SYSTRAN
Information and translation technologies

novembre 2006

RAPPORT FINAL

PLATEFORME FRANCAISE OUTILEX

(RNTL : OUTILEX)

OUTILEX-RNTL n° 02.4.93.0590

Table des matières

Introduction

La dernière phase du projet Outilex par SYSTRAN a porté sur les trois axes suivants :

- Conception, implémentation de filtres documentaires de seconde génération permettant l'extraction d'unité textuelles d'un format source, la représentation de ces unités dans un format XML normalisé, la préservation d'information typographique et des éléments structurels minimums permettant la reconstruction du format initial.
- Améliorations du module de tokenisation unicode (décrit dans le rapport d'avancement numéro 4)
- Recherche sur le développement d'un module générique de « transfert » opérant sur des automates de phrases
- Finalisation d'un lexique bilingue en correspondance avec les lexiques monolingues distribués par l'université de Marne la Vallée.

Ces librairies sont maintenues via le serveur cvs public maintenu à SYSTRAN.

L'architecture de ces modules a été repensée pour fonctionner de manière incrémentale en environnement multi-threadé.

Travail déjà effectué

Les points précédents complètent le travail déjà effectué et décrit dans les rapports d'avancement précédemment :

- Définition de méta-structure de représentation de données (Phase 1)
- Implémentation de mécanisme permettant de gérer ces méta-représentations, en particulier par la génération de classes d'objets « C » à partir de ces méta-représentations linguistiques (Phase 2)
- Implémentation de classes équivalentes permettant un chargement dynamique des structures de donnée (vs. Chargement statiques). Les deux approches étant complémentaires (LingFeatures). (Phase 3)
- Implémentation de modules de tokenisation unicode – incluant la reconnaissance d'entité textuelles de premier niveau (Phase 4)

Filtres documentaires de seconde génération

Contexte

Les filtres documentaires sont les premiers maillons de tout processus de traitement automatique sur des documents électroniques. Si il est relativement facile d'extraire le texte de document html simple, la tâche se complique pour des formats tels que rtf, doc, pdf, ou LaTeX. Les obstacles sont de plusieurs natures : écriture d'un parseur correspondant au format considéré, stratégies de « reflowing » pour des formats comme le pdf, ou « calcul » de l'affichage (par exemple, en html, une partie du contenu textuel peut être généré par du javascript, ou en LaTeX, il est possible de générer du contenu par le biais de macro).
De plus, dans le cadre de ce projet, nous avons développé une approche plus fine permettant non seulement l'extraction de ce contenu, mais aussi la reconstruction du format original (quand possible) de manière par exemple à pouvoir reconstruire le document après transformation (par exemple, après une traduction), ou à annoter le texte source avec des annotations natives au format (utilisation de couleur, « popup », …).
Ce développement a été effectué dans la logique du standard XLIFF [1]utilisé dans la localisation et élaboré par le consortium OASIS (…) : l'extraction génère un format XLIFF contenant :

- le texte extrait par « paragraphe » logique
- l'information suffisante pour la reconstruction du format initial (le document extrait contient le squelette de document mais pas forcément tous les détails) sous la forme de tag
- Support de formes « transformées » dans le format – permettant par exemple de conserver dans un même document en parallèle la forme source et la forme traduite
- une transcription de la typographie du format initial sous la forme de markup – cette information permet la conservation de la typographie en cas de réarrangement des différents éléments de la phrase.

De plus, nous avons aussi géré la possibilité de rajouter des annotations utilisateurs dans le format natif convertit en markup XML générique après conversion.

Ainsi, ce module permet d'effectuer des opérations complexes sur différents types de documents de manière totalement uniforme – le format interne étant unique – Ce format est documenté dans l'annexe 1 de ce document.

Nouvelles fonctionnalités

Les fonctionnalités propres à cette nouvelle génération sont décrites dans la table ci-dessous :

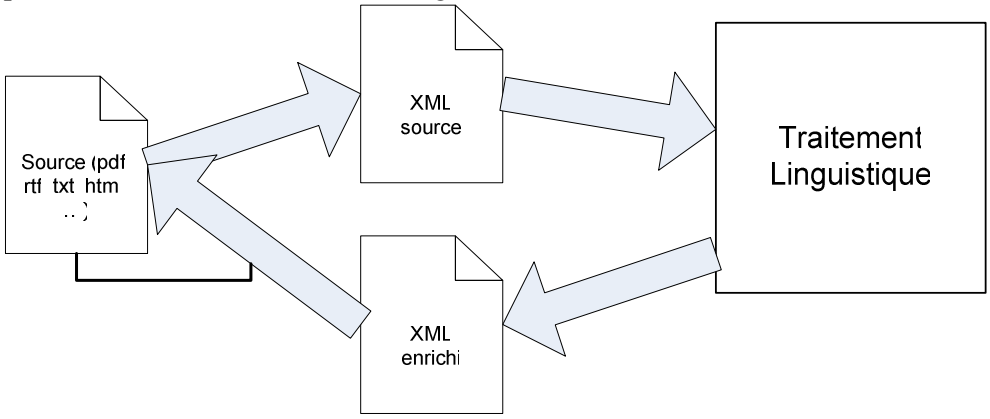
Catégorie	Fonctionnalités	Première Génération	Seconde Génération
Formats d'entrée	HTML	√	√
	TXT	√	√
	RTF		√
	PDF		√[2]
	LaTeX		en cours
	Word 97[3]		en cours
Gestion des charsets	Encodage utf-8	√	√

[1] http://www.oasis-open.org/committees/xliff/
[2] Basée sur la bilbliothèque xpdf
[3] Fichier .doc natif utilisé jusqu'à Windows XP

Catégorie	Fonctionnalités	Première Génération	Seconde Génération
	Support charsets iconv		√
	Auto-détection		√
Format XML Interne Généré	Paragraphes	√	√
	Structure source/cible		√
	Tags Structurels	Partiel	√
	Tags Typographiques		√
	Annotation Externe		√
Reconversion	Vers Format Textuel	√	√
	Vers Format Natif		√
	Annotation textuelle		√
	Annotation Native		√
Architecture	Parseur Incrémental		√
	Support Multi-thread		√

Principe

Le principe du « filtre » est résumé dans la figure suivante :



- Le fichier source est converti en un fichier XML générique indépendant de son format d'origine (cf. Figure 2 - Fichier HTML source, Figure 3 - Fichier XML provenant du format HTML et Figure 1 - Format XML provenant du format RTF)
- Ce format normalisé passe au travers d'un traitement linguistique dont la première étape est le découpage en token– cf. Figure 4 - Deuxième paragraphe après tokenisation)
- Au cours du traitement linguistique, les nœuds « paragraphes » sont modifiés : par exemple par la segmentation, l'identification d'entités la traduction. Dans le XML résultat les paragraphes contiennent les contenus sources et cibles (cf. figure : Figure 5 - Document après traitement linguistique (ici traduction))
- Le format source est regénéré contenant le contenu modifié et du markup additionnel. (voir Figure 6)

Architecture

Les caractéristiques principales de ces filtres sont les suivantes :

- Librairie avec API « C » : l'ensemble de la bibliothèque est sous la forme d'une librairie dont l'API est restreint à un ensemble de fonctions C pour permettre l'utilisation de cette librairie dans des contextes multiples. Cette API est présentée en annexe 2.

- Architecture modulaire : chaque filtre est une librairie détectée à l'initialisation, permettant l'ajout facile de nouveaux filtres documentaires.
- Support multi-thread – la conception de la librairie permet son utilisation en mode « multi-thread »
- Parseur incrémental : la génération des différents paragraphes se fait sous forme incrémentale : il est ainsi possible de parser de gros documents en gardant une empreinte mémoire minimale.

Les librairies externes suivantes sont utilisées :

- librairie libxml2 (développée par Daniel Veillard, http://www.xmlsoft.org).
- librairie iconv (http://www.gnu.org/software/libiconv/).
- librairie xpdf (http://gnuwin32.sourceforge.net/packages/xpdf.htm)

Description de la structure

La structure générée est décrite par la dtd de « **Erreur ! Source du renvoi introuvable.** »). Les éléments clefs de ce format sont les suivants :

- un document est représenté par un nœud « `document` » portant son type en attribut
- un document est une séquence de « `par` » (paragraphes) correspondant à des paragraphes logiques du fichier source
- un paragraphe contient soit du texte, soit une suite de token, soit des séquences de « `tu` » (phrases)
- une séquence de « `tu` » avant traitement est représentée par un nœud simple « `tu` »
- une séquence de « `tu` » après traitement est représentée par une paire `tu`, `transtu` dans un nœud `tu_group`
- des nœuds « `tag` » portent l'information permettant une reconstruction du format source
- les nœuds « `ts` » portent de l'information typographique sur la suite des mots suivants les tags. Il s'agit de nœuds d'état et non pas d'annotation
- des annotations additionnelles peuvent être apportées par des couples de tag `bmark`/`emark` – la correspondance bmark/emark est faite par l'attribut id. Cette représentation par couple de tag permet en particulier une imbrication des annotations : par exemple, dans la séquence suivante deux annotations `dict_1` et `dict_2` se superposent.

```
<bmark type='dict' id='dict_1'/>voiture de <bmark type='dict'
id='dict_2'/>course<emark id='dict_1'/> rapide<emark id='dict_2'/>
```

Exemples

Les trois figures ci-dessous décrivent des exemples de fichier généré pour un même fichier source en format HTML ou RTF.

```
1  <?xml version="1.0"?>
2  <!DOCTYPE document PUBLIC "-//SYSTRAN//DTD XML Internal Format 1.0//EN" "http://www.systransoft.com/XML/sts/1.
3  <document original_format="rtf">
4      <par id="p1">
5          <ts font_id="0" lang="1036"/>Un
6              <tag>}{\rtlch\fcs1\ab\af0\ltrch\fcs0\b\insrsid11404022\charrsid11404022\hich\af0\dbch\af11\loch\f0</tag>
7          <ts bold="1" font_id="0" lang="1036"/>fichier <ts font_id="0" lang="1036"/>d'exemple
8              <tag>}{\rtlch\fcs1\ai\af0\ltrch\fcs0\i\insrsid11404022\charrsid11404022\hich\af0\dbch\af11\loch\f0</tag>
9          <ts font_id="0" italic="1" lang="1036"
10             />simple<tag>}{\rtlch\fcs1\af0\ltrch\fcs0\insrsid11404022\hich\af0\dbch\af11\loch\f0</tag>
11         <ts font_id="0" lang="1036"/>.</par>
12 </document>
13
```

Figure 1 - Format XML provenant du format RTF

```
<html>
 <head>
  <title>Un fichier d'exemple simple</title>
 </head>
 <body lang=FR style='tab-interval:35.4pt'>
  <div class=Section1>
   <p class=MsoNormal>Un <b>fichier</b> d'exemple <i>simple</i>.</p>
  </div>
 </body>
</html>
```

Figure 2 - Fichier HTML source

```
<?xml version="1.0"?>
<!DOCTYPE document PUBLIC "-//SYSTRAN//DTD XML Internal Format 1.0//EN" "http://www.systransoft.com/XML/sts
<document>
    <par id="1" notypeset="true">
        <ts nval=""/>
        <bmark type="title" action="set" id="title_1"/>
        <tag html_block="1">&lt;title&gt;</tag>Un fichier d&#x2019;exemple simple<tag
            html_block="1">&lt;/title&gt;</tag>
        <emark id="title_1"/>
    </par>
    <par id="2">
        <ts nval=""/>Un <ts bold="1" lang="FR"/>fichier<ts lang="FR" nval="K%FR,%%"/>
        d&#x2019;exemple <ts italic="1" lang="FR"/>simple<ts lang="FR"/>.</par>
</document>
```

Figure 3 - Fichier XML provenant du format HTML

```
    <par id="2">
        <ts italic="1" lang="FR"/>
        <token type="word" id="t0" alph="latin" case="capit">A</token>
        <token type="word" id="t5" alph="latin" pos="adj:base">simple</token>
        <ts bold="1" lang="FR"/>
        <token type="word" id="t2" alph="latin" pos="noun:common">file</token>
        <ts lang="FR"/>
        <token type="word" id="t3" alph="latin" pos="prep">of</token>
        <token type="word" id="t4" alph="latin" pos="noun:common">example</token>
        <token type="punctuation" id="t6">.</token>
    </par>
```

Figure 4 - Deuxième paragraphe après tokenisation

```
<par id="2">
    <tu_group id="s2">
        <tu>
            <ts lang="FR"/>
            <token type="word" id="t1" alph="latin" case="capit">Un</token>
            <ts bold="1" lang="FR"/>
            <token type="word" id="t2" alph="latin">fichier</token>
            <ts lang="FR"/>
            <token source="d&#x2019;" type="word" id="t3" alph="latin">d'</token>
            <token type="word" id="t4" alph="latin">exemple</token>
            <ts italic="1" lang="FR"/>
            <token type="word" id="t5" alph="latin">simple</token>
            <ts lang="FR"/>
            <token type="punctuation" id="t6">.</token>
        </tu>
        <transtu>
            <ts italic="1" lang="FR"/>
            <token type="word" id="t0" alph="latin" case="capit">A</token>
            <token type="word" id="t5" alph="latin" pos="adj:base">simple</token>
            <ts bold="1" lang="FR"/>
            <token type="word" id="t2" alph="latin" pos="noun:common">file</token>
            <ts lang="FR"/>
            <token type="word" id="t3" alph="latin" pos="prep">of</token>
            <token type="word" id="t4" alph="latin" pos="noun:common">example</token>
            <token type="punctuation" id="t6">.</token>
        </transtu>
    </tu_group>
</par>
```

Figure 5 - Document après traitement linguistique (ici traduction)

```
<html>
 <head>
  <title>A simple file of example</title>
 </head>
 <body lang="FR" style="tab-interval:35.4pt">
  <div class="Section1">
   <p class="MsoNormal"><i>A simple</i> <b>file</b> of example.</p>
  </div>
 </body>
</html>
```

Figure 6 - Format source HTML regénéré avec la modification - voir la typographie modifiée, et la structure préservée du document.

Module Générique de « transfert » - démonstration

Principe

Comme expérience d'une utilisation complète de la plateforme Outilex.

Un module prototypique de transfert a été expérimenté à l'aide des briques disponibles : ce code ne prétend pas à développer un moteur de traduction mais est plus fourni à titre de démonstration.

Ce prototype effectue les opérations suivantes sur le français anglais:

- Filtrage documentaire incrémental : pour chaque paragraphe généré :
  - génération du nœud paragraphe correspondant
  - tokenisation du paragraphe
  - analyse morphologique des mots de la phrase
  - désambiguïsation locale (pour cette étape, l'analyse syntaxique de SYSTRAN est invoquée). En l'absence de ce module, le code utilise la première analyse.
  - consultation du lexique de transfert – addition d'une annotation permettant de conserver l'ensemble des sens
  - transfert des mots grammaticaux
  - « transfert » structurel :
    - Correspondance des temps pour les verbes
    - Correspondance des nombres pour les noms
  - Opération de réarrangement locale (nom/adjectif)
  - Génération de la morphologie adéquate pour les noms et les verbes.
- Reconstitution du document xml complet correspondant au document source
- Génération du document traduit dans le format initial

La qualité obtenue d'un processus aussi simple est évidemment médiocre, mais montre l'intégration des différentes étapes et montre aussi les limites des représentations actuelles.
Les principaux axes pour améliorer ce prototype passe par des améliorations de la structure de donnée de l'automate de la phrase :

- Ajout de « lien » entre les différents éléments de la phrase pour représenter des relations syntaxiques
- Possibilités de représenter de manière transversale un arbre syntaxique en conservant les chemins alternatifs décrits avec la structure d'automate de la phrase
- Outils de transformation de l'automate de la phrase pour modéliser un transfert plus profond qu'une substitution mot à mot
- Pondération des différents chemins

De plus, compléter cette approche par de nouvelles techniques basées sur corpus (Language Model) pourrait donner une autre dimension à l'approche purement déclarative.

Lexique Bilingue Français-Anglais

En parallèle du travail de développement effectué dans le cadre de ce projet, nous avons également préparé des lexiques bilingues français-anglais permettant de réaliser en particulier l'expérience décrite précédemment.

Ces lexiques ont la structure suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<Dictionary src="fr" tgt="en" version="1.3">
     <Entry lang="fr">
          <Lemma pos="NOUN ">source</Lemma>
          <frequency corpus="corpusname" value="X"/>
          <meaning lang="en">
               <lemma>sens1</lemma>
               <expression>a1=b1</expression>
          </meaning>
          <meaning lang="en">
               <lemma>sens2</lemma>
               <expression>a2=b2</expression>
          </meaning>
     </Entry>
</Dictionary>
```

Chaque entrée du lexique (**Entry**) décrit un lemme (**Lemma**) sa fréquence observée sur un corpus de référence, et l'ensemble de ses sens possibles dans un but de traduction automatique.
Chaque sens est représenté dans un nœud (**meaning**) qui contient le lemme en langue cible (**lemma**), ainsi que une ou plusieurs expressions ou ce sens s'utilise (**expression**).

Les sens sont ordonnés par ordre de vraisemblance sur un domaine général.

Ce lexique a été construit sur une base de fréquence >=3 des formes lemmatisées observée dans un corpus de ~1M de phrases. Les mots grammaticaux n'ont pas été produit car la notion d'équivalence directe fait beaucoup moins de sens.

Ce lexique est mis à disposition par le biais du site Web d'OUTILEX.

Ce bilingue dans sa version 1.3, à cette date, contient environ 22,000 lemmes, pour 24,000 sens et 8000 expressions associées.

Annexe 1 : schéma du format XML interne

```
<!-- DTD for Internal XML format - Outilex SYSTRAN $Revision: 1.2 $ -->
<!-- Public ID: -//SYSTRAN//DTD XML Internal Format 1.0//EN -->

<!ELEMENT document (par|tag|ts)*>
<!-- MIMETYPE du document-->
<!ATTLIST document original_format CDATA #IMPLIED>

<!-- Noeuds paragraphes -->
<!ELEMENT par (#PCDATA|tu_group|tu|ts|token|tag|bmark|emark)*>
<!ATTLIST par id        CDATA  #IMPLIED
              notypeset (true) #IMPLIED
              subflowid CDATA  #IMPLIED>

<!-- Dans le cas d'une représentation source/cible, représentation du texte dans
des groupes -->
<!ELEMENT tu_group (tu,transtu?)>
<!ATTLIST tu_group id NMTOKEN #REQUIRED>

<!-- Nœud phrase (source) -->
<!ELEMENT tu (token|tag|ts|bmark|emark)*>
<!ATTLIST tu id NMTOKEN #IMPLIED
             lang CDATA #IMPLIED>

<!-- Nœud phrase (cible) -->
<!ELEMENT transtu (token|tag|ts|bmark|emark)*>

<!-- Nœud phrase (token) - ces nœuds portent plusieurs attributs de type,
alphabet, attributs -->
<!ELEMENT token (#PCDATA)>
<!ATTLIST token type (undef|separator|numeric|punctuation|symbol|entity|word
                      |tag|unknown) #REQUIRED
                case (upper|capit|mixed|na) #IMPLIED
                alph (na|undef|latin|greek|cyrillic|arabic|cjkv) #IMPLIED
                id   NMTOKEN #REQUIRED
                pos  NMTOKEN #IMPLIED
                subt CDATA   #IMPLIED
                no_entity_misc (1) #IMPLIED>

<!-- Mécanisme d'annotation, une marque est représentée par un bmark/emark -->
<!ELEMENT bmark EMPTY>
<!ATTLIST bmark id       NMTOKEN #REQUIRED
                type     CDATA   #REQUIRED
                disabled (1)     #IMPLIED
                value    CDATA   #IMPLIED>

<!ELEMENT emark EMPTY>
<!ATTLIST emark id NMTOKEN #REQUIRED>

<!-- Etat typographique du texte suivant -->
<!ELEMENT ts EMPTY>
<!ATTLIST ts nval      CDATA #IMPLIED
             font_id   CDATA #IMPLIED
             generic   CDATA #IMPLIED
             fsize     CDATA #IMPLIED
             bold      (0|1) #IMPLIED
             italic    (0|1) #IMPLIED
             shadow    (0|1) #IMPLIED
             color     CDATA #IMPLIED
             strike    (0|simple|double) #IMPLIED
             outline   (0|1) #IMPLIED
             underline (0|continuous|double|dotted|word) #IMPLIED
             capitals  (0|all) #IMPLIED
             lang      CDATA #IMPLIED
             link      CDATA #IMPLIED>
```

```
<!-- Tag - permettant de représenter des élements structurels du format initial
pour la reconstruction -->
<!ELEMENT tag (#PCDATA)>
<!ATTLIST tag space      (0|1)           #IMPLIED
              html_block (1)             #IMPLIED
              restricted (source|target) #IMPLIED>
```

Annexe 2 : Interface externe bibliothèque

```
struct Document;
typedef struct Document *DocumentPtr;

struct Document {
  /** mimetype of the document */
  char *mimetype;
  /** encoding of the document */
  char *mimeencoding;
  /** status of the filter
   *  =0 format not detected and parsing not started
   *  =5 format detected, parsing not started
   *  <95 parsing in progress
   *  100 parsing completed
   */
  int filter_status;

  /** complete xmltree - at the end of the process */
  xmlDocPtr xmltree;
  /** number of paragraphs extracted to far */
  int nbpars;

  /** single paragraph - in incremental mode */
  xmlNodePtr par;

  /** nb of words - after tokenisation */
  int nbwords;

  /** if an error happen - detail of the error is given here
      - 0 normally */
  int nerror;
  /** if an error happen - detail of the error is given here
      - NULL normally */
  char *error;

  /** possible warning during processing of the file
      - NULL normally */
  char *warning;

  /** filter in charge of the parsing */
  void *filter;
  /** internal data for the filter */
  void *private;
};
```

Functions

DocumentPtr	**InitParserFile** (const char *filename) *Initialize a parser for a given file.*
int	**FreeDocument** (**DocumentPtr** d) *Free document structure.*
DocumentPtr	**InitParserBuf** (const char *buffer, int bufsize) *Initialize a parser for a buffer in memory.*
void	**SetOption** (**DocumentPtr** d, const char *buffer, const char *value) *Pass option to parse the document.*
int	**FormatCharsetDetection** (**DocumentPtr** d, const char *format, const char *charset)

	Format and Charset Detection.
int	**ParseDocument** (**DocumentPtr** d, int incremental) *Parse **Document**.*
int	**RegenerateToFile** (**DocumentPtr** d, const char *filename, int source, int markup, const char *format) *Regenerate **Document** to a file.*
int	**RegenerateToMemBuffer** (**DocumentPtr** d, const char **pbuffer, int *psize, int source, int markup, const char *format) *Regenerate **Document** to a memory buffer.*

Typedef Documentation

typedef struct Document* DocumentPtr

Definition at line **3** of file **docfilter.h**.

Function Documentation

int FormatCharsetDetection (DocumentPtr *d,*
const char * *format,*
const char * *charset*
)

Format and Charset Detection.

Parameters:

d	the current document
format	mimetype of the format - if known or NULL for autodetection
charset	encoding of the file - if known or NULL for autodetection

int FreeDocument (DocumentPtr *d*)

Free document structure.

Parameters:

filename the filename

Returns:

a valid document structure ptr, 0 if a problem occurs

DocumentPtr InitParserBuf (const char * *buffer,*
int *bufsize*
)

Initialize a parser for a buffer in memory.

Parameters:

buffer pointer to a buffer in memory

bufsize size of the buffer

Returns:

a DocumentPtr - 0 if a problem occurs, nerror field in the returned structure is !=0 if there is a problem with the document

DocumentPtr InitParserFile (const char * *filename*)

Initialize a parser for a given file.

Parameters:

filename the filename

Returns:

a valid document structure ptr, 0 if a problem occurs

```
int ParseDocument ( DocumentPtr d,
                    int         incremental
                  )
```

Parse **Document**.

Parameters:

d the current document
incremental boolean - true for incremental parsing

Returns:

status of the parse

```
int RegenerateToFile ( DocumentPtr d,
                       const char * filename
                       ,
                       int         source,
                       int         markup,
                       const char * format
                     )
```

Regenerate **Document** to a file.

Parameters:

d the current document
filename name of the file to regenerate
source boolean - source or target
markup boolean - generate markup

Returns:

status of the parse

```
int RegenerateToMemBuffer ( DocumentPtr d,
                            const char ** pbuffer,
                            int *       psize,
                            int         source,
                            int         markup
                            ,
                            const char * format
                          )
```

Regenerate **Document** to a memory buffer.

Parameters:

d	the current document
filename	name of the file to regenerate
source	boolean - source or target
markup	boolean - generate markup

Returns:

status of the parse

**void SetOption (DocumentPtr *d,*
const char * *buffer*
,
const char * *value*
)**

Pass option to parse the document.

Parameters:

d	the current document
buffer	name of the option
value	value of the option