Last=>To);
      when Ada.Strings.Backward =>
         Find_Token(Source=>Mirror(Source), Set=>Set, Test=>Test,
                    First=>From, Last=>To);
         if To /= 0 then
            Mirror(Size=>Size(Source),
                   One_Position=>From, Another_Position=>To);
         end if;  -- To /= 0
   end case;  -- Going
end Find_Pattern;


procedure Find_Pattern (Source : in Decid_String;
                        Low_Bound : in Positive;
                        High_Bound : in Natural;
                        Set : in Ada.Strings.Maps.Character_Set;
                        Test : in Ada.Strings.Membership :=
                          Ada.Strings.Inside;
                        Going : in Ada.Strings.Direction :=
                          Ada.Strings.Forward;
                        From : out Positive;
                        To : out Natural;
                        Control_Index : in Boolean :=
                          Index_Should_Be_Controlled) is
   Index_Translation : Natural;
begin
   Find_Pattern(Source=>Chunk(Source=>Source,
                              From=>Low_Bound,
                              To=>High_Bound,
                              Control_Index=>Control_Index),
                Set=>Set,
                Test=>Test,
                Going=>Going,
                From=>From,
                To=>To);
   if To > 0 then
      Index_Translation := Low_Bound -1;
      From := From + Index_Translation;
      To := To + Index_Translation;
   end if;
end Find_Pattern;


procedure Read (Object : out Decid_String;
                File : in Ada.Text_IO.File_Type :=
                  Ada.Text_IO.Current_Input) is
   Size : Natural;
   Buffer : String (1..Standard_Max_Size +1);
begin
   Ada.Text_IO.Get_Line (File => File, Item => Buffer, Last => Size);
   if Size = Buffer'Last then
      raise Nasty_Look;
   else
      Object := To_Decid_String (Buffer (1..Size));
   end if;
exception
   when Nasty_Look =>
      Logs.Write (Kind => Logs.Warning,
                  Localization => "Decid_Strings_G.Read",
                  Exception_Name => "Decid_Strings_G.Nasty_Look",
                  Propagated => True);
      Logs.Write (Data => Buffer);
      raise;
end Read;


procedure Write_Stream_Decid (Stream : access Root_Stream_type'Class;
                              Object : Decid_string) is
   S : String := To_String (Object);
begin
   Natural'Write (Stream, S'length);
   String'Write  (Stream, To_String (Object));
end Write_Stream_Decid;


procedure Read_Stream_Decid (Stream : access Root_Stream_type'Class;
                             Object : out Decid_string) is
   S_Length : Natural;

begin
   Object := Null_Decid_String;
   Natural'read (Stream, S_length);
   declare
      S : String (1 .. S_Length);
   begin
      String'Read (Stream, S);
      Object := To_Decid_String (S);
   end ;
end read_Stream_Decid;


end Decid_Strings_G;
```

b) Ligne

```
with Decid_Strings_G;
with Sizes;

package Lines is

   package Decid_Strings is
     new Decid_Strings_G (Size_Group => Sizes.L);

   type Line is new Decid_Strings.Decid_String;

   Null_Line : constant Line := Line (Decid_Strings.Null_Constant);

end Lines;
```

c) Chemin (nom complet d'un fichier)

```
with Decid_Strings_G;
with Sizes;

package Paths is

  package Decid_Strings is
    new Decid_Strings_G (Size_Group => Sizes.M);

  type Path is new Decid_Strings.Decid_String;

end Paths;
```

d) Identifieur générique (en SGML)

```
with Ada.Finalization;
with Ada.Strings;
with Decid_Strings_G;
with Lines;
with Sizes;

package Sgml_Gis is

   type Generic_Identifier is private;

   Null_Gi : constant Generic_Identifier;

   function To_Gi (Object : in String) return Generic_Identifier;

   function Get_Image (Object : in Generic_Identifier) return String;

   function Get_Size (Object : in Generic_Identifier) return Natural;

   function Is_Null (Object : in Generic_Identifier) return Boolean;

   function Find_Gi (Source : in Lines.Line;
                     Object : in Generic_Identifier;
                     Going : in Ada.Strings.Direction) return Natural;

private

   package Decid_Strings is new
     Decid_Strings_G (Size_Group => Sizes.S);

   type Gi_Name is new Decid_Strings.Decid_String;

   type Generic_Identifier is new Ada.Finalization.Controlled with
      record
         Name : Gi_Name;
      end record;

   procedure Initialize (Object : in out Generic_Identifier);

   procedure Adjust (Object : in out Generic_Identifier);

   Null_Gi : constant Generic_Identifier :=
     (Ada.Finalization.Controlled with
      Name => Gi_Name (Decid_Strings.Null_Constant));

end Sgml_Gis;

================================================================================
================================================================================

with Ada.Strings.Maps.Constants;
with Segments.Sgml;

package body Sgml_Gis is

   procedure Initialize (Object : in out Generic_Identifier) is
   begin
      Object.Name :=
        Transliterate (Source => Object.Name,
                       Mapping =>
Ada.Strings.Maps.Constants.Upper_Case_Map);
   end Initialize;

   procedure Adjust (Object : in out Generic_Identifier) is
   begin
      Object.Name :=
        Transliterate (Source => Object.Name,
                       Mapping =>
Ada.Strings.Maps.Constants.Upper_Case_Map);
   end Adjust;
```

```
   function To_Gi (Object : in String) return Generic_Identifier is
   begin
      return (Ada.Finalization.Controlled with
              Name => To_Decid_String (Object));
   end To_Gi;

   function Get_Image (Object : Generic_Identifier) return String is
   begin
      return To_String (Object.Name);
   end Get_Image;

   function Get_Size (Object : in Generic_Identifier) return Natural is
   begin
      return Size (Object.Name);
   end Get_Size;

   function Is_Null (Object : in Generic_Identifier) return Boolean is
   begin
      return Is_Empty (Object.Name);
   end Is_Null;

   function Find_Gi (Source : in Lines.Line;
                     Object : in Generic_Identifier;
                     Going : in Ada.Strings.Direction) return Natural is
   begin
      return Segments.Sgml.Find_Name (Source => Source,
                                      Object => Get_Image (Object),
                                      Going => Going,
                                      Mapping =>

Ada.Strings.Maps.Constants.Upper_Case_Map);
   end Find_Gi;

end Sgml_Gis;
```

3. Les segments

a) Eléments généraux de définition

```
with Ada.Characters.Latin_1;
with Ada.Strings.Maps;
with Ada.Strings.Maps.Constants;
with Lines;

package Segments is

   type Segment_Level is (Meta, Data);

private

------------------------------------------------------------------------
-- Characters sets --

   function "and" (Left, Right : in Ada.Strings.Maps.Character_Set)
                   return Ada.Strings.Maps.Character_Set
     renames Ada.Strings.Maps."and";

   function "or" (Left, Right : in Ada.Strings.Maps.Character_Set)
                   return Ada.Strings.Maps.Character_Set
     renames Ada.Strings.Maps."or";

   function "not" (Right : in Ada.Strings.Maps.Character_Set)
                   return Ada.Strings.Maps.Character_Set
     renames Ada.Strings.Maps."not";

   No_Spacing_Set : constant Ada.Strings.Maps.Character_Set :=
     (Ada.Strings.Maps.Constants.Control_Set
      or Ada.Strings.Maps.To_Set(Ada.Characters.Latin_1.Grave)  -- 96
      or Ada.Strings.Maps.To_Set(Ada.Characters.Latin_1.Acute)) -- 180
     and not Ada.Strings.Maps.To_Set(Ada.Characters.Latin_1.HT);

   Spacing_Set : constant Ada.Strings.Maps.Character_Set :=
     Ada.Strings.Maps.To_Set(Ada.Characters.Latin_1.HT)
     or Ada.Strings.Maps.To_Set(Ada.Characters.Latin_1.Space)
     or Ada.Strings.Maps.To_Set(Ada.Characters.Latin_1.No_Break_Space);

   No_Sign_Set : constant Ada.Strings.Maps.Character_Set :=
     Ada.Strings.Maps.Constants.Control_Set
     or Ada.Strings.Maps.To_Set(Ada.Characters.Latin_1.Space)
     or Ada.Strings.Maps.To_Set(Ada.Characters.Latin_1.No_Break_Space);

------------------------------------------------------------------------
```

```
   procedure Clean (Data : in out Lines.Line;
                    Garbage : in Ada.Strings.Maps.Character_Set);

   procedure Clean (Data : in out Lines.Line;
                    Garbage : in Ada.Strings.Maps.Character_Set;
                    Replacement : in Character);

   procedure Clean_Input (Data : in out Lines.Line);

   function Is_Significant (Object : in Lines.Line)
                            return Boolean;
-- call Clean_Input (Object) before.

end Segments;

================================================================================
================================================================================

package body Segments is

--------------------------------------------------------------------------------

   Simplify_Characters_Map : constant Ada.Strings.Maps.Character_Mapping :=
     Ada.Strings.Maps.To_Mapping
     (Ada.Characters.Latin_1.Left_Angle_Quotation &   -- 171
      Ada.Characters.Latin_1.Right_Angle_Quotation &  -- 187
      Ada.Characters.Latin_1.Soft_Hyphen,             -- 173
      Ada.Characters.Latin_1.Quotation &              -- 34
      Ada.Characters.Latin_1.Quotation &
      Ada.Characters.Latin_1.Hyphen);                 -- 45

   procedure Clean (Data : in out Lines.Line;
                    Garbage : in Ada.Strings.Maps.Character_Set) is
      Low : Positive;
      High : Natural;

   begin
      loop
         Lines.Find_Pattern(Source => Data,
                            Set => Garbage,
                            Test => Ada.Strings.Inside,
                            Going => Ada.Strings.Forward,
                            From => Low,
                            To => High);
         exit when High = 0;
         Lines.Delete(Source => Data, From => Low, To => High);
      end loop;
   end Clean;

   procedure Clean (Data : in out Lines.Line;
                    Garbage : in Ada.Strings.Maps.Character_Set;
                    Replacement : in Character) is
      Low : Positive;
      High : Natural;
      Cleaned : Lines.Line := Lines.Null_Decid_String;

      function "&" (Left : in Lines.Line; Right : in String)
                    return Lines.Line
      renames Lines."&";

      function "&" (Left : in Lines.Line; Right : in Character)
                   return Lines.Line
      renames Lines."&";

   begin
      loop
         Lines.Find_Pattern(Source => Data,
                            Set => Garbage,
                            Test => Ada.Strings.Inside,
                            Going => Ada.Strings.Forward,
                            From => Low,
                            To => High);
         exit when High = 0;
         Cleaned := Cleaned &
           Lines.Get_String(Source => Data, From => 1, To => Low-1) &
           Replacement;
         Data := Lines.Chunk(Source => Data,
                             From => High+1,
                             To => Lines.Size(Data));
      end loop;
      Data := Lines.Append(Left => Cleaned, Right => Data);
   end Clean;

------------------------------------------------------------------------------

   procedure Clean_Input (Data : in out Lines.Line) is
   begin
      Clean(Data => Data,
            Garbage => No_Spacing_Set);
      Clean(Data => Data,
            Garbage => Spacing_Set,
            Replacement => Ada.Characters.Latin_1.Space);
      Lines.Transliterate (Source => Data,
                           Mapping => Simplify_Characters_Map);
   end Clean_Input;

   function Is_Significant (Object : in Lines.Line)
                            return Boolean is
   begin
      return not (Lines.Is_Empty (Object) or
                  Lines.Are_The_Same (Object,
                                      String'(1 =>

Ada.Characters.Latin_1.Space)));
   end Is_Significant;
```

```
------------------------------------------------------------------------

end Segments;
```

b) Les segments dans les fichiers texte (.txt)

Base de définition des segments

```
with Ada.Text_IO;
with Lines;

package Segments.Txt is

   type Segment is abstract tagged private;

   procedure Write (Object : in Segment;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output)
      is abstract;

   function Get_Level (Object : in Segment)
     return Segments.Segment_Level
     is abstract;

private

   type Segment is abstract tagged null record;

   procedure Set_Segment (Object : in out Segment);

end Segments.Txt;

========================================================================
========================================================================

with Ada.Characters.Latin_1;
with Ada.Strings.Maps;
with Ada.Strings.Maps.Constants;

package body Segments.Txt is

   procedure Set_Segment (Object : in out Segment) is
   begin
      null;
   end Set_Segment;

end Segments.Txt;
```

Segment de type data : le contenu d'un alinéa

```
with Ada.Text_IO;
with Lines;

package Segments.Txt.Data is

   type Segment is new Segments.Txt.Segment with private;

   function Create (Content : in Lines.Line)
                    return Segment;

   procedure Write (Object : in Segment;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output);

   function Get_Level (Object : in Segment)
                       return Segments.Segment_Level;

   function Get_Content (Object : in Segment)
                         return Lines.Line;

   function Test_Content (Object : in Segment;
                          Value : in Lines.Line := Lines.Null_Decid_String)
                          return Boolean;

   procedure Set_Content (Object : in out Segment;
                          Value : in Lines.Line);

private

   type Segment is new Segments.Txt.Segment with
      record
         Content : Lines.Line := Lines.Null_Decid_String;
      end record;

   procedure Set_Segment (Object : in out Segment;
                          Content : in Lines.Line);

end Segments.Txt.Data;

========================================================================
========================================================================

package body Segments.Txt.Data is

------------------------------------------------------------------------

   function Get_Level (Object : in Segment)
                       return Segments.Segment_Level is
   begin
      return Segments.Data;
```

```
   end Get_Level;

   function Get_Content (Object : in Segment)
                         return Lines.Line is
   begin
      return Object.Content;
   end Get_Content;

   function Test_Content (Object : in Segment;
                          Value : in Lines.Line := Lines.Null_Decid_String)
                          return Boolean is
   begin
      return Lines.Are_The_Same (Get_Content (Object), Value);
   end Test_Content;

   procedure Set_Content (Object : in out Segment;
                          Value : in Lines.Line) is
   begin
      Object.Content := Value;
   end Set_Content;

-------------------------------------------------------------------------

   procedure Set_Segment (Object : in out Segment;
                          Content : in Lines.Line) is
   begin
      Set_Segment (Object => Segments.Txt.Segment (Object));
      Set_Content (Object => Object, Value => Content);
   end Set_Segment;

   function Create (Content : in Lines.Line)
                    return Segment is
      Result : Segment;
   begin
      Set_Segment (Object => Result, Content => Content);
      return Result;
   end Create;

   procedure Write (Object : in Segment;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output) is
   begin
      Ada.Text_IO.Put(File => File,
                      Item => Lines.To_String (Get_Content(Object)));
   end Write;

-------------------------------------------------------------------------

end Segments.Txt.Data;
```

Segment de type méta : séparation structurante

```
with Sizes;

package Segments.Txt.Meta is

   type Segment is new Segments.Txt.Segment with private;

   type Separator_Kind is (Vertical_Space, Horizontal_Space);

   subtype Separator_Size is Sizes.S_Natural;

   function Create (Separator : in Separator_Kind;
                    Size : in Separator_Size)
                    return Segment;

   procedure Write (Object : in Segment;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output);

   function Get_Level (Object : in Segment)
                       return Segments.Segment_Level;

   function Get_Separator (Object : in Segment)
                           return Separator_Kind;

   procedure Set_Separator (Object : in out Segment;
                            Value : in Separator_Kind);

   function Get_Size (Object : in Segment)
                      return Separator_Size;

   procedure Set_Size (Object : in out Segment;
                       Value : in Separator_Size);

private

   type Segment is new Segments.Txt.Segment with
      record
         Separator : Separator_Kind := Vertical_Space;
         Size : Separator_Size := 0;
      end record;

   procedure Set_Segment (Object : in out Segment;
                          Separator : in Separator_Kind;
                          Size : in Separator_Size);

end Segments.Txt.Meta;
```

==========
==========

```
package body Segments.Txt.Meta is

   function Get_Level (Object : in Segment)
                       return Segments.Segment_Level is
   begin
      return Segments.Meta;
   end Get_Level;

   function Get_Separator (Object : in Segment)
                           return Separator_Kind is
   begin
      return Object.Separator;
   end Get_Separator;

   procedure Set_Separator (Object : in out Segment;
                            Value : in Separator_Kind) is
   begin
      Object.Separator := Value;
   end Set_Separator;

   function Get_Size (Object : in Segment)
     return Separator_Size is
   begin
      return Object.Size;
   end Get_Size;

   procedure Set_Size (Object : in out Segment;
                       Value : in Separator_Size) is
   begin
      Object.Size := Value;
   end Set_Size;

-------------------------------------------------------------------------

   procedure Set_Segment (Object : in out Segment;
                          Separator : in Separator_Kind;
                          Size : in Separator_Size) is
   begin
      Set_Segment (Object => Segments.Txt.Segment (Object));
      Set_Separator (Object => Object, Value => Separator);
      Set_Size (Object => Object, Value => Size);
   end Set_Segment;

   function Create (Separator : in Separator_Kind;
                    Size : in Separator_Size)
                    return Segment is
      Result : Segment;
   begin
      Set_Segment (Object => Result,
                   Separator => Separator, Size => Size);
      return Result;
   end Create;

   procedure Write (Object : in Segment;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output) is
   begin
      Ada.Text_IO.New_Line(File => File);
      Ada.Text_IO.New_Line(File => File);
   end Write;

end Segments.Txt.Meta;
```

Parcours des segments d'un fichier texte simple

```
with Ada.Text_IO;
with Segments.Txt.Data;

package Segments.Txt.Plain_Ascii is

   generic
      with procedure Action (Object : in Segments.Txt.Data.Segment);
   procedure For_Each_Segment_G (File : in out Ada.Text_IO.File_Type);

end Segments.Txt.Plain_Ascii;

=============================================================================
=============================================================================

with Ada.Characters.Latin_1;
with Lines;

package body Segments.Txt.Plain_Ascii is

   function Is_Significant (Input : in Lines.Line) return Boolean is
   begin
      return not (Lines.Are_The_Same (Input, Lines.Null_Decid_String) or
                  Lines.Are_The_Same(Input,
                                     String'(1 =>

Ada.Characters.Latin_1.Space)));
   end Is_Significant;

   function Read (File : in Ada.Text_IO.File_Type)
                  return Segments.Txt.Data.Segment is
      Input : Lines.Line;
   begin
      while not Ada.Text_IO.End_Of_File (File => File) loop
         Lines.Read (File => File, Object => Input);
         Segments.Clean_Input (Input);
         exit when Is_Significant (Input);
      end loop;  -- while not ...End_Of_File...
      return Segments.Txt.Data.Create(Content => Input);
   end Read;
```

```
--   generic
--      with procedure Action (Object : in Segments.Txt.Data.Segment);
   procedure For_Each_Segment_G (File : in out Ada.Text_IO.File_Type) is
   begin
      Ada.Text_IO.Reset (File => File, Mode => Ada.Text_IO.In_File);
      while not Ada.Text_IO.End_Of_File (File) loop
         Action (Read (File => File));
      end loop;  -- while not ...End_Of_File...
   end For_Each_Segment_G;

end Segments.Txt.Plain_Ascii;
```

Parcours des segments d'un fichier texte mis en forme

```
with Ada.Text_IO;
with Polymorphic_Objects_G;
with Lines;
with Segments.Txt.Data;
with Segments.Txt.Meta;

use Segments.Txt.Data;
use Segments.Txt.Meta;

package Segments.Txt.Laid_Out_Ascii is

------------------------------------------------------------------------

   package Polymorphic_Objects is new
     Polymorphic_Objects_G (Object_Root => Segments.Txt.Segment);

   subtype Segment is Polymorphic_Objects.Polymorphic_Object;
------------------------------------------------------------------------

   function To_Segment is new
     Polymorphic_Objects.To_Polymorphic_G (Object_Child =>
                                             Segments.Txt.Data.Segment);

   function To_Segment is new
     Polymorphic_Objects.To_Polymorphic_G (Object_Child =>
                                             Segments.Txt.Meta.Segment);

   function To_Data is new
     Polymorphic_Objects.To_Child_G (Object_Child =>
                                       Segments.Txt.Data.Segment);

   function To_Meta is new
     Polymorphic_Objects.To_Child_G (Object_Child =>
                                       Segments.Txt.Meta.Segment);

------------------------------------------------------------------------

   procedure Write (Object : in Segments.Txt.Laid_Out_Ascii.Segment;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output);

------------------------------------------------------------------------

   function Get_Level (Object : in Segments.Txt.Laid_Out_Ascii.Segment)
                       return Segments.Segment_Level;

------------------------------------------------------------------------

   function Get_Content is new
     Polymorphic_Objects.Child_Function_G (Object_Child =>
                                             Segments.Txt.Data.Segment,
                                           Result_Type => Lines.Line,
                                           Action =>

Segments.Txt.Data.Get_Content);

   procedure Set_Content is new
     Polymorphic_Objects.Child_Procedure_G (Object_Child =>
                                              Segments.Txt.Data.Segment,
                                            Value_Type => Lines.Line,
                                            Action =>

Segments.Txt.Data.Set_Content);

   function Get_Separator is new
     Polymorphic_Objects.Child_Function_G (Object_Child =>
                                             Segments.Txt.Meta.Segment,
                                           Result_Type =>

Segments.Txt.Meta.Separator_Kind,
                                           Action =>

Segments.Txt.Meta.Get_Separator);

   procedure Set_Separator is new
     Polymorphic_Objects.Child_Procedure_G (Object_Child =>
                                              Segments.Txt.Meta.Segment,
                                            Value_Type =>

Segments.Txt.Meta.Separator_Kind,
                                            Action =>

Segments.Txt.Meta.Set_Separator);

   function Get_Size is new
     Polymorphic_Objects.Child_Function_G (Object_Child =>
                                             Segments.Txt.Meta.Segment,
                                           Result_Type =>
```

```
Segments.Txt.Meta.Separator_Size,
                                          Action =>
                                            Segments.Txt.Meta.Get_Size);

   procedure Set_Size is new
     Polymorphic_Objects.Child_Procedure_G (Object_Child =>
                                              Segments.Txt.Meta.Segment,
                                            Value_Type =>

Segments.Txt.Meta.Separator_Size,
                                            Action =>
                                              Segments.Txt.Meta.Set_Size);

------------------------------------------------------------------------

   generic
      with procedure Action_Data (Object : in Segments.Txt.Data.Segment);
      with procedure Action_Meta (Object : in Segments.Txt.Meta.Segment);
   procedure For_Each_Segment_G (File : in out Ada.Text_IO.File_Type);

end Segments.Txt.Laid_Out_Ascii;

========================================================================
========================================================================

with Ada.Characters.Latin_1;
with Ada.Strings.Maps;
with Lines;

package body Segments.Txt.Laid_Out_Ascii is

------------------------------------------------------------------------

   procedure Write (Object : in Segments.Txt.Laid_Out_Ascii.Segment;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output) is
   begin
      Write (Object => Polymorphic_Objects.Get_Details (Object).all,
             File => File);
   end Write;

------------------------------------------------------------------------

   function Get_Level (Object : in Segments.Txt.Laid_Out_Ascii.Segment)
                       return Segments.Segment_Level is
   begin
     return Get_Level (Polymorphic_Objects.Get_Details (Object) .all);
   end Get_Level;

------------------------------------------------------------------------
   Null_Segment : Segments.Txt.Laid_Out_Ascii.Segment :=
     To_Segment(Segments.Txt.Data.Create(Content =>
Lines.Null_Decid_String));

   function Has_Signs (Data : in Lines.Line) return Boolean is
   begin
      return not (Lines.Are_The_Same (Data, Lines.Null_Decid_String)
             or Lines.Are_The_Same(Data,
                                   String'(1 =>

Ada.Characters.Latin_1.Space)));
   end Has_Signs;

   function Has_Blank_Beginning (Data : in Lines.Line) return Boolean is
   begin
      return Lines.Get_Character (Source => Data, Index => 1) =
        Ada.Characters.Latin_1.Space;
   end Has_Blank_Beginning;

   procedure Read (Current_Segment :
                     in out Segments.Txt.Laid_Out_Ascii.Segment;
                   File : in Ada.Text_IO.File_Type :=
                     Ada.Text_IO.Current_Input;
                   Next_Segment :
                     out Segments.Txt.Laid_Out_Ascii.Segment) is
      Data_Input, Current_Content : Lines.Line;

      function "&" (Left, Right : in Lines.Line)
                    return Lines.Line
        renames Lines."&";

      function "&" (Left : in Lines.Line; Right : in Character)
                    return Lines.Line
      renames Lines."&";

   begin
      case Get_Level (Current_Segment) is
         when Segments.Data =>
            loop
               if Ada.Text_IO.End_Of_File (File => File) then
                  Next_Segment :=
                    To_Segment (Segments.Txt.Meta.Create
                                (Separator =>
                                   Segments.Txt.Meta.Vertical_Space,
                                 Size => 0));
                  exit;
               end if;  -- Ada.Text_IO.End_Of_File (File => File)
               Lines.Read (File => File, Object => Data_Input);
               Segments.Clean_Input(Data_Input);
               case Has_Signs (Data_Input) is
                  when True =>
                     Current_Content := Get_Content (Current_Segment);
                     case Ada.Strings.Maps.Is_In
                       (Element =>
```

```
                        Lines.Get_Character (Source => Current_Content,
                                             Index =>
                                               Lines.Size
(Current_Content)),
                       Set => Segments.Spacing_Set)
                      is
                       when True =>
                          Set_Content (Object => Current_Segment,
                                       Value => Current_Content &
Data_Input);
                       when False =>
                          Data_Input :=
                            (Get_Content (Current_Segment) &
                             Ada.Characters.Latin_1.Space) &
                            Data_Input;
                          Set_Content (Object => Current_Segment,
                                       Value => Data_Input);
                    end case;  -- ...Is_In (...Spacing_Set)
                 when False =>
                    Next_Segment :=
                      To_Segment (Segments.Txt.Meta.Create
                                  (Separator =>
                                     Segments.Txt.Meta.Vertical_Space,
                                   Size => 1));
                    exit;
              end case;  -- Has_Signs (Data_Input)
           end loop;
        when Segments.Meta =>
           loop
              if Ada.Text_IO.End_Of_File (File => File) then
                 Next_Segment :=
                   To_Segment (Segments.Txt.Meta.Create
                               (Separator =>
                                  Segments.Txt.Meta.Vertical_Space,
                                Size => 0));
                 exit;
              end if;  -- Ada.Text_IO.End_Of_File (File => File)
              Lines.Read (File => File, Object => Data_Input);
              Segments.Clean_Input (Data_Input);
              case Has_Signs (Data_Input) is
                 when True =>
                    if Has_Blank_Beginning (Data_Input) then
                       Lines.Delete (Source => Data_Input,
                                     From => 1, To => 1);
                    end if;  -- Has_Blank_Beginning (Data_Input)
                    Next_Segment :=
                      To_Segment (Segments.Txt.Data.Create (Content =>

Data_Input));
                    exit;
                 when False =>
                    Set_Size (Object => Current_Segment,
                              Value => Get_Size (Current_Segment) + 1);
```

```
              end case;  -- Has_Signs (Data_Input)
           end loop;
     end case;  -- Get_Level (Current_Segment)
end Read;

procedure Read_First (Current_Segment :
                        out Segments.Txt.Laid_Out_Ascii.Segment;
                      File : in Ada.Text_IO.File_Type :=
                        Ada.Text_IO.Current_Input;
                      Next_Segment :
                        out Segments.Txt.Laid_Out_Ascii.Segment) is
   Data_Input : Lines.Line;

begin
   Lines.Read (File => File, Object => Data_Input);
   Segments.Clean_Input (Data_Input);
   case Has_Signs (Data_Input) is
      when True =>
         case Has_Blank_Beginning (Data_Input) is
            when True =>
               Current_Segment :=
                 To_Segment (Segments.Txt.Meta.Create
                             (Separator =>
                                Segments.Txt.Meta.Horizontal_Space,
                              Size => 0));
               Next_Segment :=
                 To_Segment (Segments.Txt.Data.Create
                             (Content =>
                                Lines.Delete (Source =>
                                                Data_Input,
                                              From => 1,
                                              To => 1)));
            when False =>
               Current_Segment :=
                 To_Segment (Segments.Txt.Data.Create (Content =>
                                                       Data_Input));
               Read (Current_Segment => Current_Segment,
                     File => File,
                     Next_Segment => Next_Segment);
         end case;  -- Has_Blank_Beginning (Data_Input)
      when False =>
         Current_Segment :=
           To_Segment (Segments.Txt.Meta.Create
                       (Separator =>
                          Segments.Txt.Meta.Vertical_Space,
                        Size => 1));
         case Ada.Text_IO.End_Of_File (File) is
            when True =>
               Next_Segment :=
                 To_Segment (Segments.Txt.Meta.Create
                             (Separator =>
                                Segments.Txt.Meta.Vertical_Space,
                              Size => 0));
```

```
               when False =>
                  Read (Current_Segment => Current_Segment,
                        File => File,
                        Next_Segment => Next_Segment);
            end case;  -- Ada.Text_IO.End_Of_File (File)
      end case;  -- Has_Signs(Data_Input)
   end Read_First;

--   generic
--      with procedure Action_Data (Object : in Segments.Txt.Data.Segment);
--      with procedure Action_Meta (Object : in Segments.Txt.Meta.Segment);
   procedure For_Each_Segment_G (File : in out Ada.Text_IO.File_Type) is
      Current_Segment, Next_Segment : Segments.Txt.Laid_Out_Ascii.Segment;

      procedure Action is
      begin
         case Get_Level (Current_Segment) is
            when Segments.Data =>
               Action_Data (To_Data (Current_Segment));
            when Segments.Meta =>
               Action_Meta (To_Meta (Current_Segment));
         end case;  -- Get_Level (Current_Segment)
         Current_Segment := Next_Segment;
      end Action;

   begin
      Ada.Text_IO.Reset (File => File, Mode => Ada.Text_IO.In_File);
      if not Ada.Text_IO.End_Of_File (File => File) then
-- First segment
         Read_First (Current_Segment => Current_Segment,
                     File => File,
                     Next_Segment => Next_Segment);
         Action;
-- Following segments
         while not Ada.Text_IO.End_Of_File(File) loop
            Read (Current_Segment => Current_Segment,
                  File => File,
                  Next_Segment => Next_Segment);
            Action;
         end loop;  -- while not ...End_Of_File...
-- Last segment
         case Get_Level (Current_Segment) is
            when Segments.Data =>
               Action_Data (To_Data (Current_Segment));
            when Segments.Meta =>
               Action_Meta (To_Meta (Current_Segment));
         end case;  -- Get_Level (Current_Segment)
      end if;  -- if not ...End_Of_File...
   end For_Each_Segment_G;

end Segments.Txt.Laid_Out_Ascii;
```

c) *Les segments dans les fichiers SGML*

Base de définition des segments

```
with Ada.Characters.Latin_1;
with Ada.Strings.Maps;
with Ada.Strings.Maps.Constants;
with Ada.Text_IO;
with Lines;

package Segments.Sgml is

   use Ada;

  type Segment is abstract tagged private;

   Syntax_Error : exception;

   procedure Write (Object : in Segment;
                    File   : in Text_IO.File_Type :=
Text_IO.Current_Output)
      is abstract;

   function Get_Level (Object : in Segment)
     return Segments.Segment_Level
     is abstract;

-------------------------------------------------------------------------

   function Is_Like_Name (Object : in String) return Boolean;

   procedure Find_Name
     (Source : in     Lines.Line;
      From   :    out Positive;
      To     :    out Natural;
      Going  : in     Strings.Direction := Strings.Forward);

   function Find_Name
     (Source  : in Lines.Line;
      Object  : in String;
      Going   : in Strings.Direction := Ada.Strings.Forward;
      Mapping : in Strings.Maps.Character_Mapping := Strings.Maps.Identity)
                      return Natural;

-------------------------------------------------------------------------

private

   type Segment is abstract tagged null record;
```

```
   procedure Set_Segment (Object : in out Segment);

------------------------------------------------------------------------
-- tags --

   Stago : constant String :=
--      "<";
     String'(1=>Ada.Characters.Latin_1.Less_Than_Sign);
-- SGML "start-tag open"

   Etago : constant String :=
--      "</";
     String'(1=>Ada.Characters.Latin_1.Less_Than_Sign,
             2=>Ada.Characters.Latin_1.Solidus);
-- SGML "end-tag open"

   Tagc : constant String :=
--      ">";
     String'(1=>Ada.Characters.Latin_1.Greater_Than_Sign);
-- SGML "tag close"

------------------------------------------------------------------------
-- Characters sets --

   function "or" (Left, Right : in Ada.Strings.Maps.Character_Set)
                  return Ada.Strings.Maps.Character_Set
     renames Ada.Strings.Maps."or";

   Name_Character_Set : constant Ada.Strings.Maps.Character_Set :=
     Ada.Strings.Maps.Constants.Alphanumeric_Set
     or Ada.Strings.Maps.To_Set(Ada.Characters.Latin_1.Full_Stop)
     or Ada.Strings.Maps.To_Set(Ada.Characters.Latin_1.Hyphen);

   Name_Start_Character_Set : constant Ada.Strings.Maps.Character_Set :=
     Ada.Strings.Maps.Constants.Basic_Set;
-- SGML states that Name_Start_Character_Set is a subset of
Name_Character_Set.

   Digit_Set : constant Ada.Strings.Maps.Character_Set :=
     Ada.Strings.Maps.Constants.Decimal_Digit_Set;

------------------------------------------------------------------------

end Segments.Sgml;

========================================================================
========================================================================

with Ada.Strings;
with Decid_Maps;
with Logs;

package body Segments.Sgml is

   procedure Set_Segment (Object : in out Segment) is
   begin
      null;
   end Set_Segment;

------------------------------------------------------------------------

   function Is_Like_Name (Object : in String) return Boolean is
   begin
     case Object'Length is
        when 0 =>
           return False;
        when others =>
           return
             Ada.Strings.Maps.Is_In (Element => Object (Object'First),
                                     Set => Name_Start_Character_Set)
             and
             Decid_Maps.Has_Characters (Object =>
                                          Object
(Object'First+1..Object'Last),
                                        In_Set => Name_Character_Set);
     end case;  -- Object'Length
   end Is_Like_Name;

   procedure Find_With_First_Character (Source : in Lines.Line;
                                        First_Character_Set :
                                          in
Ada.Strings.Maps.Character_Set;
                                        Following_Characters_Set :
                                          in
Ada.Strings.Maps.Character_Set;
                                        From : out Positive;
                                        To : out Natural;
                                        Going : in Ada.Strings.Direction :=
                                          Ada.Strings.Forward) is
      Low : Positive;
   begin
      Lines.Find_Pattern (Source => Source,
                          Set => First_Character_Set,
                          Test => Ada.Strings.Inside,
                          Going => Going,
                          From => From,
                          To => To);
      if To > 0 then
         case Going is
            when Ada.Strings.Forward =>
               Lines.Find_Pattern (Source =>
                                     Lines.Delete (Source => Source,
                                                   From => 1,
                                                   To => From),
                                   Set => Following_Characters_Set,
                                   Test => Ada.Strings.Inside,
```

```
                                Going => Ada.Strings.Forward,
                                From => Low,
                                To => To);
               case Low = 1 is
                  when True =>
                     To := To + From;
                  when False =>
                     To := From;
               end case;  -- Low = 1
            when Ada.Strings.Backward =>
               Low := To;
               Lines.Find_Pattern (Source => Source,
                                   Set => Following_Characters_Set,
                                   Test => Ada.Strings.Inside,
                                   Going => Ada.Strings.Backward,
                                   From => From,
                                   To => To);
               case From <= Low +1 and Low <= To is
                  when True =>
                     case From > 1 is
                        when True =>
                           Lines.Find_Pattern
                             (Source => Lines.Delete (Source => Source,
                                                      From => 1,
                                                      To => From -2),
                              Set => First_Character_Set,
                              Test => Ada.Strings.Inside,
                              Going => Ada.Strings.Forward,
                              From => From,
                              To => Low);
                        when False =>
                           Lines.Find_Pattern (Source => Source,
                                               Set => First_Character_Set,
                                               Test => Ada.Strings.Inside,
                                               Going =>
Ada.Strings.Forward,
                                               From => From,
                                               To => Low);
                     end case;  -- From > 1
                  when False =>
                     From := Low;
                     To := Low;
               end case;  -- From <= Low +1 and Low < To
         end case;  -- Going
      end if;  -- To > 0
   end Find_With_First_Character;


   procedure Find_Name (Source : in Lines.Line;
                        From : out Positive;
                        To : out Natural;
                        Going : in Ada.Strings.Direction :=
                          Ada.Strings.Forward) is
   begin
      Find_With_First_Character (Source => Source,
                                 First_Character_Set =>
                                   Name_Start_Character_Set,
                                 Following_Characters_Set =>
                                   Name_Character_Set,
                                 From => From,
                                 To => To,
                                 Going => Going);
   end Find_Name;

   function Find_Name (Source : in Lines.Line;
                       Object : in String;
                       Going : in Ada.Strings.Direction :=
                         Ada.Strings.Forward;
                       Mapping : in Ada.Strings.Maps.Character_Mapping :=
                         Ada.Strings.Maps.Identity)
                       return Natural is
      Low, High : Natural;
   begin
      if not Is_Like_Name (Object) then
         Logs.Write (Kind => Logs.Warning,
                     Localization => "function Segments.Sgml.Find_Name");
         Logs.Write (Message => "The string " & Object &
                     " to be searched is not a SGML name " &
                     "([A-Za-z][0-9A-Za-z.-]*).",
                     Ends_The_Line => True);
      end if;
      Low := Lines.Find_String (Source => Source, Object => Object,
                                Going => Going, Mapping => Mapping);
      if Low > 0 then  -- checking of the borders : Object should not be
                       -- the part of a name in Source
         High := Low + Object'Length;
         if High <= Lines.Size (Source) and then
           Ada.Strings.Maps.Is_In (Element =>
                                     Lines.Get_Character (Source => Source,
                                                          Index => High),
                                   Set => Name_Character_Set) then
            return 0;
         end if;  -- High <= Lines.Size (Source) and...
         if Low > 1 and then
           Ada.Strings.Maps.Is_In (Element =>
                                     Lines.Get_Character (Source => Source,
                                                          Index => Low -1),
                                   Set => Name_Character_Set) then
            return 0;
         end if;  -- Low > 1
      end if;  -- Low > 0
      return Low;
   end Find_Name;

end Segments.Sgml;
```

Les segments de type data : contenu textuel

```
with Ada.Characters.Latin_1;
with Ada.Text_IO;
with Lines;

package Segments.Sgml.Data is

   use Ada;

   type Segment is new Segments.Sgml.Segment with private;

   Unknown_Sgml_Entity : exception;

   function Create (Content : in Lines.Line) return Segment;

   procedure Write (Object : in Segment;
                    File   : in Text_IO.File_Type :=
Text_IO.Current_Output);

   procedure Write
     (Object            : in Segment;
      Entities_Resolved : in Boolean;
      File              : in Text_IO.File_Type := Text_IO.Current_Output);

   function Get_Level (Object : in Segment) return Segments.Segment_Level;

   function Get_Content (Object            : in Segment;
                         Entities_Resolved : in Boolean)
                         return Lines.Line;

   procedure Set_Content (Object : in out Segment;
                          Value  : in     Lines.Line);

private

------------------------------------------------------------------------------

   Ero : constant String := String'(1 => Ada.Characters.Latin_1.Ampersand);
-- SGML "opens a named entity reference"
   Refc : constant String := String'(1 =>
Ada.Characters.Latin_1.Semicolon);
-- SGML "closes a reference"

------------------------------------------------------------------------------

   type Segment is new Segments.Sgml.Segment with
      record
         Content : Lines.Line := Lines.Null_Decid_String;
      end record;

   procedure Set_Segment (Object : in out Segment;
                          Content : in Lines.Line);

end Segments.Sgml.Data;

==============================================================================
==============================================================================

with Ada.IO_Exceptions;
with Logs;
with Parameters;
with Read_File;
with Sizes;
with Table_Of_Strings_And_Static_Values_G;

package body Segments.Sgml.Data is

   package Entities is
     new Table_Of_Strings_And_Static_Values_G (Character_Type => Character,
                                               String_Type    => String,
                                               Less           => "<",
                                               Equals         => "=",
                                               Value_Type     =>
Lines.Line);

   package Reader is new Read_File
     (Max_Field       => Sizes.S_Value,
      Max_Line_Length => Lines.Decid_Strings.Standard_Max_Size);

   Entity_Data_Name : String :=
     Parameters.Sgml_Entities_Data_File_Name;

-- description of the format for entity data :
-- * field separator is equal sign (=)
-- * every data line has got at least 5 fields
-- * only the second and the last field may be empty
--   (but they still have to exist).
-- 1st field is the name of the SGML entity
-- 2nd field is the replacement string, which translates the entity
-- 3rd field is the ISO character set considered (capitalized)
--     cf. Iso_Set type
-- 4th to last but one field are Latin-1 codes
--     for characters of replacement string
-- last field is the current name to refer to the entity

   Entity_To_Diacritic_Table : Entities.Table_Type;
   Entity_To_Diacritic_Table_Is_Initialized : Boolean := False;


   procedure Init_Entities is

      Duplicated_Entries : Boolean := False;
      Last : Natural;
      Replacement : Lines.Line;
```

```
   begin
      if Entities.Is_Empty (Table => Entity_To_Diacritic_Table) then
         Reader.Open (Name => Entity_Data_Name);
         Reader.Set_Separators (To => "=");
         while not Reader.End_Of_File loop
            Reader.Fetch;
            Last := Reader.Number_Of_Fields;
            if Last > 1 then
               Lines.Reset (Replacement);
               for I in 4..Last-1 loop
                  Replacement :=
                    Lines.Append (Left => Replacement,
                                  Right => Character'Val
(Reader.Field(I)));
               end loop;  --  for
               Entities.Insert (Table => Entity_To_Diacritic_Table,
                                Duplicate_Item => Duplicated_Entries,
                                Key => Reader.Field(1),
                                Value => Replacement);
               if Duplicated_Entries then
                  Logs.Write (Kind => Logs.Warning,
                              Localization =>
                               "Segments.Data_Sgml.Init_Entities");
                  Logs.Write (Message => "The entity " & Reader.Field(1) &
                              " is duplicated in file " &
                              Entity_Data_Name &
                              " ; (only the first translation is
considered).",
                              Ends_The_Line => True);
               end if;
            end if;
         end loop;  --  while
         Reader.Close;
         Entity_To_Diacritic_Table_Is_Initialized := True;
      end if;
   exception
      when Ada.IO_Exceptions.Name_Error =>
         Logs.Write (Kind => Logs.Error,
                     Localization =>
                       "procedure Segments.Data_Sgml.Init_Entities",
                     Exception_Name => "Ada.IO_Exceptions.Name_Error",
                     Propagated => True);
         Logs.Write (Message => "File " & Entity_Data_Name & "containing "
&
                     "data about SGML entities has not been found.",
                     Ends_The_Line => True);
         raise;
   end Init_Entities;


   function Translate_Entity (Name : in String) return Lines.Line is
   begin
      case Name (1) = Ada.Characters.Latin_1.Number_Sign is
         when True =>
            return Lines.To_Decid_String
              (String'(1 =>
                           Character'Val (Integer'Value (Name
(2..Name'Last)))));
         when False =>
            return Entities.Value (Table => Entity_To_Diacritic_Table,
                                   Key => Name);
      end case;  -- Name (1) = Ada.Characters.Latin_1.Number_Sign
   exception
      when Entities.Missing_Item_Error =>
         case Entity_To_Diacritic_Table_Is_Initialized is
            when True =>
               Logs.Write (Kind => Logs.Warning,
                           Localization =>
                             "function
Segments.Data_Sgml.Translate_Entity",
                           Exception_Name =>

"Segments.Data_Sgml.Entities.Missing_Item_Error",
                           Propagated => True,
                           Propagation_Name =>
                             "Segments.Data_Sgml.Unknown_Sgml_Entity");
               Logs.Write (Message => "Entity &" & Name &
                           "; is not described in the entity data file " &
                           Entity_Data_Name &
                           " ; it has not been translated.",
                           Ends_The_Line => True);
               raise Unknown_Sgml_Entity;
            when False =>
               Init_Entities;
               return Translate_Entity (Name => Name);
         end case;
   end Translate_Entity;


   function Translate_Entities (Object : in Lines.Line)
                                return Lines.Line
   is
      Not_Yet_Read : Lines.Line := Object;
      Result       : Lines.Line := Lines.Null_Decid_String;

      Ero_Position, Refc_Position, Cutting_Position : Natural;

   begin
      while not Lines.Is_Empty (Not_Yet_Read) loop
         Ero_Position := Lines.Find_String (Source => Not_Yet_Read,
                                            Object => Ero,
                                            Going  => Ada.Strings.Forward);
         exit when Ero_Position = 0;

         Cutting_Position := Ero_Position-1;
```

```
         Result :=
           Lines.Append (Left => Result,
                         Right => Lines.Chunk (Source => Not_Yet_Read,
                                               From   => 1,
                                               To     =>
Cutting_Position));

         Lines.Delete (Source => Not_Yet_Read,
                       From   => 1,
                       To     => Cutting_Position);

         Refc_Position := Lines.Find_String (Source => Not_Yet_Read,
                                             Object => Refc,
                                             Going => Ada.Strings.Forward);

         exit when Refc_Position = 0;

         Cutting_Position := Refc_Position-1;

         begin
            Result :=
              Lines.Append (Left => Result,
                            Right =>
                Translate_Entity (Lines.Get_String (Source =>
                                                    Not_Yet_Read,
                                                  From =>
                                                    Ero'Length+1,
                                                  To =>
                                                    Cutting_Position)));
            Cutting_Position := Cutting_Position + Refc'Length;
         exception
            when Unknown_Sgml_Entity =>
              Cutting_Position := Cutting_Position + Refc'Length;
              Result :=
                Lines.Append
                (Left  => Result,
                 Right => Lines.Chunk (Source => Not_Yet_Read,
                                       From   => 1,
                                       To     => Cutting_Position));
         end;

         Lines.Delete (Source => Not_Yet_Read,
                       From   => 1,
                       To     => Cutting_Position);
      end loop;
      return Lines.Append (Left => Result, Right => Not_Yet_Read);
   end Translate_Entities;

-------------------------------------------------------------------------

   function Get_Level (Object : in Segment) return Segments.Segment_Level
is
   begin
      return Segments.Data;
   end Get_Level;

   function Get_Content (Object            : in Segment;
                         Entities_Resolved : in Boolean)
                         return Lines.Line is
   begin
      case Entities_Resolved is
         when True =>
            return Translate_Entities (Object.Content);
         when False =>
            return Object.Content;
      end case;  -- Entities_Resolved
   end Get_Content;

   procedure Set_Content (Object : in out Segment;
                          Value  : in     Lines.Line) is
   begin
      Object.Content := Value;
   end Set_Content;

-------------------------------------------------------------------------

   procedure Set_Segment (Object  : in out Segment;
                          Content : in     Lines.Line) is
   begin
      Set_Segment (Object => Segments.Sgml.Segment (Object));
      Object.Content := Content;
   end Set_Segment;

   function Create (Content : in Lines.Line) return Segment is
      Result : Segment;
   begin
      Set_Segment (Object => Result, Content => Content);
      return Result;
   end Create;

   procedure Write
     (Object            : in Segment;
      Entities_Resolved : in Boolean;
      File              : in Text_IO.File_Type := Text_IO.Current_Output)
is
   begin
      Ada.Text_IO.Put
        (File => File,
         Item => Lines.To_String (Get_Content
                                  (Object            => Object,
                                   Entities_Resolved =>
Entities_Resolved)));
   end Write;

   procedure Write
```

```
     (Object : in Segment;
      File   : in Text_IO.File_Type := Text_IO.Current_Output) is
   begin
      Write (Object            => Object,
             Entities_Resolved => False,
             File              => File);
   end Write;

end Segments.Sgml.Data;
```

Les segments de type méta : les balises

```
with Lines;
with Sgml_Gis;

generic
   type Tag_Category is (<>);
   with function To_Tag (Object : in Sgml_Gis.Generic_Identifier)
     return Tag_Category is <>;

package Segments.Sgml.Meta_G is

   type Segment_G is abstract new Segments.Sgml.Segment with private;

   procedure Write (Tag : in Tag_Category;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output) is abstract;

   function Get_Level (Object : in Segment_G)
                       return Segments.Segment_Level;

   function Is_Start_Tag (Object : in Segment_G)
                          return Boolean is abstract;

   function Get_Tag (Object : in Segment_G)
                     return Tag_Category;

-------------------------------------------------------------------------------

   Empty_Tag : Tag_Category := To_Tag (Sgml_Gis.Null_Gi);

   function Is_Empty_Tag (Object : in Tag_Category) return Boolean;

-------------------------------------------------------------------------------

private

   type Segment_G is abstract new Segments.Sgml.Segment with
      record
         Gi : Sgml_Gis.Generic_Identifier := Sgml_Gis.Null_Gi;
         Tag : Tag_Category := Empty_Tag;
      end record;

   procedure Set_Segment (Object : in out Segment_G;
                          Content : in Lines.Line;
                          Index_Gi_End : out Natural);

   procedure Set_Segment (Object : in out Segment_G;
                          Tag : in Tag_Category);

end Segments.Sgml.Meta_G;

===============================================================================
===============================================================================

with Ada.Strings;

package body Segments.Sgml.Meta_G is

-------------------------------------------------------------------------------
   function Is_Empty_Tag (Object : in Tag_Category) return Boolean is
   begin
      return Object = Empty_Tag;
   end Is_Empty_Tag;

-------------------------------------------------------------------------------

   function Get_Level (Object : in Segment_G)
                       return Segments.Segment_Level is
   begin
      return Segments.Meta;
   end;

   function Get_Tag (Object : in Segment_G)
                     return Tag_Category is
   begin
      return Object.Tag;
   end Get_Tag;

   procedure Set_Segment (Object : in out Segment_G;
                          Content : in Lines.Line;
                          Index_Gi_End : out Natural) is
      Index_Gi_Beginning : Positive;
      Tag_Gi : Sgml_Gis.Generic_Identifier;
   begin
      Segments.Sgml.Find_Name (Source => Content,
                               From => Index_Gi_Beginning,
                               To => Index_Gi_End,
                               Going => Ada.Strings.Forward);
      Set_Segment (Object => Segments.Sgml.Segment (Object));
      Tag_Gi :=
        Sgml_Gis.To_Gi (Lines.Get_String (Source => Content,
                                          From => Index_Gi_Beginning,
                                          To => Index_Gi_End));
```

```
      Object.Gi := Tag_Gi;
      Object.Tag := To_Tag (Tag_Gi);
   end Set_Segment;

   procedure Set_Segment (Object : in out Segment_G;
                          Tag : in Tag_Category) is
   begin
      Set_Segment (Object => Segments.Sgml.Segment (Object));
      Object.Gi := Sgml_Gis.To_Gi (Tag_Category'Image (Tag));
      Object.Tag := Tag;
   end Set_Segment;

end Segments.Sgml.Meta_G;
```

Balises ouvrantes

```
with Ada.Text_IO;
with Lines;
with Paths;
with Sgml_Gis;

generic
package Segments.Sgml.Meta_G.Stag_G is

   type Segment is new Segment_G with private;

   function Create (Content : in Lines.Line)
                    return Segment;

   function Create (Tag : in Tag_Category)
                    return Segment;

   procedure Write (Object : in Segment;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output);

   procedure Write (Tag : in Tag_Category;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output);

   function Is_Start_Tag (Object : in Segment) return Boolean;

   procedure Get_Attribute (Object : in Segment;
                            Name : in Sgml_Gis.Generic_Identifier;
                            Value : out Lines.Line;
                            Is_Present : out Boolean);

   procedure Get_Attribute (Object : in Segment;
                            Name : in Sgml_Gis.Generic_Identifier;
                            Value : out Paths.Path;
                            Is_Present : out Boolean);

   procedure Get_Attribute (Object : in Segment;
                            Name : in String;
                            Value : out Lines.Line;
                            Is_Present : out Boolean);

   procedure Get_Attribute (Object : in Segment;
                            Name : in String;
                            Value : out Paths.Path;
                            Is_Present : out Boolean);

   procedure Set_Attribute (Object : in out Segment;
                            Name : in Sgml_Gis.Generic_Identifier;
                            Value : in Lines.Line;
                            Add_If_Not_Present : in Boolean);

   procedure Set_Attribute (Object : in out Segment;
                            Name : in Sgml_Gis.Generic_Identifier;
                            Value : in Paths.Path;
                            Add_If_Not_Present : in Boolean);

private

   type Segment is new Segment_G with
      record
         Attributes : Lines.Line := Lines.Null_Decid_String;
      end record;

   procedure Set_Segment (Object : in out Segment;
                          Content : in Lines.Line);

   procedure Set_Segment (Object : in out Segment;
                          Tag : in Tag_Category);

end Segments.Sgml.Meta_G.Stag_G;

================================================================================
================================================================================

with Ada.Characters.Latin_1;
with Ada.Strings;
with Ada.Strings.Maps;
with Logs;

package body Segments.Sgml.Meta_G.Stag_G is

--------------------------------------------------------------------------------
-- Attributes --

   Vi : constant String :=
     String'(1=>Ada.Characters.Latin_1.Equals_Sign);
-- SGML "value indicator"
```

```
   Lit : constant String :=
     String'(1=>Ada.Characters.Latin_1.Quotation);
-- SGML "literal delimiter"

   Lita : constant String :=
     String'(1=>Ada.Characters.Latin_1.Apostrophe);
-- SGML "literal delimiter (alternative)"

------------------------------------------------------------------------
   procedure Find_Attribute (Object : in Lines.Line;
                             Name : in Sgml_Gis.Generic_Identifier;
                             Name_From : out Natural;
                             Value : out Lines.Line;
                             Value_From : out Natural) is
-- Name_From = 0 indicates that the attribute has not been found
      Field_Beginning, Field_End, Index, Index_Alternate : Natural;
      Index_Translation : Natural := 0;

      procedure Move_Forward (To : in Natural) is
      begin
         Lines.Delete (Source => Value, From => 1, To => To);
         Index_Translation := Index_Translation + To;
      end Move_Forward;

      procedure Find_Value (Literal_Delimiter : in String) is
      begin
         if not Lines.Has_Characters (Object =>
                                        Lines.Chunk (Source => Value,
                                                     From => 1,
                                                     To => Index -1),
                                      In_Set =>
                                        Segments.No_Sign_Set) then
            raise Syntax_Error;  -- signs between VI and LIT
         end if;
         Value_From := Index_Translation +
           Index + Literal_Delimiter'Length -1;
         Move_Forward (To => Value_From - Index_Translation);
         Index := Lines.Find_String (Source => Value,
                                     Object => Literal_Delimiter,
                                     Going => Ada.Strings.Forward);
         if Index = 0 then  -- LIT is not closed
            raise Syntax_Error;
         end if;
         Value := Lines.Chunk (Source => Value,
                               From => 1, To => Index -1);
      end Find_Value;

   begin
      Value := Object;
      loop
         Field_End := Lines.Find_String ( Source => Value,
                                          Object => Vi,
                                          Going => Ada.Strings.Forward);
         if Field_End = 0 then  -- no more attributes
            Name_From := 0;
            Value_From := 0;
            exit;
         end if;  -- Field_end = 0
         Field_Beginning :=
           Sgml_Gis.Find_Gi (Source => Lines.Chunk (Source => Value,
                                                    From => 1,
                                                    To => Field_End),
                             Object => Name,
                             Going => Ada.Strings.Backward);
         if Field_Beginning > 0 and then
           Lines.Has_Characters (Object =>
                                   Lines.Chunk (Source => Value,
                                                From =>
                                                  Field_Beginning +
                                                Sgml_Gis.Get_Size (Name),
                                                To => Field_End -1),
                                 In_Set => Segments.No_Sign_Set) then
-- no significant character between the attribute name and the value
indicator
-- either the attribute is present, or there is a SGML syntax error
            Name_From := Field_Beginning + Index_Translation;
            Move_Forward (To => Field_End);
            Field_End := Lines.Find_String ( Source => Value,
                                             Object => Vi,
                                             Going => Ada.Strings.Forward);
            Index :=
              Lines.Find_String (Source => Value,
                                 Object => Lit,
                                 Going => Ada.Strings.Forward);
            Index_Alternate :=
              Lines.Find_String (Source => Value,
                                 Object => Lita,
                                 Going => Ada.Strings.Forward);
            if 0 < Index and
              (Index < Index_Alternate or Index_Alternate = 0) and
              (Index < Field_End or Field_End = 0) then
               Find_Value (Literal_Delimiter => Lit);
            elsif 0 < Index_Alternate and
              (Index_Alternate < Index or Index = 0) and
              (Index_Alternate < Field_End or Field_End = 0) then
               Index := Index_Alternate;
               Find_Value (Literal_Delimiter => Lita);
            else  -- Field_End is the smallest > 0 (or all are null)
               if Field_End > 0 then
                  Value := Lines.Chunk (Source => Value,
                                        From => 1, To => Field_End -1);
                  Segments.Sgml.Find_Name (Source => Value,
                                           From => Field_Beginning,
                                           To => Field_End,
                                           Going => Ada.Strings.Backward);
                  case Field_End > 0 is
```

```
                  when True =>
                     Value := Lines.Chunk (Source => Value,
                                           From => 1,
                                           To => Field_Beginning -1);
                  when False =>
                     raise Syntax_Error;
               end case;  -- Field_End > 0
            end if;  -- Field_End > 0
            Segments.Sgml.Find_Name (Source => Value,
                                     From => Value_From,
                                     To => Field_End,
                                     Going => Ada.Strings.Forward);
            Value := Lines.Chunk (Source => Value,
                                  From => Value_From,
                                  To => Field_End);
         end if;  -- Index / Index_Alternate / Field_End
         exit;
      else  -- the attribute is not present in the field
         Move_Forward (To => Field_End);
      end if;  -- Field_Beginning > 0 and then Is_Subset...
   end loop;
exception
   when Syntax_Error =>
      Logs.Write (Kind => Logs.Warning,
                  Localization => "procedure
Segments.Sgml.Meta.Get_Attr",
                  Exception_Name => "Segments.Sgml.Meta.Syntax_Error",
                  Propagated => False);
      Logs.Write (Message => "The attribute " & Sgml_Gis.Get_Image
(Name) &
                  " may not be correctly delimited by a couple of " &
                  Lit & " or " & Lita & " in the tag :",
                  Ends_The_Line => True);
      Logs.Write (Data =>
                    Stago & Lines.To_String (Object) & Tagc);
      Name_From := 0;
      Value_From := 0;
end Find_Attribute;

--------------------------------------------------------------------------

function Is_Start_Tag (Object : in Segment) return Boolean is
begin
   return True;
end Is_Start_Tag;

procedure Get_Attribute (Object : in Segment;
                         Name : in Sgml_Gis.Generic_Identifier;
                         Value : out Lines.Line;
                         Is_Present : out Boolean) is
   Name_Position, Value_Position : Natural;
begin
   Find_Attribute (Object => Object.Attributes,
                   Name => Name,
                   Name_From => Name_Position,
                   Value => Value,
                   Value_From => Value_Position);
   Is_Present := (Name_Position > 0);
end Get_Attribute;

procedure Get_Attribute (Object : in Segment;
                         Name : in Sgml_Gis.Generic_Identifier;
                         Value : out Paths.Path;
                         Is_Present : out Boolean) is
   Path_As_A_Line : Lines.Line;
begin
   Get_Attribute (Object => Object,
                  Name => Name,
                  Value => Path_As_A_Line,
                  Is_Present => Is_Present);
   case Is_Present is
      when True =>
         Value := Paths.To_Decid_String (Lines.To_String
(Path_As_A_Line));
      when False =>
         Value := Paths.Null_Decid_String;
   end case;  -- Is_Present
end Get_Attribute;

procedure Get_Attribute (Object : in Segment;
                         Name : in String;
                         Value : out Lines.Line;
                         Is_Present : out Boolean) is
begin
   Get_Attribute (Object => Object,
                  Name => Sgml_Gis.To_Gi (Name),
                  Value => Value,
                  Is_Present => Is_Present);
end Get_Attribute;

procedure Get_Attribute (Object : in Segment;
                         Name : in String;
                         Value : out Paths.Path;
                         Is_Present : out Boolean) is
begin
   Get_Attribute (Object => Object,
                  Name => Sgml_Gis.To_Gi (Name),
                  Value => Value,
                  Is_Present => Is_Present);
end Get_Attribute;

procedure Set_Attribute (Object : in out Segment;
                         Name : in Sgml_Gis.Generic_Identifier;
                         Value : in Lines.Line;
                         Add_If_Not_Present : in Boolean) is
   Name_Position, Value_Position : Natural;
```

```
      Old_Value : Lines.Line;
   begin
      Find_Attribute (Object => Object.Attributes,
                      Name => Name,
                      Name_From => Name_Position,
                      Value => Old_Value,
                      Value_From => Value_Position);
      case Name_Position is
         when 0 =>
            if Add_If_Not_Present then
               Object.Attributes :=
                 Lines.Append (Left => Object.Attributes,
                               Right => " " & Sgml_Gis.Get_Image (Name) &
                               " " & Vi & " " & Lit &
                               Lines.To_String (Value) & Lit);
            end if;  -- Add_If_Not_Present
         when others =>
            Object.Attributes :=
              Lines.Replace (Source => Object.Attributes,
                             From => Value_Position,
                             To => Value_Position + Lines.Size (Old_Value)
-1,
                             By => Value);
      end case;  -- Name_Position
   end Set_Attribute;

   procedure Set_Attribute (Object : in out Segment;
                            Name : in Sgml_Gis.Generic_Identifier;
                            Value : in Paths.Path;
                            Add_If_Not_Present : in Boolean) is
   begin
      Set_Attribute (Object => Object,
                     Name => Name,
                     Value => Lines.To_Decid_String
(Paths.To_String(Value)),
                     Add_If_Not_Present => Add_If_Not_Present);
   end Set_Attribute;

   procedure Set_Segment (Object : in out Segment;
                          Content : in Lines.Line) is
      Index : Natural;
   begin
      Set_Segment (Object => Segment_G (Object), Content => Content,
                   Index_Gi_End => Index);
      Object.Attributes := Lines.Delete (Source => Content,
                                         From => 1,
                                         To => Index);
      if Index = 0 then
         raise Segments.Sgml.Syntax_Error;
      end if;  -- Index = 0
   exception
      when Segments.Sgml.Syntax_Error =>
         Logs.Write (Kind => Logs.Warning,
                     Localization => "Segments.Sgml.Meta_G.Stag",
                     Exception_Name => "Segments.Sgml.Syntax_Error",
                     Propagated => False);
         Logs.Write (Message => "Empty open tag found : " &
                     "the tag is ignored.",
                     Ends_The_Line => True);
   end Set_Segment;

   procedure Set_Segment (Object : in out Segment;
                          Tag : in Tag_Category) is
   begin
      Set_Segment (Object => Segment_G (Object), Tag => Tag);
   end Set_Segment;

   function Create (Content : in Lines.Line)
                    return Segment is
      Result : Segment;
   begin
      Set_Segment (Object => Result, Content => Content);
      return Result;
   end Create;

   function Create (Tag : in Tag_Category)
                    return Segment is
      Result : Segment;
   begin
      Set_Segment (Object => Result, Tag => Tag);
      return Result;
   end Create;

-------------------------------------------------------------------------

   procedure Write (Object : in Segment;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output) is
   begin
      Ada.Text_IO.Put (File => File,
                       Item => Segments.Sgml.Stago &
                       Tag_Category'Image (Get_Tag (Object)) &
                       Lines.To_String (Object.Attributes) &
                       Segments.Sgml.Tagc);
   end Write;

   procedure Write (Tag : in Tag_Category;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output) is
   begin
      Ada.Text_IO.Put (File => File,
                       Item => Segments.Sgml.Stago &
                       Tag_Category'Image (Tag) &
                       Segments.Sgml.Tagc);
   end Write;
```

```
end Segments.Sgml.Meta_G.Stag_G;
```

Balises fermantes

```
with Ada.Text_IO;
with Lines;

generic
package Segments.Sgml.Meta_G.Etag_G is

   type Segment is new Segment_G with private;

   function Create (Content : in Lines.Line) return Segment;

   function Create (Tag : in Tag_Category)
                    return Segment;

   procedure Write (Object : in Segment;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output);

   procedure Write_Short (Object : in Segment;
                          File : in Ada.Text_IO.File_Type :=
                            Ada.Text_IO.Current_Output);

   procedure Write (Tag : in Tag_Category;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output);

   procedure Write_Short (Tag : in Tag_Category;
                          File : in Ada.Text_IO.File_Type :=
                            Ada.Text_IO.Current_Output);

   function Is_Start_Tag (Object : in Segment)
                          return Boolean;

private

   type Segment is new Segment_G with null record;

   procedure Set_Segment (Object : in out Segment;
                          Content : in Lines.Line);

   procedure Set_Segment (Object : in out Segment;
                          Tag : in Tag_Category);

end Segments.Sgml.Meta_G.Etag_G;

================================================================================
================================================================================
```

```
package body Segments.Sgml.Meta_G.Etag_G is

   function Is_Start_Tag (Object : in Segment)
                          return Boolean is
   begin
      return False;
   end Is_Start_Tag;

   procedure Set_Segment (Object : in out Segment;
                          Content : in Lines.Line) is
      Index : Natural;
   begin
      Set_Segment (Object => Segment_G (Object), Content => Content,
                   Index_Gi_End => Index);
   end Set_Segment;

   procedure Set_Segment (Object : in out Segment;
                          Tag : in Tag_Category) is
   begin
     Set_Segment (Object => Segment_G (Object), Tag => Tag);
   end Set_Segment;

   function Create (Content : in Lines.Line) return Segment is
      Result : Segment;
   begin
      Set_Segment (Object => Result, Content => Content);
      return Result;
   end Create;

   function Create (Tag : in Tag_Category)
                    return Segment is
      Result : Segment;
   begin
      Set_Segment (Object => Result, Tag => Tag);
      return Result;
   end Create;

   procedure Write (Object : in Segment;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output) is
   begin
      Ada.Text_IO.Put_Line (File => File,
                            Item => Segments.Sgml.Etago &
                            Tag_Category'Image (Get_Tag (Object)) &
                            Segments.Sgml.Tagc);
   end Write;

   procedure Write (Tag : in Tag_Category;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output) is
   begin
      Ada.Text_IO.Put_Line (File => File,
                            Item => Segments.Sgml.Etago &
```

```
                           Tag_Category'Image (Tag) &
                           Segments.Sgml.Tagc);
   end Write;

   procedure Write_Short (Object : in Segment;
                          File : in Ada.Text_IO.File_Type :=
                            Ada.Text_IO.Current_Output) is
   begin
      Ada.Text_IO.Put (File => File,
                       Item => Segments.Sgml.Etago &
                       Tag_Category'Image (Get_Tag (Object)) &
                       Segments.Sgml.Tagc);
   end Write_Short;

   procedure Write_Short (Tag : in Tag_Category;
                          File : in Ada.Text_IO.File_Type :=
                            Ada.Text_IO.Current_Output) is
   begin
      Ada.Text_IO.Put (File => File,
                       Item => Segments.Sgml.Etago &
                       Tag_Category'Image (Tag) &
                       Segments.Sgml.Tagc);
   end Write_Short;

end Segments.Sgml.Meta_G.Etag_G;
```

Segments de type indéfini (data ou méta)

```
with Ada.Text_IO;
with Lines;
with Polymorphic_Objects_G;
with Segments.Sgml.Data;
with Segments.Sgml.Meta_G;
with Segments.Sgml.Meta_G.Stag_G;
with Segments.Sgml.Meta_G.Etag_G;
with Sizes;

generic

   with package Meta is new Segments.Sgml.Meta_G (<>);
   with package Stag is new Meta.Stag_G (<>);
   with package Etag is new Meta.Etag_G (<>);

package Segments.Sgml.Mixed_G is

   use Ada;

   package Data renames Segments.Sgml.Data;

------------------------------------------------------------------------------

   package Polymorphic_Objects is
     new Polymorphic_Objects_G (Object_Root => Segments.Sgml.Segment);

   subtype Segment is Polymorphic_Objects.Polymorphic_Object;

------------------------------------------------------------------------------

--   function To_Segment (Object : in Data.Segment)
--                        return Segment;
   function To_Segment is
     new Polymorphic_Objects.To_Polymorphic_G (Object_Child =>
Data.Segment);

--   function To_Segment (Object : in Stag.Segment)
--                        return Segment;
   function To_Segment is
     new Polymorphic_Objects.To_Polymorphic_G (Object_Child =>
Stag.Segment);

--   function To_Segment (Object : in Etag.Segment)
--                        return Segment;
   function To_Segment is
     new Polymorphic_Objects.To_Polymorphic_G (Object_Child =>
Etag.Segment);

--   function To_Data (Object : in Segment) return Data.Segment;
   function To_Data is
     new Polymorphic_Objects.To_Child_G (Object_Child => Data.Segment);

--   function To_Stag (Object : in Segment)
--                     return Stag.Segment;
   function To_Stag is
     new Polymorphic_Objects.To_Child_G (Object_Child => Stag.Segment);

--   function To_Etag (Object : in Segment)
--                     return Etag.Segment;
   function To_Etag is
     new Polymorphic_Objects.To_Child_G (Object_Child => Etag.Segment);

   function Get_Level (Object : in Segment)
                       return Segments.Segment_Level;

   function Is_Start_Tag (Object : in Segment) return Boolean;

   procedure Write_Short
     (Tag           : in Meta.Tag_Category;
      Text          : in Lines.Line;
      Ends_The_Unit : in Boolean;
      File          : in Text_IO.File_Type := Text_IO.Current_Output);

   function Stag_To_Etag (Object : in Stag.Segment)
                          return Etag.Segment;
```

```
------------------------------------------------------------------------------

   generic
      with procedure Action_Data (Object : in Data.Segment);
      with procedure Action_Stag (Object : in Stag.Segment);
      with procedure Action_Etag (Object : in Etag.Segment);
   procedure Action_G (Object : in Segment);

-- Better, if Action_Data, Action_Stag and Action_Etag have the same name,
-- one can define Action (Object : in Segment) like Get_Level in this
package.

   generic
      with procedure Action (Object : in Segment);
   procedure For_Each_Segment_G (File : in out Ada.Text_IO.File_Type);

------------------------------------------------------------------------------

end Segments.Sgml.Mixed_G;

==============================================================================
==============================================================================

with Ada.Characters.Latin_1;
with Logs;

package body Segments.Sgml.Mixed_G is

------------------------------------------------------------------------------

   function Get_Level (Object : in Segment)
                       return Segments.Segment_Level is
      use Data;
      use Stag;
      use Etag;
   begin
      return Get_Level (Polymorphic_Objects.Get_Details (Object).all);
   exception
      when Constraint_Error =>
         pragma Debug (Logs.Write (Kind => Logs.Trace,
                                   Localization => "instance of " &
                                   "Segments.Sgml.Mixed_G.Get_Level",
                                   Exception_Name =>
                                     "Constraint_Error",
                                   Propagated => True,
                                   Propagation_Name =>
                                     "Polymorphic_Objects.Type_Error"));
         raise Polymorphic_Objects.Type_Error;
   end Get_Level;

   function Is_Start_Tag (Object : in Segment) return Boolean is
      use Meta;
      use Stag;
      use Etag;
   begin
      return Is_Start_Tag (Meta.Segment_G'Class
(Polymorphic_Objects.Get_Details (Object).all));
   exception
      when Constraint_Error =>
         pragma Debug (Logs.Write (Kind => Logs.Trace,
                                   Localization => "instance of " &
                                   "Segments.Sgml.Mixed_G.Is_Start_Tag",
                                   Exception_Name =>
                                     "Constraint_Error",
                                   Propagated => True,
                                   Propagation_Name =>
                                     "Polymorphic_Objects.Type_Error"));
         raise Polymorphic_Objects.Type_Error;
   end Is_Start_Tag;


------------------------------------------------------------------------------

   procedure Read (Current_Segment : out Segment;
                   File : in Ada.Text_IO.File_Type :=
                     Ada.Text_IO.Current_Input;
                   Buffer : in out Lines.Line;
                   Next_Beginning_Seen : in out Boolean;
                   Next_Is_Stag : in out Boolean;
                   No_More_Segments : out Boolean) is

      Cutting_Position, Cutting_Length : Natural;
      Current_Content : Lines.Line := Lines.Null_Decid_String;
      Current_Level : Segments.Segment_Level;

      function Get_Input_With_Success return Boolean is
      begin
         case Lines.Is_Empty (Buffer) is
            when True =>
               loop
                  case Ada.Text_IO.End_Of_File (File) is
                     when True =>
                        No_More_Segments := True;
                        exit;
                     when False =>
                        Lines.Read (Object => Buffer, File => File);
                        Segments.Clean_Input (Buffer);
                        exit when not Lines.Is_Empty (Buffer);
                  end case;  -- ...End_Of_File (File)
               end loop;
               return No_More_Segments = False;
            when False =>
               return True;
         end case;  -- Lines.Is_Empty (Buffer)
      end Get_Input_With_Success;
```

```
procedure Find_Opening is
   Stago_Position, Etago_Position : Natural;
begin
   Stago_Position :=
     Lines.Find_String (Source => Buffer,
                        Object => Stago,
                        Going => Ada.Strings.Forward);
   Etago_Position :=
     Lines.Find_String (Source => Buffer,
                        Object => Etago,
                        Going => Ada.Strings.Forward);
   case Stago_Position = 0 is
      when True =>
         case Etago_Position = 0 is
            when True =>
               Cutting_Position := 0;
            when False =>
               Cutting_Position := Etago_Position;
               Next_Is_Stag := False;
               Cutting_Length := Etago'Length;
               Next_Beginning_Seen := True;
         end case;  -- Etago_Position = 0
      when False =>
         Next_Beginning_Seen := True;
         case (Stago_Position < Etago_Position or
               Etago_Position = 0) is
            when True =>
               Cutting_Position := Stago_Position;
               Next_Is_Stag := True;
               Cutting_Length := Stago'Length;
            when False =>
               Cutting_Position := Etago_Position;
               Next_Is_Stag := False;
               Cutting_Length := Etago'Length;
         end case;  -- Stago_Position < Etago_Position
                    -- or Etago_Position = 0
   end case;  -- Stago_Position = 0
end Find_Opening;

procedure Find_Closing is
begin
   Cutting_Position :=
     Lines.Find_String (Source => Buffer,
                        Object => Tagc,
                        Going => Ada.Strings.Forward);
   Cutting_Length := Tagc'Length;
end Find_Closing;

procedure Transfer_Input is
begin
   Current_Content := Lines.Append (Left => Current_Content,
                                    Right => Buffer);
   Current_Content := Lines.Append (Left => Current_Content,
                                    Right =>
         Lines.To_Decid_String (String'(1 =>
Ada.Characters.Latin_1.Space)));
         Lines.Reset (Buffer);
      end Transfer_Input;

      procedure Split_Input is
      begin
         Cutting_Position := Cutting_Position -1;
         Current_Content :=
           Lines.Append (Left => Current_Content,
                         Right =>
                           Lines.Chunk (Source => Buffer,
                                        From => 1,
                                        To => Cutting_Position));
         Buffer :=
           Lines.Delete (Source => Buffer,
                         From => 1,
                         To => Cutting_Length + Cutting_Position);
      end Split_Input;

      procedure Return_Segment (Level : in Segments.Segment_Level) is
      begin
         case Level is
            when Segments.Data =>
               Segments.Clean_Input (Current_Content);
               case Segments.Is_Significant (Current_Content) is
                  when True =>
                     Current_Segment :=
                       To_Segment (Data.Create (Content =>
                                                Current_Content));
                  when False =>
                     Read (Current_Segment => Current_Segment,
                           File => File,
                           Buffer => Buffer,
                           Next_Beginning_Seen => Next_Beginning_Seen,
                           Next_Is_Stag => Next_Is_Stag,
                           No_More_Segments => No_More_Segments);
               end case;  -- Segments.Is_Significant (Current_Content)
            when Segments.Meta =>
               case Next_Is_Stag is
                  when True =>
                     Segments.Clean_Input (Current_Content);
                     case Segments.Is_Significant (Current_Content) is
                        when True =>
                           Current_Segment :=
                             To_Segment (Stag.Create (Content =>
                                                      Current_Content));
                        when False =>
                           Read (Current_Segment => Current_Segment,
                                 File => File,
                                 Buffer => Buffer,
                                 Next_Beginning_Seen =>
```

```
                                 Next_Beginning_Seen,
                                Next_Is_Stag => Next_Is_Stag,
                                No_More_Segments => No_More_Segments);
                     end case;  -- Segments.Is_Significant
(Current_Content)
                  when False =>
                     Current_Segment :=
                       To_Segment (Etag.Create (Content =>
Current_Content));
               end case;  -- Next_Is_Stag
            end case;  -- Level
         end Return_Segment;

      begin

         No_More_Segments := False;

         case Next_Beginning_Seen is
            when True =>
               Current_Level := Segments.Meta;
               Next_Beginning_Seen := False;
            when False =>
               if Get_Input_With_Success then
                  Find_Opening;
                  case Cutting_Position = 1 is
                     when True =>
                        Current_Level := Segments.Meta;
                        Buffer := Lines.Delete (Source => Buffer,
                                                From => 1,
                                                To => Cutting_Length);
                        Next_Beginning_Seen := False;
                     when False =>
                        Current_Level := Segments.Data;
                  end case;  -- Cutting_Position
               end if;  -- Get_Input_With_Success
         end case;  -- Next_Beginning_Seen

         if not No_More_Segments then
            case Current_Level is
               when Segments.Data =>
                  loop
                     case Cutting_Position = 0 is
                        when True =>
                           Transfer_Input;
                        when False =>
                           Split_Input;
                           Return_Segment (Level => Segments.Data);
                           exit;
                     end case;  -- Cutting_Position = 0
                     if not Get_Input_With_Success then
                        Return_Segment (Level => Segments.Data);
                        exit;
                     end if;  -- not Get_Input_With_Success
                     Find_Opening;
                  end loop;
               when Segments.Meta =>
                  loop
                     case Get_Input_With_Success is
                        when True =>
                           Find_Closing;
                           case Cutting_Position is
                              when 0 =>
                                 Transfer_Input;
                              when others =>
                                 Split_Input;
                                 Return_Segment (Level => Segments.Meta);
                                 exit;
                           end case;  -- Cutting_Position
                        when False =>
                           raise Segments.Sgml.Syntax_Error;
                     end case;  -- Get_Input_With_Success
                  end loop;
            end case;  -- Current_Level
         end if;  -- not No_More_Segments

         exception
         when Segments.Sgml.Syntax_Error =>
            Logs.Write (Kind => Logs.Echo,
                        Localization => "instance of " &
                        "Segments.Sgml.Mixed_G.Read",
                        Exception_Name =>
                          "Segments.Sgml.Syntax_Error",
                        Propagated => False);
            Logs.Write (Message =>
                          "Unclosed tag : no more data in the file" &
                        " to find a Tagc (>).",
                        Ends_The_Line => True);
            Return_Segment (Level => Segments.Meta);
      end Read;

-------------------------------------------------------------------------------

--    generic
--       with procedure Action_Data (Object : in Data.Segment);
--       with procedure Action_Stag (Object : in Stag.Segment);
--       with procedure Action_Etag (Object : in Etag.Segment);
      procedure Action_G (Object : in Segment) is
      begin
         case Get_Level (Object) is
            when Segments.Data =>
               Action_Data (To_Data (Object));
            when Segments.Meta =>
               case Is_Start_Tag (Object) is
                  when True =>
                     Action_Stag (To_Stag (Object));
                  when False =>
```

```
                  Action_Etag (To_Etag (Object));
            end case;  -- Is_Start_Tag (Object)
      end case;  -- Get_Level (Object)
   end Action_G;

--   generic
--      with procedure Action (Object : in Segment);
   procedure For_Each_Segment_G (File : in out Ada.Text_IO.File_Type) is

      Current_Segment : Segment;
      Buffer : Lines.Line := Lines.Null_Decid_String;
      Next_Beginning_Seen, Next_Is_Stag, No_More_Segments : Boolean :=
False;

   begin
      Ada.Text_IO.Reset (File => File, Mode => Ada.Text_IO.In_File);
      loop
         Read (Current_Segment => Current_Segment,
               File => File,
               Buffer => Buffer,
               Next_Beginning_Seen => Next_Beginning_Seen,
               Next_Is_Stag => Next_Is_Stag,
               No_More_Segments => No_More_Segments);
         exit when No_More_Segments;
         Action (Current_Segment);
      end loop;

   end For_Each_Segment_G;

------------------------------------------------------------------------------

   procedure Write_Short (Tag : in Meta.Tag_Category;
                          Text : in Lines.Line;
                          Ends_The_Unit : in Boolean;
                          File : in Ada.Text_IO.File_Type :=
                            Ada.Text_IO.Current_Output) is
   begin
      Stag.Write (Object => Stag.Create (Tag => Tag), File => File);
      Data.Write (Object => Data.Create (Content => Text), File => File);
      case Ends_The_Unit is
         when True =>
            Etag.Write_Short (Object => Etag.Create (Tag => Tag),
                              File => File);
         when False =>
            Etag.Write_Short (Object => Etag.Create (Tag => Tag),
                              File => File);
      end case;  -- Ends_The_Unit
   end Write_Short;

   function Stag_To_Etag (Object : in Stag.Segment)
                          return Etag.Segment is
   begin
      return Etag.Create (Tag => Stag.Get_Tag (Object));
   end Stag_To_Etag;

end Segments.Sgml.Mixed_G;
```

Le contexte (emboîtement des éléments ouverts)

```
with Segments.Sgml.Meta_G;
with Segments.Sgml.Meta_G.Etag_G;
with Segments.Sgml.Meta_G.Stag_G;
with Sizes;

generic
   with package Meta is new Segments.Sgml.Meta_G (<>);
   with package Stag is new Meta.Stag_G (<>);
   with package Etag is new Meta.Etag_G (<>);

   Default_Base : in Meta.Tag_Category;

   with function Is_Base_Level (Object : in Meta.Tag_Category)
                                return Boolean
   is <>;  -- generally last level before DATA ; may be omitted

   with function Is_Infra_Level (Object : in Meta.Tag_Category)
                                 return Boolean
   is <>;
   -- content of base level, mixed with DATA

   with function Is_Transparent (Object : in Meta.Tag_Category)
                                 return Boolean
   is <>;

   with function Can_Contain (Outside, Inside : in Meta.Tag_Category)
                              return Boolean
   is <>;

   with procedure Find_Implicit_Tag (From    : in     Meta.Tag_Category;
                                     Next    :    out Meta.Tag_Category;
                                     Towards : in     Meta.Tag_Category;
                                     Base    : in     Meta.Tag_Category;
                                     No_More :    out Boolean)
   is <>;

   with procedure Find_Implicit_Tag (Next    :    out Meta.Tag_Category;
                                     Towards : in     Meta.Tag_Category;
                                     Base    : in     Meta.Tag_Category;
                                     No_More :    out Boolean)
   is <>;

package Segments.Sgml.Contexts_G is

   type Tag_Context is private;
```

```
   Empty_Context : exception;

   Out_Of_Bound_Index : exception;

   function Get_Present_Context (Object : in Tag_Context)
                                 return Meta.Tag_Category;

   function Test_Present_Context (Object : in Tag_Context;
                                  Value : in Meta.Tag_Category)
                                  return Boolean;

   function Get_Base (Object : in Tag_Context)
                      return Meta.Tag_Category;

   procedure Set_Base (Object : in out Tag_Context;
                       Value : in Meta.Tag_Category);

   procedure Reset_Base (Object : in out Tag_Context);

   procedure Write (Object : in Tag_Context;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output);

private

   Depth_Max : constant Positive := Sizes.M_Value;
   subtype Depth is Positive range 1..Depth_Max;
   subtype Depth_Index is Natural range 0..Depth_Max;

   type Tag_Memory is array (Depth) of Stag.Segment;

   type Tag_Feature is array (Depth) of Boolean;

   type Tag_Context is
      record
         Tag : Tag_Memory;
         Is_Explicit : Tag_Feature;
         Base : Meta.Tag_Category := Default_Base;
         To_Open : Depth_Index := 0;
         To_Close : Depth_Index := 0;
         Index_Bound : Depth_Index := 0;
      end record;

   function Get_Segment (Object : in Tag_Context;
                         Index : in Depth_Index)
                         return Stag.Segment;

   function Get_Tag (Object : in Tag_Context;
                     Index : in Depth_Index)
                     return Meta.Tag_Category;

   function Get_To_Open_Segment (Object : in Tag_Context)
                                 return Stag.Segment;

   function Get_To_Close_Segment (Object : in Tag_Context)
                                  return Stag.Segment;

   function Get_To_Open_Tag (Object : in Tag_Context)
                             return Meta.Tag_Category;

   function Get_To_Close_Tag (Object : in Tag_Context)
                              return Meta.Tag_Category;

   function Has_No_Virtual (Object : in Tag_Context) return Boolean;

   function Segment_To_Open_Is_Explicit (Object : in Tag_Context)
                                         return Boolean;

end Segments.Sgml.Contexts_G;

================================================================================
================================================================================

with Logs;

package body Segments.Sgml.Contexts_G is

   function Get_Base (Object : in Tag_Context)
                      return Meta.Tag_Category is
   begin
      return Object.Base;
   end Get_Base;

   procedure Set_Base (Object : in out Tag_Context;
                       Value : in Meta.Tag_Category) is
   begin
      Object.Base := Value;
   end Set_Base;

   procedure Reset_Base (Object : in out Tag_Context) is
   begin
      Set_Base (Object => Object, Value => Default_Base);
   end Reset_Base;

--------------------------------------------------------------------------------

   function Get_Segment (Object : in Tag_Context;
                         Index : in Depth_Index)
                         return Stag.Segment is
   begin
      return Object.Tag (Index);
   exception
      when Constraint_Error =>
         case Index = 0 is
            when True =>
```

```
               Logs.Write (Kind => Logs.Error,
                           Localization =>
                             "Segments.Sgml.Context_G.Get_Segment",
                           Exception_Name => "Constraint_Error",
                           Propagated => True,
                           Propagation_Name =>
                             "Segments.Sgml.Context_G.Empty_Context");
               Logs.Write (Message => "Seeking for Segment whereas " &
                           "context is empty.",
                           Ends_The_Line => True);
               raise Empty_Context;
            when False =>
               pragma Debug (Logs.Write (Kind => Logs.Trace,
                                         Localization =>

"Segments.Sgml.Context_G.Get_Segment",
                                         Exception_Name =>
"Constraint_Error",
                                         Propagated => True,
                                         Propagation_Name =>

"Segments.Sgml.Context_G.Out_Of_Bound_Index"));
               raise Out_Of_Bound_Index;
         end case;  -- Index
   end Get_Segment;

   function Get_Tag (Object : in Tag_Context;
                     Index : in Depth_Index)
                     return Meta.Tag_Category is
   begin
      return Stag.Get_Tag (Get_Segment (Object => Object, Index => Index));
   exception
      when others =>
         Logs.Write (Kind => Logs.Echo,
                     Localization =>
                       "Segments.Sgml.Context_G.Get_Tag");
         raise;
   end Get_Tag;

   function Get_To_Open_Segment (Object : in Tag_Context)
                                 return Stag.Segment is
   begin
      return Get_Segment (Object => Object, Index => Object.To_Open);
   exception
      when others =>
         Logs.Write (Kind => Logs.Echo,
                     Localization =>
                       "Segments.Sgml.Context_G.Get_To_Open_Segment");
         raise;
   end Get_To_Open_Segment;

   function Get_To_Close_Segment (Object : in Tag_Context)
                                  return Stag.Segment is
   begin
      return Get_Segment (Object => Object, Index => Object.To_Close);
   exception
      when others =>
         Logs.Write (Kind => Logs.Echo,
                     Localization =>
                       "Segments.Sgml.Context_G.Get_To_Close_Segment");
         raise;
   end Get_To_Close_Segment;

   function Get_To_Open_Tag (Object : in Tag_Context)
                             return Meta.Tag_Category is
   begin
      return Get_Tag (Object => Object, Index => Object.To_Open);
   exception
      when others =>
         Logs.Write (Kind => Logs.Echo,
                     Localization =>
                       "Segments.Sgml.Context_G.Get_To_Open_Tag");
         raise;
   end Get_To_Open_Tag;

   function Get_To_Close_Tag (Object : in Tag_Context)
                              return Meta.Tag_Category is
   begin
      return Get_Tag (Object => Object, Index => Object.To_Close);
   exception
      when others =>
         Logs.Write (Kind => Logs.Echo,
                     Localization =>
                       "Segments.Sgml.Context_G.Get_To_Close_Tag");
         raise;
   end Get_To_Close_Tag;

   function Has_No_Virtual (Object : in Tag_Context) return Boolean is
   begin
      return Object.To_Open = Object.To_Close;
   end Has_No_Virtual;

   function Segment_To_Open_Is_Explicit (Object : in Tag_Context)
                                         return Boolean is
   begin
      return Object.Is_Explicit (Object.To_Open);
  exception
      when Constraint_Error =>
         case Object.To_Open = 0 is
            when True =>
               Logs.Write (Kind => Logs.Error,
                           Localization =>

"Segments.Sgml.Context_G.Segment_To_Open_Is_Explicit",
                           Exception_Name => "Constraint_Error",
                           Propagated => True,
```

```
                           Propagation_Name =>
                             "Segments.Sgml.Context_G.Empty_Context");
                Logs.Write (Message => "Seeking for Segment feature " &
                            "whereas context is empty.",
                            Ends_The_Line => True);
                raise Empty_Context;
             when False =>
                pragma Debug (Logs.Write (Kind => Logs.Trace,
                                          Localization =>

"Segments.Sgml.Context_G.Segment_To_Open_Is_Explicit",
                                          Exception_Name =>
"Constraint_Error",
                                          Propagated => True,
                                          Propagation_Name =>

"Segments.Sgml.Context_G.Out_Of_Bound_Index"));
                raise Out_Of_Bound_Index;
          end case;  -- Object.To_Open = 0
   end Segment_To_Open_Is_Explicit;

-------------------------------------------------------------------------------

   function Get_Present_Context (Object : in Tag_Context)
                                 return Meta.Tag_Category is
   begin
      return Get_To_Open_Tag (Object);
   end Get_Present_Context;

   function Test_Present_Context (Object : in Tag_Context;
                                  Value : in Meta.Tag_Category)
                                  return Boolean is
      function "=" (Left, Right : in Meta.Tag_Category)
                    return Boolean
        renames Meta."=";

   begin
      return (Object.To_Open > 0 and then Get_To_Open_Tag (Object) =
Value);
   end Test_Present_Context;

-------------------------------------------------------------------------------

   procedure Write (Object : in Tag_Context;
                    File : in Ada.Text_IO.File_Type :=
                      Ada.Text_IO.Current_Output) is
   begin
      Ada.Text_IO.Put (Item => "******", File => File);
      for I in 1..Object.To_Open loop
         Ada.Text_IO.Put (Item => Stago &
                          Meta.Tag_Category'Image
                          (Stag.Get_Tag (Object.Tag (I))) &
                          Tagc,
                          File => File);
      end loop;  -- I in 1..Object.To_Open
      Ada.Text_IO.Put (Item => "****** To_Close = " &
                       Integer'Image (Object.To_Close),
                       File => File);
      Ada.Text_IO.New_Line (File => File);
   end Write;

end Segments.Sgml.Contexts_G;
```

Les traitements contextuels

```
with Lines;
with Segments.Sgml.Data;

generic

   with procedure Action_On_Virtual_Opening (Object : in Stag.Segment);
   with procedure Action_On_Real_Opening    (Object : in Stag.Segment);
   with procedure Action_On_Virtual_Closing (Object : in Stag.Segment);
   with procedure Action_On_Real_Closing    (Object : in Stag.Segment);
-- actions are done after the change of the context,
-- because the state of the context should not be modified
-- during the operation of opening/closing

   with procedure Action_On_Data (Object : in Segments.Sgml.Data.Segment);

package Segments.Sgml.Contexts_G.Process_G is

   package Data renames Segments.Sgml.Data;

   type Tag_Context_Access is access all Tag_Context;

   procedure Init_Context (Object : in Tag_Context_Access);

   procedure Opening (Object     : in Stag.Segment;
                      Maybe_Done : in Boolean := False);

   procedure Opening (Object     : in Meta.Tag_Category;
                      Maybe_Done : in Boolean := False);

   procedure Actualize (Object : in Data.Segment);

   procedure Actualize (Object : in Lines.Line);

   procedure Closing (Object     : in Meta.Segment_G'Class;
                      Maybe_Done : in Boolean := False);

   procedure Closing (Object     : in Meta.Tag_Category;
                      Maybe_Done : in Boolean := False);
```

```
   procedure Closing_If_Last (Object : in Meta.Tag_Category);

   procedure Closing_Base_If_Not_Done;

end Segments.Sgml.Contexts_G.Process_G;

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

with Logs;

package body Segments.Sgml.Contexts_G.Process_G is

   Context : Tag_Context_Access;

   procedure Init_Context (Object : in Tag_Context_Access) is
   begin
      Context := Object;
   end Init_Context;


   function "=" (Left, Right : in Meta.Tag_Category)
                 return Boolean
     renames Meta."=";

   function "=" (Left, Right : in Stag.Segment)
                 return Boolean
     renames Stag."=";

   procedure Closing (Object : in Meta.Tag_Category;
                      Maybe_Done : in Boolean := False) is

      procedure Processing is
         Stag_Object : Stag.Segment := Get_To_Open_Segment (Context.all);
      begin
         if Is_Base_Level (Object) then
            Reset_Base (Context.all);
         end if;  -- Is_Base_Level (Object)
         case Has_No_Virtual (Context.all) is
            when True =>
               Context.all.To_Open := Context.all.To_Open -1;
               Context.all.To_Close := Context.all.To_Open;
               Action_On_Real_Closing (Object => Stag_Object);
            when False =>
               Context.all.To_Open := Context.all.To_Open -1;
               Action_On_Virtual_Closing (Object => Stag_Object);
         end case;  -- Has_No_Virtual (Context.all)
      end Processing;

   begin
      if not (Is_Transparent (Object) or
              (Maybe_Done and
               (Context.all.To_Open < Context.all.Index_Bound and then
                Get_Tag (Object => Context.all,
                         Index  => Context.all.To_Open +1) = Object))) then
         case Meta.Is_Empty_Tag (Object) is
            when True =>
               loop
                  case Context.all.To_Open = 0 is
                     when True =>
                        raise Segments.Sgml.Syntax_Error;
                     when False =>
                        case Segment_To_Open_Is_Explicit (Context.all) is
                           when True =>
                              Processing;
                              exit;
                           when False =>
                              Processing;
                        end case;  -- Segment_To_Open_Is_Explicit
(Context.all)
                  end case;  -- Context.all.To_Open = 0
               end loop;
            when False =>
               loop
                  case Context.all.To_Open = 0 is
                     when True =>
                        raise Segments.Sgml.Syntax_Error;
                     when False =>
                        case Get_To_Open_Tag (Context.all) = Object is
                           when True =>
                              Processing;
                              exit;
                           when False =>
                              case Can_Contain (Outside => Object,
                                                Inside =>
                                                  Get_To_Open_Tag
(Context.all)) is
                                 when True =>
                                    Processing;
                                 when False =>
                                    raise Segments.Sgml.Syntax_Error;
                              end case;  -- Can_Contain...
                        end case;  -- Get_To_Open_Tag (Context.all) =
Object
                  end case;  -- Context.all.To_Open = 0
               end loop;
         end case;  -- Meta.Is_Empty_Tag (Object)
      end if;  -- not (Is_Transparent (...) or (Maybe_Done and...
   exception
      when Segments.Sgml.Syntax_Error =>
         Logs.Write (Kind => Logs.Warning,
                     Localization =>
                       "Segments.Sgml.Context_G.Process_G.Closing",
                     Exception_Name => "Segments.Sgml.Syntax_Error",
                     Propagated => False);
         Logs.Write (Message => "Try to close a tag whereas " &
```

```
                  "there is no matching start tag.",
                  Ends_The_Line => True);
end Closing;

procedure Closing (Object : in Meta.Segment_G'Class;
                   Maybe_Done : in Boolean := False) is
   use Meta;
   use Stag;
   use Etag;
begin
   Closing (Object => Get_Tag (Object),
            Maybe_Done => Maybe_Done);
end Closing;

procedure Closing_If_Last (Object : in Meta.Tag_Category) is
begin
   if Test_Present_Context (Object => Context.all, Value => Object) then
      Closing (Object => Object, Maybe_Done => False);
   end if;
end Closing_If_Last;

procedure Closing_Base_If_Not_Done is
   Checked_Tag : Meta.Tag_Category;
begin
   while Context.all.To_Open > 0 loop
      Checked_Tag := Get_To_Open_Tag (Context.all);
      case Is_Infra_Level (Checked_Tag) is
         when True =>
            Closing (Object => Checked_Tag, Maybe_Done => False);
         when False =>
            if Is_Base_Level (Checked_Tag) then
               Closing (Object => Checked_Tag, Maybe_Done => False);
            end if;  -- Is_Base_Level (Checked_Tag)
            exit;
      end case;  -- Is_Infra_Level (Checked_Tag)
   end loop;  -- Context.all.To_Open > 0
end Closing_Base_If_Not_Done;

procedure Opening (Object : in Stag.Segment;
                   Maybe_Done : in Boolean := False) is
   Object_Tag            : Meta.Tag_Category := Stag.Get_Tag (Object);
   Checked_Tag, Next_Tag : Meta.Tag_Category;
   Current_Base          : Meta.Tag_Category := Get_Base (Context.all);
   Is_Last_Processing : Boolean;

   procedure Processing (Current_Object : in Stag.Segment;
                         Current_Tag    : in Meta.Tag_Category;
                         Is_Explicit    : in Boolean) is
   begin
      if Is_Base_Level (Current_Tag) then
         Set_Base (Object => Context.all, Value => Current_Tag);
      end if;  -- Is_Base_Level (Current_Tag)
      Context.all.To_Open := Context.all.To_Open +1;
      if Context.all.To_Open > Context.all.Index_Bound then
         Context.all.Index_Bound := Context.all.To_Open;
      end if;  -- Context.all.To_Open > Context.all.Index_Bound
      Context.all.Tag (Context.all.To_Open) := Current_Object;
      Context.all.Is_Explicit (Context.all.To_Open) := Is_Explicit;
      Action_On_Virtual_Opening (Object => Current_Object);
   end Processing;

begin
   if not (Meta.Is_Empty_Tag (Object_Tag) or
           Is_Transparent (Object_Tag) or
           (Maybe_Done and
            (not Has_No_Virtual (Context.all) and then
             Get_To_Open_Segment (Context.all) = Object))) then
      loop  -- closing loop
         if Context.all.To_Open = 0 then
            Find_Implicit_Tag (Next    => Next_Tag,
                               Towards => Object_Tag,
                               Base    => Current_Base,
                               No_More => Is_Last_Processing);
            exit;
         end if;  -- Context.all.To_Open = 0
         Checked_Tag := Get_To_Open_Tag (Context.all);
         case Can_Contain (Outside => Checked_Tag,
                           Inside => Object_Tag) is
            when True =>
               Find_Implicit_Tag (From    => Checked_Tag,
                                  Next    => Next_Tag,
                                  Towards => Object_Tag,
                                  Base    => Current_Base,
                                  No_More => Is_Last_Processing);
               exit;
            when False =>
               Closing (Object => Checked_Tag, Maybe_Done => False);
               Current_Base := Get_Base (Context.all);
         end case;  -- Can_Contain...
      end loop;  -- closing loop
      while not Is_Last_Processing loop  -- opening loop
         Processing (Current_Object => Stag.Create (Next_Tag),
                     Current_Tag    => Next_Tag,
                     Is_Explicit    => False);
         Checked_Tag := Next_Tag;
         Find_Implicit_Tag (From    => Checked_Tag,
                            Next    => Next_Tag,
                            Towards => Object_Tag,
                            Base    => Current_Base,
                            No_More => Is_Last_Processing);
      end loop;  -- opening loop
      Processing (Current_Object => Object,
                  Current_Tag    => Object_Tag,
                  Is_Explicit    => True);
   end if;  -- ((not Meta.Is_Empty_Tag (...)) or
```

```
               -- (not Is_Transparent (...)) or
               -- (not (Maybe_Done and...
   end Opening;

   procedure Opening (Object : in Meta.Tag_Category;
                      Maybe_Done : in Boolean := False) is
   begin
      Opening (Object => Stag.Create (Tag => Object),
               Maybe_Done => Maybe_Done);
   end Opening;

   procedure Actualize (Object : in Data.Segment) is
   begin
      case Context.all.To_Open > 0 is
         when True =>
            declare
               Checked_Tag : Meta.Tag_Category :=
                 Get_To_Open_Tag (Context.all);
            begin
               if not (Is_Base_Level (Checked_Tag) or
                       Is_Infra_Level (Checked_Tag)) then
                  Opening (Object => Context.all.Base, Maybe_Done =>
False);
               end if;  -- not (Is_Base_Level... or Is_Infra_Level...)
            end;
         when False =>
            Opening (Object => Context.all.Base, Maybe_Done => False);
      end case;  -- Context.all.To_Open > 0
      while not Has_No_Virtual (Context.all) loop
         Context.all.To_Close := Context.all.To_Close +1;
         Action_On_Real_Opening (Get_To_Close_Segment (Context.all));
      end loop;  -- while not Has_No_Virtual (Context.all)
      Action_On_Data (Object);
   end Actualize;

   procedure Actualize (Object : in Lines.Line) is
   begin
      Actualize (Data.Create (Content => Object));
   end Actualize;

end Segments.Sgml.Contexts_G.Process_G;
```

d) *Une instance de segments SGML : les segments HTML*

Définition du jeu de balises et de son comportement

```
with Sgml_Gis;

package Segments.Sgml.Html is

   type Tag is (Html, Header_Side, Title, Base_Url, Body_Side,
                Heading, High_Division, Low_Division, Dividing, Caesura,
                List, Item, Definiendum, Definiens,
                Table, Caption, Row, Header_Cell, Data_Cell,
                Alt_Media, Code, Other, Null_Tag);

   function  To_Tag (Object : in Sgml_Gis.Generic_Identifier) return Tag;

   Root_Tag     : constant Tag := Html;

   Default_Base : constant Tag := Low_Division;

   function  Is_Transparent (Object : in Tag) return Boolean;
-- Transparent means that Object should not be recorded in Tag_Context

   function  Is_Base_Level (Object : in Tag) return Boolean;
-- used for setting Base in Tag_Context

   function  Is_Infra_Level (Object : in Tag) return Boolean;

   function  Can_Contain (Outside, Inside : in Tag) return Boolean;

   procedure Find_Implicit_Tag (From    : in     Tag;
                                Next    :    out Tag;
                                Towards : in     Tag;
                                Base    : in     Tag;
                                No_More :    out Boolean);
-- using Find_Implicit_Tag supposes that Can_Contain (From, Towards)

   procedure Find_Implicit_Tag (Next    :    out Tag;
                                Towards : in     Tag;
                                Base    : in     Tag;
                                No_More :    out Boolean);

end Segments.Sgml.Html;

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

package body Segments.Sgml.Html is

   function To_Tag (Object : in Sgml_Gis.Generic_Identifier)
                    return Tag is
      Tag_Image : String := Sgml_Gis.Get_Image (Object);
   begin
      if Tag_Image = "HTML" then
         return Html;
      elsif Tag_Image = "HEAD" then
         return Header_Side;
      elsif Tag_Image = "TITLE" then
         return Title;
```

```
      elsif Tag_Image = "BASE" then
         return Base_Url;
      elsif Tag_Image = "BODY" then
         return Body_Side;
      elsif Tag_Image = "H1" or Tag_Image = "H2" or
        Tag_Image = "H3" or Tag_Image = "H4" then
         return Heading;
      elsif Tag_Image = "DIV" or Tag_Image = "CENTER" or
        Tag_Image = "BLOCKQUOTE" or Tag_Image = "FORM" or
        Tag_Image = "ADDRESS"  then
         return High_Division;
      elsif Tag_Image = "P" or Tag_Image = "PRE" or
        Tag_Image = "H5" or Tag_Image = "H6" then
         return Low_Division;
      elsif Tag_Image = "HR" then
         return Dividing;
      elsif Tag_Image = "BR" then
         return Caesura;
      elsif Tag_Image = "UL" or Tag_Image = "OL" or Tag_Image = "DL" or
        Tag_Image = "DIR" or Tag_Image = "MENU" then
         return List;
      elsif Tag_Image = "LI" then
         return Item;
      elsif Tag_Image = "DT" then
         return Definiendum;
      elsif Tag_Image = "DD" then
         return Definiens;
      elsif Tag_Image = "TABLE" then
         return Table;
      elsif Tag_Image = "CAPTION" then
         return Caption;
      elsif Tag_Image = "ROW" then
         return Row;
      elsif Tag_Image = "TH" then
         return Header_Cell;
      elsif Tag_Image = "TD" then
         return Data_Cell;
      elsif Tag_Image = "APPLET" or Tag_Image = "IMG" then
         return Alt_Media;
      elsif Tag_Image = "SCRIPT" or Tag_Image = "STYLE" then
         return Code;
      elsif Sgml_Gis.Is_Null (Object) then
         return Null_Tag;
      else
         return Other;
      end if;
   end To_Tag;

      subtype Level_Value is Positive range 1..5;
-- 1, 2 : external tags ;
-- 3 : cf. %body - structured parts
-- 4 : cf. %text - zones included in #PCDATA
-- 5 : nothing - "content" of empty tags

   type Column is (Tag_Level, Max_Level_Included);
   Level_Table : array (Tag, Column) of Level_Value :=
     (Html =>          (Tag_Level => 1, Max_Level_Included => 2),
      Header_Side =>   (Tag_Level => 2, Max_Level_Included => 3),
      Title =>         (Tag_Level => 3, Max_Level_Included => 4),
      Base_Url =>      (Tag_Level => 3, Max_Level_Included => 5),
      Body_Side =>     (Tag_Level => 2, Max_Level_Included => 3),
      Heading =>       (Tag_Level => 3, Max_Level_Included => 4),
      High_Division => (Tag_Level => 3, Max_Level_Included => 3),
      Low_Division =>  (Tag_Level => 3, Max_Level_Included => 4),
      Dividing =>      (Tag_Level => 3, Max_Level_Included => 5),
      Caesura =>       (Tag_Level => 4, Max_Level_Included => 5),
      List =>          (Tag_Level => 3, Max_Level_Included => 3),
      Item =>          (Tag_Level => 3, Max_Level_Included => 3),
      Definiendum =>   (Tag_Level => 3, Max_Level_Included => 4),
      Definiens =>     (Tag_Level => 3, Max_Level_Included => 3),
      Table =>         (Tag_Level => 3, Max_Level_Included => 3),
      Caption =>       (Tag_Level => 3, Max_Level_Included => 4),
      Row =>           (Tag_Level => 3, Max_Level_Included => 3),
      Header_Cell =>   (Tag_Level => 3, Max_Level_Included => 3),
      Data_Cell =>     (Tag_Level => 3, Max_Level_Included => 3),
      Alt_Media =>     (Tag_Level => 4, Max_Level_Included => 4),
      Code =>          (Tag_Level => 3, Max_Level_Included => 4),
      Other =>         (Tag_Level => 4, Max_Level_Included => 4),
      Null_Tag =>      (Tag_Level => 4, Max_Level_Included => 5));

function Is_Transparent (Object : in Tag) return Boolean is
begin
   return Object = Other;
end Is_Transparent;

function Is_Base_Level (Object : in Tag) return Boolean is
begin
   return Level_Table (Object, Tag_Level) = 3 and
     Level_Table (Object, Max_Level_Included) = 4;
end Is_Base_Level;

function Is_Infra_Level (Object : in Tag) return Boolean is
begin
   return Level_Table (Object, Tag_Level) > 3;
end Is_Infra_Level;

type Side is (Header_Side, Body_Side, Unknown);

function Get_Side (Object : in Tag) return Side is
begin
   case Object is
      when Html | Other | Null_Tag =>
         return Unknown;
      when Header_Side | Title | Base_Url | Code =>
         return Header_Side;
      when Body_Side | Heading |
        High_Division | Low_Division | Dividing | Caesura |
```

```
         List | Item | Definiendum | Definiens |
         Table | Caption | Row | Header_Cell | Data_Cell |
         Alt_Media =>
          return Body_Side;
   end case;  -- Object
end Get_Side;

function Can_Contain (Outside, Inside : in Tag) return Boolean is

   function No_Direct_Recursion (Outside, Inside : in Tag)
                                 return Boolean is

      function Is_Cell (Object : in Tag) return Boolean is
      begin
         return Object = Data_Cell or Object = Header_Cell;
      end Is_Cell;

   begin
      return (Outside = Inside and
              (Inside = Item or Inside = Definiens or Inside = Row)) or
        (Is_Cell (Outside) and
         (Is_Cell (Inside) or Inside = Row));
   end No_Direct_Recursion;

   function Are_Side_Coherent (Outside, Inside : in Tag)
                               return Boolean is
      Outside_Side : Side := Get_Side (Outside);
      Inside_Side : Side := Get_Side (Inside);
   begin
      return Outside_Side = Inside_Side or
        Outside_Side = Unknown or
        Inside_Side = Unknown;
   end Are_Side_Coherent;

begin
   return (Level_Table (Outside, Max_Level_Included)
           <= Level_Table (Inside, Tag_Level)) and
     Are_Side_Coherent (Outside, Inside) and
     not No_Direct_Recursion (Outside => Outside, Inside => Inside);
end Can_Contain;

procedure Find_Implicit_Tag (From    : in     Tag;
                             Next    :    out Tag;
                             Towards : in     Tag;
                             Base    : in     Tag;
                             No_More :    out Boolean) is

   Next_Level : Level_Value;

   function Get_Default_Tag (Level   : in Level_Value;
                             Of_Side : in Side) return Tag is
   begin
      case Level is
         when 1 =>
            return Html;
         when 2 =>
            case Of_Side is
               when Header_Side =>
                  return Header_Side;
               when Body_Side | Unknown =>
                  return Body_Side;
            end case;  -- Of_Side
         when 3 =>
            return Base;
         when 4 | 5 =>  -- dummy values, never asked for
            return Null_Tag;
      end case;  -- Level
   end Get_Default_Tag;

   procedure Conclude is
   begin
      Next := Towards;
      No_More := True;
   end Conclude;

begin
   case Towards is
      when Null_Tag =>
         Conclude;
      when others =>  -- it cannot be Html (because Can_Contain would be
                      -- false), nor Other (because it is transparent
                      -- and then not considered) : Get_Side (Towards)
                      -- is always significant
         Next_Level := Level_Table (From, Tag_Level) +1;
         case Level_Table (Towards, Tag_Level) > Next_Level is
            when True =>
               Next := Get_Default_Tag (Level   => Next_Level,
                                        Of_Side => Get_Side (Towards));
               No_More := False;
            when False =>
               Conclude;
         end case;  -- Level_Table (Towards, Tag_Level) > Next_Level
   end case;  -- Towards
end Find_Implicit_Tag;

procedure Find_Implicit_Tag (Next    :    out Tag;
                             Towards : in     Tag;
                             Base    : in     Tag;
                             No_More :    out Boolean) is
begin
   Next := Root_Tag;
   No_More := (Towards = Root_Tag);
end Find_Implicit_Tag;

end Segments.Sgml.Html;
```

Modules qui en découlent

Balises

```
with Segments.Sgml.Meta_G;

package Segments.Sgml.Html.Meta is new Segments.Sgml.Meta_G
  (Tag_Category => Tag);
```

Balises ouvrantes

```
with Segments.Sgml.Html.Meta;
with Segments.Sgml.Meta_G.Stag_G;

package Segments.Sgml.Html.Stag is new Segments.Sgml.Html.Meta.Stag_G;
```

Balises fermantes

```
with Segments.Sgml.Html.Meta;
with Segments.Sgml.Meta_G.Etag_G;

package Segments.Sgml.Html.Etag is new Segments.Sgml.Html.Meta.Etag_G;
```

Balises indéfinies

```
with Segments.Sgml.Html.Etag;
with Segments.Sgml.Html.Meta;
with Segments.Sgml.Html.Stag;
with Segments.Sgml.Mixed_G;

package Segments.Sgml.Html.Mixed is
  new Segments.Sgml.Mixed_G (Meta => Segments.Sgml.Html.Meta,
                             Stag => Segments.Sgml.Html.Stag,
                             Etag => Segments.Sgml.Html.Etag);
```

Contexte structurel

```
with Segments.Sgml.Html.Etag;
with Segments.Sgml.Html.Meta;
with Segments.Sgml.Html.Stag;
with Segments.Sgml.Contexts_G;

use Segments.Sgml.Html;

package Segments.Sgml.Html.Contexts is new Segments.Sgml.Contexts_G
  (Meta         => Segments.Sgml.Html.Meta,
   Stag         => Segments.Sgml.Html.Stag,
   Etag         => Segments.Sgml.Html.Etag,
   Default_Base => Segments.Sgml.Html.Default_Base);
```

e) Une autre instance de segments SGML : les segments Corpus

Définition du jeu de balises et de son comportement

```
with Sgml_Gis;

package Segments.Sgml.Decid is

   type Tag is (Corpus, Doc, Ref, Arr, Box, Text, Tit, Head, Nls,
                Idea, Expr, Blck, Scop, Item, Meta, Null_Tag);

   function To_Tag (Object : in Sgml_Gis.Generic_Identifier)
                    return Tag;

   Root_Tag     : constant Tag := Corpus;

   Default_Base : constant Tag := Nls;

   function Is_Transparent (Object : in Tag) return Boolean;
-- Transparent means that Object should not be recorded in Tag_Context

   function Is_Base_Level (Object : in Tag) return Boolean;
-- used for setting Base in Tag_Context

   function Is_Infra_Level (Object : in Tag) return Boolean;

   function Can_Contain (Outside, Inside : in Tag) return Boolean;

   procedure Find_Implicit_Tag (From    : in     Tag;
                                Next    :    out Tag;
                                Towards : in     Tag;
                                Base    : in     Tag;
                                No_More :    out Boolean);
-- using Find_Implicit_Tag supposes that Can_Contain (From, Towards)

   procedure Find_Implicit_Tag (Next    :    out Tag;
                                Towards : in     Tag;
                                Base    : in     Tag;
                                No_More :    out Boolean);

end Segments.Sgml.Decid;

================================================================================
================================================================================
```

```
with Logs;

package body Segments.Sgml.Decid is

   function To_Tag (Object : in Sgml_Gis.Generic_Identifier)
                           return Tag is
   begin
      case Sgml_Gis.Is_Null (Object) is
         when True =>
            return Null_Tag;
         when False =>
            return Tag'Value (Sgml_Gis.Get_Image (Object));
      end case;  -- Sgml_Gis.Is_Null (Object)
   exception
      when Constraint_Error =>
         Logs.Write (Kind => Logs.Error,
                     Localization => "Segments.Sgml.Decid.To_Tag",
                     Exception_Name => "Constraint_Error",
                     Propagated => True);
         Logs.Write (Message => Sgml_Gis.Get_Image (Object) &
                     " is not a Decid Tag.",
                     Ends_The_Line => True);
         raise;
   end To_Tag;

   subtype Level_Value is Natural range 0..5;
-- 0 : packaging tag ;
-- 1, 2 : external tags ;
-- 3 : text structuration
-- 4 : textual zones, inside parts
-- 5 : pure text
   type Column is (Tag_Level, Max_Level_Included);
   Level_Table : array (Tag, Column) of Level_Value :=
     (Corpus => (Tag_Level => 0, Max_Level_Included => 1),
      Doc    => (Tag_Level => 1, Max_Level_Included => 2),
      Ref    => (Tag_Level => 2, Max_Level_Included => 5),
      Arr    => (Tag_Level => 2, Max_Level_Included => 5),
      Box    => (Tag_Level => 2, Max_Level_Included => 5),
      Text   => (Tag_Level => 2, Max_Level_Included => 3),
      Tit    => (Tag_Level => 3, Max_Level_Included => 4),
      Head   => (Tag_Level => 3, Max_Level_Included => 4),
      Nls    => (Tag_Level => 3, Max_Level_Included => 4),
      Idea   => (Tag_Level => 4, Max_Level_Included => 4),
      Expr   => (Tag_Level => 4, Max_Level_Included => 5),
      Blck   => (Tag_Level => 3, Max_Level_Included => 3),
      Scop   => (Tag_Level => 3, Max_Level_Included => 3),
      Item   => (Tag_Level => 3, Max_Level_Included => 3),
      Meta   => (Tag_Level => 3, Max_Level_Included => 3),
      Null_Tag => (Tag_Level => 4, Max_Level_Included => 5));

   function Is_Transparent (Object : in Tag) return Boolean is
   begin
      return False;
   end Is_Transparent;

   function Is_Base_Level (Object : in Tag) return Boolean is
   begin
      return ((Level_Table (Object, Tag_Level) = 3 and
               Level_Table (Object, Max_Level_Included) > 3) or
              ((Object = Ref) or (Object = Arr) or (Object = Box)));
   end Is_Base_Level;

   function Is_Infra_Level (Object : in Tag) return Boolean is
   begin
      return Level_Table (Object, Tag_Level) > 3;
   end Is_Infra_Level;

   function Can_Contain (Outside, Inside : in Tag) return Boolean is

      function No_Direct_Recursion (Outside, Inside : in Tag)
                                    return Boolean is
      begin
         return (Outside = Inside and
                 (Inside = Item or Inside = Meta or Inside = Idea));
      end No_Direct_Recursion;

   begin
      return (Level_Table (Outside, Max_Level_Included)
              <= Level_Table (Inside, Tag_Level)) and
        not No_Direct_Recursion (Outside => Outside, Inside => Inside);
   end Can_Contain;

   procedure Find_Implicit_Tag (From    : in     Tag;
                                Next    :    out Tag;
                                Towards : in     Tag;
                                Base    : in     Tag;
                                No_More :    out Boolean) is
      Next_Level : Level_Value;

      function Get_Default_Tag (Level   : in Level_Value) return Tag is
      begin
         case Level is
            when 0 =>
               return Corpus;
            when 1 =>
               return Doc;
            when 2 =>
               return Text;
            when 3 =>
               return Base;
            when 4 | 5 =>  -- dummy values, never asked for
               return Null_Tag;
         end case;  -- Level
      end Get_Default_Tag;
```

```
      procedure Conclude is
      begin
         Next := Towards;
         No_More := True;
      end Conclude;

   begin
      case Towards = Null_Tag is
         when True =>
            Conclude;
         when False =>
            Next_Level := Level_Table (From, Tag_Level) +1;
            case Level_Table (Towards, Tag_Level) > Next_Level is
               when True =>
                  Next := Get_Default_Tag (Next_Level);
                  No_More := False;
               when False =>
                  Conclude;
            end case;  -- Level_Table (Towards, Tag_Level) > Next_Level
      end case;  -- Towards = Null_Tag
   end Find_Implicit_Tag;

   procedure Find_Implicit_Tag (Next    :    out Tag;
                                Towards : in     Tag;
                                Base    : in     Tag;
                                No_More :    out Boolean) is
   begin
      Next := Root_Tag;
      No_More := (Towards = Root_Tag);
   end Find_Implicit_Tag;

end Segments.Sgml.Decid;
```

Modules qui en découlent

Balises

```
with Segments.Sgml.Meta_G;

package Segments.Sgml.Decid.Meta_Pkg is new Segments.Sgml.Meta_G
  (Tag_Category => Tag);
```

Balises ouvrantes

```
with Segments.Sgml.Decid.Meta_Pkg;
with Segments.Sgml.Meta_G.Stag_G;

package Segments.Sgml.Decid.Stag is new
Segments.Sgml.Decid.Meta_Pkg.Stag_G;
```

Balises fermantes

```
with Segments.Sgml.Decid.Meta_Pkg;
with Segments.Sgml.Meta_G.Etag_G;

package Segments.Sgml.Decid.Etag is new
Segments.Sgml.Decid.Meta_Pkg.Etag_G;
```

Balises indéfinies

```
with Segments.Sgml.Decid.Etag;
with Segments.Sgml.Decid.Meta_Pkg;
with Segments.Sgml.Decid.Stag;
with Segments.Sgml.Mixed_G;

package Segments.Sgml.Decid.Mixed is
  new Segments.Sgml.Mixed_G (Meta => Segments.Sgml.Decid.Meta_Pkg,
                             Stag => Segments.Sgml.Decid.Stag,
                             Etag => Segments.Sgml.Decid.Etag);
```

Contexte structurel

```
with Segments.Sgml.Decid.Etag;
with Segments.Sgml.Decid.Meta_Pkg;
with Segments.Sgml.Decid.Stag;
with Segments.Sgml.Contexts_G;

use Segments.Sgml.Decid;

package Segments.Sgml.Decid.Contexts is new Segments.Sgml.Contexts_G
  (Meta         => Segments.Sgml.Decid.Meta_Pkg,
   Stag         => Segments.Sgml.Decid.Stag,
   Etag         => Segments.Sgml.Decid.Etag,
   Default_Base => Segments.Sgml.Decid.Default_Base);
```

4. Utilitaires

a) Tailles : la gestion des ordres de grandeur

```
package Sizes is

   type Size is (S, M, L, XL);

   XS_Value : constant Positive := 8;
   S_Value : constant Positive := 64;
   M_Value : constant Positive := 250;
   L_Value : constant Positive := 4_000;
   XL_Value : constant Positive := 64_000;

   subtype S_Positive is Positive range 1..S_Value;
   subtype M_Positive is Positive range 1..M_Value;
   subtype L_Positive is Positive range 1..L_Value;
   subtype XL_Positive is Positive range 1..XL_Value;

   subtype S_Natural is Natural range 0..S_Value;
   subtype M_Natural is Natural range 0..M_Value;
   subtype L_Natural is Natural range 0..L_Value;
   subtype XL_Natural is Natural range 0..XL_Value;

   function To_Number (Notation : in Size) return Positive;

end Sizes;
```

b) Lecture de fichiers tabulés

```
------------------------------------------------------------------------------
--|
--
--| Component
--|
--| $Id: read_file.ads,v 1.1.1.1 1998/04/17 12:20:12 obry Exp $
--|
--| Author    : Pascal Obry
--| e-mail    : 101465.2502@compuserve.com
--|
--| Machine/System Compiled/Run on : SPARC SUN-OS, Windows NT - GNAT
--|
--|
--
------------------------------------------------------------------------------

--|
--| Read Formatted Files
--| --------------------
--|
--| Each line is a set of field. Fields are separated by ' ' or Tab or
--| user defined separators.
--|
--| The file name "-" is used for standard input.
--|

generic

    Max_Field          : Positive :=    30;
    Max_Line_Length    : Positive := 1_000;

package Read_File is

   subtype Field_Range is Positive range 1 .. Max_Field;

   procedure Open (Name : in String);
   procedure Close;
   procedure Set_Separators (To : in String);
   --| define a set of separators

   procedure Fetch;
   --| read next line
   function End_Of_File
     return Boolean;

   function Is_Empty_Line
     return Boolean;

   function Number_Of_Fields
     return Natural;
   --| number of field for the current line

   function Line_Number
     return Natural;

   --| to get the separator that has been used to cut the field.
   function Separator_For_Field (N : in Field_Range)
                                 return Character;

   --| to get the value of a field as a String, Integer, Float
   function Field (N : in Field_Range)
      return String;
   function Field (N : in Field_Range)
      return Integer;
   function Field (N : in Field_Range)
      return Float;

   --| to get the value of a field as a user defined type.
   generic
```

```
      type T is (<>);
   function Field_Of_Type (N : in Field_Range)
      return T;

end Read_File;
```

c) *Un générique pour construire et manipuler des objets polymorphes*

```
with Ada.Finalization;

generic

   type Object_Root is abstract tagged private;

package Polymorphic_Objects_G is

   type Polymorphic_Object is private;

   Type_Error : exception;

-------------------------------------------------------------------------

   function Is_Undefined (Object : in Polymorphic_Object)
                          return Boolean;

   procedure Reset (Object : in out Polymorphic_Object);

-------------------------------------------------------------------------
-- Conversions --

   generic
      type Object_Child is new Object_Root with private;
   function To_Polymorphic_G (Object : in Object_Child)
                              return Polymorphic_Object;

   generic
      type Object_Child is new Object_Root with private;
   function To_Child_G (Object : in Polymorphic_Object)
                        return Object_Child;

-------------------------------------------------------------------------
-- Dispatching subprogramms --

   type Object_Access is access all Object_Root'Class;

   function Get_Details (Object : in Polymorphic_Object)
                         return Object_Access;

-- The following generics can only be used with a non abstract Action

   generic
      type Arg_1_Type is limited private;
      with function Arg_1_Default_Value return Arg_1_Type;
      with procedure Action (Object : in Object_Root;
                             Arg_1 : in Arg_1_Type :=
                               Arg_1_Default_Value);
   procedure Dispatching_Procedure_G (Object : in Polymorphic_Object;
                                      Arg_1 : in Arg_1_Type :=
                                        Arg_1_Default_Value);

   generic
      type Arg_1_Type is limited private;
      with function Arg_1_Default_Value return Arg_1_Type;
      with procedure Action (Object : in out Object_Root;
                             Arg_1 : in Arg_1_Type :=
                               Arg_1_Default_Value);
   procedure Dispatching_Modifying_Procedure_G
     (Object : in out Polymorphic_Object;
      Arg_1 : in Arg_1_Type := Arg_1_Default_Value);

   generic
      type Result_Type is limited private;
      with function Action (Object : in Object_Root)
        return Result_Type;
   function Dispatching_Function_G (Object : in Polymorphic_Object)
                                    return Result_Type;

-------------------------------------------------------------------------
-- Specific subprogramms --

   generic
      type Object_Child is new Object_Root with private;
      type Result_Type is limited private;
      with function Action (Object : in Object_Child)
                            return Result_Type;
   function Child_Function_G (Object : in Polymorphic_Object)
     return Result_Type;

   generic
      type Object_Child is new Object_Root with private;
      type Value_Type is limited private;
      with procedure Action (Object : in out Object_Child;
                             Value : in Value_Type);
   procedure Child_Procedure_G (Object : in out Polymorphic_Object;
                                Value : in Value_Type);

-------------------------------------------------------------------------

private

   type Polymorphic_Object is new Ada.Finalization.Controlled with
      record
```

```
         Details : Object_Access;
      end record;

   procedure Adjust (Object : in out Polymorphic_Object);
   procedure Finalize (Object : in out Polymorphic_Object);

end Polymorphic_Objects_G;

------------------------------------------------------------------------------
------------------------------------------------------------------------------

with Ada.Unchecked_Deallocation;
with Logs;

package body Polymorphic_Objects_G is

   function Get_Details (Object : in Polymorphic_Object)
                         return Object_Access is
   begin
      return Object.Details;
   end Get_Details;

------------------------------------------------------------------------

   function Is_Undefined (Object : in Polymorphic_Object)
                          return Boolean is
   begin
      return Get_Details (Object) = null;
   end Is_Undefined;

   procedure Reset (Object : in out Polymorphic_Object) is
   begin
      Object := Polymorphic_Object'
                (Ada.Finalization.Controlled with
                 Details => null);
   end Reset;

------------------------------------------------------------------------
-- Conversions --

--   generic
--      type Object_Child is new Object_Root with private;
   function To_Polymorphic_G (Object : in Object_Child)
     return Polymorphic_Object is
   begin
      return Polymorphic_Object'(Ada.Finalization.Controlled with
               Details => new Object_Root'Class'(Object_Root'Class
(Object)));
   end To_Polymorphic_G;

--   generic
--      type Object_Child is new Object_Root with private;
   function To_Child_G (Object : in Polymorphic_Object)
                        return Object_Child is
   begin
      return Object_Child (Object.Details.all);
   exception
      when Constraint_Error =>
         pragma Debug (Logs.Write (Kind => Logs.Trace,
                                   Localization => "instance of " &
                                   "Polymorphic_Objects_G.To_Child_G",
                                   Exception_Name =>
                                     "Constraint_Error",
                                   Propagated => True,
                                   Propagation_Name =>
                                     "Polymorphic_Objects_G.Type_Error"));
         raise Type_Error;
   end To_Child_G;

------------------------------------------------------------------------
-- Dispatching subprogramms --

--   generic
--      type Arg_1_Type is limited private;
--      with function Arg_1_Default_Value return Arg_1_Type;
--      with procedure Action (Object : in Object_Root;
--                             Arg_1 : in Arg_1_Type :=
--                               Arg_1_Default_Value) is <>;
   procedure Dispatching_Procedure_G (Object : in Polymorphic_Object;
                                      Arg_1 : in Arg_1_Type :=
                                        Arg_1_Default_Value) is
   begin
      Action (Object => Object_Root (Object.Details.all),
              Arg_1  => Arg_1);
   end Dispatching_Procedure_G;

--   generic
--      type Arg_1_Type is limited private;
--      with function Arg_1_Default_Value return Arg_1_Type;
--      with procedure Action (Object : in out Object_Root;
--                             Arg_1 : in Arg_1_Type :=
--                               Arg_1_Default_Value) is <>;
   procedure Dispatching_Modifying_Procedure_G
     (Object : in out Polymorphic_Object;
      Arg_1 : in Arg_1_Type :=
        Arg_1_Default_Value) is
   begin
      Action (Object => Object_Root (Object.Details.all), Arg_1 => Arg_1);
   end Dispatching_Modifying_Procedure_G;

--   generic
--      type Result_Type is limited private;
--      with function Action (Object : in Object_Root)
--        return Result_Type is <>;
   function Dispatching_Function_G (Object : in Polymorphic_Object)
                                    return Result_Type is
```

```
   begin
      return Action (Object => Object_Root (Object.Details.all));
   end Dispatching_Function_G;

-------------------------------------------------------------------------
-- Specific subprogramms --

--   generic
--      type Object_Child is new Object_Root with private;
--      type Result_Type is limited private;
--      with function Action (Object : in Object_Child)
--                            return Result_Type is <>;
   function Child_Function_G (Object : in Polymorphic_Object)
     return Result_Type is
      function To_Child is new
        To_Child_G (Object_Child => Object_Child);
   begin
      return Action (Object => To_Child (Object));
   end Child_Function_G;

--   generic
--      type Object_Child is new Object_Root with private;
--      type Value_Type is limited private;
--      with procedure Action (Object : in out Object_Child;
--                             Value : in Value_Type) is <>;
   procedure Child_Procedure_G (Object : in out Polymorphic_Object;
                                Value : in Value_Type) is
      Work_Object : Object_Child;
      function To_Polymorphic is
         new To_Polymorphic_G (Object_Child => Object_Child);
      function To_Child is new
        To_Child_G (Object_Child => Object_Child);
   begin
      Work_Object := To_Child (Object);
      Action (Object => Work_Object,
              Value => Value);
      Object := To_Polymorphic (Work_Object);
   end Child_Procedure_G;

-------------------------------------------------------------------------

   procedure Adjust (Object : in out Polymorphic_Object) is
   begin
      Object.Details :=
       new Object_Root'Class'(Object.Details.all);
   end Adjust;

   procedure Finalize (Object : in out Polymorphic_Object) is
      procedure Free is
        new Ada.Unchecked_Deallocation (Object => Object_Root'Class,
                                        Name => Object_Access);
   begin
      Free (Object.Details);
   end Finalize;

-------------------------------------------------------------------------

end Polymorphic_Objects_G;
```

5. Données auxiliaires

a) Entités SGML

```
aacute=á=LAT1=225=
Aacute=Á=LAT1=193=
acirc=â=LAT1=226=
Acirc=Â=LAT1=194=
agrave=à=LAT1=224=
Agrave=À=LAT1=192=
aring=å=LAT1=229=
Aring=Å=LAT1=197=
atilde=ã=LAT1=227=
Atilde=Ã=LAT1=195=
auml=ä=LAT1=228=
Auml=Ä=LAT1=196=
aelig=æ=LAT1=230=
AElig=Æ=LAT1=198=
ccedil=ç=LAT1=231=
[...]
agr=a=GRK1=97=alpha
Agr=A=GRK1=65=Alpha
bgr=b=GRK1=98=beta
[...]
aleph==TECH=0=aleph
and==TECH=0=logical and
ap==TECH=0=approximate
bottom==TECH=0=perpendicular
[...]
half=1/2=NUM=189=fraction one-half
frac12=1/2=NUM=189=fraction one-half
frac14=1/4=NUM=188=fraction one-quarter
frac34=3/4=NUM=190=fraction three-quarters
frac18=1/8=NUM=49=47=56=fraction one-eighth
frac38=3/8=NUM=51=47=56=fraction three-eighths
[...]
```