TR-IT-0314

TDMT 辞書ツール説明書 (英日版・開発者向け)

TDMT Dictionary Tool Developers Manual (English to Japanese)

鷹尾 和享　　柏岡 秀紀　　白井 諭
Kazutaka TAKAO　Hideki KASHIOKA　Satoshi SHIRAI

1999 年 9 月 30 日

概要

TDMT 辞書ツール（英日版）の開発者向けの内容について述べる。
各モジュールの概要、応用プログラム開発のための API、自動推定の方法等について記述する。なお、Visual C++ に関する予備知識が必要である。

エイ・ティ・アール音声翻訳通信研究所
ATR Interpreting Telecommunications Research Laboratories

目次

第１章

モジュールの概要

TDMT 辞書ツールは機能毎に.DLL を分けてあり、API を利用しての応用プログラムの作成が容易に行えるようになっている。

[DLL]

SPLKEY	辞書分割キーの取得
DICINTP	辞書の行解釈
DICIO	辞書ファイル I/O
DICLOGIC	自動推定ロジック
DICDLG	DialogBox 等のユーザ・インタフェース

[上記 DLL を使ったプログラム]

DICTOOL1.EXE	辞書ツール起動プログラム
SPLDIC.EXE	辞書分割ツール
ADDLIST.EXE	まとめて追加するツール
MICHIGO0.EXE	応用プログラム例：未知語を学習する仕掛け

[備考]

- SPLKEY, DICINTP, DICIO, DICLOGIC は USRDLL であり、これを使用するプログラムは必ずしも VisualC++で作っていなくても可能。また、GUI を備えている必要はなく、コンソールプログラムでもよい。(たとえば、SPLDIC.EXE と ADDLIST.EXE はコンソールプログラムである。)
- DICDLG は AFXDLL であり、これを使用するプログラムは VisualC++で MFC を使って作る必要がある。すなわち、CWinApp の派生クラスを持ち、メッセージを処理する機構を備える必要がある。
- 呼び出しシーケンスに extern "C" が付いているものは C 言語でも利用可能。
  付いていないものは C++(VisualC++)専用。

[開発環境]

Windows95 OSR2

Visual C++ 4.0

CPU : MMX-Pentium 200MHz

[モジュールの依存関係]

ソースをビルドし直す場合は以下の順序で行う必要がある。なお、DICDLG は DICTOOL1.EXE のサブプロジェクトなので、DICTOOL1 をビルドすれば DICDLG も自動的にビルドされる。参考までに、使っている DLL を併記した。

初回ビルド時は番号順に.LIB を作っていく必要がある。2 回目以降は、undocumented のものを含めて API の変更がない場合には、さわったモジュールをビルドし直すだけでよい。

<名前>	<使っている DLL>
(1) LIB : static にリンクされるので最初にビルド	
(2) SPLKEY	なし
(3) DICINTP	なし
(4) DICIO	SPCKEY,DICINTP
(5) DICLOGIC	DICINTP,DICIO
(6) DICDLG	上記 DLL 全部（DICTOOL1 をビルドすればよい）
(7) DICTOOL1.EXE	DICDLG
(8) SPLDIC.EXE	DICIO
(9) ADDLIST.EXE	DICIO

第２章

辞書ファイルの分割の仕様

１語追加する毎に巨大なファイルを読み書きするのは効率が悪いので、細かく分割して管理する。

[仕様]

・英語正規形の３文字目と６文字目を見て、アルファベットならその
大文字、それ以外の文字なら ＠ の２文字でファイル名を構成する。
なお、スペースとハイフンは読み飛ばし、文字数にカウントしない。
…したがって、729 個に分割（27 × 27）
…大文字と ＠ にするのはループを回す都合

・ただし、途中で非 ascii 文字 or 文字列終端に出会ったらそれ以降は ＠
…漢字コードに依存しない

例："at work" … WK

[特徴]

・登録の都度巨大な量を書かなくて済む
・かつ、テキスト形式で管理
・分割のルールが人間に理解しやすい
・英語正規形がわかれば高速に検索できる（ほぼ一瞬）
・サイズは十分 mule で読めるくらいのサイズになる（35 万語で確認）

[備考]

・出現形からの検索が必要なら
出現形によるキー → 正規形によるキー
のマッピング表で対処できると考えられる。
（多くは出現形＝正規形なので、例外はわずか）

・辞書をクライアントに置くとソケット通信の処理を切り離せる。
・元通り１つの巨大ファイルにするのは簡単。
・分割の方式を変更するのも簡単。
・実際には語の一部のみから検索する場合が多いかも知れない？
　（結局全ファイル検索することになる？）
・もし日本語等に適用する場合は別途検討が必要

第 3 章

API

3.1 LIB

・DLL ではなく、static にリンクされる。

・下記のように用途別（コンパイルオプション別）に分かれている。

EXE_R	.EXE 用(Release)
EXE_D	.EXE 用(Debug)
USRDLL_R	USRDLL 用(Release)
USRDLL_D	USRDLL 用(Debug)
AFXEXT_R	AFXEXT DLL 用(Release)
AFXEXT_D	AFXEXT DLL 用(Debug)

※下の説明で用語がわからない人は VisualC++のオンラインマニュアルを参照。

USRDLL : メモリ確保の空間は各 DLL や EXE で別々。
API の引数は LPCSTR 等の一般的なものに限られる。
この DLL を使う EXE は何でも可（VisualC++で作っていなくてもよい）

AFXEXT : new/delete が EXE-DLL 間で混在可能。
MFC オブジェクトの受け渡しが可能。
この DLL を使う EXE は MFC アプリに限る（CWinApp が必要）

・以下、応用プログラムに有用なもののみ説明する。

def.h

共通の#define 等

utl.h

・_mbsxxxx() に LPCSTR 等の unsigned でない引数を渡してもエラーにならないようにする。

・文字列処理等

utl_str.h

CString で iostream のような operator<< を実現する。

3.2 SPLKEY

#include <splkey.h>

リンク時　　splkey.lib

実行時　　splkey.dll

extern "C" UINT WINAPI GetSplitKey(LPCSTR str);

[機能]

辞書分割キーの取得

[引数]

str　　英語基本形

[戻り値]

bit8 ～ 15　　分割ファイル名１文字目（'@'～'Z'のどれか）

bit0 ～ 7　　分割ファイル名２文字目　　〃

extern "C" void CALLBACK GetSplitKey_rundll(HWND hwnd, HINSTANCE hinst, LPSTR lpszCmdLine, int nCmdShow);

[機能]

上記の rundll32 用（詳細は MSDN Q164787 を参照）

[例]

rundll32 SPLKEY.DLL,_GetSplitKey_rundll@16 abcdefg

3.3 DICINTP

#include <dicintp.h>

リンク時　　dicintp.lib

実行時　　dicintp.dll

extern "C" BOOL WINAPI InterpretEjLine(LPCSTR buf,
char* hinshiE, char* wordE, char* hinshiJ, char* wordJ,
char* comment, size_t siz);

[機能]

e-j 対訳辞書の行解釈

（english-to-japanese.lisp の 1 行から語・品詞を取り出す）

[引数]

buf	辞書の 1 行
hinshiE	品詞(E) を受け取るバッファ
wordE	語(E) を受け取るバッファ
hinshiJ	品詞(J) を受け取るバッファ
wordJ	語(J) を受け取るバッファ
comment	コメントを受け取るバッファ
siz	バッファサイズ

[戻り値]

真 ：正しく解釈できた

偽 ：エラー

[備考]

・""は取り除かれる。¥¥等も元に戻す。

・日本語が分割してある場合はくっつけて取り出す

[辞書の 1 行の例]

((ADJ "articular") (普通名詞 "関節"))

extern "C" BOOL WINAPI InterpretMorphLine(LPCSTR buf,

char* shutsugen, char* kihon, char* hinshi,

char* katsu1, char* katsu2, char* comment, size_t siz);

[機能]

形態素辞書（ema-eng-morph.dic）の行解釈

[引数]

buf	辞書の１行
shutsugen	出現形を受け取るバッファ
kihon	基本形を受け取るバッファ
hinshi	品詞　を受け取るバッファ
katsu1	活用１を受け取るバッファ
katsu2	活用２を受け取るバッファ
comment	コメントを受け取るバッファ
siz	バッファサイズ

[戻り値]

真：正しく解釈できた

偽：エラー

[備考]

・""は取り除かれる。¥¥等も元に戻す。

[辞書の１行の例]

(".mu.kj/mol" "^arabsym" CN NIL SUFFIX)

extern "C" BOOL WINAPI InterpretSemLine(LPCSTR buf,

char* word, char* hinshi, char* imicode, char* comment,

size_t siz);

[機能]

意味コード辞書の行解釈

[引数]

buf	辞書の１行
word	語　　を受け取るバッファ
hinshi	品詞　　を受け取るバッファ
imicode	意味コードを受け取るバッファ
comment	コメント　を受け取るバッファ
siz	バッファサイズ

[戻り値]

真 ：正しく解釈できた

偽 ：エラー

[備考]

・""は取り除かれる。¥¥等も元に戻す。

・意味コードはなくても OK。また、NIL のみなら読み飛ばす。

[辞書の１行の例]

("Japanese orange" CN "051d" "057" "928c")

extern "C" LPCSTR WINAPI GetQuotedToken(LPCSTR si, char* di, size_t siz);

[機能]

""で囲まれたトークンを取り出す（¥¥を元に戻すバージョン）

[引数]

si	source
di	取り出したトークンを受け取るバッファ
siz	バッファサイズ

[戻り値]

トークンの次の位置（si 内を指す） ／ 0 : エラー

[備考]

・""は取り除かれる。

extern "C" LPCSTR WINAPI GetQuotedToken2(LPCSTR si, char* di, size_t siz);

[機能]

""で囲まれたトークンを取り出す（¥¥はそのままのバージョン）

[引数]

si	source
di	取り出したトークンを受け取るバッファ
siz	バッファサイズ

[戻り値]

トークンの次の位置（si 内を指す） ／ 0 : エラー

[備考]

・""は取り除かれる。

extern "C" LPCSTR WINAPI GetSpacedToken(LPCSTR si, char* buf, size_t siz);

[機能]

スペースで区切られたトークンを取り出す

[引数]

si　　source

buf　　取り出したトークンを受け取るバッファ

siz　　バッファサイズ

[戻り値]

トークンの次の位置（si内を指す）　／　0：エラー

[備考]

・¥¥の処理はしない（スペースで区切る仕様なら、もともと¥"とかをする必要はないので、そういうデータはないと仮定）

extern "C" LPCSTR WINAPI GetSpacedOrKakkoedToken(LPCSTR si, char* buf,size_t siz);

[機能]

スペース or ）で区切られたトークンを取り出す

[引数]

si　　source

buf　　取り出したトークンを受け取るバッファ

siz　　バッファサイズ

[戻り値]

トークンの次の位置（si内を指す）　／　0：エラー

[備考]

・¥¥の処理はしない（スペースで区切る仕様なら、もともと¥"とかをする必要はないので、そういうデータはないと仮定）

3.4 DICIO

#include <dicio.h>

リンク時　　dicio.lib

実行時　　dicio.dll

extern "C" BOOL WINAPI IsCommentOut(LPCSTR lpszDicLine);

[機能]

コメントアウトされたものかどうかを調べる

[引数]

lpszDicLine　　辞書の１行

[戻り値]

真 ：はい

偽 ：いいえ

extern "C" UINT WINAPI GetSplitKey_ej(LPCSTR lpszDicLine);

extern "C" UINT WINAPI GetSplitKey_esem(LPCSTR lpszDicLine);

extern "C" UINT WINAPI GetSplitKey_emorph(LPCSTR lpszDicLine);

[機能]

辞書の分割キーの取得（辞書の行を丸ごと引数に渡すバージョン）

上から順に対訳辞書用、意味コード辞書用、形態素辞書用。

※先行するコメント記号は読み飛ばす

[引数]

lpszDicLine　　辞書の１行

[戻り値]

OK: bit8 ～ 15　　分割ファイル名１文字目（'@'～'Z'のどれか）

bit0 ～ 7　　分割ファイル名２文字目　　〃

エラー: 0

extern "C" BOOL WINAPI Dicio_Delete_ej(LPCSTR lpszDir, LPCSTR lpszDicLine);

extern "C" BOOL WINAPI Dicio_Delete_esem(LPCSTR lpszDir, LPCSTR lpszDicLine);

extern "C" BOOL WINAPI Dicio_Delete_emorph(LPCSTR lpszDir, LPCSTR lpszDicLine);

[機能]

辞書の１行を削除

上から順に対訳辞書用、意味コード辞書用、形態素辞書用。

[引数]

lpszDir	分割辞書ﾃﾞｨﾚｸﾄﾘ（末尾に¥が必要）
lpszDicLine	辞書の１行（CR,LF は事前に取ること）

[戻り値]

真 ：OK

偽 ：エラー

```
extern "C" BOOL WINAPI Dicio_Add_ej( LPCSTR lpszDir,
                                     LPCSTR const *lpszAddLines, UINT kosu );
extern "C" BOOL WINAPI Dicio_Add_esem( LPCSTR lpszDir,
                                     LPCSTR const *lpszAddLines, UINT kosu );
extern "C" BOOL WINAPI Dicio_Add_emorph( LPCSTR lpszDirManu,
                                     LPCSTR lpszDirReadonly,
                                     LPCSTR const *lpszAddLines, UINT kosu );
```

[機能]

辞書に追加（複数行まとめて＆重複チェック付き）

上から順に対訳辞書用、意味コード辞書用、形態素辞書用。

[引数]

lpszDir	分割辞書ﾃﾞｨﾚｸﾄﾘ（末尾に¥が必要）
lpszAddLines	追加行たち（LPCSTR の配列）… CR,LF は不要
kosu	その個数

※形態素辞書用は引数が若干異なる：

lpszDirManu	分割辞書ﾃﾞｨﾚｸﾄﾘ(-manual の手編集用)
lpszDirReadonly	分割辞書ﾃﾞｨﾚｸﾄﾘ(-manual でない ReadOnly のほう)

[戻り値]

真 ：OK

偽 ：エラー

[備考]

・重複チェックは大文字／小文字が違っても検出する。

・CutExtraSpace()は必要なら呼び出し側でする（EMORT のミス？のために空白を複数並べている場合が実際にある）

```
extern "C" void CALLBACK Add_ej_rundll( HWND hwnd, HINSTANCE hinst,
                                        LPSTR lpszCmdLine, int nCmdShow );
extern "C" void CALLBACK Add_esem_rundll( HWND hwnd, HINSTANCE hinst,
                                        LPSTR lpszCmdLine, int nCmdShow );
extern "C" void CALLBACK Add_emorph_rundll( HWND hwnd, HINSTANCE hinst,
                                        LPSTR lpszCmdLine, int nCmdShow );
```

[機能]

上記の rundll32 用（詳細は MSDN Q164787 を参照）

上から順に対訳辞書用、意味コード辞書用、形態素辞書用。

[引数]

dir¥ 追加する行

または

"dir¥¥" 追加する行

※「dir¥」と「追加する行」の間のスペースは１個だけ読み飛ばす。

※ Add_emorph_rundll は

dir_manual¥ dir_ReadOnly¥ 追加する行

[例１]

```
rundll32 DICIO.DLL,_Add_ej_rundll@16 d:¥dd¥ ((ADJ "articular") (普通名詞 "関節"))
```

[例２]

```
rundll32 DICIO.DLL,_Add_ej_rundll@16 "d:¥¥dd¥¥" ((ADJ "articular") (普通名詞 "関節"))
```

```
extern "C" BOOL WINAPI Dicio_AddFile_ej( LPCSTR lpszDir, LPCSTR fname_src );
extern "C" BOOL WINAPI Dicio_AddFile_esem( LPCSTR lpszDir, LPCSTR fname_src );
extern "C" BOOL WINAPI Dicio_AddFile_emorph( LPCSTR lpszDirManu,
                                             LPCSTR lpszDirReadonly,
                                             LPCSTR fname_src );
```

[機能]

辞書に追加（ファイルの内容をまとめて追加）

上から順に対訳辞書用、意味コード辞書用、形態素辞書用。

[引数]

lpszDir	分割辞書ﾃﾞｨﾚｸﾄﾘ（末尾に¥が必要）
fname_src	追加リストのファイル名

※形態素辞書用は引数が若干異なる：

lpszDirManu	分割辞書ﾃﾞｨﾚｸﾄﾘ(-manual の手編集用)
lpszDirReadonly	分割辞書ﾃﾞｨﾚｸﾄﾘ(-manual でない ReadOnly のほう)

[戻り値]

真 ：OK

偽 ：エラー

[備考]

・重複チェックは大文字／小文字が違っても検出する。

・CutExtraSpace()は必要なら事前にする（EMORT のミス？のために空白を複数並べている場合が実際にある）

```
extern "C" void CALLBACK AddFile_ej_rundll( HWND hwnd, HINSTANCE hinst,
                                            LPSTR lpszCmdLine, int nCmdShow );
extern "C" void CALLBACK AddFile_esem_rundll( HWND hwnd, HINSTANCE hinst,
                                            LPSTR lpszCmdLine, int nCmdShow );
extern "C" void CALLBACK AddFile_emorph_rundll( HWND hwnd, HINSTANCE hinst,
                                            LPSTR lpszCmdLine, int nCmdShow );
```

[機能]

上記の rundll32 用（詳細は MSDN Q164787 を参照）

上から順に対訳辞書用、意味コード辞書用、形態素辞書用。

[引数]

dir¥ 追加ファイル

または

"dir¥¥" "追加ファイル"

※ AddFile_emorph_rundll は

dir_manual¥ dir_ReadOnly¥ 追加ファイル

[例]

rundll32 DICIO.DLL,_AddFile_ej_rundll@16 d:¥dd¥ d:¥addlist

```
extern "C" int WINAPI Dicio_Add_ej_one( LPCSTR lpszDir, LPCSTR lpszAdd );
extern "C" int WINAPI Dicio_Add_esem_one( LPCSTR lpszDir, LPCSTR lpszAdd );
extern "C" int WINAPI Dicio_Add_emorph_one( LPCSTR lpszDirManu,
                                            LPCSTR lpszDirReadonly,
                                            LPCSTR lpszAdd );
```

[機能]

辞書に追加（１行のみ・YES か NO かを返すバージョン）

上から順に対訳辞書用、意味コード辞書用、形態素辞書用。

[引数]

lpszDir	分割辞書ﾃﾞｨﾚｸﾄﾘ（末尾に¥が必要）
lpszAdd	追加行… CR,LF は不要

※形態素辞書用は引数が若干異なる：

lpszDirManu	分割辞書ﾃﾞｨﾚｸﾄﾘ(-manual の手編集用)
lpszDirReadonly	分割辞書ﾃﾞｨﾚｸﾄﾘ(-manual でない ReadOnly のほう)

[戻り値]

IDYES：登録した

IDNO：重複チェックで「いいえ」と答えたので登録せず

IDCANCEL：エラー

[備考]

・重複チェックは大文字／小文字が違っても検出する。

・CutExtraSpace()は必要なら呼び出し側でする（EMORT のミス？のために空白を複数並べている場合が実際にある）

extern "C" LPCSTR WINAPI SearchMyself(LPCSTR lpszDir, LPCSTR lpszE, LPCSTR lpszHin=0);

[機能]

自分自身（意味コード）の検索

[引数]

lpszDir	分割辞書ディレクトリ（末尾に¥が必要）
lpszE	検索する英語
lpszHin	検索する品詞 ／ 0:何でもよい

[戻り値]

意味コード ／ 0：見つからず

[備考]

・事前に patch_mbslwr() を実行すること。

・戻り値は static のバッファを指す。

class CLockDir

CLockDir::CLockDir(LPCSTR lpszDir, DWORD dwDesiredAccess);

CLockDir::~CLockDir();

BOOL CLockDir::IsLocked() const;

[機能]

ディレクトリのロック（複数のプログラムが同時にアクセスしないように排他処理を行う）

[引数]

lpszDir	ディレクトリ名（末尾に¥を付けること）
dwDesiredAccess	GENERIC_READ | GENERIC_WRITE など（詳細は CreateFile の説明を参照）

[使い方]

スタック上に CLockDir オブジェクトを作る。

その寿命の間ロックが有効（デストラクタでロックが解除される）

[例]

```
CLockDir lock(lpszDir, GENERIC_READ | GENERIC_WRITE);
if(!lock.IsLocked())    return FALSE;
```

[備考]

cooperative な方法なので、抜けがないように呼び出し側で注意すること。

3.5 DICLOGIC

#include <diclogic.h>

リンク時　　diclogic.lib

実行時　　diclogic.dll

extern "C" BOOL WINAPI ReadMyself(LPCSTR fname);

[機能]

自分自身（意味コード辞書）をハッシュテーブルに読み込む

[引数]

fname　ファイル名

[戻り値]

真 : OK

偽 : エラー

extern "C" BOOL WINAPI ReadKadokawa(LPCSTR fname1, LPCSTR fname2);

[機能]

角川辞書をハッシュテーブルに読み込む（キャッシュ１から）

& キャッシュ２の作成

[引数]

fname1　キャッシュ１のファイル名

fname2　キャッシュ２のファイル名

[戻り値]

真 : OK

偽 : エラー

extern "C" BOOL WINAPI ReadKadokawa_by_Cache(LPCSTR fname2);

[機能]

角川辞書をハッシュテーブルに読み込む（キャッシュ２から）

[引数]

fname2　キャッシュ２のファイル名

[戻り値]

真 ：OK

偽 ：エラー

enum eSEM_RETCODE

[意味]

意味コードを自動推定した方法を表す

[値]

SEM_ALREADY	既に登録済み
SEM_MYSELF	自分自身を参照
SEM_JIRIKI1	自力で付与(1)
SEM_KADOKAWA	角川を参照
SEM_JIRIKI2	自力で付与(2)
SEM_JIRIKI3	自力で付与(3)
SEM_2GO	主要な語を探し、自分自身を参照
SEM_SPLIT_KADO	区切って再試行、角川を参照
SEM_JIRIKI4	自力で付与(4)
SEM_JIRIKI5	自力で付与(5)
SEM_NG	付与できなかった

```
extern "C" eSEM_RETCODE WINAPI AutoSemCode(LPCSTR lpszHinshiE, LPCSTR lpszE,
                                           LPCSTR lpszHinshiJ, LPCSTR lpszJ,
                                           LPCSTR lpszDirES,
                                           LPCSTR* plpszCode, LPCSTR* pInf,
                                           EZenkakuType* pKugire, EZenkakuType* pKataOnly );
```

[機能]

意味コードの自動推定

[引数]

lpszHinshiE	品詞(E)
lpszE	語(E)
lpszHinshiJ	品詞(J)
lpszJ	語(J)
lpszDirES	自分自身のディレクトリ（末尾に¥必要）
plpszCode	結果を受け取るポインタのアドレス ：角川コードの羅列("012 345")／NULL
pInf	どういう語形変化をしたかの情報を受け取るポインタのアドレス ／変化なし等で情報がない場合は NULL or "" ※ static 領域を指すので次回呼び出し後は無効になる
pKugire	カタカナ等の途中で区切れた／ZEN_ETC:そうでない
pKataOnly	区切った結果がカタカナ等のみ／ZEN_ETC:そうでない

[戻り値]

推定した方法

[備考]

lpszE は事前に小文字にすること。

```
enum EZenkakuType    {
    ZEN_ETC,
    ZEN_HIRA,
    ZEN_KATA,
    ZEN_ALPH
};
```

extern "C" int WINAPI AutoHinshi(LPCSTR lpszE, LPCSTR lpszJ,
LPCSTR* hinshiE, LPCSTR* hinshiJ);

[機能]

品詞の自動推定

[引数]

lpszE	語(E)
lpszJ	語(J)
hinshiE	品詞(E)の推定結果を受け取るポインタのアドレス
hinshiJ	品詞(J)の推定結果を受け取るポインタのアドレス

[戻り値]

真 : OK (AH_END_HIRA:最後の文字はひらがな)

偽 : 推定できず

extern "C" BOOL WINAPI AutoHinshi_pre(LPCSTR lpszE, LPCSTR* hinshiE);

[機能]

品詞推定の前処理 (日本語不要の品詞かどうかの判定)

(^arabic、^arabsym は日本語不要なので、日本語なしで推定を試行)

[引数]

lpszE	語(E)
hinshiE	品詞(E)の推定結果を受け取るポインタのアドレス

[戻り値]

真 : OK

偽 : 推定できず

extern "C" void WINAPI GetPrincipalWord(LPCSTR lpszE, char* bufPrin, char* bufRemain);

[機能]

二語以上の単語の活用形変化の準備：

意味をなす上で重要な単語までと、残りの文字列に分割

[引数]

lpszE	語(E)
bufPrin	意味をなす上で重要な単語までを受け取るバッファ
bufRemain	残りの文字列を受け取るバッファ

extern "C" void WINAPI Katsuyou_PL(LPCSTR lpszE, char* buf);

extern "C" void WINAPI Katsuyou_PAST(LPCSTR lpszE, char* buf);

extern "C" void WINAPI Katsuyou_ING(LPCSTR lpszE, char* buf);

extern "C" void WINAPI Katsuyou_3S(LPCSTR lpszE, char* buf);

extern "C" void WINAPI Katsuyou_ER(LPCSTR lpszE, char* buf);

extern "C" void WINAPI Katsuyou_EST(LPCSTR lpszE, char* buf);

[機能]

英語活用形の推定

[引数]

lpszE	語(E)
buf	活用形を受け取るバッファ

[備考]

GetPrincipalWord 実行後に呼ぶこと。

class CKadokawaCD

［機能］

角川類語新辞典 CD の読み込みを行うクラス。

［備考］

#include <kado_cd.h>

CKadokawaCD::CKadokawaCD();

CKadokawaCD::~CKadokawaCD();

［機能］

コンストラクタとデストラクタ

BOOL CKadokawaCD::Make_midasi_sort_txt(LPCSTR fname1, LPCSTR fname2,
LPCSTR fnameo);

［機能］

角川 CD から midasi_sort.txt 相当のファイルを作成

［引数］

fname1　1 つ前のファイル

fname2　角川 CD のファイル

fnameo　作成する midasi_sort.txt 相当のファイル名

static void CKadokawaCD::Scramble(char* buf);

［機能］

misasi_sort.txt 相当のファイルの行にスクランブルをかける／はずす

3.6 DICDLG

#include：個別に記述

リンク時　　dicdlg.lib

実行時　　dicdlg.dll

CDialog* AFXAPI CreateRuiModeless(LPCSTR lpszMidasi_sort, CWnd* pWnd, UINT nID);

[機能]

CRuiDialog（角川類語新辞典検索）のモードレス・ダイアログを作成

[引数]

lpszMidasi_sort	キャッシュ１（midasi_sort.txt 相当のファイル）の名前
pWnd	ダブルクリックで転送する先の Dialog
nID	ダブルクリックで転送する先の Edit コントロール

[戻り値]

作成したオブジェクト（プログラム終了時に delete すること）

[インクルード]

```
#include <dicdlg.h>
```

[例]

```
if(!theApp.m_pRui){
    … CTourokuDialog を作成（theApp.m_pTouroku） …
    theApp.m_pRui = CreateRuiModeless( theApp.midasi_sort(),
                                        theApp.m_pTouroku, IDC_EDIT_SEMCODE );
}
```

BOOL AFXAPI InitKado(LPCSTR lpszZkana, LPCSTR lpszHonmon,
LPCSTR lpszMidasi_sort, LPCSTR lpszCache2);

[機能]

角川類語新辞典 CD の初期化（読み込み・キャッシュ作成）

[引数]

lpszZkana	角川 CD の"ZKANA"のファイル名
lpszHonmon	角川 CD の"HONMON"のファイル名
lpszMidasi_sort	キャッシュ１のファイル名
lpszCache2	キャッシュ２のファイル名

[戻り値]

真 : OK

偽 : エラー

[インクルード]

#include <dicdlg.h>

[備考]

・USRDLL が複数箇所で使われると気持ち悪いので、CKadokawaCD, ReadKadokawa, ReadKadokawa_by_Cache を橋渡しする。

・角川 CD には同じ名前のファイルが複数箇所にあるので注意すること。

[例]

```
CString   fnameZ = GetProfileString("CD","zkana" ,"TITLE¥¥ZKANA");
CString   fnameH = GetProfileString("CD","honmon","RUIGO.DAT¥¥HONMON");
CString   fnameC = GetProfileString("CD","cache2","midasi2.tmp");
if( ! InitKado( fnameZ, fnameH, m_midasi_sort, fnameC )) {
    return FALSE;
}
```

class CKensakuDialog

[機能]

辞書検索条件のダイアログ

[インクルード]

#include "KensakuDialog.h"

[使い方]

普通に処理をした後、CResultDialog::Kensaku() に渡せばよい。

[例]

```
CKensakuDialog    dlg;
if(dlg.DoModal() != IDOK) return;
… CResultDialog 作成 (theApp.m_pResult) …
theApp.m_pResult->Kensaku( dlg );
```

class CResultDialog

[機能]

検索結果のダイアログ（モードレス）

[インクルード]

```
#include "ResultDialog.h"
```

[設定が必要なクラスメンバ]

```
CString   m_DirEJ;     対訳辞書のディレクトリ
CString   m_DirES;     意味コード辞書のディレクトリ
CString   m_DirEMM;    形態素辞書（-manual）のディレクトリ
CString   m_DirEMA;    形態素辞書（ReadOnly）のディレクトリ
```

[使い方]

（1）引数のないコンストラクタで作成

（2）Create(CResultDialog::IDD)を呼ぶ

（3）m_DirEJ, m_DirES, m_DirEMM, m_DirEMA を設定

[例]

```
if(!theApp.m_pResult){
    CResultDialog*     pDlg = new CResultDialog;
    pDlg->Create(CResultDialog::IDD);
    pDlg->m_DirEJ  = theApp.DirEJ();
    pDlg->m_DirES  = theApp.DirES();
    pDlg->m_DirEMM = theApp.DirEMM();
    pDlg->m_DirEMA = theApp.DirEMA();
    theApp.m_pResult = pDlg;
}
theApp.m_pResult->ShowWindow(SW_SHOW);
theApp.m_pResult->SetFocus();
```

void CResultDialog::Kensaku(CKensakuDialog& dlg);

[機能]

dlg で指定した条件で辞書検索を行う

[例]

```
CKensakuDialog    dlg;
if(dlg.DoModal() != IDOK) return;
… CResultDialog 作成 (theApp.m_pResult)  …
theApp.m_pResult->Kensaku( dlg );
```

class CTourokuDialog

[機能]

辞書登録のダイアログ（モーダル・モードレスどちらか選択可能）

[インクルード]

```
#include "TourokuDialog.h"
```

[設定が必要なクラスメンバ]

```
CString   m_DirEJ;     対訳辞書のディレクトリ
CString   m_DirES;     意味コード辞書のディレクトリ
CString   m_DirEMM;    形態素辞書（-manual）のディレクトリ
CString   m_DirEMA;    形態素辞書（ReadOnly）のディレクトリ
```

[使い方]

・モードレスの場合

```
CTourokuDialog*   pDlg = new CTourokuDialog;
pDlg->Create(CTourokuDialog::IDD);
pDlg->m_DirEJ  = theApp.DirEJ();
pDlg->m_DirES  = theApp.DirES();
pDlg->m_DirEMM = theApp.DirEMM();
pDlg->m_DirEMA = theApp.DirEMA();
pDlg->ShowWindow(SW_SHOW);
pDlg->SetFocus();
```

・モーダルの場合

```
CTourokuDialog    dlg(CTourokuDialog::IDD);
dlg.m_DirEJ  = theApp.DirEJ();
dlg.m_DirES  = theApp.DirES();
dlg.m_DirEMM = theApp.DirEMM();
dlg.m_DirEMA = theApp.DirEMA();
dlg.m_wordE = "english";
dlg.m_wordJ = "日本語";
dlg.DoModal();
```

第 4 章

自動推定

4.1 概要

●品詞

日本語と英語の語形のみを見て推定を行っている。事前に傾向を調査し、その結果に基づいて推定ロジックを作成した。たとえば、

(a) 日本語の末尾が "る" なら V/本動詞

(b) 英語の末尾が "ly" なら ADV/副詞

のような法則を、確からしいと思われる順にあてはめてみて、最初に適合した法則で推定を行っている。

●活用形

英語の語形のみを見て推定を行っている。必要となる活用形が英語品詞によって異なるので、英語品詞を推定した後で活用形を推定している。事前に傾向を調査し、その結果に基づいて推定ロジックを作成した。概略は以下の通りである。

(1) of, for 等の前置詞以降を落とし、その前の主要な語に視点を当てる。さらに、その語が過去分詞のように見える場合はもう一つ前の語に視点を当てる。

(2) 視点を当てた語に対し、複数形なら s、過去形なら ed の付加等の処理をする。

(3) (1)で落とした部分を元通りつなげる。

●意味コード

日本語と英語の語形と品詞により推定を行っている。必要なら下記の通り各種の辞書を参照している。事前に傾向を調査し、その結果に基づいて推定ロジックを作成したが、大まかに分類すると以下のようになる。

(a) 英語で自分自身（過去に手作業で付与した意味コード辞書）を調べる。

(c) 日本語で角川類語新辞典を調べる。

(b) 何も参照せずに自力で付与。(定型パターン)

また、以下の処理も併用している。

(ｱ) 語形変化をさせて調べる。(英・日とも)

(ｲ) 英語が二語以上の単語の場合、意味をなす上で重要な単語に視点をあてる。

(ｳ) 日本語の一部を区切って再試行する。

なお、プログラムでは、先に確からしい推定方法で試行し、後になるに従って怪しい方法になるようにしている。

4.2 ソース

自動推定のルーチンはは diclogic.dll にある。

autohin.cpp	品詞の自動推定
autosem.cpp	意味コードの自動推定の主要部分
jiriki.cpp	意味コードを何も参照せずに自力で推定
katsuyou.cpp	活用形の推定

4.3 関数の説明

推定ロジックを調整する場合は以下の関数を使うとよい。

static inline BOOL MatchHead(LPCSTR str, LPCSTR pattern)

[機能]

文字列 str の先頭が pattern になっているかどうかを調べる

[引数]

str　　調べる対象の文字列

pattern　一致パターン

[戻り値]

真 ：一致した

偽 ：一致せず

[インクルード]

#include <utl.h>

[例]

if(MatchHead(lpszJ,"（株）"))　　…

BOOL MatchTail(LPCSTR str, LPCSTR pattern);

[機能]

文字列 str の末尾が pattern になっているかどうかを調べる

[引数]

str　　調べる対象の文字列

pattern　一致パターン

[戻り値]

真 ：一致した

偽 ：一致せず

[備考]

全角文字列も使用可能。str の全角文字の２バイト目が間違って判定されることはない。

[インクルード]

#include <utl.h>

[例]

if(MatchTail(lpszJ,"研究所"))　　…

BOOL MatchTail_w(LPCSTR str, LPCSTR pattern);

[機能]

文字列 str の末尾が pattern になっているかどうか、かつ、その直前が英単語の境目かどうかを調べる

[引数]

str　　調べる対象の文字列

pattern　一致パターン

[戻り値]

真 ：一致した

偽 ：一致せず

[備考]

「直前が英単語の境目かどうか」とは「 」「-」「,」なら OK。

文字列の先頭のため「直前」が存在しない場合も OK。

また、このチェックがあるため、全角文字列の処理は特に必要ない。

[インクルード]

#include <utl.h>

[例]

```
if( MatchTail_w(lpszE,"method"))    …
```

BOOL ReplaceTail(char* buf, LPCSTR str, LPCSTR pattern, LPCSTR replace)

[機能]

末尾の文字列置換（str の末尾の pattern を replace に置換）

[引数]

buf　　結果の格納先（呼び出し側で十分なサイズを用意すること）

str　　対象文字列

pattern　検索パターン（＝置換前）

replace　置換後パターン

[戻り値]

TRUE : パターン一致、置換結果は buf に格納

FALSE : パターン不一致

[インクルード]

#include <utl.h>

[例]

```
if( ReplaceTail( buf, lpszE, "y", "ies" ) )        …
```

void AddTail(char* buf, LPCSTR str, LPCSTR add)

[機能]

末尾に文字列を追加

[引数]

buf　　結果の格納先（呼び出し側で十分なサイズを用意すること）

str　　対象文字列

add　　追加パターン

[インクルード]

#include <utl.h>

[例]

```
AddTail( buf, lpszE, "s" );
```

第５章

今後の課題

・日英方向等、他の言語に拡張する場合は、辞書分割の仕様を考え直す必要があるほか、意味コードや品詞の自動推定のロジックを考える必要がある。

・定型パターンなどはユーザがカスタマイズできるように改良するのもよいと思われる。

・未知語のリストから追加する機能や、まとめて追加するツールは、作業の中断に関してあまり考慮されていないが、実際の使用時には大量のリストを扱う可能性があり、途中で作業を中断し、後で再開するような仕組みを作る必要があると思われる。

・語の追加、削除等のログを取るような機能も安全面で有効と思われる。

・今後は専門用語辞書の利用等が考えられるため、複数の辞書を管理できるようにすることが必要になると思われる。

参考文献

(1) 古瀬 蔵, 隅田英一郎, 飯田 仁: “経験的知識を活用する変換主導型機械翻訳”, 情報処理学会論文誌, Vol.35, No.3, pp.414-425(1994)

(2) 大野晋, 浜西正人: “角川類語新辞典 CD-ROM 版”, 角川書店, (1989)

(3) 鷹尾 和享, 柏岡 秀紀, 白井 諭: “TDMT 辞書ツール説明書(英日版・ユーザ向け)”, ATR テクニカルレポート TR-IT-0313, (1999-9)

(4) 鷹尾 和享, 柏岡 秀紀, 白井 諭: “異なる辞書を利用した意味コードの自動付与”, 情報処理学会第 59 回全国大会, 1N-07, (1999-9)