表 2　実験 2 における自動修正の内訳

修正の正誤	修正前	修正後
正しい修正(10 個)	かいみん / ほー (誤)	かいみんほー (正)
	てつや / あけに (誤)	てつやあけに (正)
	つぶる / だけでも (誤)	つぶるだけでも (正)
	しまう / からです (誤)	しまうからです (正)
	つかれ / させない (誤)	つかれさせない (正)
	ききて / ばかり (誤)	ききてばかり (正)
	かいぜん / やく (誤)	かいぜんやく (正)
	つぼ / おし (誤)	つぼおし (正)
	つぼ / おしわ (誤)	つぼおしわ (正)
	なる / など (誤)	なるなど (正)
誤った修正(2 個) (誤)	たんみんから (正)	たん / みんから(誤)
	したぶぶんに (正)	した / ぶぶんに(誤)

3.5.3 考察

全国の点字図書館の運営しているネットワークである、「サピエ図書館」に登録されている、一般的な点訳ファイルを、無作為に 10 巻選び、あえてコーパスによる自動修正にかけたところ、合計で 87 個の分かち書きの誤りが検出された。分かち書き個所の合計は 93566 個であり、割合で表現すると 0.093 パーセントである。これについては熟練点訳者による修正は行わなかったが、自動修正で検出できなかった誤りが存在すると推測されることも考慮すると、0.1 パーセント程度の誤りは内包していると考えられる。

3.6　実験 3.　自動点訳による点訳の修正

市販、またはフリーウェアとして公開されている自動点訳エンジンを前段階として利用し、その点訳結果をコーパスにより修正した場合の有効性について検証した。

3.6.1　手順

評価指標はテスト用文書を用意し、それぞれの自動点訳エンジンで点訳し、誤りの数を計測した。テスト用文書には、異なる分野からなる、20 本の漢字かな交じり文(最小 556 字、最大 1562 字、平均 1134 字から構成されている)を用意し、熟練点訳者が点訳したものを正解とした。テスト用文書の分野は、人文社会、法律、小説、点字新聞、理療科教科書、高校生物、高校歴史、エッセイ、病院リスト、広報誌の 10 種類である。

3.6.2　結果

本研究開発システムによる修正の結果を表 3 に示す。

表 3 自動点訳による点訳に対する本研究開発システムによる修正

点訳エンジン名	修正前誤り数(個)	修正後誤り数(個)	誤り数減少率(%)
エクストラ	111	60	45.95
IbukiTenC	195	132	32.31
お点ちゃん	204	157	23.04
合計	510	349	31.56

- 「エクストラ」[2]は市販ソフトである。石川准によって開発された。
- 「Ibuki-ten」,「Ibuki-tenC」[3]は形態素解析エンジン Ibuki を使用した自動点訳エンジンである。
- 「お点ちゃん」[4]は勝沼貞幸氏によるフリーソフトである。

3.6.3　考察

いずれの自動点訳エンジンの場合も、分かち書きの誤り数は減少した。成績が最も良好であった点訳エンジンであるエクストラでの修正の内訳を詳しく調べたところ、正しい修正が 54 個、誤った修正(副作用)が 3 個で、差し引きで誤りは 51 個減少した。

副作用が発生したのは、以下の語である。

・ 異化される(化学用語) [正:いか / される→ 誤:いかされる(行かされる、生かされる)]
・ 北安東(地名) [正:きたあんどー → 誤:きた / あんどー(来た安藤)]
・ 腎内科 [正:じん / ないか→誤:じんないか(陣内か)]

また、本研究開発システムをもってしても修正することができなかった誤りについて考察したところ、以下の傾向がみられた。

a. 地名、人名以外の固有名詞(例、協立湊病院)
b. 高度に専門的な用語(例、錐体外路系、チロシンキナーゼ)
c. 点訳者によって判断が分かれる語(例、客相手等)
d. 点訳規則上、意味まで理解しないと判断できない語(例、「武田氏」は一人の個人を指す場合と、武田一族を指す場合で分かち書きが異なる。)

このうち、a.と b.は該当単語がコーパスにまったく含まれていなかった場合、c.と d.はコーパスに、切った場合と続けた場合の両方がある程度含まれている場合である。いずれの場合も、警告を表示することは可能であるものの、自動的に修正することは困難であると考えられる。

3.7 実験 4. 自動点訳後、読みの誤りを修正した後の修正
自動点訳は、読み下しについても、完璧に読み下せるわけではないので、読みの誤りを修正する必要がある。この読みの誤りを修正した部分については、分かち書きも誤っている可能性が高い。読み下しについては修正できるが、点字の分かち書きの知識は無いユーザーが使うものと仮定し、読みの誤りを修正した後、その点訳結果をコーパスにより修正した場合の有効性について検証した。

3.7.1 手順
実験 3 では、分かち書きの誤りについてのみ着目したが、もちろん同時に読みの誤りも存在しており、その数は 133 であった。この読みの誤りを単純に(分かち書きを修正せずに)修正した場合、読みの誤りは 0 となるが、新たに分かち書きの誤りは 26 増加する。この新たに発生した誤りを点訳コーパスによる自動修正で修正した。

3.7.2 結果
修正した前後の誤り数を表 4 に示す。

表 4 自動点訳後、読みの誤りを修正した際に新たに発生した分かち書きの誤りの修正

点訳エンジン名	修正前誤り数(個)	修正後誤り数(個)	減少率(%)
エクストラ	26	10	61.5

3.7.3 考察
誤り数は減少し、この場合にも本研究開発システムが有効であると考えられる。

3.8 実験 5. MeCab を使った自動点訳エンジンの試作
形態素解析エンジンの MeCab を使った自動点訳エンジンを試作し、公開されている自動点訳エンジンと性能を比較した。

3.8.1 手順
評価指標、手順は実験 3 と同じである。

3.8.2 結果
MeCab を使った自動点訳システムによる修正の結果を表 5 に示す。

表 5 MeCab を使った自動点訳エンジンと、公開されている自動点訳エンジンとの性能の比較

点訳エンジン名	修正後誤り数(個)
エクストラ	60
IbukiTenC	132

お点ちゃん	157
MeCab を使った 試作自動点訳エンジン	168

3.8.3 考察
試作した自動点訳エンジンは、いずれの公開されている自動点訳エンジンよりも悪い値となった。形態素解析の性能は当然重要であると考えられるが、作りこみによる点訳精度の向上という部分も大きいということが考察される。

3.9 実験 6a 後続文字数の短縮
点訳コーパスのファイルサイズはインデックスが 0.7G、データが 32GB 程度の合計で約 33GB である。市販ソフトに組み込む場合は、大きすぎるため、どの程度まで小さくしてもよいのかを調べた。
先行文字列は、分かち書きの辞書として使用する場合に重要であるが、自動修正においては使用しない。そのため、先行文字列は削除する。
後続文字列は、スペースを含まずに 14 文字となっているが、まずこれを短縮することを考える。

3.9.1 結果
後続文字列を短縮し、自動修正した結果を以下に示す。使用したテスト文書は、実験 2 と同じもので、使用した点訳エンジンはエクストラである。
結果を表 6 に示す。

表 6 後続文字数の短縮の効果

後続文字数	誤り数(個)
14 文字	119
12 文字	119
10 文字	119
8 文字	119
7 文字	120
6 文字	140
修正なし	182

3.9.2 考察
結果より、後続文字列は 8 文字程度までは短縮しても結果に影響がないと考えられる。

3.10 実験 6b コーパスに含まれるタイトル数の削減
次に、後続文字列を 8 文字に固定し、タイトル数を削減した場合の誤り数を調べた。使用したテスト文書は、実験 2、実験 3a と同じもので、使用した点訳エンジンはエクストラである。

3.10.1 結果
結果を表 7 に示す。

表 7　後続文字数の短縮の効果

後続文字数	誤り数(個)	ファイルサイズ
タイトル数 1/1	119	8GB
タイトル数 1/2	122	4GB
タイトル数 1/3	120	3GB
タイトル数 1/4	117	2GB
タイトル数 1/5	124	1.6GB
タイトル数 1/10	124	0.8GB
タイトル数 1/15	131	0.6GB
タイトル数 1/20	140	0.4GB
修正なし	182	

3.10.2 考察
1 つの語に対し、データを、後続文字列 8 文字(8 バイト)、マス空け(スペース)情報 1 バイトの合計 9 バイトで構成し、コーパスに含まれる点訳ファイルのタイトル数を 1/4 にすれば値サイズとデータサイズを合わせて約 3GB になり、
配布時に DVD1 枚に格納できると考えられる。
この場合、自動修正としての機能については上記の結果で示されるように、さほど性能が落ちるわけではないが、ファイル ID、ファイル内での位置情報、先行文字列の情報はコーパスに含まれないことになるので、点訳の分かち書き辞書として使用した場合の結果の見やすさは低下する。また、ある用語からそれを含む本を検索したい場合なのどの全文検索機能は持ち合わせていない。

4.まとめ
現時点では、今回試作した自動点訳エンジンに比べ、前段階として市販の自動点訳エンジンをそのまま用い、後段階としてコーパスによる修正をおこなうほうが良い結果であり、現時点ではベストの選択であることを示した。ただし、理工学など、分野によっては試作の自動点訳エンジンのほうが良い結果を示したものもある。
本研究開発システムによる後修正を行うことにより、点訳にさほど熟練していない者が点訳した場合や、自動点訳エンジンが点訳した場合において、より正確な分かち書きを行うことができることを示した。
　しかし、点訳には、分かち書きだけではなく、難しい熟語の読みの正確さや、場合によっては図書の内容についての知識、また点訳基準についての知識などが必要であり、本研究開発システムを使ったからといって、熟練点訳者と同じ点訳がすぐにできるわけではないことは勿論である。

点字自動翻訳技術は文字通り地味な仕事ではあるが、長期継続的努力により少しずつではあっても進歩していくと考えている。

References

[1] 全国視覚障害情報提供施設協会　点訳のてびき　第3版 (2002)
[2] 石川　准:自動点訳ソフト「エクストラ」,http://www.extra.co.jp/
[3] 兵藤安昭, 横平貫志, 早川哲史, 村上 裕, 池田尚志:誤り個所指摘機能をもたせた点字翻訳編集システムIBUKI-TEN, 電子情報通信学会論文誌. D-I, Vol.84, No.7, pp.1102-1111(2001).
[4] 勝沼貞幸:パソコン点訳ソフト 「お点ちゃん」, http://www17.plala.or.jp/otenchan/.
[5] 橘 美紗, 鈴木昌和: 数学文書点訳における日本語分かち書き仮名変換処理, 電子情報通信学会研究報告. TL, 思考と言語, Vol.106, No.486, pp.7-11 (2007)
[6] 北 研二, 津田和彦, 獅々堀正幹: 情報検索アルゴリズム, 共立出版(2002).
[7] 吉村賢治, 日高 達, 吉田 将: 文節数最少法を用いたべた書き日本語文の形態素解析, 情報処理学会論文誌, Vol.24, No.1, pp.40-46(1983).
[8] 『点字表記辞典改定新版』編集委員会: 点字表記辞典　改定新版(2002).
[9] 日本点字委員会: 日本点字表記法 2001 年版(2001).

B -6 複数候補の提示と修正 UI 開発

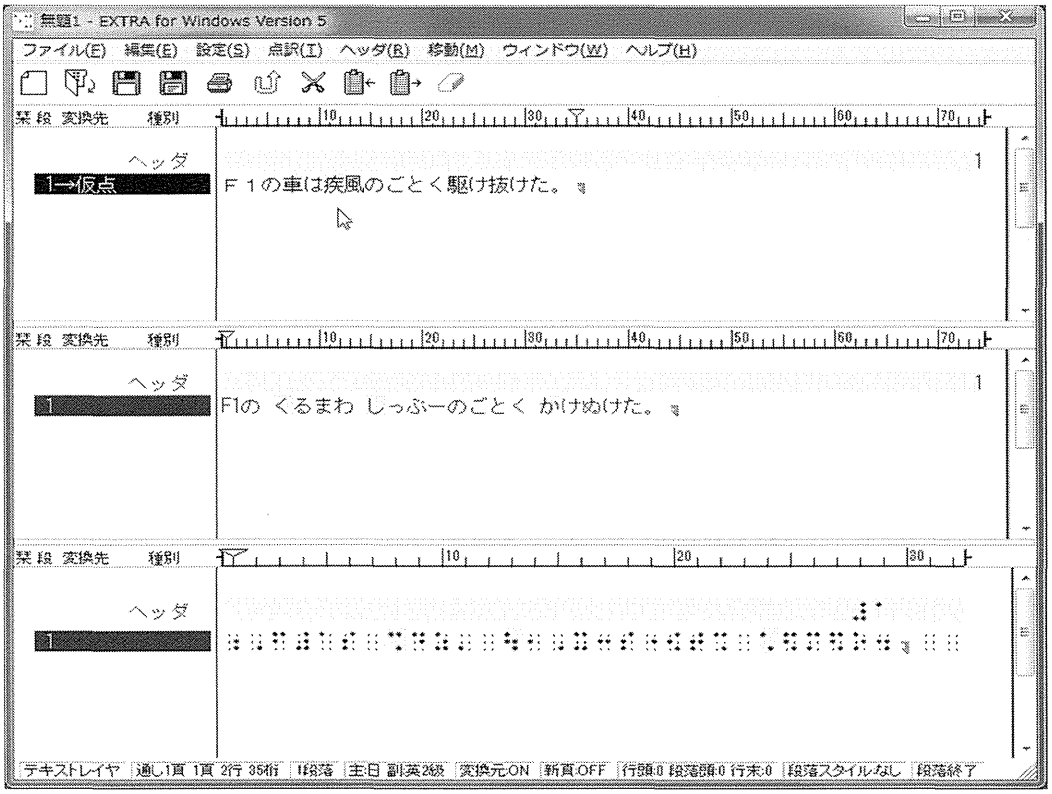
■ 目的

日本語の点訳では、テキストの漢字部分を読み情報に変換する必要がある。ところが、昨日(きのう/さくじつ)、五月(ごがつ/さつき)、紅葉(もみじ/こうよう)のように、読みが一意に定まらないことがある。とくに人名や地名などでは複数の読み候補が発生し、前後の文章だけでは判定できないものも多い。
複数の読み候補の中からどの候補を選ぶべきかという問題は、前後の品詞や用語などの情報を元にはっきりと判断できる場合もある。しかし、長年の研究にも関わらずまだ十分に確立していない知識フレームワークなどの人口知能の研究領域に及ぶ技術を必要とするものもある。
現在の自動点訳システムの限界を超えている複数候補の選択に関しては、複数候補の存在箇所を利用者に示し、人手で選択候補の中から適切な候補を選択し直すという、機械と人手による共同作業を効率よく作業できるシステムが現実的である。そこで、自動点訳システムで正確な読みが提示できなかった場合には、機械的に得られた複数候補を利用者に示し、選択肢を選び直すことで、適切な読みに校正できるユーザインターフェースを有するシステムが望まれる。

■ 研究内容
◆ 自動点訳エンジン EXTRA の解析と解析ドキュメントの作成
自動点訳エンジン EXTRA には、これまで外部インターフェースに関する資料しか存在しておらず、内部構造に関してはソースコード以外のドキュメントは存在しなかった。後述するように、複数候補の提示は、自動点訳エンジン EXTRA の基本的な構造に大きく手を入れる必要があるため、複数候補の提示の拡張を行う前段階の作業として、自動点訳エンジン EXTRA の解析と解析ドキュメントの作成を行った。解析作業は次の 2 段階で進めた。
・全 API のドキュメント化
点訳エンジンのソースコードにある全 API(C/C++言語の関数とメソッド)について解析を行い、ヘッダファイルのプロトタイプ宣言箇所に、関数の仕様に関するコメントを追記した。コメントは、Doxygen 型式で作成し、全ソースコードを Doxygen のドキュメント生成コンパイラにかけることで、最新の関数仕様書を生成できるようにした。
・内部の点訳処理や点訳辞書の解析
API ドキュメントを元に、内部の点訳処理や点訳辞書の解析を行った。解析結果は、「点訳エンジン EXDLL の構造について」と題する報告書(後述)としてまとめた。

◆ 自動点訳エンジンによる複数読み候補の提示
自動点訳エンジンによる複数読み候補の提示を行うために、自動点訳エンジン EXTRA に対して次の 2 つの点についての機能拡張を行った。

・複数読み候補の存在位置の提示
形態素解析エンジンの多くは、形態素解析時に複数の候補をラティス情報のような形で作成し、最終的に最適な候補をラティス情報の組み合わせの中から絞り込む構造を持つものが多い。しかし、自動点訳エンジン EXTRA はこのような基本設計にはなっておらず、常に単一の解を文法規則や辞書のエントリを元に作成し、次の解析に進む構造になっている。
この構造は、複数読み候補の扱いとは相反するものであり、容易には複数読み候補の存在位置を提示できない。そこで、単一の解を見つける部分について変更を行い、単一の解が定まった時点でさらにそれ以外の候補が存在するかについて探索を続ける拡張を行った。この機能拡張により、点訳時に原文テキストに対して複数候補が存在する位置に関するマーキング情報を提示できるようになった。
・複数候補の列挙
上記の拡張により得られた複数候補の存在位置について、さらに、どのような候補が存在しうるかについて列挙する機能の拡張を行った。列挙に際しては、自動点訳エンジンの持つ全ての辞書について探索を行うこととし、読み情報と共に品詞情報についても列挙できるようにした。



◆ 複数候補提示のユーザインターフェース

複数候補の提示と修正については、どのようなユーザインターフェースが実現できるかによって、大きく利用者の作業効率が左右される。ユーザインターフェースの設計に際しては、利用者が簡単に複数読み候補の存在を把握でき、読みの修正が簡単となるように配慮した。

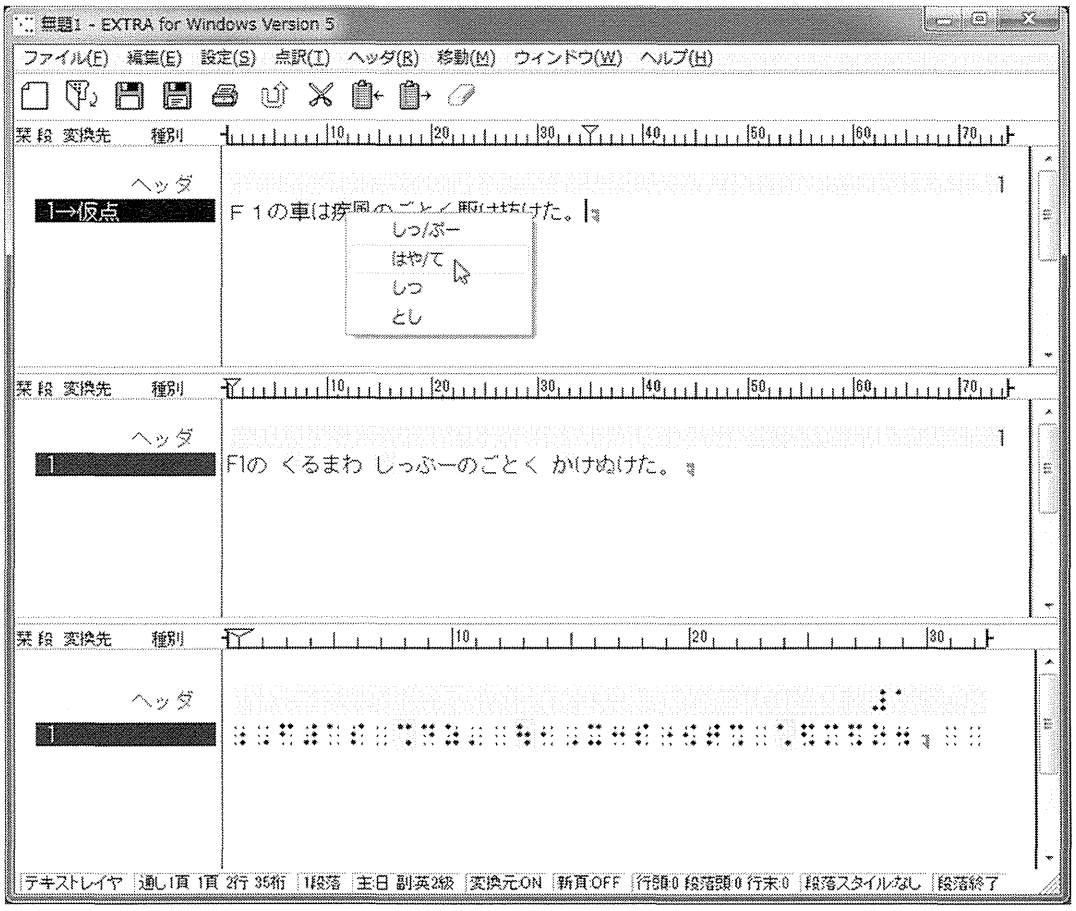
自動点訳システムのユーザインターフェースとしては、点訳エンジンから得られた複数候補の存在位置を、原文テキスト、点訳結果の仮名表現(仮名レイヤ)、点訳結果(点字レイヤ)にそれぞれピンク色の背景色で示すようにした。

【ユーザインターフェース画面：複数候補の存在位置表示】

ピンク色の文字の上で、マウスの右ボタンを押すと、その複数候補箇所で想定される複数候補の一覧が、ポップアップメニューとして表示される。利用者は、このポップアップメニューの中から望ましい候補を選択することで、簡単に適切な読みに修正できる。

【ユーザインターフェース画面：複数候補ポップアップメニュー】

この読みの修正は何度でも繰り返し行える。間違った場合には、再度マウスの右ボタンをクリックし、読みの候補を選択し直す。



■ 研究成果
本研究により、自動点訳時に読みが間違った場合に、複数の読み候補を選択し直すことで容易に人手による読みの修正作業が行えるようになった。この研究成果は、次期バージョンの自動点訳システムに組み込み、一般利用できるようにリリースする計画である。

■ 点訳エンジン EXDLL の構造について
□ 1 章 EXTRA 自動点訳システム

EXTRA 自動点訳システムは日本語および英語のテキストを点字情報に高い精度で点訳するシステムです。

◆ 1-1 目的と機能
次のような主言語、副言語の組み合わせによる点訳をサポートしています。
表示　主言語 副言語
J-E1　日本語 英語１級
E1-J　英語１級　　日本語
J-E2　日本語 英語２級
E2-J　英語２級　　日本語
J-JC　日本語 情報処理
JC-J　情報処理　　日本語
J-NC　日本語 北米情報処理
NC-J　北米情報処理　日本語

自動点訳の際に、元のテキストと、点訳後のテキストとの間の対応情報を取得することもできます。
アプリケーションのために自動的に全角を含む文字列を点訳する関数の他に読みに変換を行ったり、点訳のみをおこなうこともできます。

□ 2章 使用方法

◆ 2-1 基本的な使い方
一番単純な方法はZenToBrTabRHinを使って全角を直接変換します。また全角を半角にするZenToHan、半角を点字にするHanToBrなどがあります。

◆ 2-2 呼び出し方法
STDAPIインターフェースを使ったDLLとなっています。ポインターを扱える言語で使用することができます。

◆ 2-3 使用例
EXDLLは次のように、exdll.dllのDLLより外部公開されているインタフェースを取得し、関数をコールすることで使用します。

```
HANDLE hLib = ::LoadLibrary("exdll.dll"); /* exdllを取り出します */
FARPROC fpZTB = ::GetProcAddress(hLib, "ZenToHan"); /* ZenToHanのアドレスを取り出します */
(*fpZTB)(zenkaku_str, kana_buf, (short) sizeof kana_buf - 1, IndexTableZB); /* 点訳を行います */
::FreeLibrary(hLib); /* exdllの使用を終了します */
```

◆ 2-4 実行環境
プログラムはWin32APIとCライブラリーで構成されています。ほとんどのWindows OS(95/98/Me/2000/XP/Vista/7)で動作します。また、Windows CE用のexdll.dllも存在します。

exdll.dllの実行環境としては以下の設定が必要です。

・ 辞書が入っているデレクトリを参照するためのレジストリ情報
・ 辞書ファイル

□ 3章 点訳辞書

点訳用の辞書はプログラムソースデレクトリbrlにある拡張子.orgから作成されます。作成に使用されるプログラムはmakedic1.exe, makedic2.exeです。makedic1は英数をキーにした辞書を作成します。makedic2は漢字をキーにした辞書を作成します。
ここでは各辞書ファイル（拡張子は.dic)がどのソース(拡張子は.org)から作成され、各辞書ソースの読み方を説明します。
以下の説明の中で辞書番号とはexdllソース上でint dic_page[DIC_QTY]の要素番号として扱われる番号です。通常はsearch_page関数の第一引数として使われるケースがほとんどです。

◆ 3-1 kanjuku.dic
kanjuku.dicはプログラムで辞書番号0として扱われます。
このファイルは通常の熟語(主に名詞)から半角カタカナにする辞書です。辞書のソースはbrl.orgです。
辞書ソースファイルbrl.orgの形式は次の通りです。
漢字 読み タイプ(コンディションコード)
漢字の部分は漢字がそのままの形式で入っています。読みの部分は半角のカタカナで漢字の1文字に相当する部分が/で区切られて入っています。
タイプは次のタイプがあります。

タイプ	内容
!	人名
#	人姓
%	名詞
"	固有名詞
K	か行五段動詞語幹
S	さ行五段動詞語幹

T　　た行五段動詞語幹
N　　な行五段動詞語幹
H　　は行五段動詞語幹
M　　ま行五段動詞語幹
R　　ら行五段動詞語幹
W　　わ行五段動詞語幹
G　　が行五段動詞語幹
B　　ば行五段動詞語幹
I　　一段動詞語幹
C　　さ行サ変動詞語幹
Z　　ざ行サ変動詞語幹
A　　形容詞語幹
V　　形容動詞語幹
(　　区
&　　市
)　　町（チョー）
*　　町（マチ）
+　　村（ムラ）
.　　村（ソン）（brl8 では使用不可）
-　　その他
0　　動詞の命令他
1　　動詞の未然
2　　動詞の連用
3　　１段動詞の語幹
4　　動詞の仮定形
5　　動詞の終止、連帯
6　　さ変動詞の連用形
=　　助詞
_　　数字'

コンディションコードは必ず指定する必要がありません。この辞書レコードが接続出来る１つ前の品詞コードを表します。

タイプ 内容
x　　直前が固有名詞か人姓
y　　直前が全角数字以外
l　　前が漢字熟語とかたかな
^　　口語文書の場合'

◆ 3-2 tankan.dic

tankan.dic はプログラム上で辞書番号１として扱われます。
このファイルは単漢字の漢字１文字→半角カタカナの変換をします。辞書のソースは brl9.org です。
辞書ソースファイルの形式は kanjuku.dic と同一です。
コンディションコードはありません。

◆ 3-3 katakana.dic
katakana.dic はプログラム上で辞書番号２として扱われます。
全角カタカナで始まる辞書です。辞書のソースは brl10.org です。
ファイルの形式は kanjuku.dic と同一です。
コンディションコードはありません。

◆ 3-4 kankana.dic
kankana.dic はプログラム上では辞書番号３として扱われます。
送りがなが付いている熟語辞書です。辞書のソースは brl7.org がです。
ファイルの形式は kanjuku.dic と同一です。
コンディションコードは次の通りです。

コード 内容
h　　直前が(72:動詞の連用, 76:さ変動詞の連用形)
q　　直前が(73:１段動詞の語幹)
u　　直前が(76:さ変動詞の連用形)
d　　直前(20:熟語, 21:さ変動詞化できる熟語, 30:固有名詞, 40:人姓, 50:単漢字, 86:全角の特殊記号のうちとじかっこ), かたかな
l　　直前(21:さ変動詞化できる熟語)
p　　かたかな
y　　(82:全角の数字)以外
z　　地名

◆ 3-5 hiragana.dic
hiragana.dic はプログラム上では辞書番号４として扱われます。
ひらがなから始まる辞書です。brl8.org がソースです。
ファイルの形式は kanjuku.dic と同一です。
コンディションコードは以下の通りです。

コード 内容
x　　直前が(0:行頭, 9:半角および全角スペース)以外
^　　直前がひらがな以外
q　　直前が(73:１段動詞の語幹)
u　　直前が(76:さ変動詞の連用形)でない
h　　直前が(72:動詞の連用, 76:さ変動詞の連用形)
j　　直前が(70:動詞の命令他, 75:動詞の終止、連帯)
o　　直前が(74:動詞の仮定形)でない
e　　直前が(71:動詞の未然)
f　　直前が、漢字、かたかな、ひらがな登録語以外
g　　直前(30:固有名詞, 40:人姓, 41:人名)
d　　(81:ひらがな文字, 5:その他の半角, 0:行頭)以外
y　　(82:全角の数字, 1:半角数字)
z　　地名
@　　口語的文脈
n　　直前が(82:全角の数字, 1:半角数字)

◆ 3-6 zenhan.dic
zenhan.dic はプログラムで辞書番号５として扱われます。
全角から半角、はんかくから全角の変換辞書です。
このファイルは次の２つの辞書ソースから作成されます。
全角→半角(辞書ソース:zenhan.org), 半角→全角(辞書ソース:hanzen.org)
ファイルの形式は次の通りです。

"元文字列","先文字列"

◆ 3-7 engbrl.dic
engbrl.dic はプログラムで辞書番号６として扱われます。
全角の記号、全角のカナ、半角英語を点字表記にする辞書です。
辞書ソースは kigoubrl.org(全角の記号), kanabrl.org(全角カナ), engbrl.org(英語半角)です。

kigoubrl.org の形式は次の通りです。

"漢字文字１文字","点字表記"

kanabrl.org の形式は次の通りです。

"全角カタカナ","点字形式"

engbrl.org の形式は次の通りです。

"元の英語単語","点字形式",タイプ

タイプの意味は不明です。

◆ 3-8 user.dic
user.dic プログラム上では辞書番号７として扱われます。
ユーザが作成する辞書なのでソースはありません。

□ ４章 辞書のファイルについて

◆ 4-1 辞書ビルド方法
辞書は brl デレクトリーで makedic.bat を起動すると作成されます。

ビルドの各ステップの内容

◇ 4-1-1 逆変換用辞書の作成

mkedic1 を使用して逆変換用の辞書を作成します。
辞書元 変換後辞書
zenhan.org+hanzen.org　　engbrl.dic
kigoubrl.org+kanbrl.org+engbrl.org　engbrl.dic

◇ 4-1-2 漢字から点字辞書の作成

makedoc2 を使用して次の変換を行います。
辞書元 変換後辞書
brl.org kanjuku.dic
brl7.org　　kankana.dic
brl8.org　　hiragana.dic
brl9.org　　tankan.dic
brl10.org　　katakana.dic

◇ 4-1-3 makedic1 の使い方

makedic1 は逆変換用辞書変換ユーティリティです。コマンドラインから次の形式で起動します。
mkedic1 filename
この例では filename.org を入力ファイルとして filename.dic を作成します。

◇ 4-1-4 makedic2 の使い方

makedic2 は全角用辞書変換ユーティリティです。コマンドラインから次の形式で起動します。
mkedic2 filename
この例では filename.org を入力ファイルとして filename.dic を作成します。

□ 5章 点訳辞書の内部構造
辞書は次のような構成で保存されています。

1. バージョンヘッダー
2. ブロックの繰り返し

◆ 5-1 バージョンヘッダー

バージョンヘッダーはデータファイルの識別のためにある物でプログラムは何も必要としません。
終了がわかるように終了コードとして^Z(0x1a)が入っています。

◆ 5-2 ブロック

ブロックには次の形式でブロックサイズの固定長(DIC_BLOCK_SIZE 0x4000)で区切られたデータになっています。

1. ブロックヘッダー（後記）
2. エントリーブロック（後記）
3. データブロック（後記:繰り返し）

◇ 5-2-1 ブロックヘッダー

1. スタートエントリ（16bit 1word）
2. エンドエントリ （16bit 1word）
3. ブロックエンドのオフセット（16bit 1word）
4. エントリブロックのサイズ（16bit 1word）
5. エントリブロック（後記）
6. データブロック（後記）
7. 0データのパディング

◇ 5-2-2 エントリブロック

1. int_kanji(16bit 1word)は1文字目の漢字が入っています。
2. offset(16bit 1word)はこのエントリがデータブロックのどの位置に存在のかが入っています。

◇ 5-2-3 データブロック

データブロックは次の3つのデータが連続して入っていて0で区切られています。品詞タイプの最後の0がありますから0と0の連続で1レコードが終了します。

1. 漢字の続き(8bit 0で終了） 漢字の2文字以降が入っています。1文字目はエントリブロックのint_kanjiに記載されています。
2. 変換結果(8bit 0で終了) 辞書のソースと同じ物が入っています。
3. 品詞タイプ、コンディション(8bit 0で終了） 辞書のソースと同じ物が入っています。

◇ 5-2-4 0データのパディング

ブロックサイズに合わせるために、余った領域は0になっています。

□ 6章 EXTRA点訳エンジンの内部構造

本章では、EXTRA点訳エンジンの内部でどのような処理を行って点訳した点字情報を取得しているかについて解説します。

◆ 6-1 STEP1: 漢字から仮名に変換

ポインタの文字形式を見て文字形式に応じた辞書を引きます(ztok)。
辞書にあった場合(各種 brlxx)
辞書の内容と前のコンディションで次に呼ぶ brlxx を決めます。
辞書とマッチした場合、辞書の品詞と辞書が要求する条件(condition)が合致するか確認します。
この辞書が合致した場合は
この辞書の品種を次の辞書検索の時に使用するように保存します。
この辞書から取り出した内容には辞書に入っている区切り文字/と文節の区切り目の~がはいっています。
strget 関数より品種を hindex にセットし出力バッファに入れます。
set_index_zen により b_index から rindex 逆引きインデックスを作成します。

例:
漢字の変換を -> カンジノ~ヘンカンヲ~

◆ 6-2 STEP2: 仮名から点字に変換

外国語引用符、外字符、数符、英大文字符の処理を行います(xcap)。
仮名、記号であればテーブルで直接変換しします。
英単語の場合には短縮用の表現を辞書を引いて確定します。

◆ 6-3 STEP3: 文字位置対応表を作成します

出力文字列から文字区切りマークの'~'を削除しながら、文字位置対応表(index)と結果文字列から元文字列位置に対応する表(rindex)を作成します。

□ 7 章 点訳エンジンの内部品詞コード

点字エンジンでは次の内部コードを使って品詞を管理しています。

コード	内容
0	行頭
1	半角数字
2	半角英字
3	半角カナ
4	半角かな区切り
5	その他の半角
9	半角および全角スペース
20	熟語
21	さ変動詞化できる熟語
30	固有名詞
31	市、区、町、村
40	人姓
41	人名
50	単漢字
51	接頭辞
60	カタカナ単語
61	カタカナ文字
70	動詞の命令他
71	動詞の未然
72	動詞の連用
73	１段動詞の語幹
74	動詞の仮定形
75	動詞の終止、連帯

76　さ変動詞の連用形
77　促音便のっまでの動詞
79　形容詞、形容動詞の語幹
80　ひらがな単語
81　ひらがな文字
82　全角の数字
83　全角のアルファベット
84　全角の特殊記号
85　ひらがなの"っ"、"ご"および"お"
86　全角の特殊記号のうちとじかっこ
87　全角の特殊記号のうち開きかっこ
88　ひらがなのを
89　形容詞、形容動詞、接続詞、副詞、その他残余的登録
90　助詞
99　定義文字
100　ユーザー単語

□ 8章 辞書アクセスロジック
ここでは辞書ファイルを exdll がどのように使用しているかを解説します。

◆ 8-1 辞書ファイルのオープン
辞書ファイルは全て Windows の GlobalMemory に読み出します。(過去のコードが EMS を使って書かれていた関係でページの概念があります)(実装場所 exdinit.c:check_size_dic)

◆ 8-2 辞書番号について

辞書番号	辞書名	辞書ソース（説明）
0	kanjuku.dic	地名(brl.org):漢字の地名→半角カタカナ
1	tankan.dic	単漢字(brl9.org):漢字１文字→半角カタカナ
2	katakana.dic	カタカナ(brl10.org):全角カタカナで始まる→半角カタカナ
3	kankana.dic	一般辞書(brl7.org):漢字→半角カタカナ
4	hiragana.dic	ひらがな(brl8.org):全角ひらがなで始まる→半角カタカナ
5	zenhan.dic	(zenhan.org+hanzen.org)　全角→半角，半角→全角
6	engbrl.dic	全角記号(kigoubrl.org):全角記号→点字表記，全角カナ(kanabrl.org)全角カナ→点字表記，全角キゴウ→点字表記,全角カナ→点字表記，半角英語(engbrl.org):半角英語→点字表記
7	user.dic	ユーザ辞書(ソースはありません)

◆ 8-3 辞書番号のアクセス方法

#define DIC_QTY 8 このプログラムで扱える最大の辞書インデックスのサイズ
extern HGLOBAL mem_handle; メモリハンドル 全ての dic_page の容量 * 0x4000 のメモリバッファ
int dic[DIC_QTY] = { 0 }; 辞書のファイルハンドル
int dic_page[DIC_QTY] = { 0 }; 辞書のページポインター
int dic_max[DIC_QTY] = { 0 }; 全ての辞書のサイズ
int dic_crypto[DIC_QTY] = { 0 }; 辞書が暗号化されているかのフラグ 0:暗号化されていない, <>0:暗号化されている
DLL のエントリ関数で temp_d_ptr = (char *) GlobalLock(mem_handle);を行い辞書メモリをロックします。このメモリを変換ロジックに送る。このポインターは d_ptr として使われています。
search_in_page で実際の辞書ポインターを検索します。

□ 9章 変換ロジックの解説

◆ 9-1 漢字からカタカナ

全角をカナにする ztok 関数を C++の形式に変更しました。

◇ 9-1-1 ztok の初期化部分
まず、current_type(現在のタイプ), old1_type(１つ前のタイプ), old2_type(2 つ前のタイプ)という変数を初期化します。これは辞書から受け取ったタイプ（品詞情報)を保存しておくもので、辞書のの条件を判定するために使用します。

ZtoK::ztok()では出力文字列バッファ output_buffer に４文字のスペースに入力文字列を加えます。
つぎに、プログラムは直接出力文字列バッファーを編集するようなループが構造になっています。
old2_type = old1_type, old1_type = current_type;を行い現在のステートを保存します。

◇ 9-1-2 ztok のメインループ
ループではタイプが「ひらがな」,「漢字」,「漢字のカナ」,「漢字の特殊記号」,「半角数字」 ,「半角文字」を認識して各処理ロジックにふりわけられます。

◇ 9-1-3 ztok の最終処理
最後に処理を行ったデータを出力文字列バッファーに出力します。

◇ 9-1-4 ひらがなの変換:zenHiraConvert

プログラムは次の順でチェックを行いそれぞれの関数で処理されます。
1. ユーザ辞書の検索 (userdic(8, ...)
2. ひらがなの検索 (brlh)
見つかった場合は strget で変換後の'/'を元に index_ptr, rindex_ptr を設定します。
この関数は必ず変更を行います。

◇ 9-1-5 通常辞書を使ったひらがな変換:brlh

type2 = 0 に設定します。
前の品詞(*brl)が(20:熟語, 21:さ変動詞化できる熟語, 30:固有名詞, 31:市、区、町、村, 40:人姓, 41:人名, 50:単漢字, 60:カタカナ単語, 61:カタカナ文字, 72:動詞の連用, 73:１段動詞の語幹, 75:動詞の終止、連帯,80:ひらがな単語, 82:全角の数字, 83:全角のアルファベット, 89:形容詞、形容動詞、接続詞、副詞、その他残余的登録, 1:半角数字, 2:半角英字, 3:半角カナ, 90:助詞)で、入力文字が[０-９Ａ-Ｚａ-ｚあ-ん]ならば、現状のタイプを助詞(type:90)にして[ひらがなから半角カタカナにします(brl8(0...) hiragana.dic 辞書番号 4, brl8.org] もし見つかれば助詞の活用形の処理を行います。
その条件でなければ 0 を返します。
どのケースも、辞書からの検索が成功した時点で、strget,set_index_zen で変換後文字列の位置情報(index)を調整します。

◇ 9-1-6 漢字熟語:kanjiJukugoConver

ユーザ辞書の検索 (userdic(8, ...)を行います。
１文字の漢字である事がわかった場合は[単漢字辞書の検索 brl9(0, ...) 辞書番号 1: tankan.dic (brl9.org)]を行います。
その条件か単漢字署にマッチしなかった場合は[一般辞書の変換 brl7(&current_type..) : kankana.dic (brl7.org)]を行います。
その条件でマッチしなかった場合は、[漢数字処理 ktoi]を行います。
その条件でマッチしなかった場合で、接頭語の条件にかかる場合は[単漢字辞書の検索 brl9)]を行います。
どのケースも、辞書からの検索が成功した時点で、strget,set_index_zen で変換後文字列の位置情報(index)を調整します。

◆ 9-2 カタカナ英数から点訳

◇ 9-2-1 HanToBrR 解説

まず辞書バッファの変数、入力文字列のサイズなどをチェックし問題が無ければ、次の動作を行います。
1. IndexTable[0]に入力文字列のサイズ(strlen)を入れる
2. 辞書ポインターを作成します。
3. set_l_ex_braille_r を次の形式で呼び出します。
set_l_ex_braille_r((char) ForeignModeM, (char) ForeignModeS, (char) LangM, (char) LangS, OutStr, sOutLimit, 3, temp_d_ptr, lpsIndexTable, lpsRIndexTable, 0);
4. 辞書ポインターをハンドルに戻します。

1. メイン言語(lang_m), サブ言語(lang_s), メイン言語のモード(f_mode_m), サブ言語のモード(f_mode_s)の組み合わせから bcode を作成します。
2. s_func が 3 の時だけ ex_braille_r(bcode, out_ptr, limit, d_ptr, index_ptr, rindex_ptr, raw_mode);を呼び出します。
ex_braille_r(BCode bcode, uchar * line, short limit, char *d_ptr, short *index_ptr, short *rindex_ptr, char raw_mode)
1.chk_kanji(line);
2a.bcode が(B_US1_JP0, B_US2_JP0, B_JP0_US1, B_JP0_US2, B_JPU1_US1,B_JPU1_US2)ならば btoe_r を呼び出します。
2b.bcode が(B_JP0_JP_JOUHOU, B_JPJOUHOU_JP0)ならば ktoj_r を呼び出します。
2c.bcode が(B_JP0_US_JOUHOU, B_US_JOUHOU_JP0, B_JP_JOUHOU_JP_MUHENKAN)ならば kton_r を呼び出す。

カナ文字列から点字を行います。ここでは(日本語情報処理) BCode 3,4 を例にして説明します。

void kton_r(uchar *instr, short limit, BCode bcode, char *d_ptr,short *index_ptr,short *rindex_ptr);
引数:instr 入力文字列
引数:limit 出力バッファの大きさバイト数単位
引数:index_ptr 出力：点字から入力への場所を入れた配列。0 番目に文字列のバイト長、1 番以降は入力文字列 0 バイト目の出力文字列のオフセット位置に変換されたことを表します。このテーブルは出力で入力文字列のバイト数＋１が必要です。不要な時は NULL を指定指定してください。
引数:index_ptr 出力：点字から入力への場所を入れた配列。0 番目に文字列のバイト長、1 番以降は入力文字列 0 バイト目の出力文字列のオフセット位置に変換されたことを表します。このテーブルは出力で入力文字列のバイト数＋１が必要です。不要な時は NULL を指定指定してください。

内部動作
uchar x[MAX_LEN+1]という変数を作成し、元の instr をコピーします。
uchar line[MAX_LEN+1]という変数を作成し、文字長 0 として初期化します。この出力バッファーに入れる文字列となります。
指示が B_JP_JOUHOU_JP_MUHENKAN の場合は、次の動作を行います。
  x[i]に対して次の動作を行います。
    半角のカナならば ktob で変換してインデックスを修正します。
    漢字ならば stob で変換してインデックスを修正します。
  他のケースに関しては省略

◇ 9-3-2 stob 解説

全角記号、全角カナ、英語半角を点字表記にします。
使用辞書　6: engbrl.dic (kigoubrl.org+kanabrl.org+engbrl.org)　全角キゴウ→点字表記, 全角カナ→点字表記, 英語半角→点字表記

int stob(uchar *buffer, uchar *x, int ptr, char *d_ptr);
引数:buffer 出力バッファ（半角カタカナ出力）
引数:x 入力　（ptr を加算した所から探す）

引数:ptr 入力のオフセット
引数:d_ptr 辞書バッファー
return 変換を行った場合は 0 以外を返します。

内部動作
search_in_page を使って 6 番の辞書から探します。もし見つからなければ return 0 をおこないます。
辞書バッファーから文字列を検索します。もし見つかったら buffer に出力します。

□ 10 章 API 一覧

ここでは、exdll より外部公開されており、アプリケーション側から利用できる関数仕様について説明します。
本章の API は、exd.h というヘッダファイルに関数宣言が定義されています。

◆ 10-1 kydll 関数

EXDLL のほとんどの機能を packet_ptr の情報により呼び出します。

EXDLL_EXPORT kydll(kydll_packet FAR *packet_ptr);
引数:packet_ptr 機能指定の構造体変数
返り値:必ず 0 を返します。s_func にエラーコードが設定されます。

◆ 10-2 ZenToBrTabRHin 関数

全角を点字にします。

EXDLL_EXPORT ZenToBrTabRHin(char* InStr, char* OutStr, short LangM, short LangS, short ForeignModeM, short ForeignModeS, short sOutLimit, short* lpsIndexTable, short* lpsRIndexTable, short* lpsHinTable, short sTabs);

引数:InStr 入力文字列
引数:OutStr 出力バッファ
引数:sOutLimit 出力バッファへの文字列の長さの制限を指定します。
引数:LnagM 主言語のコード 1:英語 0x41: 日本語
引数:LangS 補助言語のコード 1:英語 0x41: 日本語
引数:ForeignModeM 後述の主要言語種別(LangM)に対する点訳の水準を指定
引数:ForeignModeS 後述の補助言語種別(LangS)に対する点訳の水準を指定
引数:lpsIndexTable 出力：0 番目に文字列のバイト長、1 番以降は入力文字列 0 バイト目の出力文字列のオフセット位置に変換されたことを表します。このテーブルは出力で入力文字列のバイト数＋１が必要です。不要な時は NULL を指定してください
引数:lpsRIndexTable 出力：点字から入力への場所を入れた配列。0 番目に文字列のバイト長、1 番以降は入力文字列 0 バイト目の出力文字列のオフセット位置に変換されたことを表します。このテーブルは出力で入力文字列のバイト数＋１が必要です。不要な時は NULL を指定指定してください
引数:lpsRIndexTable 出力：点字から入力への場所を入れた配列。0 番目に文字列のバイト長、1 番以降は入力文字列 0 バイト目の出力文字列のオフセット位置に変換されたことを表します。このテーブルは出力で入力文字列のバイト数＋１が必要です。不要な時は NULL を指定指定してください
引数:lpsHinTable 出力: 品種テーブル
引数:sTabs 入力全角文字列内のタブ・コードをスペース何個に展開するか指定
返り値:成功の時に TRUE、失敗の時に FALSE を返します。

◆ 10-3 ZenToBrTabR 関数

全角を点字にします。

EXDLL_EXPORT ZenToBrTabR(char* InStr, char* OutStr, short LangM, short LangS, short ForeignModeM, short ForeignModeS, short sOutLimit, short* lpsIndexTable, short* lpsRIndexTable, short sTabs);
引数:InStr 入力文字列
引数:OutStr 出力バッファ
引数:LnagM 主言語のコード 1:英語　0x41: 日本語
引数:LangS 補助言語のコード　1:英語　0x41: 日本語
引数:ForeignModeM 後述の主要言語種別(LangM)に対する点訳の水準を指定
引数:ForeignModeS 後述の補助言語種別(LangS)に対する点訳の水準を指定
引数:sOutLimit 出力バッファへの文字列の長さの制限を指定
引数:lpsIndexTable 出力：0番目に文字列のバイト長、1番以降は入力文字列0バイト目の出力文字列のオフセット位置に変換されたことを表します。このテーブルは出力で入力文字列のバイト数＋1が必要です。不要な時はNULLを指定してください
引数:sTabs 入力全角文字列内のタブ・コードをスペース何個に展開するか指定
返り値:成功の時にTRUE、失敗の時にFALSEを返します。

◆ 10-4 ZenToBrTabHin 関数

全角を点字にします。

EXDLL_EXPORT ZenToBrTabHin(char* InStr, char* OutStr, short LangM, short LangS, short ForeignModeM, short ForeignModeS, short sOutLimit, short* lpsIndexTable, short* lpsHinTable, short sTabs);

引数:InStr 入力文字列
引数:OutStr 出力バッファ
引数:sOutLimit 出力バッファへの文字列の長さの制限を指定します。
引数:LnagM 主言語のコード 1:英語　0x41: 日本語
引数:LangS 補助言語のコード　1:英語　0x41: 日本語
引数:ForeignModeM 後述の主要言語種別(LangM)に対する点訳の水準を指定
引数:ForeignModeS 後述の補助言語種別(LangS)に対する点訳の水準を指定
引数:lpsIndexTable 出力：0番目に文字列のバイト長、1番以降は入力文字列0バイト目の出力文字列のオフセット位置に変換されたことを表します。このテーブルは出力で入力文字列のバイト数＋1が必要です。不要な時はNULLを指定してください
引数:lpsRIndexTable 出力：点字から入力への場所を入れた配列。0番目に文字列のバイト長、1番以降は入力文字列0バイト目の出力文字列のオフセット位置に変換されたことを表します。このテーブルは出力で入力文字列のバイト数＋1が必要です。不要な時はNULLを指定指定してください
引数:lpsRIndexTable 出力：点字から入力への場所を入れた配列。0番目に文字列のバイト長、1番以降は入力文字列0バイト目の出力文字列のオフセット位置に変換されたことを表します。このテーブルは出力で入力文字列のバイト数＋1が必要です。不要な時はNULLを指定指定してください
引数:lpsHinTable 出力: 品種テーブル
引数:sTabs 入力全角文字列内のタブ・コードをスペース何個に展開するか指定
返り値:成功の時にTRUE、失敗の時にFALSEを返します。

◆ 10-5 ZenToBrTab 関数

全角を点字にします。

EXDLL_EXPORT ZenToBrTab(char* InStr, char* OutStr, short LangM, short LangS, short ForeignModeM, short ForeignModeS, short sOutLimit, short* lpsIndexTable, short sTabs);

引数:InStr 入力文字列
引数:OutStr 出力バッファ
引数:sOutLimit 出力バッファへの文字列の長さの制限を指定します。

引数:LnagM 主言語のコード 1:英語　0x41: 日本語
引数:LangS 補助言語のコード　1:英語　0x41: 日本語
引数:ForeignModeM 後述の主要言語種別(LangM)に対する点訳の水準を指定
引数:ForeignModeS 後述の補助言語種別(LangS)に対する点訳の水準を指定
引数:lpsIndexTable 出力：0番目に文字列のバイト長、1番以降は入力文字列0バイト目の出力文字列のオフセット位置に変換されたことを表します。このテーブルは出力で入力文字列のバイト数＋1が必要です。不要な時はNULLを指定してください
引数:lpsRIndexTable 出力：点字から入力への場所を入れた配列。0番目に文字列のバイト長、1番以降は入力文字列0バイト目の出力文字列のオフセット位置に変換されたことを表します。このテーブルは出力で入力文字列のバイト数＋1が必要です。不要な時はNULLを指定指定してください
引数:lpsRIndexTable 出力：点字から入力への場所を入れた配列。0番目に文字列のバイト長、1番以降は入力文字列0バイト目の出力文字列のオフセット位置に変換されたことを表します。このテーブルは出力で入力文字列のバイト数＋1が必要です。不要な時はNULLを指定指定してください
引数:lpsHinTable 出力: 品種テーブル
引数:sTabs 入力全角文字列内のタブ・コードをスペース何個に展開するか指定
返り値:成功の時にTRUE、失敗の時にFALSEを返します。

◆ 10-6 ZenToBrRHin 関数

全角を点字にします。

EXDLL_EXPORT ZenToBrRHin(char* InStr, char* OutStr, short LangM, short LangS, short ForeignModeM, short ForeignModeS, short sOutLimit, short* lpsIndexTable, short* lpsRIndexTable, short* lpsHinTable);

引数:InStr 入力文字列
引数:OutStr 出力バッファ
引数:sOutLimit 出力バッファへの文字列の長さの制限を指定。
引数:LnagM 主言語のコード 1:英語　0x41: 日本語
引数:LangS 補助言語のコード　1:英語　0x41: 日本語
引数:ForeignModeM 後述の主要言語種別(LangM)に対する点訳の水準を指定
引数:ForeignModeS 後述の補助言語種別(LangS)に対する点訳の水準を指定
引数:lpsIndexTable 出力：0番目に文字列のバイト長、1番以降は入力文字列0バイト目の出力文字列のオフセット位置に変換されたことを表します。このテーブルは出力で入力文字列のバイト数＋1が必要です。不要な時はNULLを指定してください
引数:lpsRIndexTable 出力：点字から入力への場所を入れた配列。0番目に文字列のバイト長、1番以降は入力文字列0バイト目の出力文字列のオフセット位置に変換されたことを表します。このテーブルは出力で入力文字列のバイト数＋1が必要です。不要な時はNULLを指定してください
引数:lpsRIndexTable 出力：点字から入力への場所を入れた配列。0番目に文字列のバイト長、1番以降は入力文字列0バイト目の出力文字列のオフセット位置に変換されたことを表します。このテーブルは出力で入力文字列のバイト数＋1が必要です。不要な時はNULLを指定してください
引数:lpsHinTable 出力: 品種テーブル
返り値:成功の時にTRUE、失敗の時にFALSEを返します。

◆ 10-7 ZenToBr 関数

全角を点字にします。

EXDLL_EXPORT ZenToBr(char* InStr, char* OutStr, short LangM, short LangS, short ForeignModeM, short ForeignModeS, short sOutLimit, short* lpsIndexTable);

引数:InStr 入力文字列
引数:OutStr 出力バッファ
引数:sOutLimit 出力バッファへの文字列の長さの制限を指定。