縦書きエディットとユニコード

プログラム開発言語デルファイ用の「ユニコード対応の縦書き
エディット」コンポーネントの開発過程で経験した諸問題を
とりあげて解説

Borland Delphi Ver.5 対応

発　行
YahagiOffice
2006

<はじめに>

ＰＣで開発を始めたのは20年以上前で正確なところは忘れてしまった。ＮＥＣ製のＰＣ－８０００の時代だった。Ｎ８８ＢＡＳＩＣがなつかしい。まもなくＰＣ－８８００に、そしてすぐにＰＣ－９８００に買い換えた。ＢＡＳＩＣにあきたらずアセンブラに手をつけたのだが、開発ツールがなく、当時の『Ｉ／Ｏ』だったか『マイコン』誌に掲載されたアセンブラ･ツールの１６進データを手打ちして用いた。蛇足だがサンプルプログラムなどではなく、アセンブラ･ツールそのものを打ち込んだのだ。

つぎに手をつけたのはＣである。ＭＳ－Ｃとボーランドを使った。ボーランドのほうがなじみやすく長く使った。こうしているうちに、ＯＳがＤＯＳを経てウインドウズにかわっていった。ウインドウズの開発ツールとして、最初に使ったのはＶｉｓｕａｌ－Ｃだったが、Ｃ＋＋を勉強しているうちに、急に突貫工事のようなあるツールの作成依頼があって、納期の問題からＶｉｓｕａｌＢａｓｉｃをつかった。これは今でも多くのユーザーがささえているすばらしいツールなのだが、製品が完成してもなぜか好きになれなかった。やたらにまわりくどい（しかたないのだろうが）し、かゆいところには手が届かないという不満を持っただけだった。

こうしているうちにＢｏｒｌａｎｄ社のＤｅｌｐｈｉを知った。「Ｐａｓｃａｌ言語は厳格できびしい」とかのうわさでルーズな気分が抜けない自分はちょっと敬遠していた。だがＣでなじんでいたし、「めっぽうコンパイルが速い」というＢｏｒｌａｎｄの製品はやってみる気にさせた。

決定的にＤｅｌｐｈｉにのめりこんだのは、ウインドウズでも実行ファイルをＥＸＥ１本でできる、という点だった。ＶｉｓｕａｌＢａｓｉｃではうんざりしていたし、Ｃは何といっても得る結果は１００点なのだが開発経緯がつらい（単なる面倒がりや）。Ｄｅｌｐｈｉは自分が主に作る「その場のツール」開発にぴったり、これはいいという感触が深まるばかりだった。

Ｄｅｌｐｈｉも人の子、とうぜん長所ばかりではなく、短所もあるのだが、使う目的から決定的な欠陥はない。十分だ。これが、私の経験からきた、Ｄｅｌｐｈｉへの思いだ。

Ｄｅｌｐｈｉのいいところは、容易にコンポーネントが作れることかもしれない。最初から用意されているコンポーネントも膨大な数があって、標準的な開発には特別こまることはない。しかし開発者にも好みがあって、ちょっと自分の好みを反映させたいなと思ったような場合に､元のコンポーネントを継承して好きなように化粧をほどこせる。慣れれば、最初から目的のコンポーネントを作成できる。

他の開発ツールも同じかもしれないが、インターネットの世界には、さまざまなコンポーネントが登録されている。Ｄｅｌｐｈｉの先輩達の作品をいろいろと試してみるのはたいへん勉強になる。

また、私がＤｅｌｐｈｉに関して多くの情報を得るのはＤｅｌｐｈｉのメーリングリストだ。初心者から達人までがいろいろな情報を交換している。これをときどきのぞくだけで、深い刺激をうける。

一方では残念なこともある。書店でＤｅｌｐｈｉ関係の書籍をみてみると、最近はそのコーナーが閉められた店もある。Ｄｅｌｐｈｉのバージョンは７まで追従してきたのだが、５の安定性、５周辺の情報量、５のコンポーネントの多さなどから、私の開発ニーズでは特にこれ以上先にいく理由がない。マイクロソフトのnetFrameworkなどの流れを考えると、これらを抑えておくことは必要であっても、実践的には実績と安定性が最優先となることが多い。つまり、Ｄｅｌｐｈｉ５からまだしばらく離れなれられない。

さて、縦書きエディットコンポーネントだが、以前から使用していた宛名印字のツール用に必要で作ったものである。

住所録は汎用的なエクセルで管理しているのがいちばん便利で、もっとも多いのではないだろうか。

エクセルはワードと同様に、ウインドウズの当初からユニコードに対応した。それならせっかく作る縦書きエディットコンポーネントもユニコード対応にしようと決めた。

改めて言うことではないかもしれないが、「縦書きエディット」というのは、Ｄｅｌｐ

ｈｉ標準添付のEditコントロールの縦書き版だ。こんなもの、当然インターネットで探せばごろごろしているに違いないと思って探してみたのだが、意外に見つからない。どうも、ニーズがないようだ。だが、自分は自分で作って使っている宛名印字ソフトの改良版で使いたい。探してもないなら、その作成を自分でやってみるか、ということで作ったものである。

縦書きは日本語では当然使われるものだが、それを実現するというのは、結構やっかいなものだ、ということをやってみてわかった。同じように、ユニコード対応ということも、予想に反してめんどうであることを知った。Ｄｅｌｐｈｉはもともと Widestring=Unicode対応ではなかったのか、と首をかしげたくなるような現実に直面する連続だった。

驚いたことのひとつにフォントの実状もある。

このあたりの問題は、Ｄｅｌｐｈｉ、コンポーネント、縦書き、ユニコードに関係のない方にはまったく必要の無い話だ。逆に、そのようなことに関心があり問題に直面しているひとには、少しは役立つのではないかと思っている。この書籍は、そのような目的で、作成中のメモを整理したものである。

縦書きエディットのソースの解説を、関係する知識と実験のサンプルプログラムの紹介もしている。

井の中の蛙という例えがある。ここに記したものはほとんど自分でちょこちょこと動かして得た知識だ。プログラミングについては師匠をもたない。ゆえに、組織で学んでいる方々から見るとそんなものと思うものもあるに違いない。そのあたりも含めた関連情報を教えていただけたらありがたいと思っている。

２００６年６月　YahagiOffice（矢萩光也）

目 次

はじめに …… i

第１部　文字あれこれ …… 1

1.1. 宛名印刷のいろいろ …… 1
年賀状 …… 1
大量の挨拶状 …… 1
ラベル印刷 …… 1
個人情報保護法 …… 2
気になることのまとめ …… 2
1.2. 住所録はどう管理する …… 2
「筆まめ」などの印刷ソフト …… 2
フリーソフトの住所録 …… 2
エクセル …… 2
ワードなど …… 3
気になることのまとめ …… 3
1.3. 使用できる文字コド範囲の問題 …… 3
ＪＩＳとユニコード …… 3
フォントと書体 …… 8
IMEパッドとIME …… 9
横書き明朝フォントで縦書きが可能か …… 9
外字の扱い …… 11
経済通産省のＪＩＳコード …… 12
WidowsVistaがJIS2004に対応 …… 12
気になることのまとめ …… 13
1.4. 文字の配置と郵便番号 …… 13
郵便番号枠 …… 13
カスタマバコード …… 13
事業所用の郵便番号 …… 14
書体とサイズ …… 14
宛名印刷に必要な項目と文字の長さ …… 14
縦置きと横置き、縦書きと横書き …… 15
気になることのまとめ …… 15

第２部　文字処理プログラミング …… 16

2.1. デルファイとコンポーネント …… 16
デルファイの強み …… 16
コンポーネント作成の必要 …… 16
2.1. デルファイで文字処理 …… 16
LabelとEdit …… 16
MemoとRichEdit …… 17
HEditorというメモコンポーネントのすぐれもの …… 17
ImageやPainBoxに描画する …… 17
印刷はPrinter.Canvasに描画する …… 18
気になることのまとめ …… 18
特徴と機能 …… 18
インストールと使い方 …… 19
プロパティ …… 19
イベント …… 20

入力中の編集 …… 20
編集中のショートカットキー …… 20
コントロールの移動キ …… 20
TateEditの基本設計 …… 20
「縦書きエディット」作成過程での課題 …… 20
気になることのまとめ …… 21
2.3.縦書きエディットコンポーネント作成過程でのとまどい …… 21
CaptionとText …… 21
ユニコードの入力インターフェイス …… 22
TateEditを何から継承してつくるか …… 22
キーイベントでの処理の問題 …… 22
ＩＭＥからユニコードで漢字を得る …… 23
ＡＮＫ（半角文字）を得るにはどうするか …… 23
苦肉のMessageToAscii …… 24
ＡＮＫと漢字の合成で編集文字列にする …… 28
気になることのまとめ …… 29
2.4.縦書きエディットコンポーネントでの文字描画処理と扁平処理 …… 29
文字描画処理をするにあたって …… 29
縦書きをする方法 …… 29
LogFontを使っての注意点など …… 30
文字のHeightの指定 …… 31
ユニコードの印字のための準備 …… 31
文字の表示オプション（プロパティのAlignment） …… 33
文字の扁平 …… 34
扁平横書きのしかた …… 34
扁平縦書きのしかた …… 37
JustifyExのOffsetExはどう使うか …… 39
文字列描画エリアの計算 …… 41
太字、斜体、下線、着色 …… 42
気になることのまとめ …… 42
ＩＭＥの複雑なふるまい …… 42
入力中の編集処理 …… 42
ＩＭＥ経由の漢字とＡＮＫとコントロールキの分離 …… 43
ユニコード文字のカット＆ペースト …… 43
ドラッグ＆ドロップ …… 45
気になることのまとめ …… 44
2.5.PmUnicodeViewerツール …… 44
特徴と機能 …… 44
画面の情報 …… 44
インストールされているフォントリスト …… 45
CharSetを得る …… 48
JISShiftJISUnicodeEUCUTF8句点のコード変換 …… 50
気になることのまとめ …… 52
2.6.PmHagakiAtenaハガキ宛名印刷ツール …… 52
特徴と機能 …… 52
エクセルファイルから読み込む …… 53
TntUnicodeWareコンポーネントの利用 …… 55
プレビューの処理 …… 56
Controlの位置とサイズの動的変更 …… 56
TateEdit機能と特殊プロパティの効用 …… 57
ユニコード文字の表示 …… 58
カスタマーバコードの作成 …… 58
気になることのまとめ …… 59

付録　Delphi Tips ……… 60

XpStyleに対応する……… 60
文字コードポイントの変換……… 61
使用説明書を表示する……… 65
タイトルバーのアイコンにメニューを追加する……… 67
全角スペースを含んだTrim……… 68
ユニコード対応のposWとPosWP……… 69
ＣＳＶ１行の操作parse……… 70
Booleanとstringの交換……… 76
文字列リストからダブりを除く……… 77

第1部　文字あれこれ

１．１．宛名印刷のいろいろ

年賀状

一般的に話題として、宛名の印刷についていちばん多いのは、やはり年賀状だ。毎年11月の後半になると、ＰＣショップや書店には、年賀状ソフトとその関係の書籍が山積みされる。インターネット関係でもそのＰＲがところ狭しとひしめいている。
私もかつては「筆まめ」を使っていたことがあるし、パソコンを購入するとたいていは何かの年賀状ソフトがバンドルされているので、多くの方々は気楽に使っているのではないだろうか。
新年にいただく年賀状の宛名でいちばん多いのがパソコンで印刷したものだ。手書きと業者に依頼して印刷してもらったものは合わせても私の場合は２割程度だった。
ということは、そうとう多くの方々が宛名印刷ソフトを利用しているということである。

大量の挨拶状

私は仕事がらさまざまな宛名印刷に接することが多い。賀状だけでなく、季節のあいさつ、移転のあいさつ、社名変更のあいさつ、イベントの案内、選挙やそのＤＭなどだ。
もちろんデータの整理や入力から行う。こちらで印刷までする場合は、出力のことを考えてデータを扱う。データ持ち込みも多い。この場合はまずデータの整理をしなければならない。
・郵便番号があるかないか
・住所が最新になっているかどうか
・都道府県をつけるかどうか
・カスタマーバーコードを印刷するかどうか
・読めない文字の扱い
・外国住所の扱い
・印刷の体裁をどうするか
こうした点で方針を確認していっせいに処理に入る。近年の市町村合併ブームで住所や郵便番号の扱いは結構めんどうである。郵便番号関係では、３桁から５桁、今は７桁だが持ち込まれるデータには今だ３桁がある。大規模事業所や公共機関は専用の郵便番号を持っている。これも、一般郵便番号との混乱がおこりやすい。

ラベル印刷

宛名の印刷といっても、プロの現場ではいろいろある。
各種封筒への印刷というのがあるが、これはさまざまな大きさの封筒に宛名を直接印刷するものだ。専用の印刷機は別にして、いまはパソコンでもプリンターが対応しているのが多いので身近に使っているかたも多いと思う。封筒のサイズを特定して印刷するのは比較的容易だが、さまざなな大きさの封筒に印刷するとなると宛名領域の配置には注意しながらおこなわなければならない。封筒は厚みがあるのでプリンターを痛めることがあるので、これも考慮しなければなららない。
ハガキ印刷も年賀状などのようにハガキに１枚ごと印刷するのが基本ではあるが、大量の場合はＡ４の用紙に４枚、Ａ３の用紙に８枚あらかじめ印刷しておいたものに宛名を印刷することになる。印刷した後で、断裁するのだ。
宛名ラベルに印刷するということも同じように多い。
これは、剥がして貼る作業がともなうが、相手はハガキでも、印刷しにくい封筒でも何にでも使える。

シートは汎用で発売されていて、Ａ４サイズで２列×６枚（１２枚)、３列×７枚（２１枚)、４列×５枚（２０枚）などがある。

個人情報保護法

宛名などの住所録を個人で使用している分にはすべて自己責任だが、会社で大量に扱うときは注意しなければならないことがある。それは、２００５年４月から施行された「個人情報の保護に関する法律」。

この法律ができて以来なぜか公共機関や大企業から大量の個人データが流出していることが報じられているのは皮肉なことだ。ウィニィを通じてとか、パソコンの紛失とか、担当者が規則で禁止されているにもかかわらず、個人ＰＣへのコピーをして自宅に持ち帰ってウィルスを通じて流出するとかだ。

もちろん、本来の目的外で使用したり、リスト業者に不法に売るなどは明確な犯罪だ。

社会は個人のあつまりなので、人間が介在する以上どうしても穴が生じてしまう。それを法律での規則と漏洩しないためのさまざまな対策を前向きに実行することで少しでも防ごうというものだ。

公共機関や大手から個人情報をともなう仕事が流れるときには、適切な対策を講じているという認証を得た（プライバシーマーク）事業体でないといけないようになったのである。

気になることのまとめ

宛名などの住所録を扱う場合は、何の目的でどのようなツールを使って行うのか。データは過不足なく用意したのか。手元にデータがあるときには管理がいきとどいているのか。

１.２. 住所録はどう管理する

年賀状ソフトの住所録

実際に年賀状ソフトでの宛名印刷をする場合のことを考えてみよう。一般的にこの種のソフトについて触れて感じたことを述べたいだけだ。これらは発売当初からすばらしい人気だった。パソコンが普及するときにワードなどとならんで身近に使用することができるツールだったからであろう。同種に乗り物で行き先までの時間、経費を見ることができる「駅スパート」がある。

横道にそれたが、年賀状ソフトがパソコンを使う一つの目的になり、日本のパソコン普及に大きく貢献した。これで日本語の文字入力、データ管理というもの、印刷すること、実用に役立つことを実感した人が実際に多い。

こうしたソフトは、当然独自性がある。これが個性でもあり、売り込みたいところでもある。おのずとこれはデータの管理方式に反映し、独自のデータフォーマットを採用しているのがほとんどだ。汎用のCSVファイルを読み込んだりするのは当然としても、CSVへの書き出しができるのは以前は少なかった。

フリーソフトの住所録

フリーソフトのVertorなどをみればわかるように、住所録ソフトは数多くある。フリーソフトは作者のセンスと使用者のセンスが合致したときに使い心地がいい。触ってみると、同じ開発をするものとしていつも感心させられる。いろいろなアイデアをみることができるのは楽しみでもある。

エクセル

エクセルとワードはパソコンを購入するとたいていバンドルされているので、パソコンを使

っている方なら知っているはずだ。パソコンはソフトがなければ本当にただのハコである。ワープロと表計算は個人でも使うし職場では欠かせないツールであろう。インターネットでのブラウジング、Eメールの交換、お絵かきとデジカメ撮影の写真管理はパソコンのユーザーならまずトライしてみるはずだ。
　年賀状などで使うデータを年賀状ソフトではなく、エクセルをその目的で使っているというひとはまず職場でデータ管理に触れている方が多いようだ。エクセルデータはパソコンの世界では「汎用」といっても過言ではないほどだから、応用が期待できる。エクセルそのもので年賀状を印刷するというのまず聞かない。ここで管理しているデータをCSVデータにして、ワードで差込印刷をするのだ。

ワードなど

　エクセルで作ったデータからワードで差込印刷する方法は、多く利用されている。ワードで住所や宛先などの項目の印刷用のテンプレートを用意して、各項目にデータを関連付ける。
　ワードでは、こうした差込印刷の機能を使わずに、A4のラベルへの印刷をするのなら、そのイメージどおりA4の用紙に文字を配置して、ｄｏｃファイル自身を住所管理データそのものとして使っているというケースも多々ある。まあ、このケースは「データを管理する」というにはちょっと遠いのだが、現実にはよくみる。エクセルで数字が管理できるということは知っていても具体的な使い方は「そうとう難しいに違いない。自分ではきっとできない」と思って、合計などの金額を電卓たたいてその数字を打ち込んでいたというのも見ているが、けっして笑えない。

気になることのまとめ

　せっかくコンピュータという「計算・管理」の得意な装置を使っているのだから、住所録や宛名リストはデータとして管理したい。その場合はやはり汎用のエクセルが適している。しかも、あれでもこれでもはやめ、エクセル１本でするのがいい。それをワードにでも、年賀状ソフトへでも自由に渡せる。ハガキだけでなく、ラベルなど、さまざまなものへの印刷で利用できる。

１.３．使用できる文字コード範囲の問題

ＪＩＳとユニコード

　普通にパソコンで文字を入力したり、見ているだけでは文字についての疑問を感じることはないと思える。だが、このような経験はないだろうか。
　メモ帳やエディタを使っていて、ＩＥ（インターネット・エクスプローラ）、ワードやエクセルではちゃんと文字が表示されているのに、文字が [?] ［ ▮ ］に化けている、なぜだ。
　理由は、化けたその文字は、ＪＩＳで規定する第１水準と第２水準の範囲にはない文字だからだ。
　では、その文字は何なのか。というと、それはユニコード（あるいは外字）という範囲の文字なのである。
　ウインドウズでは、使用できる文字コードの範囲をユニコードとしている。ウインドウズが全世界共通の基本ソフトであることから、世界のさまざまな文字に対応しているからそれはあたりまえでもあり、必要な基準であったのだ。アルファベット圏も漢字圏も含み、左から右へ、右から左へ、上から下へもサポートしなければならないという宿命を持っている。（実は、Windows95、Windows98、WindowsMeはユニコードＯＳではない。Windows2000、WindowsＸｐから内部一表でない一でUnicodeを使うようにしたのだ。ここでウインドウズと呼んでいるのはWindowsXp以降のこと。）
　ユニコードはApple社、IBM、Microsoft社など米国コンピュータ会社が中心となって提唱し、1993年ISO（国際標準化機構）で世界標準規格「ISO/IBC 10646」として採用された。世界中

のすべての文字を16ビット（2バイト）で表現できるようにしたものだが、後日の改定で大幅な拡張がされた。
　世界中の文字を共存させられるといっても当然限界があり、無理をともなっている。何事にも完璧はなく、過信はできない。
　アルファベット圏の話題はおいておき、漢字圏は文字数が多い。ＣＪＫという言葉がひんぱんにでてくるが、中国、日本、韓国で使用する漢字のこと。中国は大陸の簡体字と台湾の繁体字がある。韓国（朝鮮）はハングルだけでなく漢字も使用されている。日本語だけでも包摂で同じとみなされている漢字の扱いに、同じと思える字でも字形が加わるので判断はきわめて困難だ。簡体字と繁体字はあきらかな字形の相違があるが、点やはねなどの微妙な違いはほとんど混乱を増幅するような問題だ。だが、コンピュータ化が進む中では決めなければ間に合わない。ということから、急遽基本ソフトを作っている米国主導で決まった。
　社会を動かす文字コードの問題を十分とはいえない討議のなかで決まったものがユニコードだ。何度か調整もあった。大きなくくりとしては多少の矛盾をかかえながらも実用上のスタンダード化していっているのは事実だ。
　ＪＩＳは、１～９４区×１～９４点、つまり９４×９４のマトリックスに第１、２水準を配置したもの。シフトＪＩＳは区と点の合成を１６ビットで8140から順に表示できるように番号をつけかえたもの。新しいＪＩＳでは句点では不足したために、面を導入。１面に第１～３水準を、２面に第4水準2436字と1910字とを配置し、面句点への拡張をおこなった。
　従来のＪＩＳ表は１面で、空いている場所に「外字」「ＮＥＣ補助」「ＩＢＭ補助」があった。これらは、別の場所の機種依存文字とともに別のコードとして正式に採用されたのだが、従来のエリアは別の文字のコードになる。

JISコードの制定と改定

	1978年	1983年	1990(基本)	1990(補助)	2000年	2004年
初版	JIS X 0208		JIS X 0212		JIS X 0213	
第１水準	2,965字	1,965字	1,965字	5,801字	-	-
第２水準	3,384字	3,388字		3,390字	-	-
第３水準	*	*		*	5,801字	1,908字
第４水準	*	*		*	-	2,436字
非 漢 字	453字	524字	577字	266字	266字	661字
合　計	6,802字	6,877字		12,999字	-	4,383字

*JISは一見系統だち改訂で継承されているように見えるが、前に規定された版は基本的に無関係とみたほうがわかりやすい。表に上げた文字数などは参考までのメモで、簡単にピックアップできないのでよくわからない。
*1990年の規定にもとづいた［JIS X 0208：1990］を当時の新ＪＩＳとしてＸｐ等で採用。
*1997年には1983年改訂での包摂基準に整合性を追及。
*2000年に文部科学省が表外漢字(常用漢字表にない漢字)の字体を定義。（漢字 10,040字）区点から面区点を採用した。
*2004年、経済通産省は「JIS漢字コード表の改正について ―168字の例示字形を変更―」（JIS X 0213:2004）を発表。（漢字 10,050字）
*2004年に改正された戸籍法施行規則はJIS X 0213:2004の印刷標準字体を採用した。

ＪＩＳ第１水準　　8140～9872（シフトＪＩＳコード）
ＪＩＳ第２水準　　989F～EAA4
ＩＢＭ選定拡張文字　FD40～EEFC　388字
ＮＥＣ選定特殊文字　FA40～FC4B
ＪＩＳ第３水準
ＪＩＳ第４水準
Adobe−Japan1−4（ヒラギノ）　　15,444グリフ（漢字 9,138字）
Adobe−Japan1−5（ヒラギノ）　　20,317グリフ（漢字 12,676字）
Adobe−Japan1−6
ユニコード　　　　0001～******（ユニコード）

ユニコードは16bit（65536字）が最大として規格をスタートしたものだが、最近これでは不足することとなり、32bitのサロゲート・ペアという仕組みを用意している。

下記は、ユニコードの範囲の分類である。

0000－007F	Basic Latin	基本ラテン
0080－00FF	Latin-1 Supplement	ラテン１補助
0100－017F	Latin Extended-A	ラテン拡張Ａ
0180－024F	Latin Extended-B	ラテン拡張Ｂ
0250－02AF	IPA Extensions	ＩＰＡ拡張
02B0－02FF	Spacing Modifier Letters	スペース調整文字
0300－036F	Combining Diacritical Marks	結合分音記号
0370－03FF	Greek and Coptic	ギリシャ
0400－04FF	Cyrillic	キリル
0500－052F	Cyrillic Supplement	キリル補助
0530－058F	Armenian	アルメニア
0590－05FF	Hebrew	ヘブライ
0600－06FF	Arabic	アラビア
0700－074F	Syriac	シリア
0750－077F	Arabic Supplement	アラビア補助
0780－07BF	Thaana	ターナ
0900－097F	Devanagari	デバナガリ
0980－09FF	Bengali	ベンガル
0A00－0A7F	Gurmukhi	グルムキー
0A80－0AFF	Gujarati	グジャラート
0B00－0B7F	Oriya	オリヤー
0B80－0BFF	Tamil	タミール
0C00－0C7F	Telugu	テルグ
0C80－0CFF	Kannada	カナラ
0D00－0D7F	Malayalam	マラヤラム
0D80－0DFF	Sinhala	シンハラ
0E00－0E7F	Thai	タイ
0E80－0EFF	Lao	ラオス
0F00－0FFF	Tibetan	チベット
1000－109F	Myanmar	ミャンマー
10A0－10FF	Georgian	グルジア
1100－11FF	Hangul Jamo	ハングル字母
1200－137F	Ethiopic	エソピック
1380－139F	Ethiopic Supplement	エソピック補助
13A0－13FF	Cherokee	チョロキー
1400－167F	Unified Canadian Aboriginal Syllabics	アボリジニ
1680－169F	Ogham	オガム
16A0－16FF	Runic	ルーニック
1700－171F	Tagalog	タガログ
1720－173F	Hanunoo	ハヌノー
1740－175F	Buhid	ブハイド
1760－177F	Tagbanwa	タグバンワ
1780－17FF	Khmer	クメール
1800－18AF	Mongolian	モンゴル
1900－194F	Limbu	リムブ
1950－197F	Tai Le	タイレ
1980－19DF	New Tai Lue	新タイル
19E0－19FF	Khmer Symbols	クメール記号
1A00－1A1F	Buginese	バギニーズ
1D00－1D7F	Phonetic Extensions	ホンチック拡張
1D80－1DBF	Phonetic Extensions Supplement	ホンチック拡張補助
1DC0－1DFF	Combining Diacritical Marks Supplement	分音符号補足
1E00－1EFF	Latin Extended Additional	ラテン拡張追加

1F00－1FFF	Greek Extended	ギリシャ拡張
2000－206F	General Punctuation	一般句読点
2070－209F	Superscripts and Subscripts	上付き・下付き
20A0－20CF	Currency Symbols	通貨記号
20D0－20FF	Combining Diacritical Marks for Symbols	分音符号記号
2100－214F	Letterlike Symbols	文字様記号
2150－218F	Number Forms	数字形
2190－21FF	Arrows	矢印
2200－22FF	Mathematical Operators	数学記号
2300－23FF	Miscellaneous Technical	技術記号
2400－243F	Control Pictures	制御記号
2440－245F	Optical Character Recognition	ＯＣＲ
2460－24FF	Enclosed Alphanumerics	囲み英数字
2500－257F	Box Drawing	罫線素片
2580－259F	Block Elements	ブロック要素
25A0－25FF	Geometric Shapes	幾何学模様
2600－26FF	Miscellaneous Symbols	その他記号
2700－27BF	Dingbats	装飾記号
27C0－27EF	Miscellaneous Mathematical Symbols-A	その他の数学記号Ａ
27F0－27FF	Supplemental Arrows-A	矢印補足Ａ
2800－28FF	Braille Patterns	ブライユ記号
2900－297F	Supplemental Arrows-B	矢印補足Ｂ
2980－29FF	Miscellaneous Mathematical Symbols-B	その他の数学記号Ｂ
2A00－2AFF	Supplemental Mathematical Operators	数学操作記号
2B00－2BFF	Miscellaneous Symbols and Arrows	その他の矢印記号
2C00－2C5F	Glagolitic	グラゴール
2C80－2CFF	Coptic	コプト
2D00－2D2F	Georgian Supplement	ジョージア補足
2D30－2D7F	Tifinagh	タイファナグ
2D80－2DDF	Ethiopic Extended	エチオピア拡張
2E00－2E7F	Supplemental Punctuation	句点補足
2E80－2EFF	CJK Radicals Supplement	ＣＫＪ偏旁補助
2F00－2FDF	Kangxi Radicals	カンキ偏旁
2FF0－2FFF	Ideographic Description Characters	表意文字
3000－303F	CJK Symbols and Punctuation	ＣＪＫ記号・分音
3040－309F	Hiragana	ひらかな
30A0－30FF	Katakana	カタカナ
3100－312F	Bopomofo	注音字母
3130－318F	Hangul Compatibility Jamo	ハングル互換字母
3190－319F	Kanbun	漢文
31A0－31BF	Bopomofo Extended	囲みＣＪＫ月
31C0－31EF	CJK Strokes	ＣＪＫ互換文字
31F0－31FF	Katakana Phonetic Extensions	ＣＪＫ統合漢字
3200－32FF	Enclosed CJK Letters and Months	ＣＪＫ内部文字付月
3300－33FF	CJK Compatibility	ＣＪＫ互換
3400－4DBF	CJK Unified Ideographs Extension A	ＣＪＫ統合表意文字拡張Ａ
4DC0－4DFF	Yijing Hexagram Symbols	Ｙｉｊｉｎｇ記号
4E00－9FFF	CJK Unified Ideographs	ＣＪＫ統合表意文字
A000－A48F	Yi Syllables	Ｙｉ音節
A490－A4CF	Yi Radicals	Ｙｉ偏旁
A700－A71F	Modifier Tone Letters	装飾子
A800－A82F	Syloti Nagri	シロチ・ナグリ
AAA1－ABCF	Undefined Area	（空白領域）
AC00－D7AF	Hangul Syllables	ハングル
D800－DB7F	High Surrogates	私用領域高１
DB80－DBFF	High Private Use Surrogates	私用領域高２
DC00－DFFF	Low Surrogates	私用領域低１
E000－F8FF	Private Use Area	私用領域低２
F900－FAFF	CJK Compatibility Ideographs	ＣＪＫ互換漢字
FB00－FB4F	Alphabetic Presentation Forms	アルファベット表示形

FB50－FDFF	Arabic Presentation Forms-A	アラビア表示形Ａ
FE00－FE0F	Variation Selectors	変化記号
FE10－FE1F	Vertical Forms	縦形
FE20－FE2F	Combining Half Marks	結合半角記号
FE30－FE4F	CJK Compatibility Forms	ＣＪＫ互換形
FE50－FE6F	Small Form Variants	小字形
FE70－FEFF	Arabic Presentation Forms-B	アラビア表示形Ｂ
FF00－FFEF	Halfwidth and Fullwidth Forms	半角形・全角形
FFF0－FFFF	Specials	特殊文字
10000－1007F	Linear B Syllabary	
10080－100FF	Linear B Ideograms	
10100－1013F	Aegean Numbers	
10140－1018F	Ancient Greek Numbers	
10300－1032F	Old Italic	
10330－1034F	Gothic	
10380－1039F	Ugaritic	
103A0－103DF	Old Persian	
10400－1044F	Deseret	
10450－1047F	Shavian	
10480－104AF	Osmanya	
10800－1083F	Cypriot Syllabary	
10A00－10A5F	Kharoshthi	
1D000－1D0FF	Byzantine Musical Symbols	
1D100－1D1FF	Musical Symbols	
1D200－1D24F	Ancient Greek Musical Notation	
1D300－1D35F	Tai Xuan Jing Symbols	
1D400－1D7FF	Mathematical Alphanumeric Symbols	
20000－2A6DF	CJK Unified Ideographs Extension B	JIS第３、４水準の一部
2F800－2FA1F	CJK Compatibility Ideographs Supplement	
E0000－E007F	Tags	
E0100－E01EF	Variation Selectors Supplement	
F0000－FFFFF	Supplementary Private Use Area-A	
100000－10FFFF	Supplementary Private Use Area-B	

ウインドウズの前身のＭＳＤＯＳではシフトＪＩＳを採用していた。

シフトＪＩＳは、ＭＳＤＯＳが開発された時点のＪＩＳコードをもとにして１バイト系文字であるＡＳＣＩＩと同時に使用できるようにコードシフトして用意したものだ。当時すでに専用ワープロなどでもＪＩＳコードをベースにして使用していたのだが、コードをシフトしての採用はマイクロソフトが独自に行ったものである。

１６×１６ドット、あるいは２４×２４ドットの「漢字ＲＯＭ」がハードとしてボードに搭載されていて、ここからフォント情報を得て画面に表示していた。プリンターでは別途プリンター自身に漢字ＲＯＭを持っていて、印刷ではこちらが使われる仕組みが基本だった。

後継のウインドウズになったときにＤＯＳと互換を取る必要があった。ウインドウズは世界対応からユニコードを使えるように仕組みつつも、前面はシフトＪＩＳが採用されたのだ。

ウインドウズになった時点で、フォント情報は「漢字ＲＯＭ」を使う方式を廃止し、すべてをソフトウエア的に処理するようにした。印刷もプリンターに漢字ＲＯＭの搭載を不要とし、画面の情報がそのまま紙に出力できるようになった。そのため自由にいくつでもフォントをインストールし、複数のフォントを同じ画面に使用で切るようになった。まあ、ＭＡＣに追いついたというのが正しいのだが、画期的な前進をみたのだ。日本語ウインドウズは、標準のフォントとしてＭＳ明朝とＭＳゴシックを搭載した。

ＪＩＳは実際にパソコンに漢字ＲＯＭあるいはフォントとして搭載されるときに、パソコンＯＳの違いや、メーカーの思惑から未定義のエリアの使い方が違った。ＭＡＣとＤＯＳ以前のＭＥＣ製パソコン、ＩＢＭパソコン、そしてウインドウズでは、いわゆる機種依存文字といわれるエリアの相違を生んだ。このブロックの文字は文字化けをおこすので「なるべく使わないように」といわれているのだが、現実には「半角カナ」、[①][㈱]などの機種依存の記号、[髙]などは無視して市民権を得ている。使えてしまうものを「使うな」というのは、しょせん無理なのだ。

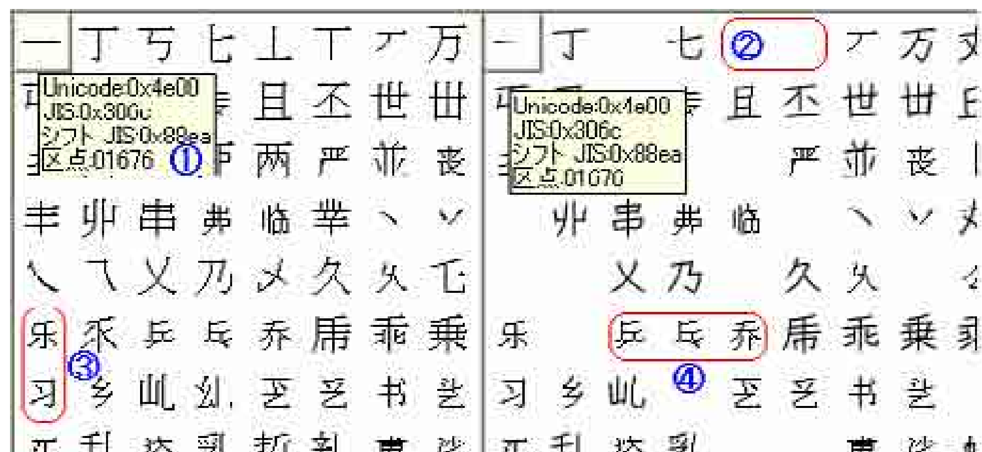
さて、ＪＩＳ第３、４水準といわれて、すぐに理解できるひとはどれほどいるのだろうか。
パソコンではＩＭＥパッドでくくりを見ようとしても、簡単にはわからない。第１、２水準のように並んでいるわけではないからだ。ほとんどはユニコードの範囲にいりまじっているものだからだ。経済通産省のペーパーによるドキュメントであれば、２面に表になっているというものなのだ。同じような問題に、「常用漢字」「人名漢字」というものがある。これらも、ＪＩＳのコード表のようには見られない。つまり、コードがまとまったブロックのようなものとして寄せられないからだ。
ＪＩＳ補助漢字(JIS X 0212)というのもあり、第３、４水準でおよそ９００字が含まれる。
いわゆる「通産省系」と「文部省系」の政府内での考え方の不一致が、社会の現場での運用に混乱を起こしているのではないかと勘ぐる人もいるが、あながち間違いではなさそうだ。

フォントと書体

ウインドウズとして、基本的な仕組みは完成したといってよい。だが、運用上ではＤＯＳをひきづったことから問題をかかえた。あくまでも表面はシフトＪＩＳで通したことだ。日本語は横書きだけではなく、縦書きもあるのだが、縦書きは基本ソフトでほとんど考慮されていないといってよい。
WindowsXpまでの時点の話としては、ＭＳ明朝、ＭＳゴシックともにワードでの使用を考えてユニコードの範囲の文字デザインを作った。シフトＪＩＳ第１、２水準はクリアしているのだが、ユニコードの範囲の文字についてはフルサポートしているとはいいがたい。それは、ＭＳ明朝、ＭＳゴシックともに縦書きをすると横書きでは出ていたユニコードのフォントがでないのだ。
横書き用のＭＳゴシックでも、ユニコードの範囲は驚くべきことに、ＭＳ明朝のフォントを代替でだしている始末だ。つまり、インチキ。
これは言い過ぎというひともいるかもしれない。実は、Arial Unicode MS というゴシック体フォントがる。Unicode 2.1対応で50,377グリフを収録している。表示してみると日本語を知らない外人が作ったと連想しそうな見にくい文字デザイン。そればかりかフォントの高さに互換性がなく、ワードでは突然行間がパーッと開いてしまう。使えるといはいいがたく、アリバイ作りで添付しているだけ、知っても使っても欲しくないのではないか、と勘ぐりたくなるようなモノ。これはＭＳゴシックの裏ゴシックなのか、実態なのか、どのような関係のものなのかよくわからない。Unicodeというだけあって、確かにユニコードをサポートしている。が、これをワードやエクセルで指定して使用しているケースはどの程度あるのだろうか。
それを考えるとやはり、インチキではなく対応しているといえるのは、唯一ＭＳ明朝だけ（横書きの場合）。
ウインドウズには、ＭＳ明朝、ゴシックの他に、ＨＧ系のフォントがいくつかバンドルされている。これらのユニコード対応はどうだろうか。全滅。シフトＪＩＳの範囲内で使用しているぶんにはなんら問題はないが、一歩ユニコードの範囲まで目を広げるとこうした問題がでてくる。
もちろん、パソコンを文字処理、プロの印刷のプリプロセスの道具として位置づけ、使っている現場はちょっと違う。アドビ社のイラストレータ、インデザインなどとそれに類する他社のプロ用のツールでは、プロ用のフォントを使う。１セット数万から数十万するフォントで、文字デザイン（グリフ）がフルサポートされている。ユニコードだけではなく、いわゆる異字体についてもサポートされている。ＣＩＤフォントである。
さて、フォントと書体だが、厳密には別物だ。その説明をしてもしかたない。おおかた同じと思うのも許してよい。
文章を印刷という方向から見ると、いくつかの属性がある。文字の形（個別に名前あり）、大きさ、色。装飾として、扁平率、カーニング、傾き、下線など。フォントがインストールされていれば、文字の形を名前で指定すれば、形だけを変更できる。大変便利な仕組みだ。厳密にいえば、文字の基本ピッチの相違などがあるので、字詰めが変わったりするのだが、基本はこのようにさまざまな属性を変更することによって、印刷の体裁を任意に変更できる。組版をするということは、その詳細を使いこなすことだ。

IMEパッドとIME

IMEパッドというだけあって、IMEとIMEパッドは一体のようだ。
手書き、文字一覧、ソフトキーボード、画数、部首、音声からの入力をサポートしている。
音声は興味があることはあるのだが、使ったことはない。ソフトキーボードはまさにキーボードからの操作のエミュレーションをしている。画数と部首は、入力したい文字がよみ変換ででてこなくても明確にわかっていれば利用できる。手書きは実際に漢字を手書きする。これは、結構使用する機能だ。
さて、文字一覧だが、①JISかUnicodeかの選択、②漢字表のエリア選択、③フォント名が指定できる。JISに「漢字３」というブロックがあるので漢字第３水準かと思いきや、これはIBM補助漢字だ。
IMEパッドのスペックはすばらしい。これだけで、日本語の入力のすべてがサポートされている。
IMEパッドでフォントのサイズが変更できると申し分なのだが。
Unicodeにして、フォントをいろいろ変更してみてみると面白い。



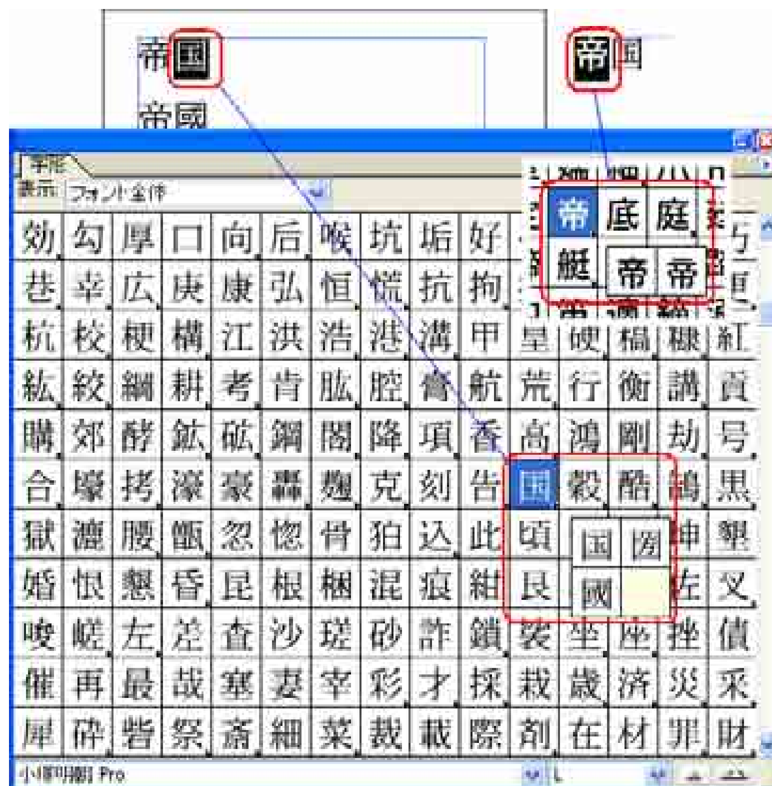
IMEパッドの「文字一覧」の一部分
左はMS明朝、右はY.OzFont

① は「一」についてのコードが表示されている。
② フォントの文字デザインが存在しないもの。MS明朝にはあっても、すべてのホントにデザインがあるわけではないことがわかる。
③ 文字がひとまわり小さいもの。これは、中国の簡体字のようなのだが、さだかではない。IMEパッドの説明をみても説明されていないようなのだ。
④ MS明朝体で表示されている文字が、デザインを持っていないフォントでそのまま「代替」で表示されている。よくわからないのは、デザインがない文字のすべてが「代替」されているわけではないことである。

異字体とＣＩＤフォント

漢字には、異字体という問題がある。同じ漢字でも、手書きでくずした文字がデザインされたもので中国の簡体字のようなもの、戦前まで使用されていた古いデザインの活字、渡邊の「ナベ」に見られるようなちょっとしたデザインの相違文字等さまざまである。
これらは、現在での使われ方や経緯からＪＩＳ第２、第３水準、あるいはユニコードに納められているばあがある。しかし、当然古書体デザインの場合などの多くは包摂という概念で同一視され、コードを持たない場合もあるのだ。
アドビ社が中心になってこうした異字体をＣＩＤフォントというもので実現している。
つぎの図は、イラストレータでの異字体選択画面だ。

帝国の「国」は異字体で表示されている文字はいずれもコードを持つが、「帝」の場合、立の１画目が横線になっている文字があるが、これはコードをもたない異字体だ。
フォントは「小塚明朝」だが、文字デザインをきちっと持ったもフォントであれば利用できる。実際には、プロの現場が使用するフォント以外ではこのようなデザインを保持していない。



横書き明朝フォントで縦書きが可能か

「横書き用のＭＳ明朝しかユニコードを正しく表示できないようだ」という現実は問題。特に、ハガキ等への縦書き宛名印刷を考えた場合に、楷書体、行書体、教科書体などが「代替」でしか使えないというのは、ユニコードをあきらめろというに等しいことになる。
まあ、楷書体、行書体、教科書体等はせめてあきらめたとしても、明朝体だけでも縦書きはできないものだろうか。
横に正しく書ける文字を、どうせ１文字づつ描画するのなら、横書き用のフォントをそのまま使えばいいではないか、と思いつく。やっかいな表示の処理などもいらないし、いいことづくめのように見える。
だが、結論としてこれは無理である。
日本語の文字を扱うときに、別の問題も考慮しなければならないからだ。
縦書き用フォントでプロポーショナルというのはいったいどうなるのだろうか。１文字単位の文字幅情報が得られないので対応できない。すべての文字が同じ幅になってしまう。
もっとやっかいな問題がある。拗音、促音、音引き、句点だ。これらは縦書きと横書きでは、フォントの仮想枠内での位置が違うのである。限られた文字だけとはいえ、この補正処理はやりたくない処理。
ということで、けっきょく横書き用フォントを縦書きで使うのはできない。ＰＣに搭載されるフォントはさきざき、ユニコード部分についても書体を持つようになるのを期待するしかないわけだ。
ちなみに、横書き住所でよく使う３－２－１のようなマイナスを使う表記だが、縦書きにすると記号のマイナスは正しく変換されない。半角のマイナスでは短すぎ、ダッシュでは長す

ぎる。これはワードなどでも同じで、仕組みを「日本人でない、しかも技術系の人々」が中心になって作られたなごりを感じる。

外字の扱い

エディタで見たときに文字が [?] [▮] [・] に化けるときに、もうひとつ考えなければならないのは外字である。

外字は、ユーザー定義できる文字のことで、誰かが作字したものだ。別の人が作ったワードやエクセルのファイルを印刷したときに、印刷するパソコンに同じ外字が定義（くみこみ）されていないと表示されないのは当然である。

最初に外字をつかったひとが、使ったことを忘れてしまう。他のひとにファイルを渡すときに外字ファイルを渡すのを思いつくのは困難だ。また、外字ファイルをもらったからといって簡単に自分のパソコンにいれていいわけでもない。自分が使わなければならない外字が使えなくなる。一緒にしようとしても、コードが重なっているかもしれない。そのときだけしか使わないファイルなら、わざわざ外字をインストールはしたくない、というような問題だ。

仕事となると、もっと大きな問題をはらむ。

たとえば、住所の場合。東京都葛飾区…という場合、葛は現在[メ]を使うデザインになっている。これで１００％ビジネス文書では意味を貫いているのだが、葛飾は古くから[人]を使うのだという人がいる。確かにこの字は正しい字形ではない。いわゆる「活字」で一般化していた字形とは異なる。ところがパソコンではこの字形のフォントがない。ないフォントは外字で作って使うとなる。官公庁から印刷の仕事を受けるときに、それを業者に求める。業者は外字を作り対応しようとするが文字数が多い場合にもれずに処理するのは大変な労力を要する。それでいて、作字とチェックに要した負担の経費を払うことはない。納期が延びるわけでもない。

これを平気で指示するこの官公庁のひとははっきりいって不適格者。このひとをそのような立場に配属した上司の不認識だ。この一事が犯罪といってもいい深刻な問題として改善して欲しいと思うのは、筆者だけではあるまい。これは、単に後述するWindow Vista が登場した後、すなわちフォントが普及した時点の後で求めれば言い話。

ついでだから、具体的に事実を指摘しておこう。第１に、官公庁の印刷物は「ビジネス文書」であって「芸術作品」ではないという点だ。ビジネス文書は合理性こそ命（いのち）である。はやく、むだなく、用件が伝達できればいい。葛飾と普通に書いたことでのビジネス文書としての問題点などゼロである。第２に業者にかけた負荷についてノーペイというのは詐欺という犯罪だという認識がないことである。現場で発生する無駄な損失、しかもそれが大きければ大きいほど自分が得したような倒錯した姿勢は民間企業なら許されるはずがない。

外字にまつわる理不尽な問題をもうひとつ。しょっちゅうあるはなし。印刷物に出てくるある人名なのだが、それがやはりＪＩＳ外の文字でユニコードにもない文字を使っている。[祷]と[禱] という文字がある。示＋寿を使えというのである。印刷物の内容は会議の資料で人名にこだわる必要のあるものではない。理由は、発注者の「上司の名前だから」というだけのもの。普段は禱をつかっているという。

取り上げたなまなましい例は、いずれも文書としては他人に渡ったら正確に用をたさなくなるという、デジタル時代に逆行するモノだ。外字を使うことでこうした文書は互換性を捨てなければならない運命を背負うのだ。

外字はまことにやっかいなモノだ。可能な限り使わないに限る。

ＪＩＳ外でも一部機種依存文字としてだがウインドウズであれば使用できる文字がある。ＮＥＣ選定とＩＢＭ選定がある。これがＪＩＳ第２水準の後ろにあるのだが、この２つの選定文字は基本的には同じ文字だ。ということは、同じ文字なのに２つのコードがあるということだ。マイクロソフトが利便性からここにあげたのだろうが、混乱のもとをも同時に作ってくれた。

外字は使用できるコードの範囲がある。

特別なエリアのコード配置

Shift JIS外字エリア	F040～F9FC
Unicode外字エリア	A000～A757
True/OpenTypeFont外字エリア	E000～F8FF
縦文字エリア	EB40～EFFC

経済通産省のＪＩＳコード

経済通産省は２００４年の２月に不可解な発表をした。どのような立場で、どんな権限でそんなことをいいだしたのか、首をかしげるような発表だった。「JIS漢字コード表の改正について —168字の例示字形を変更—」（JIS X 0213:2004）というものだ。

「JISは、漢字に対する符号（コード）を定める規格であり、字形は規定していない。このため、JISにおいては、今回の改正によって変更された字形と変更前の字形は、どちらも同じものとして取り扱っている。したがって今回の改正が、パソコンなどに搭載される字形の変更を求めるものではない」と述べながら、一方では「今回の改正によってパソコンなどに搭載される字形が、徐々に印刷標準字体に変更されることが期待される」といっている。

そもそもＪＩＳは文字コードを決めたもので、文字のデザインについては規制していないし、できようもない。書体はデザインの分野に属するものである。紛らわしいが同一とみなしたものは包摂というくくりであいまいさをいわば公的に許し、社会的に実績を積んできた。

ところが、ここでは事実上書体デザインそのものではないのだが、いままで包摂であった文字の字形に公然と口をはさんできたのである。

現在パソコンで一般化されている文字の字形について、正しくないものがある。それはこの168文字だ、といいたいようだ。ＪＩＳを管掌するところから出してきたところが奇異なのである。「正しくない」文字の原型は、ＪＩＳを最初に決めた時点で公表されたコード表の字形にあるのだ。このときはこととき で「正かった」はずのものだ。文字は生き物であり、時代とともに変化するのはいうまでもない。ためしに、大正時代から戦前の活字をみてみればわかる。むずかしそうな字形をしている。それらは、戦後改革されて現在見ているような字形に変化したのだ。が、JIS X 0213:2004は、ほとんどが「正字」「旧字」に逆戻りしている。それをこれから使う字のモトにしたい、というものである。

指摘された文字はそれが「正字」「旧字」に違いないことは皆みとめる、ということと、それを「印刷標準字体に変更されること」とは別である。むしろワープロとパソコンの普及以来の実績を否定し、変更してしまうことでの混乱こそが問題なのだ。

正しい字形が「漏れている」というのであれば、単にユニコードへの追加をすればよかっただけなのではないだろうか。

JIS X 0213:2004は、JIS X 0208：1990を「継承したのではなく、新規にＪＩＳを決めた」ものなのだ。「葛」「飴」などはホントはウソ字だった、これを１９８３年に「例示字形」で示したのが間違いだった、２０余年世にでたのは幻だったと思って欲しい。この事実自身がいまいましいので、抹殺したい。新しいフォントが出て現場は混乱するのは承知だが、なかったものだと思って永遠に忘れ、再スタートして欲しい、というのが彼らの姿勢なのである。

正式に「奈良県葛城市」と字形まで採用したところも、こっそりと移行して欲しい、といったところかもしれない。（ウソ字、間違い字というよりも、略字。一歩ゆずっても俗字。しかも、２０年以上経て立派に市民権を得た正式な字形ではないのかネ。）

WidowsVistaがJIS2004に対応

ウインドウズの新版 Windows Vista が２００７年初には発売される予定だ。２００６年の５月マイクロソフトはこの新しいＯＳに搭載するフォントについて発表した。標準フォントとしてＭＳ明朝２書体、ＭＳゴシック３書体、そして新しい仕組みとしてのクリアタイプフォントの「メイリオMeirio」である。そしてこれらは、JIS X 0213:2004対応とのことなのだ。

同時に明朝とゴシックについては、Ｘｐ用もダウンロードして使えるようにするという。

Ｘｐ搭載のフォントはJIS X 0208：1990で普及しきっている。同じテキストを新ＯＳで表示（印刷）すると別の字形が表示されるという混乱が目に見えているのだ。

確かに、字形はミズモノで、社会の移り変わりを反映して変化するものだ。だが、あえてこのような形での混乱の増幅は必要なのだろうか。マイクロソフトは、時の政府の発表を常に最新として、過去とは無関係にそれを採用して先に進んでいくだけ。

ちなみに、クリアタイプフォントとは、フォントのレンダリング機能の相違のようだ。この仕組みが入っているのかどうかが違い、と思えばいいのかな。

Verdana 10point in black and white
Verdana Italic 10point in black and white

Verdana 10point in ClearType
Verdana Italic 10point in ClearType

ちなみに、メイリオのかなとカナは見るとどうも横長の丸ゴジといった感じだ。MS UI Gothicがやや縦長だったのを逆にしたもののようだ。

気になることのまとめ

政府と官僚はその場の思いつきだけで勝手に方針を変更し、強引に押し通す。マイクロソフトは右から左にそれを採用する。現場の混乱には関心がない。Windows Vista が使われだすと、同じワードのドキュメントでも、別のＯＳで印刷したものとあきらかな違いがでてくることを心にとめておくことだ。

１.４. 文字の配置と郵便番号

郵便番号枠

官製ハガキや市販のハガキには右上に赤の枠で郵便番号欄が印刷されている。
この位置に手書きで正しく郵便番号を記載すれば、郵便局の自動読取装置で確実に分別処理され最速の配達ルートに乗るのである。郵便番号と住所は合致していないといけない。違えば機械ははじき、手作業にまわるために配達がウンと遅くなる。
旧郵政省、いま総務省のホームページをみれば、手書きの時代の説明が読める。読み取りやすい字形と書き方の注意までわかるのだ。
住所や宛名はいま大半プリンターでの印字だ。ラベルを貼ったりしているので、多くはこの郵便番号枠が無視されている。でも、大丈夫。読取装置は改良されて、住所の上(横)に活字体で印刷されたものなら、小さくてもちゃんと読み取ってくれる。
ということで、郵便番号の枠は手書きする場合のガイドといったところ。
機械は読んだ後、目で見えないインクで宛名固有のカスタマーバーコードを印字していく。
トラックで各郵便局に配送され、配達局ではカスタマーバーコードを頼りに配達のルートの順で並べ替えられる。配達員はこの順で届ける。無駄がない、ということになる。実態はどうか関心があるが、それはここでのテーマではない。

カスタマーバーコード

一般にはカスタマーバーコードはあまりなじみがないかもしれない。だが、郵便局での処理から見ると、欠かせないポイントになっているのだ。これで、全国の配達局への分配をおこない、配達のルートの配達順番を決める。
１戸を特定しているのだ。これは地図上の配達ルート（道路も考慮した）とも関係しているから何ともすばらしい。住宅は新たに建築され、壊されもする。道路も同じだし、近年の市町村合併で住所や郵便番号はどんどん変化する。
だが、このシステムのおかげで日本の郵便配達制度はいま近ければ翌日配達のようなすばらしいサービスが実現しているのだ。手作業ではノウハウが人間に蓄積されなければならず、こうはいかない。正職員をひとりでもへらし、アルバイト社員でまかなうという郵政民営化方針を進める中では欠かせないバックボーンになっているのだ。
１０００通以上の郵便物を扱う大口の場合は、郵便を使う側があらかじめカスタマーバーコードを印字し、郵便番号ごとにまとめて局に持ち込めば５％以上の割引が受けられる制度もある。ＤＭを扱う業者は当然これを利用している。

郵便番号を寄せてカスタマーバーコード付きで宛名を印刷すればいいだけなので容易に実現できるのだ。
総務省のホームページではカスタマーバーコードの規格がわかるようにしている。

事業所用の郵便番号

郵便番号には、一般的に地名番地をベースにしたものの他に、官公庁ベース、企業事業所ベース、大きなビルディングベースとある。正式には「大口事業所等個別番号」と言うようだ。１日の平均５０通を越える郵便物がある必要がある。
正しい郵便番号が明記されていれば、宛名の住所を省略していきなり市役所、会社名を書いてもちゃんと届く。いずれにしても、最終的に配達を処理する郵便局に連結している。
地域番地をベースにした一般の郵便番号と、官公庁ベース、企業事業所ベース、大きなビルディングベース、私書箱はフェーズが違うので当然ダブルわけだが、このあたりのことについてはどうなっているのかよくわからない。また、官公庁ベース、企業事業所ベース、大きなビルディングベースの番号は消えたり、変わったりすることがひんぱんに起こりえる。したがって、最新版への更新は常時おこなう必要が出てくる。素人目にはちょっと無理があるのではないかと思うが、総務省はこのシステムが将来にわたってベストだと導入したようだ。
官公庁ベース、企業事業所ベース、大きなビルディングベースの郵便番号は、その場にいる関係者にはなじみがあり知っているということなのだが、他からそれを知るのはそうたやすいことではない、のが気になる。最新でかつフルカバーした状態のツールはどうなっているのだろうか。すくなくとも、ＭＳ ＩＭＥの変換やエクセルなどのアドインでは十分とはいいがたいのだが…。

「〒」マーク

「〒」マークについて触れたい。住所録の郵便番号の前に必ずといっていいほどこれを付けるのをみる。パソコンで宛名の印刷する場合も当然出てくるのだが、このマークは総務省の郵便番号の書き方の説明をみると、付けないほうがいいのだ。確かに、郵便番号の誤読に結びつく可能性を増している。〒マークはつけないほうが賢明である。

書体とサイズ

郵便局の自動読取装置が判断しやすいと思える字形とサイズについては、郵便番号の仕様書に記述されている。だが、現実をみると、さまざまであることがわかる。
印刷する活字体であれば、８ポイント程度のものからある。明確に判読できればサイズは常識的な判断のレベルでよい、ということのようだ。
書体は、読みやすいものとして、いわゆるゴシック体が望ましいとなっているが、現実には明朝体のほうが多いと思う。むしろ、問題は毛筆筆記体とか古いイメージの読みにくい書体などを使っているケースであろう。現場ではなるべく使わないようにしている。まあ、実際は世界に誇るＯＣＲである総務省の読取装置はよほどの手書きくずし文字でないかぎり大丈夫のようである。活字体はほぼ１００パーセント、手書きもていねいに書かれているものはほとんどＯＫ、というすぐれものなのである。

宛名印刷に必要な項目と文字の長さ

明朝宛名の印刷と関係して考えると、住所の持ち方はほぼ決まった、といってよい。住所１、住所２の２つに分けて管理するのがベストだ。
地番までを住所１とし、建物室番などは住所２とする。エクセル等でデータを提供してもらうときには、まず、この確認。分かれていなければ、分離する作業から始める。
１行の長さを考えたときに、たいていはこれでおさまる。
つぎに、会社名や肩書きだが、なかには長いのがある。経験では外資系の企業名でカタカナよみがつながっていて数十文字におよぶものがあった。肩書きも、やや大きな（？）企業で第何、なになに推進本部の、第何マーケテングリサーチなんたらかんたら、……と名刺では小さ

な文字で３行のもの。こうした会社名や肩書きと、わずか数文字の会社名とを同じサイズ内に体裁よく一律に印刷できるのは至難のわざといっていい。

名前もばかにできない。日本人の名前はほとんど問題にならないが、外国人の名前があるときだ。アルファベットなら半角文字で処理できても、カタカナで長い文字には処理にへいこうする。もちろん、会社名も肩書きも名前も当人には何のトガもないのはいうまでもない。が、処理するとなるとこうしたことにもきちんと対応するかどうかが、大事になってくる。

ワードの差込印刷をする場合、データを作成するにはちょっと注意しなければならないことがある。実際に印刷したときに、ラベルの枠からはみでないように文字の長さをチェックすることである。単に特定の字詰めで改行するだけではいいとはいえない。総務省の書き方の説明をみるまでもなく、住所番地などの文字のかたまりの「泣き別れ」をなるべくしないようにすることである。

都道府県市町村郡字のちょうどいいところで改行する。番地もそうである。例として１５７２番地なら、この数字の４文字を途中で改行するのは酷である。

後述する縦エディットのサンプルであるPmHagakiAtena では、こうしためんどうをいっきに解決しているのだ。

縦置きと横置き、縦書きと横書き

日本では縦書きという文化がもともとだから、これをきちんとできるようにする必要がある。宛名の印刷には、ハガキも縦置きと横置きがあるので、それぞれに縦書き横書きできることが求められる。

また、縦置きの場合は、郵便番号をどう書くのか。すなわち、ハガキだと郵便番号欄の位置に横書きするのか、それとも無視して縦書きで住所の一部として書くのかということである。

気になることのまとめ

事業所郵便番号などをエクセルなどで簡単に処理できればいい。

長い文字列、読みやすい字体とサイズ、縦書きを忘れずに。

第2部　文字処理プログラミング

２.１.　デルファイとコンポーネント

デルファイの強み

　プログラム作成といってもピンからきりまである。
　大規模なシステム、多重のネットワークは別にして、小規模のネットワークからスタンドアロンのツールを作成するプログラミングのツールとして最適だと思うのはDelphiだ。
　しかも、DelphiのV5まで。内容としてはV7までは同じようなものだが、その後の版は筆者にとって必要がないのでわからないのが正直。
　V5では古すぎるのではないかと思われるふしもあるが、なんら問題なく新旧マシンでちゃんと動いている。むしろ、新版はトレンドに振り回されているきらいを感じないでもない。
　デルファイはいわずと知れたパスカル言語であるが、デルファイの作りはＣ系の下層言語とはいえＶＢでもできないようなことまで手が届くようになっている。Ｃでできないことは何もないというのであるならデルファイでもできない。
　デルファイで大変気に入ったことは実行プログラム１本がつくられ、それを別のマシンに持っていって作動させることができるという手軽さ。ＶＢでのようにちょっとしたプログラムでも数えきれないＤＬＬやらが一緒に持っていく必要があるということがない。つまり、インストーラなどいらない。
　ほんとに必要なときだけ、インストーラを用意すればいいだ。

コンポーネント作成の必要

　デルファイは、標準でも３桁近くのコンポーネントが用意されていてたいていはこれで十分まにあう。
　しかし、ひととおりできるようになると、人間ぜいたくになるもので、好き嫌いがでてきたり、ここをもう少し変えられないかということがままある。
　デルファイのよさは、このへんへの対応がよくできているのだ。
　何を実現したいのかということが明確であれば、こんな手がある、あんな手がある、この方がいいとか解決の道が必ず見つかるもの。
　他の言語も基本的には同じだが、ウェブサイトには先人たちの貴重な体験と解決策があことが多い。少なくともそこには重要なヒントがある。
　縦書きエディット自身では、背景透明化をコンポーネントでどう実現するのかという例を、１０件程度参考にさせてもらった。
　いずれにしても、必要だと思えば手軽に作成できるのがデルファイのいいところだ。

２.１.　デルファイで文字処理

LabelとEdit

　デルファイでの文字処理では、まずLabelとEditがある。
　これは、いうまでもなく、開いているFormにコンポーネットパレットからドロップして貼り付けるだけで使える。画面、すなわち自分が作成する画面に文字を表示したり、文字を入力させるときにひんぱんに使用する。
　Labelは貼り付けて、プロパティを設定するだけで基本的に処理完了する。Captionに文字列、Fontにフォント名、サイズ、色、下線などの属性をセットする。

Editもそうだがコンポーネントの縦書きはない。表示するだけなら、固定長のフォントを選んで、改行などで処理すればいいだけなので、ふだんこれで苦労することはない。
Editは、ラベルと比べると入力をさせる点が大きく異なる機能だ。

MemoとRichEdit

Editが１行を処理するのを主な目的にしているのに対し、MemoとRichEditは、複数行の処理をするものだ。いずれも、縦書きワープロのような縦書き機能はない。
しかし、通常の文字列処理ではこの２つがあれば不足することはまずない。
RichEditは、Memoでは実現できない文字単位での装飾ができる。Memoでは文字列に対するフォントの設定は、全文字に対する設定と同一になる。RichEditでは、文字単位なので相当複雑なことができる。まあ、簡易のワープロのようなことだ。これにイメージをはつるけられるとか、線、矩形、丸などのシェープが使えるとほんとのワープロになるのだが、その機能は標準ではもたない。
RichEditの仕様自身ではイメージを扱える。例えばワード自身は複雑なページを作成してもそれをRichEditのファイル形式であるRTFで保存し、読み込めば完璧に再現できる。
デルファイのRichEditではそうはいかない。

HEditorというメモコンポーネントのすぐれもの

Memoはこれはこれで多く利用されているのだが、ちょっと気が利いたMemoを作ろうとなると表現不足といえる。このあたりをカバーして有名なのがHEditorである。本田勝彦さんの作品で次のような特徴（紹介文）がある。

■ 予約語・記号・文字列・数値・コメント領域・半角文字・全角文字及び半角カタカナ・url・メールアドレスを識別し、それぞれに、背景色、文字色、フォントスタイルを指定
■ {}, (* *), <!-- -->, など、特定の文字列で囲まれた領域を複数行に渡って別色表示
■ 選択領域の背景色・文字色を指定
■ [EOF] マーク、改行マーク、アンダーラインを表示
■ レフトマージン、トップマージン、行間マージン、文字間隔を指定
■ １行文字列のオーナードロー
■ WordWrap 時に、行頭禁則文字／行末禁則文字による追い出し、改行文字のぶら下げ、句読点のぶら下げ、英文ワードラップを行う
■ Undo, Redo
■ 選択された行のインデント・アンインデント
■ ３２Ｋの制限がない。

ここに書いてあることがそのままMemoの不足分でもある。

ImageやPainBoxに描画する

EditやMemoなどは、文字列はテいわゆるキストとしてメモリに記憶されている。ファイルにMemoから書き出すのも簡単だ。
Memo1.lines.SaveToFile(filename); で書き出し、LoadFromFileメソッドで読み出せる。
これに比べて、同じ「画面に文字列を表示する」ことでも、ImageやPainBoxに描画するのは、イメージとしてであってテキストとしてのメモリへの保存はされない。また、PaintBoxでは消えたら「再描画」をして復活させる必要がある。エベントのOnPaintに記述する。このようなやりかたはWindowsのアプリケーションでは普通に知っている必要がある。コンポーネントの作成では当然このような書き方をする。

```
//Imageに文字列描画
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
  Image1.Canvas.Font.Size := 20;
  Image1.Canvas.TextOut(10,10,'Aa1文字列');
end;

//PaintBoxに文字列描画
procedure TForm1.PaintBox1Paint(Sender: TObject);
begin
  PaintBox1.Canvas.Font.Size := 20;
  PaintBox1.Canvas.TextOut(10,10,'Aa1文字列');
end;
```

印刷はPrinter.Canvasに描画する

「テキストを書く」ではなく、「描画する」方法は、プリンターに印刷するときにも使う。
DOSの時代にはプリンターに印刷するとは、プリンターにたいして「テキストをコードで送る」ように書けばよかった。
Windowsでは、Printer.Canvasにたいして「描画する」ことで印刷ができる。
Printerは、Formにコントロールをペタッと貼り付けるようなわけにはいかない。
uses Printers; とすることで、プリンター操作のコードが記述できるようになる。

```
uses Printers;
//プリンターに文字列描画
procedure TForm1.Button3Click(Sender: TObject);
begin
  Printer.BeginDoc;
  Printer.Canvas.TextOut(10,10,'Aa1文字列');
  Printer.EndDoc;
end;
```

TextOutのパラメータは、x位置、y位置、文字列である。xとyの単位は、デバイスのドットである。ディスプレイなら左上からのドット数。同様にプリンターなら用紙の左上端からのドット数である。
ドット数ということは、解像度に依存する。筆者の場合は、１０分の１ミリ単位を基本とし、画面あるいはプリンターに描画するときにはそれぞれの解像度に合わせて数値を変換して使用する。

```
sd := mmmToScreenDot(mmm);
pd := mmmToPrinterDot(mmm):
```

このようにして使用できるように、関数を作って用意しておく。
なお、文字列を描画する例を紹介したが、イメージについても描画の仕方は同じである。

気になることのまとめ

デルファイなどのコントロールには縦書きはない。MemoのいたらないところはHEditorが使える。RichEditはイメージが処理できない。「文字列を書く」と「Canvasに描画する」を使い分ける。位置の指定はドットでそれはデバイスの解像度に依存する。

２.２.縦書きエディットコンポーネントの概略

特徴と機能

Delphi5用TEditの縦書き版コンポーネント。

なるべくTEditに似せた外見にしている。
プロパティを少し拡張している。
①背景透過のビューモード
②横書きと縦書き
③エリア内扁平表示
④Unicode(WideString)を実現。

インストールと使い方

通常のコンポーネントと同じようにインストールする。
MyCompoのパレットにアイコンができる。
削除は、通常のコンポーネントの削除と同じ。

パレットからドロップしたときの、Verticalプロパティの初期値はFalse（横書き）なので、貼り付け後にプロパティを変更する。

TateEditXXXX.zipファイルは、次のファイルがパックされている。
(XXXXは、V2.0.0.1なら、2001とバージョンを読み替える)
TateEdit.pas .... コンポーネントソース
TateEdit.dcr .... アイコン
TateEdit.dcu .... コンパイルされたもの
readme.txt ...... この説明書

プロパティ

拡張機能ためTEditとは微妙に異なるものがある。
TateEditで拡張したものは次のもの。

Autosize
　入力エリアを自動的に調整する。TateEditの初期値はFalse
ALignment=(alLeft,alCenter,alRight,alJustify,alJustifyEx)
文字の表示方法
　alLeft　左あわせ
　alCenter　中心に表示
　alRight　左あわせ
　alJustify　均等割り(左右あわせ)
　alJustifyEx　OffsetEx以内の文字列は均等割り、超えた場合は左あわせ
BorderStyle=(bsSingle,bsRised,bsNone,bsXpStyle)
　bsSingle　TEditの初期値に同じ(入力エリアがくぼんだイメージ)
　bsRised　入力エリアが浮き上がったイメージ
　bsNone　凹凸なし
　bsXpStyle　Delphi5ではmanufestなどでXpStyleに対応できなかったので、XpStyleにしたときの初期値の１つに似せたイメージにしたもの
BevelSize
　枠幅
Enabled
　入力操作はできなくなる。(Transparent時以外は、文字色がグレイになる)
NextFocusByEnter
　[Enter]キーで次のアクティブなコントロールにフォーカスを移す
Offset
　Offsetは、左上から文字の表示開始位置。左、上から指定のドットずらす
OffsetEx
　alJustifyExで使用する極マイナーなプロパティ(詳しくは別項で説明)

Transparent

背景を透明にする（透明時、入力中の文字列の一部の編集機能が使用できない）

ReadOnly

TEditとはやや異なる。ReadOnlyすなわち表示のみの場合は、ラベルのように動作させる目的をもつ。配置した矩形から長さがはみ出すような文字列をセットした場合に、範囲内に収まるように文字の扁平をかけている。ただし、alLeftの場合は扁平されない。

Vertical

初期値はTEditと同じ縦書きだが、VerticalがON時には縦書きになる。

イベント

Editと同じ。

入力中の編集

アクティブになった直後は､カーソルは文字の最終位置｡プロパティのAutoSelectによって､フォーカスを受けたときの選択状態がかわる。

その後、マウスで、文字列の任意の箇所にカーソルを移動する。

編集中のショートカットキー（/の次のキーは、縦書きモードの場合）

矢印	カーソル移動
BS	前１文字削除
DEL	後１文字削除
Shist+DEL	クリア
Ctrl+A	すべて選択
Ctrl+C	クリップボードへコピー
Ctrl+V	クリップボードから貼り付け
Ctrl+X	切り取り
Ctrl+Y	縦横の切り替え
Ctrl+R	編集中からReadOnlyへ（逆はできない）
Ctrl+J	シフトＪＩＳコード入力
Ctrl+U	ユニコード入力
Ctrl+V	１回だけのUndo
INS	挿入/上書モード切替
Shift+→/↓	カーソルから前の１文字選択追加
Shift+←/↑	カーソルから後の１文字選択追加
Ctrl+→/↓	文字列の最後に移動
Ctrl+←/↑	文字列の最初に移動
マウスでドラッグ	SelTextにする(TransparentがOFFのとき)
コントロールの移動	
Tab, →/↓	次のコントロールへ
Shift+Tab, ←/↑	前のコントロールへ
Enter	NextFocusByEnterがTrueのときに次のコントロールへ

コントロールの移動キー

Tab、Shift+Tab、NextFocusByEnterがONであれば、Enterで次に移動する。

ReadOnlyがTrueであれば、↓、↑で移動する。

「縦書きエディット」作成過程での課題

・Editコンポーネントにどこまで似せるか

AutoSize, AutoSelectの初期値は、Editの使用目的の違いからFalseにしたなど。
・縦書きと横書きを別のコンポーネントにするか
Verticalプロパティで判断することとし一体にした。Editには及ばないので、別にすることも考えたが、同じプロパティを維持したまま切り替えができる便利さを優先させた。
そのため、Create時の初期値はVerticalをFalseとした。
・Xpスタイルへの対応をどうするか
開発ツールDelphiのバージョンがV5で、V6以降の機能に対応していないため。
これは、単に標準の設定時の描画を似せただけで、画面の設定を変えている場合などには対応していないので、オマケ的なもの。
・背景透明化と矛盾する文字列の選択をどうするか
Editに近くしたのだが、透明化時に文字選択機能を捨てた。入力を伴って使う場合は非透明、ビューとして使うときは透明が推薦である。
・ユニコード文字列をどう取得するか。
①IME　②IMEパッド　③クリップボードからのペースト　④コード入力
これらの機能には最小限でも対応できなければならない。
・編集機能をどこまでつけるか。
Undoは見送った。キー入力に関係して１回だけのUndo、Redoとした。

気になることのまとめ

縦書きEditを使ったサンプルアプリケーション PmHagakiAtena で利用してみたように、普通に使用するぶんにはほぼ満足できるできになったのではないかと思う。
単にEditを使っているときはまったく気にしないことでも、実際にそれもどきを作るだけでそうとう入り組んだ問題ととりくまなければならないことが分る。先達者の究明に頭がさがりっぱなしである。

２.３.縦書きエディットコンポーネント作成過程でのとまどい

WideString時のCaptionとText

Captionというのは、ラベルなどの場合は「表示する文字」ともいえる。フォームなどのタイトルバーがある場合は、そこに表示される文字列。
Editなどでは、Captionのプロパティは見えず、Textというプロパティがあり、同様に表示する文字であり、編集する文字列。
CaptionあるいはTextは、TCaptionと規定されている。実態はString。
Editのコンポーネントをフォームに貼り付けてみる。貼り付けられたコンポーネントは、Edit1という名前が自動的につく。２つ貼れば、２つめはEdit2になる。同時に、Textのプロパティに名前が初期値としてはいる。
また、オブジェクトインスペクタでTextプロパティの値を変更すると、フォームに貼り付けたEditのテキストもリアルタイムで更新される。
なぜ、CaptionとTextのことを取り上げたのかというと、TateEditでユニコードを扱おうとすると、思ったような動きをしなくなるからである。
Textプロパティの型をWideStringにしたとたんに、このような動作にはならない。デルファイの仕組みはCaption/TextがStringであることを前提にしているからだ。
そのために、TextはWideStringとし、Captionは非公開でそのまま残し、Captionが変更されたときに自動的に発生するメッセージCM_TEXTCHANGEDをなるべく反映させるようにした。CaptionからTextの方向なので、常にそうするのは無意味でかつ、TextがStringになってしまう。TextからCapptionの方向はイベントを発生させるだけのもの。
しかし、こうした対応では解決できないのがTextプロパティを変更したときの即時更新だ。これは、解決方法が見つからないままだが、実用上特に気にならないのでこのままにとどめた。

ユニコードの入力インターフェイス

「劍の右側が刀のけん」がユニコードでＪＩＳにはない文字である。
ユニコードを扱うといったときに、その文字をそもそもどのようにして得たらいいのだろうか。次のようなケースを想定してみる。
・ＩＭＥパッドで手書きする。
・ＩＭＥパッドの「文字一覧」から入力する。
・ＩＭＥ辞書に登録されている文字を変換して入力する。
・既存のワードやエクセルの文字列の一部をカット＆ペーストで持ってくる。
・ユニコードを４桁16進で直接入力する。
日本語の一文字一文字に、これはユニコード、これはＪＩＳと書いているわけではないので、いや書いてあったとしても面倒なことには違いない。
コンポーネントを作る側は、使われ方を無視できない。考えられるさまざまな方法をクリアしておかないと、使用者は使わないただのゴミを作りことになる。
①キーボードからの入力、すなわちＩＭＥ経由への対応
②クリップボード経由への対応
③コンポーネント内部での文字列の挿入処理
以上３つの対応は必ず必要である。

TateEditを何から継承してつくるか

ユニコードを描画できても、TateEditコンポーネント実現までにはまだ遠い。表示する文字を得なければならない。まず、キーボードからの入力。
本当はEditから継承してコンポーネントを作るのが正当なのかも知れない。そうすると入力はもしかして楽に作れるような気がする。しかし、実際には表示だけを縦書きに変えるとしても、得なければならないのはユニコードなので、どちらにしても同じところにいきつくような思いがした。どうせなら、不明なことのすべてを掌握していくのが勉強になるのではないかという理由で、TCoustomControlからの継承に決めた。
「入力をともなうものはTCustomControlから」作るようだ。
だが、その前に、部品としてのユニコードを含む入力のモジュールを用意する必要がある。コンポーネントではない普通のアプリの方法でいろいろとトライしてみた。
①アスキーの文字列を得るのは、OnKeyDown、OnKeyOress、OnKeyUpなどのエベントで捉えられる。
②漢字の文字列を得るのは、基本的には同じ。入手したKeyコードを調べて漢字の第１桁目なら次の文字を得て、合成して漢字とする。
③TABなどのコントロールキー、編集キーも処理しなければならない。しかし、これも同じキーイベントで処理できる。
通常のシフトＪＩＳなら、これだけで済む。

キーイベントでの処理の問題

キーイベントでは、たやすくキーから入力された文字を取得できる。だが、ユニコードを得られそうにない。そもそも、キー入力時点では必要がないのかもしれない。
ユニコードを得るとしたら、ＩＭＥ辞書変換の結果か、コードによる入力だけだ。ここではシフトＪＩＳで文字を得て、それからWideString（ユニコード）にするということは考えられるが、これではせっかくのユニコードが「?」でしか取得できないのでだめ。
キーイベントの内部でやっていることは、WindowsのMessageのWM_KEYDOWNなどだ。漢字を得るのはＩＥＭ関係のメッセージである。
そこで、procedure FSubclassProc(var Message: TMessage); を用意して、キーイベント関係、特にＩＭＥまわりをモニタしてみることにした。
Messageの種別、wParam、lParamの内容を、Form上のMemoなどに表示して確認してみようというのである。

ＩＭＥからユニコードで漢字を得る

注目は、WM_IME_COMPOSITIONというMessage。
ここで、FTemp := GetIMECompString(GCS_COMPSTR); のようにして文字を得てみる。GCS_COMPSTRは引数だた、変換された文字が必要だというような意味のときに使用する。
GetIMECompStringのなかで、ImmGetCompositionStringWというAPIを使ってユニコードを得る。

```
//変換モード時にIMEから文字を得る
function GetIMECompString(Index: DWORD): widestring;
var
  hIM: HIMC;     // IMEのコンテキスト
  PC: PWideChar;
  Len: Integer;
begin
  hIM := ImmGetContext(Form1.Handle);
  Len := ImmGetCompositionStringW(hIM,Index,nil,0);
  GetMem(PC,Len*2+1);
  try
    FillChar(PC^,Len*2+1,#0);
    ImmGetCompositionStringW(hIM,Index,PC,Len);
    result := PC;
  finally
    FreeMem(PC);
  end;
end;
```

ＡＮＫ（半角文字）を得るにはどうするか

半角文字を得るのは意外と苦労。キーイベント内ならKeyがそのままアスキーコードだ。Messageでは、wParam、lParamに情報があるのは分るのだが、アスキーコードではなさそう。キーボードのキーＩＤなのである。物理的なキーの押下情報で、ShiftStateとあわせることで求めるアスキーコードになる。
デルファイのOnKeyPressで得られるKeyは、すでに処理されたものなのだ。
当然Windowsはその処理モジュールが備わっているはずだが、見つけられない。こんなものすぐ見つかると思っていたのだが。

仮想キーコードをアスキーコードに変換するというので、MapVirtualKeyがそれかなと思ったのだが、どうやら目的が違うようだ。
まあ、情報そのものとしてのキーＩＤとシフト情報があるわけだから自前で処理してもいいのだが。いろいろ試行錯誤して調べていると、
MapVirtualKeyを見ていく中で、GetKeyNameText、ToAsciiExというのをみつけた。何かふに落ちないままなのだが、これを使えばアスキーコードとコントロールコードが得られそうだと考えた。
このようにするのが確かなのかはいまだ不明。常識的にはもっともっと簡便にアスキーコードを得られるはずだ。

苦肉のMessageToAscii

CKey := MessageToAscii(Message,WKey); のようにして、MessageToAscii関数を使う。VF_***というのは、TateEditのために自前で宣言したもの。値はWkeyに設定される。通常のアスキー文字ならCkeyにResultで返る。
テンキーの処理（NumLockの状態を見て値を変える)、ファンクションキーなどの取得などたいていのキーを得られる。

```
//TMessageからAsciiコードにする
//Asciiとコントロールコードを返す
//Char:Result=0のときは、WKeyにControlCodeを返す
function MessageToAscii(var message: TMessage; var WKey: TfunkValue): Char;
var
  NombreTecla: array [0..100] of char;
  Handle     : THandle;
  TID        : integer;
  KbdState   : TKeyboardState;
  KbdLang    : HKL;
  ch         : array [0..1] of char;
  ds         : string;
begin
  ch := Char(0);
  Result := ch[0];
  WKey := VF_NONE;

  Handle := GetForegroundWindow;
  TID := GetWindowThreadProcessId(Handle,NIL);
  GetKeyboardState(KbdState);
  FShiftState := KeyboardStateToShiftState(KbdState);

  KbdLang := GetKeyboardLayout(TID);
  //文字列で仮想キーの名前を得る
  GetKeyNameText(Message.lParam,@NombreTecla,SizeOf(NombreTecla));

  //キーの押下の判断
  if ((Message.lParam shr 31) and 1)=1 then begin
    //上げた
  end else if ((Message.lParam shr 30) and 1)=1 then begin
    //再入力
  end else begin
    //押下
    if ToAsciiEx(Message.wParam,MapVirtualKey(Message.wParam,0),
      KbdState,@ch,0,KbdLang)>=1 then begin
```

```
ds := ch;
case Ord(ch[0]) of //Ctrl+KEYを判断
  $00: begin
    Wkey := VF_NONE;
  end;
  $01: begin
    WKey := VF_SELECTALL;
  end;                  //A
  $02: begin
    Wkey := VF_NONE;
  end;                  //B
  $03: begin
    WKey := VF_COPY;
  end;                  //C
  $04: begin
    Wkey := VF_NONE;
  end;                  //D
  $05: begin
    Wkey := VF_NONE;
  end;                  //E
  $06: begin
    Wkey := VF_NONE;
  end;                  //F
  $07: begin
    Wkey := VF_NONE;
  end;                  //G
  $08: begin
    WKey := VF_BACK;
  end;                  //H
  $09: begin
    WKey := VF_TAB;
  end;                  //I
  $0A: begin
    Wkey := VF_JISCODE;
  end;                  //J
  $0B: begin
    Wkey := VF_NONE;
  end;                  //K
  $0C: begin
    WKey := VF_CLEAR;
  end;                  //L
  $0D: begin
    WKey := VF_RETURN;
  end;                  //M
  $0E: begin
    Wkey := VF_NONE;
  end;                  //N
  $0F: begin
    Wkey := VF_NONE;
  end;                  //O
  $10: begin
    WKey := VF_DELETE;
  end;                  //P
```

```
    $11: begin
      Wkey := VF_NONE;
    end;                  //Q
    $12: begin
      WKey := VF_CHGREADONLY;
    end;                  //R
    $13: begin
      Wkey := VF_NONE;
    end;                  //S
    $14: begin
      Wkey := VF_NONE;
    end;                  //T
    $15: begin
      Wkey := VF_UNICODE;
    end;                  //U
    $16: begin
      WKey := VF_PASTE;
    end;                  //V
    $17: begin
      Wkey := VF_NONE;
    end;                  //W
    $18: begin
      WKey := VF_CUT;
    end;                  //X
    $19: begin
      WKey := VF_CHGVERTICAL;
    end;                  //Y
    $1A: begin
      WKey := VF_UNDO;
    end;                  //Z
    $1B: begin
      WKey := VF_ESCAPE;
    end;
    $1C: begin
      WKey := VF_RIGHT;
    end;
    $1D: begin
      WKey := VF_LEFT;
    end;
    $1E: begin
      WKey := VF_UP;
    end;
    $1F: begin
      WKey := VF_DOWN;
    end;
  end;
  if (length(ds)<2) and (ch[0]>=Char($7e)) then begin
    WKey := VF_NONE;
  end;
end else begin
  //-----------------------------------
  //Tab: to NextActiveTabNoControl
  //Alt,F11,SysReq,PrtSc,Fn: 取得不可
```

```
//F12: 注意：強制エラー
//----------------------------------
ds := string(NombreTecla);
if ds='' then begin
  ds := intToHex(Message.wParam,4);
  if ds='002D' then begin
    WKey := VF_INSERT;
  end else if ds='002E' then begin
    WKey := VF_DELETE;
  end else if ds='00F0' then begin
    Wkey := VF_NONE; // 'CapsLock';
  end;
end;

//文字列からの判断
if ds='Up' then begin
  WKey := VF_UP;
end else if ds='Down' then begin
  WKey := VF_DOWN;
end else if ds='Left' then begin
  WKey := VF_LEFT;
end else if ds='Right' then begin
  WKey := VF_RIGHT;
end else if ds='Delete' then begin
  WKey := VF_DELETE;
end else if ds='Insert' then begin
  WKey := VF_INSERT;
end else if ds='Ctrl' then begin
  Wkey := VF_NONE;
end else if ds='Shift' then begin
  Wkey := VF_NONE;
end else if ds='Right Ctrl' then begin
  Wkey := VF_NONE;
end else if ds='NumLock' then begin
  Wkey := VF_NONE;
end else if ds='Pause' then begin
  Wkey := VF_NONE;
end else if ds='Scroll Lock' then begin
  Wkey := VF_NONE;
end else if ds='Application' then begin
  Wkey := VF_NONE;
end else if ds='Left Windows' then begin
  Wkey := VF_NONE;
end else if ds='Right Windows' then begin
  Wkey := VF_NONE;
end;

if ds='F1' then begin
  Wkey := VF_NONE;
end else if ds='F2' then begin
  Wkey := VF_NONE;
end else if ds='F3' then begin
  Wkey := VF_NONE;
```

```
      end else if ds='F4' then begin
        Wkey := VF_NONE;
      end else if ds='F5' then begin
        Wkey := VF_NONE;
      end else if ds='F6' then begin
        Wkey := VF_NONE;
      end else if ds='F7' then begin
        Wkey := VF_NONE;
      end else if ds='F8' then begin
        Wkey := VF_NONE;
      end else if ds='F9' then begin
        Wkey := VF_NONE;
      end else if ds='F10' then begin
        Wkey := VF_NONE;
      end else if ds='F11' then begin
        Wkey := VF_NONE;
      end else if ds='F12' then begin
        Wkey := VF_NONE;
      end;

      //テンキーの無NumLock時のName変更
      if ds='Num 7' then begin
        WKey := VF_HOME;
      end else if ds='Num 8' then begin
        WKey := VF_UP;
      end else if ds='Num 9' then begin
        WKey := VF_PRIOR;
      end else if ds='Num 4' then begin
        WKey := VF_LEFT;
      end else if ds='Num 6' then begin
        WKey := VF_RIGHT;
      end else if ds='Num 1' then begin
        WKey := VF_END;
      end else if ds='Num 2' then begin
        WKey := VF_DOWN;
      end else if ds='Num 3' then begin
        WKey := VF_NEXT;
      end else if ds='Num 0' then begin
        WKey := VF_INSERT;
      end else if ds='Num Del' then begin
        WKey := VF_DELETE;
      end;
    end;
    Result := ch[0];
  end;
end;
```

ＡＮＫと漢字の合成で編集文字列にする

　これで、ようやくＡＮＫと漢字が取得できるようになった。最終的には双方を合成してひとつの編集文字列にしなければならない。後にそれを、各種編集キーの操作で処理することになる。

　キーボードからのキーをなまで処理しようとする場合に、編集のためのキーをも正確に得な

ければならない。そのために、リストのごとく冗長ではあるが編集で必要と思えるキーについて漏らさないようにしているのだ。

気になることのまとめ

１行といえども、文字を入力する処理をちょっとでも下層でするとなると、なかなか甘いものではない。入力インターベースの周辺の知識とEditの動きをまんべんなく調べる周到さが求められる。
ＩＭＥは今だ良くわからないというのが感想だ。

２.４.縦書きエディットコンポーネントでの文字描画処理と扁平処理

文字描画処理をするにあたって

デルファイでの文字の描画自身については先にもちょっと触れたとおりである。TateEditの描画を考えると、いくつかの疑問が生じる。
Editにはないいくつかの機能をつけてしまったためである。

①縦書きと横書き
②Alignmentと扁平処理
③背景の透明化
④入力モードと閲覧モード

なるべく、矛盾がおこらないように注意してみたつもりだ。

縦書きをする方法

横に文字を描画するのはデルファイでは簡単なことは先にふれた。
特に表示するCanvasがわかっていれば、TextOutというメソッドを使えるので、これで不自由することはない。
だが、縦書きとなるとそのようなメソッドはない。フォントは、ご存知のようにフォント名の最初に＠が付くのは縦書き用フォントとして用意されている。これをそのままフォントに指定してTextOutすればいいのかかなどと思って実行してみても、９０度傾いた状態で横に表示されるだけである。
文字列を上から下に向かって表示したいわけなのでこれではだめだ。
結論として、LogFontというのを通じて縦書き用フォントをさらに９０度傾け、印字方向を指定することによって普通に見えるようなる。

```
//sample1:LogFontで縦書き
procedure TForm1.Button1Click(Sender: TObject);
var
  OrgFont,MyFont: THandle;
  LogFont: TLogFont;
  fName,strs: string;
  fHeight: integer;
  x,y,fw,fh: integer;
begin
  //------------------------
  //パラメータ変数
  //------------------------
  fName := '@ＭＳ 明朝';                    //@で縦書き
  fHeight := 40;
  x := 50;
```

```
  y := 50;
  strs := '漢字縦書き1A';
  fw := 40;
  fh := 300;

  //-------------------------
  //フォントの設定前処理
  //-------------------------
  with LogFont do begin
    StrCopy(lfFaceName,PChar(fName));            //フォント名
    lfCharSet := SHIFTJIS_CHARSET;               //CHARSET
    lfHeight := fHeight;                         //１文字の高さ
    lfEscapement  := 2700;                       //確度（単位1/10度）
    lfOrientation := 2700;                       //      （反時計回り）
    lfWeight := FW_NORMAL;                       //太字レベル指定
    lfItalic := 0;                               //斜体する・しない
    lfUnderline := 0;                            //下線する・しない
    lfStrikeOut := 0;                            //消線する・しない
    lfOutPrecision  := OUT_DEFAULT_PRECIS;       //
    lfClipPrecision := CLIP_DEFAULT_PRECIS;      //
    lfQuality := DEFAULT_QUALITY;                //
    lfPitchAndFamily := DEFAULT_PITCH or FF_DONTCARE; //
  end;
  MyFont := CreateFontIndirect(LogFont);         //Create
  OrgFont := SelectObject(Canvas.Handle,MyFont);     //保存と設定

  //--------------------------
  //フォント使用（文字列表示）
  //--------------------------
  Canvas.Brush.Style := bsClear;
  Canvas.Rectangle(x,y,x+fw,y+fh);
  SetTextAlign(Canvas.Handle,VTA_LEFT or VTA_TOP);   //左上を表示原点に
  Canvas.TextOut(x,y,strs);                      //表示

  //--------------------------
  //フォントの設定後処理
  //--------------------------
  SelectObject(Canvas.Handle,OrgFont);           //復帰
  DeleteObject(MyFont);                          //Delete
end;
```

LogFontを使っての注意点など

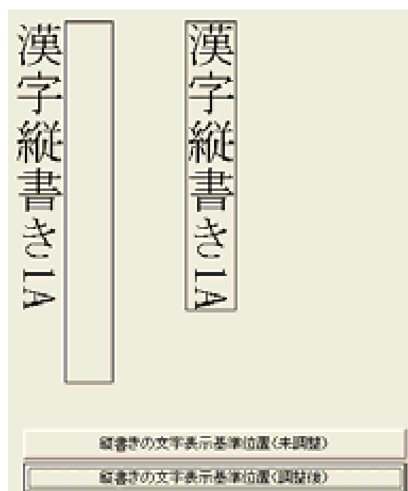
　デルファイのTextOutをそのまま使って縦書きができることがわかった。LogFont自身はデルファイで用意しているのではなく、Windowsが内部で用意していて、それを横から利用するような方法となる。デルファイはこのような技が自前のように使えるので便利だ。

　ただ、その仕掛けはというよりも、WindowsのほうはＣ言語を前提にされていることから、フォント名を設定するくだりなどは、Stringへの代入のようなわけにはいかない。そのために、strcopyを使って設定している。Windowsの仕掛けを使うときはこのようにするのだと憶えてしまえばいいわけである。

　デルファイのbeginとend、createとfreeのように、Ｃでも設定のためのオブジェクトを作成したら絶対に開放するというのは動かせないルールだ。これも、ウインドウズの仕掛けを利用するときに忘れてならないこと。

コメントに表示原点というのがある。lfEscapementで1/10度づつ反時計回りで指定が可能。つまり、微妙な斜めの方向への描画もできるということだが回転には中心がある。当然確度を与えると、意図した描画原点x,yと実際の表示に相違ができたように見える。

サンプルでは、縦書きと横書きで同じx,yを指定したたとしたときに、１文字目の漢字がちょうど重なるように指定している。x,yが１文字の仮想枠の左上で一致する。



文字のHeightの指定

LogFontでは、文字のサイズを通常使用しているポイントではなく、表示するデバイスの解像度に応じたドット数で指定する。

ポイントは、デバイスを考慮しない共通のサイズだが、ドット数での指定となるとLogFont.lfHeight := fh;などとする場合に、fhを自分で決めなければならない。

１ポイントは1/72インチです。10ポイントといった場合は、10/72インチの文字の高さということになる。

画面に表示しよう、つまりディスプレイを描画デバイスとしたときは、ディスプレイの解像度がどうなっているのかを知る必要がある。

```
//x pt(10分の1Point 11.5pt等)→画面dot
function PointToScreenDot(x: Extended): integer;
var
  DotFromScreen: integer;
begin
  if x<=1 then begin
    Result := 1;
    exit;
  end;
  DotFromScreen   := Screen.PixelsPerInch;
  Result := RoundOff(x*DotFromScreen/72.0);
  if Result<=1 then begin
    Result := 1;
  end;
end;
```

ユニコードの印字のための準備

いよいよユニコードの描画だが、結論として TextOutWを使う。デルファイの用意しているものではなく、Windowsの資産を利用するもの。

TextOutW(Canvas.Handle,x,y,PWideChar(wstrs),Length(wstrs));
出力デバイスのCanvas.Handleをパラメータで与える。Wが付くのはWideString対応、すなわちユニコード対応ということ。
別に見かけは、Canvas.TextOutと変わらない。このように、ちょっとめんどうな手続きを経なければユニコードは表示できないということだけ理解しておく必要がある。
Windows(98以降)や、デルファイは内部では「ユニコードを使っている」などといわれる。Windowsはこの基本ソフトと一体ともいえるオフィス製品でそれなりの実現をみている。しかし、デルファイの場合は、少なくともV5(V7までしか使ったことがない)までの仕様は「あくまでもＪＩＳコードの範囲の文字について内部でユニコード文字に変換して処理している」というのが正確で、ユニコードが自然と使用できるというのとは違うようだ。

```
//sample3:描画エリア計算とユニコード表示
procedure TForm1.Button2Click(Sender: TObject);
var
  OrgFont,MyFont: THandle;
  LogFont: TLogFont;
  fName,strs: string;
  wstrs,ws1: WideString;
  fHeight: integer;
  x,y,fw,fh: integer;
  i,sw,sh,wslen: integer;
  wsz: TSize;
begin
  //------------------------
  //パラメータ変数
  //------------------------
  fName := '@ＭＳ 明朝';                    //@で縦書き
  fHeight := 40;
  x := 150;
  y := 50;
  wstrs := '漢字縦書き1A';

  //------------------------
  //フォントの設定前処理
  //------------------------
  with LogFont do begin
    StrCopy(lfFaceName,PChar(fName));           //フォント名
    lfCharSet := SHIFTJIS_CHARSET;              //CHARSET
    lfHeight := fHeight;                     //１文字の高さ
    lfEscapement  := 2700;                      //確度（単位1/10度）
    lfOrientation := 2700;                    //    （反時計回り）
    lfWeight := FW_NORMAL;                       //太字レベル指定
    lfItalic := 0;                          //斜体する・しない
    lfUnderline := 0;                         //下線する・しない
    lfStrikeOut := 0;                         //消線する・しない
    lfOutPrecision  := OUT_DEFAULT_PRECIS;          //
    lfClipPrecision := CLIP_DEFAULT_PRECIS;        //
    lfQuality := DEFAULT_QUALITY;                //
    lfPitchAndFamily := DEFAULT_PITCH or FF_DONTCARE; //
  end;
  MyFont := CreateFontIndirect(LogFont);           //Create
  OrgFont := SelectObject(Canvas.Handle,MyFont);     //保存と設定
```

```
  //--------------------------
  //描画エリアの取得
  //--------------------------
  wslen := length(wstrs);
  sw := 0;
  sh := 0;

  for i:=1 to wslen do begin
    ws1 := wstrs[i];    //１文字を！
    GetTextExtentPoint32W(Canvas.Handle,PWideChar(ws1),1,wsz);
    sw := sw+wsz.cx;    //文字幅
  end;
  sh := wsz.cy;         //文字高さ
  fh := sw;
  fw := sh;

  //--------------------------
  //フォント使用（文字列表示）
  //--------------------------
  Canvas.Brush.Style := bsClear;
  Canvas.Rectangle(x,y,x+fw,y+fh);
  SetTextAlign(Canvas.Handle,VTA_LEFT or VTA_TOP);    //左上を表示原点に
  TextOutW(Canvas.Handle,x,y,PWideChar(wstrs),Length(wstrs));
  //--------------------------
  //フォントの設定後処理
  //--------------------------
  SelectObject(Canvas.Handle,OrgFont);              //復帰
  DeleteObject(MyFont);                             //Delete
end;
```

文字の表示オプション（プロパティのAlignment）

プロパティのAlignmentでは、alLeft（左寄せ）、alCenter（中央合わせ）、alRight（右寄せ）、alJustify（均等合わせ）を用意している。

文字を表示するときに、通常は日本語なので左から右へ行う。文字間隔は専門用語で「ベタ」「ベタ詰め」といって、文字と文字の間隔は空けない。つまり、フォントの幅分だけ表示基点を進める。

本文の文字間は通常ベタが原則だ。これは、日本語の組版という世界では基本中の基本になる。一定のルールにもとづいて「気持ち」狭めたり、広げたりの処理をすることがあるが、それでも限度はわずかとおもってよい。特に行間との関係を考慮したときに、文字間が狭め得る値以上に広い場合、間が抜けたイメージになるもの。きちんとした理由がない限り、文字間「ベタ」原則を守るようにする。

「左寄せ」は、左から右へ。「右寄せ」は、右にそろえるということだ。電卓の数字入力のイメージ。「中央合わせ」は、エリアの中央に文字が表示される。

「均等合わせ」というのは、エリアに均等に文字を配置すること。この場合に限って、文字間が広がる。

考え方の概念は下記のようになる。

Editの表示エリア、すなわちwidthが300(単位ドット)とする。

文字列は、sample2では、'漢字縦書き1A'でした。文字の高さを40として計算してみると、文字幅に要するサイズは240×42とでた。文字数は7文字。ユニコードの表示を前提にしているので、特に断りがない限り、WideStringのLengthをいう。目で数える文字数と同じ。

・左寄せの場合：300-(240)=60
　文字の右側が、60空く。

・右寄せの場合：
　文字の左側が、60空く。
・中央合わせの場合：60÷2=30
　文字の左右が、30づつ空く。
・均等合わせの場合：60÷(7-1=6)=10
　文字間がフォント幅＋10の値になる。
　均等合わせの場合の留意点がある。
　エリア幅が、303と半端な場合である。
・均等合わせの場合：63÷(6)=10、余り3
　７文字中最初の３文字の文字間がフォント幅＋11とし、
　残りの３文字をフォント幅+10とする。
　余り分を１ドットづつ追加して処理する。

実際の印字では、左側の空きといった箇所が、印字開始位置になるである。したがって、右寄せの場合は60､中央合わせの場合は30がオフセット値として､x+offsetの位置から表示する。

以上の概念は、横書きで説明したが、縦書きでもまったく同じである。

ちなみに、SetTextCharacterExtra(W)というAPIでは、文字の間隔を任意に設定できるのだが、余りの処理はできないので使うことはできない。

文字の扁平

表示エリアに納まりきらない長い文字列の処理を考える。

文字列が長く、エリアの幅が300で、文字の計算幅がそれを超えた360とかだった場合である。

Editなどでは、左にスクロールして文字列の右端がエリアの右端に必ず表示されるような動作になる。

TateEditでは、当初入力を受け付けるモード、すなわちReadOnlyがFalseならEditと同じ扱いにしようと考えていた。しかし、ワードやイラストレータなどでの編集の詳細を見てみると、Editとは異なる。表示されている文字の状態で入力を受け付けている。それなら、同じようにやれるだけＷＹＳＷＹＧに似せてみようと試みた。

それは、幅360の文字列を幅300のエリアに横幅を縮めて、その状態のまま入力をさせてみようということ。

扁平というのは、上下と左右の比率を変えて、どちらかを伸縮させるもの。ここでは、上下の文字の高さをそのままにして、左右幅を表示エリア幅まで縮小してみる。

入力のことは後の説明にして、まずは扁平表示。

表示する前に、通常の正体（せいたい：扁平をしないそのままの状態のこと）で必要な表示エリアを計算する必要がある。

幅360とでたとする。300÷360＝0.83。83％に横を縮小すればよいことになる。

縦100％、横83％で、LogFontのlfWidthとlfHeightに値を指定すればいいわけだ。

LogFontにこのように指定して、再度表示エリアの計算をする。１文字の平均的なサイズを正しく指定したからといって、文字列全体の長さがぴったりと目的のサイズとなるわけではない。360を前後にやや長くなるか、短くなるかになる。

目的の描画エリアのサイズと計算した値を比較し、差分を文字間で調整するのである。

例としての文字列は、'漢字扁平の縦書き1A'。文字数は10。再計算の結果は306、つまり6だけ超えている。

補正値１は、(300-306) div (10-1)=0;

補正値２は、(300-306) mod (10-1)=-6;

超えていれば、補正値２はマイナスとなり、文字間を1づつ６回詰める。少なければ、補正値はプラスで字間を1つづ値の回数だけ広げることになる。

扁平横書きのしかた

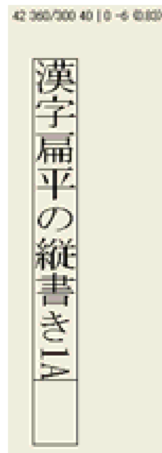
この例では、横に83%の扁平をかけている。

```
//sample4:扁平横書き
procedure TForm1.Button1Click(Sender: TObject);
var
  OrgFont,MyFont: THandle;
  LogFont: TLogFont;
  fName: string;
  wstrs,ws1: WideString;
  fHeight: integer;
  x,y,fw,fh: integer;
  i,sw,sh,wslen: integer;
  wsz: TSize;
  Henpei: Extended;
  ereaH,fntW,hoseipt1,hoseipt2: integer;
begin
  //-------------------------
  //パラメータ変数
  //-------------------------
  fName := 'ＭＳ 明朝';                    //@で縦書き
  fHeight := 40;
  x := 150;
  y := 50;
  wstrs := '漢字扁平の縦書き1A';
  ereaH := 300;                            //表示エリア最大

  //-------------------------
  //フォントの設定前処理
  //-------------------------
  with LogFont do begin
    StrCopy(lfFaceName,PChar(fName));          //フォント名
    lfCharSet := SHIFTJIS_CHARSET;             //CHARSET
    lfHeight := fHeight;                       //１文字の高さ
    lfEscapement  := 0;                        //確度（単位1/10度）
    lfOrientation := 0;                        //      （反時計回り）
    lfWeight := FW_NORMAL;                     //太字レベル指定
    lfItalic := 0;                             //斜体する・しない
    lfUnderline := 0;                          //下線する・しない
    lfStrikeOut := 0;                          //消線する・しない
    lfOutPrecision  := OUT_DEFAULT_PRECIS;     //
    lfClipPrecision := CLIP_DEFAULT_PRECIS;    //
    lfQuality := DEFAULT_QUALITY;              //
    lfPitchAndFamily := DEFAULT_PITCH or FF_DONTCARE; //
  end;
  MyFont := CreateFontIndirect(LogFont);       //Create
  OrgFont := SelectObject(Canvas.Handle,MyFont);    //保存と設定

  //--------------------------
  //描画エリアの取得
  //--------------------------
  wslen := length(wstrs);
```

```
sw := 0;
sh := 0;

for i:=1 to wslen do begin
  ws1 := wstrs[i];    //１文字を！
  GetTextExtentPoint32W(Canvas.Handle,PWideChar(ws1),1,wsz);
  sw := sw+wsz.cx;
end;
sh := wsz.cy;
fw := sw;             //文字幅
fh := sh;             //文字高さ

hoseipt1 := 0;        //文字間補正値1
hoseipt2 := 0;        //文字間補正値2

//--------------------------
//表示エリアを超える場合
//扁平の計算
//--------------------------
if sw>ereaH then begin
  fntW := Canvas.textwidth('■');
  henpei := ereaH/sw;
  LogFont.lfWidth := round(fntW*henpei/2);
  MyFont := CreateFontIndirect(LogFont);              //Create
  OrgFont := SelectObject(Canvas.Handle,MyFont);      //保存と設定

  sw := 0;
  for i:=1 to wslen do begin
    ws1 := wstrs[i];    //１文字を！
    GetTextExtentPoint32W(Canvas.Handle,PWideChar(ws1),1,wsz);
    sw := sw+wsz.cx;    //文字幅
  end;

  hoseipt1 := (ereaH-sw) div (wslen-1);              //ピッチ補正１
  hoseipt2 := (ereaH-sw) mod (wslen-1);              //ピッチ補正２
end;

//--------------------------
//フォント使用（文字列表示）
//--------------------------
Canvas.Brush.Style := bsClear;
Canvas.Rectangle(x,y,x+fw,y+fh);
Canvas.Rectangle(x,y,x+ereaH,y+fh);

SetTextAlign(Canvas.Handle,VTA_TOP);
for i:=1 to wslen do begin
  ws1 := wstrs[i];    //１文字を！
  TextOutW(Canvas.Handle,x,y,PWideChar(ws1),1);
  GetTextExtentPoint32W(Canvas.Handle,PWideChar(ws1),1,wsz);
  x := x+wsz.cx+hoseipt1;
  if i<=abs(hoseipt2) then begin
    if hoseipt2<0 then begin
      x := x-1;
```

```
    end else begin
      x := x+1;
    end;
  end;
end;

//--------------------------
//フォントの設定後処理
//--------------------------
SelectObject(Canvas.Handle,OrgFont);                //復帰
DeleteObject(MyFont);                               //Delete
end;
```

扁平縦書きのしかた



縦書きの場合も考え方は同じである。
プログラム上での違いをコメントで表示している。
①フォントを縦書き用のフォントにする
②270ど傾ける
③表示原点を文字の左上にする
④１文字ごとの位置を進める
⑤確認用の矩形表示でプログラムには関係ない
　広いほうが、正体で計算したエリア。
　狭いほうが指定のエリア

```
//sample5:扁平縦書き
procedure TForm1.Button2Click(Sender: TObject);
var
  OrgFont,MyFont: THandle;
  LogFont: TLogFont;
  fName: string;
  wstrs,ws1: WideString;
  fHeight: integer;
  x,y,fw,fh: integer;
  i,sw,sh,wslen: integer;
  wsz: TSize;
  Henpei: Extended;
  ereaH,fntW,hoseipt1,hoseipt2: integer;
begin
  //-------------------------
  //パラメータ変数
  //-------------------------
  fName := '@ＭＳ 明朝';                        //①@で縦書き
  fHeight := 40;
  x := 150;
  y := 50;
  wstrs := '漢字扁平の縦書き1A';
  ereaH := 300;                                //表示エリア最大
```

```
//------------------------
//フォントの設定前処理
//------------------------
with LogFont do begin
  StrCopy(lfFaceName,PChar(fName));                //フォント名
  lfCharSet := SHIFTJIS_CHARSET;                   //CHARSET
  lfHeight := fHeight;                             //１文字の高さ
  lfEscapement  := 2700;                           //②1確度（単位1/10度）
  lfOrientation := 2700;                           //②2      （反時計回り）
  lfWeight := FW_NORMAL;                           //太字レベル指定
  lfItalic := 0;                                   //斜体する・しない
  lfUnderline := 0;                                //下線する・しない
  lfStrikeOut := 0;                                //消線する・しない
  lfOutPrecision  := OUT_DEFAULT_PRECIS;           //
  lfClipPrecision := CLIP_DEFAULT_PRECIS;          //
  lfQuality := DEFAULT_QUALITY;                    //
  lfPitchAndFamily := DEFAULT_PITCH or FF_DONTCARE; //
end;
MyFont := CreateFontIndirect(LogFont);             //Create
OrgFont := SelectObject(Canvas.Handle,MyFont);     //保存と設定

//-------------------------
//描画エリアの取得
//-------------------------
wslen := length(wstrs);
sw := 0;
sh := 0;

for i:=1 to wslen do begin
  ws1 := wstrs[i];   //１文字を！
  GetTextExtentPoint32W(Canvas.Handle,PWideChar(ws1),1,wsz);
  sw := sw+wsz.cx;
end;
sh := wsz.cy;
fw := sw;          //文字幅
fh := sh;          //文字高さ

hoseipt1 := 0;       //文字間補正値1
hoseipt2 := 0;       //文字間補正値2

//-------------------------
//表示エリアを超える場合
//扁平の計算
//-------------------------
if sw>ereaH then begin
  fntW := Canvas.textwidth('■');
  henpei := ereaH/sw;
  LogFont.lfWidth := round(fntW*henpei/2);
  MyFont := CreateFontIndirect(LogFont);           //Create
  OrgFont := SelectObject(Canvas.Handle,MyFont);   //保存と設定

  sw := 0;
```

```
    for i:=1 to wslen do begin
      ws1 := wstrs[i];    //１文字を！
      GetTextExtentPoint32W(Canvas.Handle,PWideChar(ws1),1,wsz);
      sw := sw+wsz.cx;    //文字幅
    end;

    hoseipt1 := (ereaH-sw) div (wslen-1);              //ピッチ補正１
    hoseipt2 := (ereaH-sw) mod (wslen-1);              //ピッチ補正２
  end;

  //--------------------------
  //フォント使用（文字列表示）
  //--------------------------
  Canvas.Brush.Style := bsClear;
  Canvas.Rectangle(x,y,x+fh,y+fw);                     //⑤1
  Canvas.Rectangle(x,y,x+fh,y+ereaH);                  //⑤2

  SetTextAlign(Canvas.Handle,VTA_LEFT or VTA_TOP);     //③表示原点
  for i:=1 to wslen do begin
    ws1 := wstrs[i];    //１文字を！
    TextOutW(Canvas.Handle,x,y,PWideChar(ws1),1);
    GetTextExtentPoint32W(Canvas.Handle,PWideChar(ws1),1,wsz);
    y := y+wsz.cx+hoseipt1;                            //④1
    if i<=abs(hoseipt2) then begin
      if hoseipt2<0 then begin
        y := y-1;                                      //④2
      end else begin
        y := y+1;                                      //④3
      end;
    end;
  end;

  //--------------------------
  //フォントの設定後処理
  //--------------------------
  SelectObject(Canvas.Handle,OrgFont);                 //復帰
  DeleteObject(MyFont);                                //Delete
end;
```

JustifyExのOffsetExはどう使うか

機能そのものはさておいて、こんなことをプロパティに入れるのはいかがなものか、と思われることだろう。それは、ともかく、この説明をつぎにする。

縦書きの宛名印刷の氏名の箇所を想定してみる。日本人の氏名は姓が２文字、名が２文字は統計的に多いほうである。しかし、姓１文字から４文字程度、名が１文字から４文字程度は想定しなければならない。

日本人でない場合はこんな前提はすべて飛んでしまう。ファーストネームとファミリーネームをあわせると３０文字に及ぶものもある、という具合だ。

日本人に限っても、２文字から８文字程度の幅があり、いずれの場合でもそれなりに自然に見える方法はないものだろうか、というのがテーマ。

好みによって姓と名のあいだにブランクスペースをいれ、最後に敬称を追加する。

image02は、人名２文字から15文字の例です。ハガキは１件１枚なので比較することはないのだが、それにしても短いののは不自然だし、長いのは文字の大きさによってはハガキからは

み出てしまいそうである。
image03に注目。多く出現すると思える７文字以内については「均等Justify」をしている。ここまでは描画してもいいという最大が③の線だとする。これを超えるものはすべて扁平をかけて左右（上下）そろいにしている。①と③のあいだの長さのものは正体のまま左（上）寄せで表示している。
AlignmentをJyustifyExとして、OffsetExに①の値を設定する。こうすることで、多少長い文字列でも、短い文字列でも、最適と思える印字を実現できるのである。

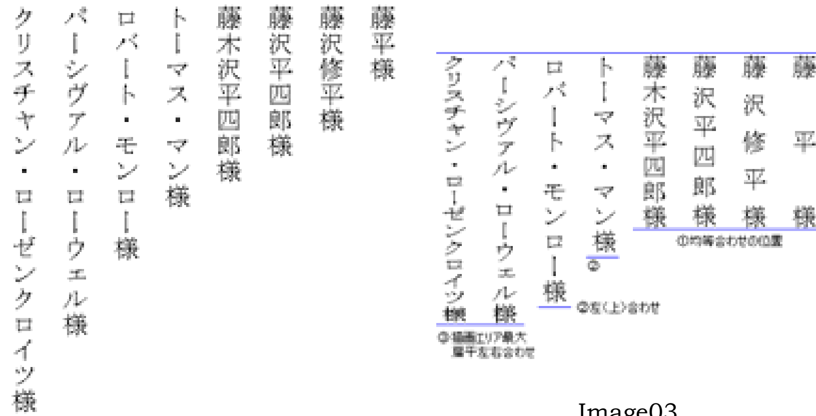
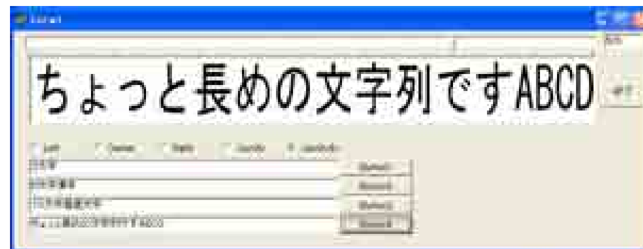
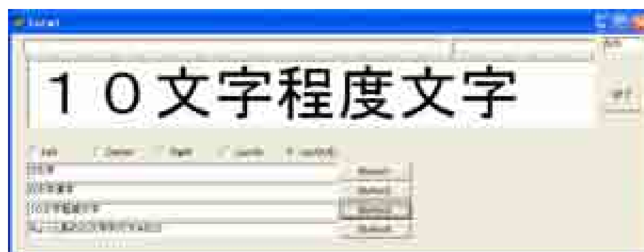


Image03

Image02



上のImageは、OffetExとJustifyExを任意に変更して機能を確認するためのツールの画面だ。
指定の位置OffsetExを境に、それ以内なら均等をおこない、エリアの範囲内なら正体で左寄せ、エリアを越えたときには扁平をするということが理解できる。



文字列描画エリアの計算

TextOutでは、これから描画しようとしたときの描画エリアのサイズは簡単にわかる。Canvas.TextWidthとCanvas.Textheightである。

漢字は原則として正方形なので、計算で得られるのだが、プロポーショナルのひらがな、カタカナが入ると複雑である。欧文フォントは原則としてプロポーショナルである。

コンポーネントを自作し文字を表示する場合には、Paintのなかで文字を描画するが、エリアに注意をはらう必要がある。はみ出さないようにしなければならない。はみ出る場合は扁平をかける、中央に表示する、右寄せにする、などの処理をする場合はどうしても、実際に表示する前に、正確なエリアを得なけばならないい。

問題は、扱う文字がユニコードの場合である。デルファイ（日本語）の提供している範疇はＪＩＳなので、ユニコードを扱うときにはさまざなな工夫が必要になる。

まして､文字に扁平をかけるとか斜体にするとかの装飾をした場合の変化等も要注意である。斜め方向に描画する場合はさらに角度に応じた計算式を自前で用意する。

ここでは直角方向だけを念頭に起く。さまざまな方法があるようだが、正確な値を得られたのは、下記の方法だけだった。

一般的にはDrawText(W)という文字描画APIにDT_CALCRECTを与えて実行すると実際の描画はされずにエリアの値が計算されて返える。しかし、ユニコードに対応しているのかというとどうも正しい計算はされないようなのだ。

```
//sample2:描画エリアの計算
var
  wsz: TSize;
  ws1 : WideChar;
  wslen: integer;
  wstrs: widestring;
  i: integer;
  sw,sh: integer;
begin
  //
  //sample1
  // (下記の実行前にsample1で文字の大きさ等を設定する)
  //

  wstrs := '漢字縦書き1A';  //ユニコード文字も可能
  wslen := length(wslen);
  sw := 0;
  sh := 0;

  for i:=1 to wslen do begin
    ws1 := wstrs[i];    //１文字を！
    GetTextExtentPoint32W(Canvas.Handle,PWideChar(ws1),1,wsz);
    sw := sw+wsz.cx;    //文字幅
  end;
```

```
    sh := wsz.cy;          //文字高さ
  end;
```

エリアが得られれば、sample1で、直接代入した fw := 40;と、 fh := 300;の正確な値を事前に得て、設定することができる。

太字、斜体、下線、着色

TateEditではこのことには深入りしなかった。なりゆきまかせとした。
Editでは特に何かをしているようではない。厳密には斜体や太字にすると正確な描画エリアの計算は誤差が生じる。しかし、現時点ではそのような使われ方は思いつかなかったので、現状のままとした。将来的に影響が大きい場合に再考することにする。

気になることのまとめ

文字列の描画は、プログラミングでは避けて通れない。プリンターにたいして任意の何かを印刷する処理をする場合には絶対に必要な知識といえる。ただ、デルファイでは出力デバイスのCanvasにたいする描画であるという点で共通の方式であるために、画面への描画の知識とノウハウがすべて運用できる。
文字列の描画についてはここで扱った問題を理解すればおおかた解決できる。

２.５.縦書きエディットコンポーネントでの編集処理

ＩＭＥの複雑なふるまい

ＩＭＥは意外に奥が深いもの。漢字入力の途中というのは、入力中、変換中、候補からの選択中、次入力でデフォルトの変換決定、候補を選んで決定などの状態がある。
ＡＮＫはまだしも、漢字の場合は「仮決定状態」で次の入力の動作をしたときに、仮決定が明らかな決定に変わり、ここで編集文字列をいったん決定して、入力の動作の処理に移っていく。
そして、次の入力のキーが押された場合に、その文字や変換候補文字の表示位置をもカーソルとともに進める必要がある。
「決定されている文字列」の現在のカーソル位置に対して、ＡＮＫあるいは漢字で明らかに「決定された時点」で、「取得した文字列」を挿入し、カーソル位置を進める。これは、入力モードが「挿入」のきとで、「上書き」の場合には、得た文字列の長さ分のカーソル以降の既存の文字列をいったん削除してから挿入するという動きになる。

入力中の編集処理

入力中の文字列については、編集ができなければならない。
TEditの編集をいろいろと見てみると、けっこう複雑なことをしている。
まず、フォーカスを受けたときに、プロパティのAutoSelectにもとづいて、全選択状態で入力編集処理を開始するかが決まる。
カーソルの移動は矢印だ。TateEditでは、縦書きモードか横書きかで、上下矢印、左右矢印を切り替えてる。
矢印は、シフトキーを押していると、１文字づつの文字列選択数に変わる。コントロールキーを押していると、右端か左端にカーソル移動する。
ＢＳとＤＥＬ、挿入モード上書きモード、ＴＡＢでのコントロールの移動がサポートされている。
おもしろいのは、右クリックで「標準？」的な編集項目がポップアップで表示されること。ここで、「右から左に読む」「Unicode制御文字の表示」「Unicode制御文字の挿入」というのが

ある。
「右から左に読む」というのは、右寄せの電卓入力のような動きをするのだが、いったい、どう使うのだろうか。
「Unicode制御文字の…」の項目はグレイになっていて不明。
また、「元に戻す」がサポートされている。これは、ぜひともTateEditでも実現したいのだが、ちから及ばず実装方法がわからなかった。
「切り取り」「コピー」「貼り付け」「削除」「全て選択」は編集の基本機能。使い方としてはメニューを出すよりもショートカットを使うケースが多いように思われる。
「ＩＭＥを開く／閉じる」「再変換」はＩＭＥの操作だが、前者はキー操作でするのが普通かなと思う。再変換は便利だ。ワープロを使っているときなどは、ＩＭＥのバージョンにもよるが、再変換ができて、それはそれで便利そう。

ＩＭＥ経由の漢字とＡＮＫとコントロールキーの分離

具体的に編集処理をコーディングするには、まずコンポーネントに入ってくるキーを大きくＩＭＥ経由の漢字とＡＮＫとコントロールキーに分ける。
描画用（TateEditで得る文字列）の「入力された文字列」は自動的に得られるわけではない。漢字入力モードとＡＮＫのモードは基本的には別で、双方のモードから必要な情報を別々に得る。得た情報を文字列にするものとコントロールコードにふりわける。「決定」のタイミングで「決定済みの既存の文字列」と「未決定の漢字文字列」と「ＡＮＫ」を結合する。
漢字モードでのコントロールコードはＩＭＥ内での編集が主で「決定」のタイミングをどう得るかが微妙な問題なのである。先にも触れたように「仮決定」状態で新たな入力が始まると無言で「仮決定」は「決定」になる。同時に新たな「未決定」が始まる、という具合だ。
ＩＭＥからのメッセージをとらまえて連続的に処理しなければならない。
ＡＮＫのモードでは、コントロールキーは取りやすい。押されているキーから判断する。
コントロールキーであれば、1キーごとに編集処理をして画面に反映させていくのである。

ユニコード文字のカット＆ペースト

文字入力では、キーボードからだけでなく、ＩＭＥパッドからの入力も受け付けなければならない。しかし、これは、ＩＭＥのキー入力処理を経てなされるようだ。そのために、ＩＭＥ入力の箇所が完成していればそのまま何もしなくても入力ができた。
もう一つは、クリップボード経由の入力、つまりカット＆ペースト。
カット＆ペーストは大変便利な機能である。しかし、どのようにすればWideStringのクリップボード操作が実現できるのか。
シフトＪＩＳだけを扱っているときには気にすることはほとんどなく、便利さに甘えていたのだが、TateEditには欠かせない機能なのでどうしても実現する必要がある。
クリップボード内でユニコードを保持しているのはCF_UNICODETEXTの型。この型でバッファにセットし、この型があればテキスト文字を取得できることがわかった。試行錯誤して、下記のようなモジュールを作成して使った。

```
//uses ClipBrd
procedure Clipboard_SetBuffer(Format: Word; var Buffer; Size: Integer);
var
  Data: THandle;
  DataPtr: Pointer;
begin
  with Clipboard do begin
    Open;
    try
      Data := GlobalAlloc(GMEM_MOVEABLE+GMEM_DDESHARE,Size);
      try
        DataPtr := GlobalLock(Data);
```

```
      try
        Move(Buffer,DataPtr^,Size);
        Clear;
        SetClipboardData(Format,Data);
      finally
        GlobalUnlock(Data);
      end;
    except
      GlobalFree(Data);
      raise;
    end;
  finally
    Close;
  end;
 end;
end;
//Unicodeのセット
procedure ClipBoard_SetAsWideTex(const textW: WideString);
begin
 Clipboard.Open;
 try
   Clipboard.AsText := textW;
   Clipboard_SetBuffer(CF_UNICODETEXT,
     PWideChar(textW)^,(Length(textW)+1)*SizeOf(WideChar));
 finally
   Clipboard.Close;
 end;
end;
//Unicodeの取得
function ClipBoard_GetAsWideTex(): WideString;
var
 Data: THandle;
begin
 with Clipboard do begin
   Open;
   Data := GetClipboardData(CF_UNICODETEXT);
   try
     if Data<>0 then begin
       Result := PWideChar(GlobalLock(Data));
     end else begin
       Result := '';
     end;
   finally
     if Data<>0 then begin
       GlobalUnlock(Data);
     end;
     Close;
   end;
   if (Data=0) or (Result='') then begin
     Result := Clipboard.AsText;
   end;
 end;
end;
```

ドラッグ＆ドロップ

TateEditではあきらめた。HEditorを作成された本田さんの実装の説明をみると可能なのですが、いわゆるMemoと違いTateEditは１行の入力に特化しているのでパスした。

気になることのまとめ

1文字単位での編集などの処理をコーディングするのはつらい。横書きであればデルファイ標準で十分すぎるほど提供されているので、このような必要性が基本的にないのだ。TateEditゆえのテーマなのである。

編集処理自身についてはコード例は掲載していないがソースコードをみれば参考になるかもしれない。

2.5.PmUnicodeViewer ツール

特徴と機能

PmUnicodeViewerdツールは、単にUnicodeを調べるものだが、プログラミングに関連していくつか説明してく。

デルファイでフォント名をリストアップする一般的方法は２つ。ひとつは、Screen.Fonts。もうひとつはAPIを使って、列挙する方法。これらは同じものを持ってきているものではないようだ。後者の方が多く持ってきているようだ。

このツールでは、フォントの属性を調べて、TrueTypeか、固定幅か、縦書き用か、SHIFTJISかをチェックしている。

コード表のマトリックスは、StringGridを使っている。問題は、セルにテキストとして文字を表示するときに、Unicodeは表示できないことだ。そこで、OnDrawを使い、この中で自前で描画している。フォントのいわゆる仮想の縦横サイズがそれぞれ違う。基本的にデザイン時のルール違反なのだと思うが、求めたポイントを超えた大きさで、しかも描画位置が左も上も勝手なオフセット位置からなされているようなのもみうけられる。

つまり、このまま描画すると、StringGridのセルからはみ出るものが結構あるわけだ。この場合そうとうみっともないマトリックスになる。

文字を描画するときに、エリアをクリッピングする描き方がSHIFTJISにはある(DrawText)のだが、ユニコード対応APIがわからない。そのために、別のCanvasに描画して、セルのサイズにあわせてコピーすることでクリッピングをしている。

つぎは、文字コード変換である。文字コードは、さまざまな経緯から1種類ではない。ＩＥでも、文字のエンコードは、ＪＩＳ、シフトＪＩＳ、ＵＴＦ８などを標準でサポートしている。これらのコード変換の方法はいろいろあるようだが、筆者がつねづね使っている方法を使用した。

画面の情報

①ＰＣにインストールされている全てのフォントをリストアップして、どのフォントでも指定できる。

②選んだフォントの情報を表示する。
デルファイで取得するScreen.Fontsは、Ｓ。固定幅フォントはＦ。トゥルータイプフォントはＴ。CharSetがSHIFTJISならＪ。縦書きフォントはＶ。また、使用できる全フォント数、CharSet名を表示する。

③文字の書体デザインを大きなサイズで表示する。
フォント自身にユニコード書体がデザインされているかどうかを目で見て確認できる。漢字対応フォントでもデザインされていなければ、「・」か、ファミリーの代替フォントを表示

されるので、書体デザインフル装備のフォントか否かを確認できる。
④縦書き用フォントも表示できる。
横書きで対応していても、縦書きで対応していないというフォントがあることが確認できる。
⑤選択した文字のコードを表示する。
Unicode、UTF8、ShiftJIS、JIS、句点、EUCの６種類を表示する。

縦書きと横書き、明朝とゴシックのいずれでも、ちゃんとしたデザインで表示できるフォントは何かを調べてみるのに最適のツールではないかと思う。
フル装備のフォントは何点か公開されているようだ。それらをインストールして使用する方法もある。
やや細めのゴシック体としては、@New Gulim, New Gulimがかろうじて使えそうだ。明朝系では、@NSimSun, @PMingLiU, @SimSun, @MingLiU, NSimSun, PMingLiU, SimSun, MingLiU, ＭＳ Ｐ明朝, ＭＳ 明朝。ＭＳ明朝以外はききなれないフォントだ。当然普段見慣れないデザインである。しかし、同じ書体で縦書き、横書きがサポートされているというのはそれなりに有用かと思われる。だが、日本語で多用される括弧や音引きやハイフォンなどのいわゆるヤクモノが縦書きでも正しく用意されているかをチェックする必要がある。意外とダメなのが多い。すると、「同じ書体のままで縦書きと横書きを」というのにはなかなか使用で気なのである。

インストールされているフォントリスト

EnumFontProcのなかでフォントの属性の情報を得ている。

```
//フォントのチェック
function EnumFontProc(lpelf: PEnumLogFont; lpntm: PNewTextMetric;
  nFontType: integer; lParam: LParam): integer; stdcall;
var
  i: integer;
begin
  result := 1;
  if MyCallId=0 then begin
    TStringList(lParam).Add(string(lpelf^.elfLogFont.lfFaceName));
    exit;
```

```
  end;
  MyFonts[MyIndex].Name := lpelf^.elfLogFont.lfFaceName;
  MyFonts[MyIndex].CSet := lpelf^.elfLogFont.lfCharSet;
  MyFonts[MyIndex].Atr := 0;
  if (lpelf^.elfLogFont.lfCharSet=SHIFTJIS_CHARSET) then begin
    MyFonts[MyIndex].Atr := MyFonts[MyIndex].Atr or FSHIFTJISFONTS;
  end;
  if MyFonts[MyIndex].Name[1]='@' then begin
    MyFonts[MyIndex].Atr := MyFonts[MyIndex].Atr or FVERTICALFONTS;
  end;
  if (lpelf^.elfLogFont.lfPitchAndFamily and 3)=FIXED_PITCH then begin
    MyFonts[MyIndex].Atr := MyFonts[MyIndex].Atr or FFIXEDFONTS;
  end;
  if (lpntm.tmPitchAndfamily and TMPF_TRUETYPE)>0 then begin
    MyFonts[MyIndex].Atr := MyFonts[MyIndex].Atr or FTRUETYPEFONTS;
  end;
  for i:=0 to MySFontCnt-1 do begin
    if MyFonts[MyIndex].Name=Screen.Fonts[i] then begin
      MyFonts[MyIndex].Atr := MyFonts[MyIndex].Atr or FSCREENFONTS;
      break;
    end;
  end;
  inc(MyIndex);
end;

//インストールされているすべてのフォントを得る
//ListBox1.にセットする
procedure TForm1.GetAllFont();
var
  DC: HDC;
  fList: TStringList;
begin
  MySFontCnt := Screen.Fonts.Count;

  fList := TStringList.Create;
  try
    MyCallId := 0; //数を得るだけ
    DC := GetDC(0);
    EnumFontFamilies(DC,nil,@EnumFontProc,LParam(fList));
    MyFontsCount := Largest(MySFontCnt,fList.Count);

    SetLength(MyFonts,MyFontsCount);
//showmessage(intToStr(MyFontsCount)+'  '+intTOStr(length(MyFonts)));
    //------------------------------------------------------
    MyIndex := 0;
    MyCallId := 1;
    EnumFontFamilies(DC,nil,@EnumFontProc,LParam(fList));
    ReleaseDC(0,DC);

    ListBox1.items := fList;
    ListBox1.Sorted := True;
  finally
    fList.Free;
```

```
  end;
end;
```

CharSetを得る

Form1のStatusBar1.Panels[2].textに表示している。
あわせて、指定のフォントの横書き縦書き、トゥルータイプ、固定ピッチ、スクリーンフォントなどの情報も表示している。

```
//CharSetを得る
//
function TForm1.getFontsIndex(Fname: string): integer;
var
  idx,i,Atr,Cset: integer;
  strs: string;
begin
  Atr := 0;
  Result := -1;
  idx := -1;
  StatusBar1.Panels[0].text := '';
  StatusBar1.Panels[1].text := '';
  StatusBar1.Panels[2].text := '';
  StatusBar1.Panels[3].text := fName;
  Cset := 0;
  for i:=0 to length(MyFonts)-1 do begin
    if MyFonts[i].Name=fName then begin
      idx  := i;
      Atr  := MyFonts[i].Atr;
      CSet := MyFonts[i].Cset;
      break;
    end;
  end;
  if idx<0 then begin
    exit;
  end;
  //
  strs := 'UNKNOWN_CAHRSET';
  if Cset=0 then begin //ANSI_CHARSET        = 0;
    strs := 'ANSI_CHARSET';
  end;
  if Cset=1 then begin // DEFAULT_CHARSET    = 1;
    strs := 'DEFAULT_CHARSET';
  end;
  if Cset=2 then begin // SYMBOL_CHARSET     = 2;
    strs := 'SYMBOL_CHARSET';
  end;
  if Cset=$80 then begin // SHIFTJIS_CHARSET = $80;
    strs := 'SHIFTJIS_CHARSET';
  end;
  if Cset=129 then begin // HANGEUL_CHARSET  = 129;
    strs := 'HANGEUL_CHARSET';
  end;
  if Cset=134 then begin // GB2312_CHARSET   = 134;
```

```
    strs := 'GB2312_CHARSET';
  end;
  if Cset=136 then begin // CHINESEBIG5_CHARSET = 136;
    strs := 'CHINESEBIG5_CHARSET';
  end;
  if Cset=255 then begin // OEM_CHARSET      = 255;
    strs := 'OEM_CHARSET';
  end;
  if Cset=130 then begin // JOHAB_CHARSET    = 130;
    strs := 'JOHAB_CHARSET';
  end;
  if Cset=177 then begin // HEBREW_CHARSET   = 177;
    strs := 'HEBREW_CHARSET';
  end;
  if Cset=178 then begin // ARABIC_CHARSET   = 178;
    strs := 'ARABIC_CHARSET';
  end;
  if Cset=161 then begin // GREEK_CHARSET    = 161;
    strs := 'GREEK_CHARSET';
  end;
  if Cset=162 then begin // TURKISH_CHARSET  = 162;
    strs := 'TURKISH_CHARSET';
  end;
  if Cset=163 then begin // VIETNAMESE_CHARSET= 163;
    strs := 'VIETNAMESE_CHARSET';
  end;
  if Cset=222 then begin // THAI_CHARSET     = 222;
    strs := 'THAI_CHARSET';
  end;
  if Cset=238 then begin // EASTEUROPE_CHARSET= 238;
    strs := 'EASTEUROPE_CHARSET';
  end;
  if Cset=204 then begin // RUSSIAN_CHARSET  = 204;
    strs := 'RUSSIAN_CHARSET';
  end;
  if Cset=77 then begin // MAC_CHARSET       = 77;
    strs := 'MAC_CHARSET';
  end;
  if Cset=186 then begin // BALTIC_CHARSET   = 186;
    strs := 'BALTIC_CHARSET';
  end;
  StatusBar1.Panels[2].text := strs;

  strs := '-----';
//  FSCREENFONTS  =$1;
//  FFIXEDFONTS   =$2;
//  FTRUETYPEFONTS=$4;
//  FSHIFTJISFONTS=$10;
//  FVERTICALFONTS=$20;
  if (FSCREENFONTS and Atr)<>0 then begin
    strs[1] := 'S';
  end;
  if (FFIXEDFONTS and Atr)<>0 then begin
```

```
    strs[2] := 'F';
  end;
  if (FTRUETYPEFONTS and Atr)<>0 then begin
    strs[3] := 'T';
  end;
  if (FSHIFTJISFONTS and Atr)<>0 then begin
    strs[4] := 'J';
  end;
  if (FVERTICALFONTS and Atr)<>0 then begin
    strs[5] := 'V';
  end;
  StatusBar1.Panels[1].text := strs;
  //
  Result := idx;
end;
```

JIS-ShiftJIS-Unicode-EUC-UTF8-句点のコード変換

◆シフトJISからJIS

```
//ShiftJIS->JIS
function ShiftJISToJIS(C: Word): Word;
var
  Hi,Lo: Word;
begin
  Hi := (C shr 8) and $FF;
  Lo := C and $FF;
  if Hi <= $9F then begin
    Dec(Hi,$71);
  end else begin
    Dec(Hi,$B1);
  end;
  Hi := Hi * 2 + 1;
  if Lo > $7F then begin
    Dec(Lo);
  end;
  if Lo >= $9E then begin
    Dec(Lo,$7D);
    Inc(Hi);
  end else begin
    Dec(Lo,$1F);
  end;
  Result := (Hi shl 8) or Lo;
end;
```

◆JISから句点

```
function JisToKuten(N: WORD): WORD; assembler;
asm
  sub ax,2020h;
end;
```

◆UnicodeからUTF8

```
//Unicode -> UTF8へ変換
procedure UnitoUTF8(unich: pWidechar; var j: cardinal; UTFCh :Pchar;
  var i: cardinal);
type
  T_DoubleByte = record
    case integer of
      0: (LoByte,HiByte: byte);
      1: (WByte: word);
    end;
var
  dummy,uniw: T_DoubleByte;
begin
  //16ビットUnicode   -> UTF8形式Unicode
  //00000000 0xxxxxxx -> 0xxxxxxx
  uniw.WByte := word(uniCh[j]);
  dummy.WByte := Word(UniCh[j+1]);
  if (uniw.WByte=0) //and ( dummy.WByte =0)
    then begin
    j := 2147483647;
    exit;
  end;

  if not(bool((uniw.LoByte and $80)) or(uniw.HiByte<>0)) then begin
    byte(UTFCh[i]) := byte(uniw.LoByte) and $7F;
    // byte(UTFCh[i+1]) := 0;
    i := i+1;
    j := j+1;
    exit;
  end;

  //16ビットUnicode   -> UTF8形式Unicode
  //00000xxx xxyyyyyy -> 110xxxxx 10yyyyyy
  if ((uniw.HiByte) and $f8)=0 then begin
     dummy.WByte := (uniW.WByte*4); //xxxxx
     byte(UTFCh[i]) := (dummy.HiByte or $c0) and $DF;
     byte(UTFCh[i+1]):= (uniW.LoByte and $3F) or $80; //yyyyyyy
     i := i+2;
     j := j+1;
     exit;
   end;

  //16ビットUnicode   -> UTF8形式Unicode
  //xxxxyyyy yyzzzzzz -> 1110xxxx 10yyyyyy 10zzzzzz
  dummy.HiByte := uniw.HiByte; //uniCh[j+1]; //xxxxx
  byte(UTFCh[i]) := (((dummy.HiByte div 16))and $0f) or $E0;
  dummy.WByte := uniw.WByte; //yyyyyyy
  byte(UTFCh[i+1]) := ((dummy.WByte div 64) and $3f) or $80;
  // uniw.HiByte := 0;//uniCh[j+1]; //zzzzzz
  byte(UTFCh[i+2]) := (uniw.LoByte and $3f) or $80;
  i := i+3;
  j := j+1;
end;
```

◆シフトJISからUTF8

```
//
function StringtoUTF8(wscd: Word): PChar;
var
  indexUni,IndexUTF: Cardinal;
  utf8: PChar;
  unich2: pWideChar;
  wc: WideChar;
begin
  UTF8 := StrAlloc(9);
  wc := WideChar(wscd);
  unich2 := @wc;
  indexUni := 0;
  indexUTF := 0;
  UnitoUTF8((unich2),indexUni,UTF8,indexUTF);
  //UnitoUTF8((unich2),indexUni,UTF8,indexUTF);
  byte(UTF8[indexUTF])   := 0;
  byte(UTF8[IndexUTF+1]) := 0;
  byte(UTF8[IndexUTF+2]) := 0;
  Result := utf8;
end;
```

気になることのまとめ

Unicodeは拡張があいつぎ、FFFFの16ビットで管理できる最大を超えてしまっている。
FFFFを超えたケースについては、このツールはサポートしていない。コード変換の処理が追いつかないためだ。JISの規格が前の規格を踏襲しないで変化していることもある。
このあたりは、もうすこし情報が集まってから必要であれば対応したいところだ。
ＪＩＳ第３、４水準を寄せるのもそうだがFFFFを越えたユニコードのコード変換は変換表を使うしかないのである。

2.6.PmHagakiAtenaハガキ宛名印刷ツール

特徴と機能

単に宛名を印刷するといってもなかなか手ごわい問題がありる。

・どのような項目が可能か
・各項目について書体、サイズが個別に指定できるか
・各項目の印字位置を任意に指定できるか
・縦書き、横書きが選べるか
・背景つきでプレビューが可能か
・ユニコード文字を扱えるか
・１枚単位以外の印刷もできるか
・カスタマーバーコードは扱えるか
・実行時に微妙な位置やサイズなどを変更できるか
・設定が複数できるか

最後の４～５項は相当専門的な機能になる。
このほかに、設定などの取り扱いがどの程度やりやすいか、ということで違ってくる。

PmHagakiAtenaハガキ宛名印刷ツールでの特徴は、このあたりをカバーしたもの。

第１． 用紙の種別は３種類である。
ハガキ、Ａ４に４枚のもの、Ａ３に８枚のもの。
４枚、８枚については、１ページ目どの位置から宛名を印刷するかを指定できる。
第２．カスタマーバーコードの印刷に対応している。
カスタマーバーコード付き宛名の印刷で送料割引を受けられる。
１０００枚以上で５％割引のメリットがある。
第３．横書き、縦書きモードを切り替えることができる。
横書きとして用意した宛名データを、縦書きに自動変換している。
プレビューで確認できる。
第４．印字位置と文字フォントを任意に指定できる 。
文字の大きさや書体、印字位置を自分で指定できる。
設定を保存再利用できる。
第５．エクセルのデータファイルを直接読み込める。
一元的にエクセルで管理している住所録をそのまま利用できる。
エクセルで表現できるＪＩＳ外のユニコード文字にも対応している。

エクセルファイルから読み込む

宛名印刷ツールの場合の処理の流れは次のようにする。

印刷で必要な項目は、①宛名、②郵便番号、③住所１、④住所２、⑤勤務先など、⑥部署名など、⑦肩書きなど、⑧連名など、⑨敬称です。顧客リストなどの住所管理では、その他に電話番号、ファックス番号、携帯番号、ＵＲＬ、メールアドレス、顧客区分等々がある。

まず､エクセルで住所リストを整理します｡宛名印刷に必要な項目だけを残し他を削除する。宛名印刷で使用する項目の順に列をそろえる。敬称が必要な場合は継承列を追加し､「様」「先生」「御中」などをいれる。敬称は、空欄でも、オプション設定で任意の敬称を使うことができるのだが､あらかじめファイルに設定しておけば､各あて先ごとに任意の指定が可能になる。

宛名印刷用の仮ファイルなので、ここでいったん別名で保存しておく。

具体的にエクセルのファイルを作成するときのことについて述べる。

(1) 列の項目名（必須）

これは、このシステムでは固定です。なぜ固定かというと、１つはこのシステムで安定的に処理ができるため。２つ目は、列の順が都合で多少変わった場合でもそのまま使用できるようにするためだ。

サンプルのデータから１行目のデータだけをコピーして作成するのがベスト。

下記に、項目名の１行をあげておく。

" ",宛先|Atena,郵便番号|YubinNo,住所１|Jyusho1,住所２|Jyusho2,
会社名|Kaishamei,肩書き|Katagaki,敬称|Keisho1,連名|Renmei,
敬称2|Keisho2,コード|CBCode,印刷コード|PrintCBC," ",,,

(2) 宛名を編集するときの留意点

宛名や住所を整理する場合の一般的な注意がある。

・宛名：姓と名の区切りにスペースをつけるほうが何となく分かりやすかったり、バランスがいいということがある。もちろん、付けなくてもいいことなのだが、付ける場合はすべて統一しておくことである。
・住所等：基本的に住所１と住所２に分けるようにする。住所１は、郵便番号で変換した都道府県に番地を含めたものとする。住所２にはアパート、マンションなどの住宅名と室番などを記す。
・基本的にすべての文字を全角で書いてかまわない。しかし全角で室番などが出るのはあまり気持ちがいいものではない、という意見もある。一番自然なのは、郵便番号変

換で表示される文字列はすべて漢字が適切。それ以外で出てくる英数文字は半角とする。ただし、全角の方がぴったりくる場合はそのようにする。

郵便番号：123-4567
住 所 １：東京都千代田区神田駿河台3-2-1
住 所 ２：吉澤ビル402B
宛　　名：Ｍ＆Ｘコーポレーション

・都道府県名を付けるか。付けたほうがより確実に場所を特定するのでいいと思う。

なお、半角のカナはいうまでもなく使わないことだ。

宛名は、まさに他人様の目にふれるものなので、きちんとしたルールで統一することのが大事だ。

(3) 住所、人名の文字の問題

ＪＩＳで規定している日本語のＪＩＳコードは、コンピュータが使用されるに伴って整備してきている。

基本的に全国の地名で使用されている文字をカバーするように留意されたものなので、地名で文字がないというケースはほとんどないはずだ。だが、人名はＪＩＳでカバーしきれないときがある。（人名漢字が規定される前に届けられた名前）

戸籍にはよみがなく、名前が記号のように登録する日本のシステムなので、登録した本人の思い込みによる「誤字」「俗字」が多く手書きで登録された。コンピュータで処理する必要が念頭にないのだからやむをえないかったのだろう。

そのためコンピュータで扱うには基本的な無理が生じる。

本来の漢字にない、活字にない、明らかな誤字（創作漢字）をどう扱うかという問題がある。

ユニコードの範囲に含まれるものは表示できるが注意（CSVなどテキスト化の時点で消える)。誤字、俗字は正字で表現がすることも選択できるが、本人や編集責任者が満足かは別になる。

活字にない文字（誤字、俗字）でも本人がガンとして「自分だけの正しい文字」として強く主張する場合もあるからだ。

住所録の場合は、次のような制限をもうけることが必要ではないだろうか。

①エクセルで管理し、かつオフィス等ユニコードを扱えるツールで印刷まで処理する場合は、ユニコードを用いる。（ユニコードを印刷で扱えない場合は、あくまでもＪＩＳの範囲とする）

②これ以外の外字は使わない。ひらがな等のよみで表示する。（自分で外字を作り、自分で印刷する場合に限って外字を使う）

つまり、他人がデータを扱う段になったとたんに、互換性と取り扱いが大きく制限されるため、少なくとも外字は扱えない。ユニコードはくれぐれも事前に確認しておかなければ、文字は消えると思ったほうがいい。

(4) ＣＳＶファイルで失われる文字がある

エクセルで作成した住所録を宛名印刷用のファイルにする。その場合、多くはＣＳＶファイルとしている。

ＣＳＶファイルとは、項目をカンマで区分し、１レコード１行のテキストファイルのことだ。メモ帳などのエディタで表示編集が可能で広く普及している。

このときに、エクセルで表示されていた文字が半角の?に化けていることがある。この文字化けした文字がユニコード文字。例として、伊川けんーのけんが劍の右側が刀。

また、・に化ける場合は外字だ。吉澤の吉が、土に口。ユニコードにもないので自分だけの外字を作ってあてていたようなときの例。

けんーは、剣ーか劍ーかひらがなでけんーと表示するしかない。吉澤は士に口で住所データ上ではやむをえない。

ＣＳＶファイルはテキストファイルであるゆえに、たいていのソフトで読み書きができるのだが、制限はあくまでもＪＩＳで規定する範囲の文字しか扱えないことだ。

Ｍ＆Ｘコーポレーション,123-4567,東京都千代田区神田駿河台3-2-1,⒧㉝澤ビル402B,伊川?ー

ＣＳＶファイルの注意点は、いったんＣＳＶになってしまえば、?や・の元の文字が何であったのか、作成した本人以外は誰もわからない、という状態になることである。

　このようにして作成した、エクセルのファイルを、このツールではメニューの開くからそのまま読み込める。

TntUnicodeWareコンポーネントの利用

ツールを自作するときに、クリアしなければならないことがある。

①エクセルの直接把握する。

　エクセルで管理するデータをＣＳＶを経由しないで直接ユニコードのままのデータを把握する必要がある。ＣＳＶを得てしまえば欠落した状態で処理しかできなくなるからだ。

②ユニコードの表示と印刷。

　得たデータを画面に表示、プリンターに印刷するにはどうするか、ということだ。

③背景を透明化したプレビュー。

　これは、結局最後に「縦書きエディット」を作ることになった理由のひとつだ。プレビューは表示だけなので「縦ラベル」でも当初はよかったのだが、ＷＹＳＷＹＧでの入力を実現したくなり「縦エディット」のコンポーネントを作ることになったのである。

④文字の均等割り付けと扁平化。

　特定のサイズに文字を均等に配置するというのは、ワードではよくみる機能だが、そのエリアに入りきらない場合は扁平をかける機能の実現が必要である。特定のエリアに入りきらないときはフォントを小さくするというエクセルなどのやり方は採用できない。

　PmHagakiAtenaハガキ宛名印刷ツールでは、解決するために結果的には、いくつかのコンポーネントを使っている。

　まず、ユニコードを扱わなければならないので、標準のコンポーネントでは実現できない。TntUnicodeコンポーネントを使っている。TntWare社のコンポーネントはそれを標準と取り替えることによってユニコードにたやすく対応できるようになっているので、大変便利なものだ。すばらしいものを提供してくれている。

　次に、エクセルからデータを読み取るのだが、M&I様提供のXBiff180コンポーネントを使っている。エクセルからデータを読み込むのは、エクセル関係のコンポーネントが標準でも多く出されているので利用されている方々も多いと思うが、使い方が今ひとつであるのと、速度が遅い。そのようななかでXBiffは十分な満足を与えてくれる。

　ただこのままでは、読み込むデータの型がStringであってWideStringでないので、ソースを一部書き換えてWideStringが得られるようにして使っている。書き出しは、エクセルで十分な

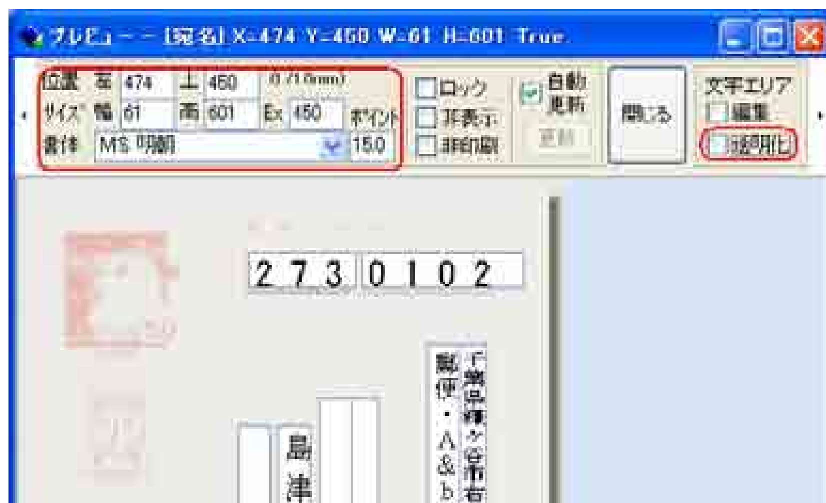
のでしない。
　あとは、プレビューで印刷イメージを表示し同時に設定も可能にしたいという要求をみたすために、TateEditを使っている。このコントロールの状態を、保存するのにObjStreamV1.03（VRAMの魔術師様）を使っている。かつて佐々木＠六角堂氏の「実行時デザインシステムVer2.2」というものを使っていたのだが、開発環境の問題からこちらの方式に変更したものだ。運用中のコントロールの動的変更も可能なすばらしいシステムだった。

プレビューの処理

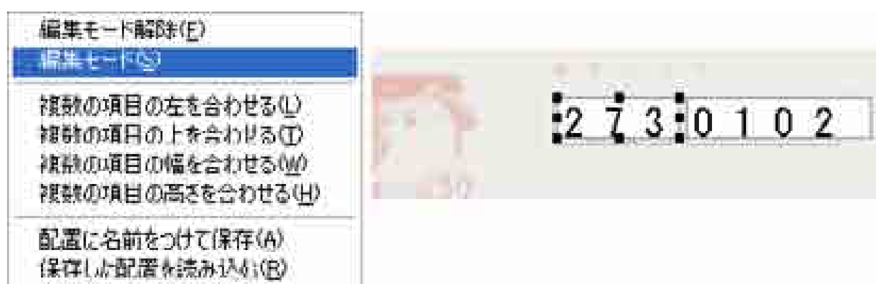
　PmHagakiAtenaハガキ宛名印刷ツールでは、ハガキへの宛名の文字は位置状態をプレビューできる。
　プレビューのフォームに配置したパネル。このパネルは画像の読み込みが可能になっていてハガキのイメージを確認するためにある。このパネルは、別途指定可能なズーム（50～100%）にもとづいてサイズが変わる。
　このパネルに配置されたTateEditに、メイン画面でさしている宛名の情報がテキストとしてセットされる。セットするときに、それぞれのTateEditの位置とサイズをズームに応じて計算している。文字のサイズも同様である。
　注意が必要なのは、元、つまりズームしない時点の位置とサイズ情報をべっと変数でちゃんと持っていることである。持たないで、現在表示しているTateEditなりのコントロールに対して再計算を繰り返すうちに、誤差から元がわからなくなってしまうからである。画面の解像度はそうとう粗いので必ず元をおさえておくことである。
　このツールは各コントロールの動的な変更を可能にしている。すなわち、プレビュー画面で位置やサイズやフォント情報の変更などが可能なのだ。このためにも、元を別途持っていて、未決定のうちは画面上だけで自由に変更し、「決定」したタイミングで、元の内容を変更するという手法をとっている。

Controlの位置とサイズの動的変更

まず、透明化のチェックをはずすと、TateEditの位置が明確になる。
そして、マウスでクリックして指定したコントロールの情報が左上に表示される。タイトルバーには名称が表示されるので確認する。そして、位置などの情報が表示される。単位は左上を基点にした10分の１ミリだ。



数値の変更は、このEditに直接入力してリターンを押す。
もうひとつの方法は、コントロールの上で右クリックメニューをだして、編集モードを選ぶ。すると、コントロールの周囲にグラブが表示される。これをつまんでサイズを変えたり、マウスでドラッグして位置を移動したりする。
このモードからの解除は、同じように右クリックメニューを表示して「編集モード解除」をする。パネル上のほかの箇所をクリックしても解除される。

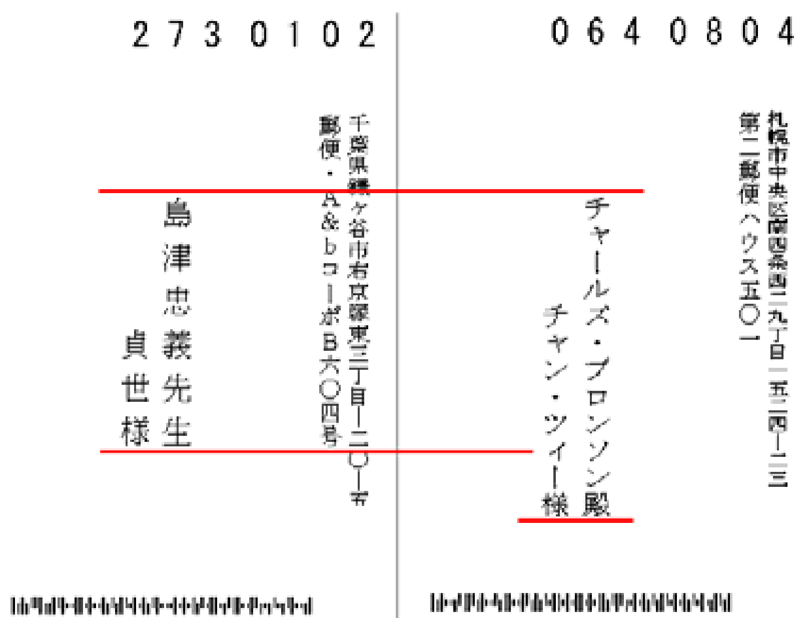


プレビュー画面の自動更新にチェックがあれば、移動した時点で、元情報の更新をしている。なければ、更新ボタンが押された時点で変更を元に反映させる。逆に更新ボタンを押さなければ、画面が変わった時点で変更はキャンセルされる。
グラブを表示したコントロールの動的変更だが、StretchHandleというAnthony Scott氏の作品がある。これに移動したときのイベント追加などをおこなって利用している。
配置した情報は、次回起動時に再現される。
なお、現状はそれとは別に保存読み込み（読み込めば、それが最新の状態の情報になる）ができる。これを利用すれば、宛名ツールとして、初期値で用意しているハガキ4種類だけでなく、ラベルなどへの応用が可能だ。

TateEdit機能と特殊プロパティの効用

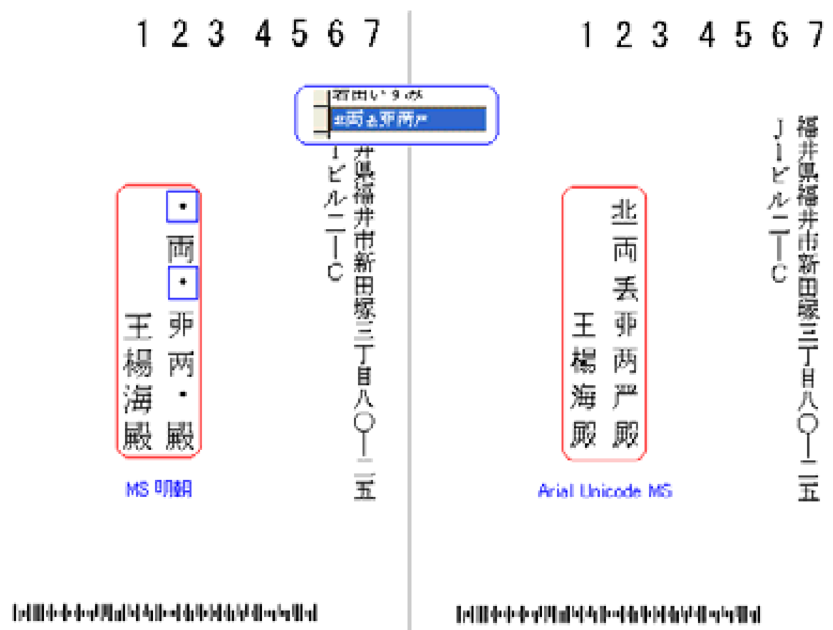
JyustifyExとOffsetExの活用

TateEditのプロパティに入れた特別な機能をこのツールで生かしている。



このように、名前が長い場合でも違和感がないようになっている。

ユニコード文字の表示



[北両丟丣両严] が、ＭＳ明朝横では正しく表示できるが、ＭＳ明朝縦ではごらんのように表示できない。

カスタマーバーコードの作成

カスタマーバーコードはどうしているか。
まず、総務省のカスタマーバーコードについての仕様書をインターネット上から得た。
カスタマーバーコードは、郵便番号と住所の組み合わせであることがわかる。
しかも、エクセルのアドインである「郵便番号変換」で、住所から郵便番号を得られる

ばかりか、同じような処理でカスタマーバーコード自身も得られる。
　ただ、得られるのは生の第１次状態のコードだ。これを仕様書にそって必要な最終コードにするのは難しいことではない。
　このような変換をして得たコードを、バーコードとして表示するわけだ。パネルに直接各方法もあるのだが、ここではバーコードのために特化したフォントを新たに作成して対応している。
　i-JapanPost.ttfがそれだ。
　パネル上のTateEditのフォントに指定して、９ポイントで表示している。

気になることのまとめ

　エクセルでユニコード文字を使い、ＭＳ明朝で縦書きで印刷。文字が出ない、という問題は残る。これは横書きにする以外にすすめられない。だが、さきざき、フル対応のフォントが提供されれば解決される。
　表面に、送り主である自分の住所氏名を表記したいというニーズがある。これは、PmHagakiAtenaの機能からはずれている。先の課題としたい。

付録　Delphi Tips

XpStyle に対応する

WindowsXpのスタイルにするのは、それなりに見え方が変化する。マイクロソフトの98のサポートが今年終了する。Xpを使っているユーザーが大半になったため、マシンがＸｐなら対応したほうがいいかもしれない。Delphi6以上であれば楽に対応できるのだが、著者の場合は、Delphi5が標準であるために最小限の対応になる。

具体的には次のようなmanifestをつくり、Exeと同じフォルダにおいておけばいいというものなのだ。しかし、著者の場合はリソースファイルにして使いまわしている。

つまり、menifestからリソースコンパイラでresファイルにし、これをUnit1.pasでインクルードする方式だ。これだと、manifestを置かなくても、Exeだけで実行できる。他人に使ってもらうツールならたいへん便利といえる。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly（次のスペースで行続き）
 xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
<assemblyIdentity
 version="1.0.0.0"
 processorArchitecture="X86"
 name="YahagiOffice.Pageman.PmHagakiAtena"
 type="win32"
/>
<description>Description</description>
<dependency>
   <dependentAssembly>
      <assemblyIdentity
         type="win32"
         name="Microsoft.Windows.Common-Controls"
         version="6.0.0.0"
         processorArchitecture="X86"
         publicKeyToken="6595b64144ccf1df"
         language="*"
      />
   </dependentAssembly>
</dependency>
```

PmHagakiAtena.exe.manifestというファイル名で保存する。
つぎに、同じフォルダに、Pm.rcというファイル名で下記の内容にする。

```
#define CREATEPROCESS_MANIFEST_RESOURCE_ID  1
#define RT_MANIFEST  24
CREATEPROCESS_MANIFEST_RESOURCE_ID（次のスペースで行続き）
  RT_MANIFEST  "PmHagakiAtena.exe.manifest"
```

リソースコンパイラを実行する。
brcc32.exe Pm.rc
pm.resが作成される。

このファイルを、Unit1.pasで指定する。

{$ +.DFM}の次の行あたりに、下記のように1行書いておく。
{$R pm.res}

ただし、XpStyleには過剰な期待はできない。動かないコントロールもあるので、上記のpm.resを指定するだけで期待の動作をしないときは、削除しておくことである。

文字のコードポイントの変換

◆シフトＪＩＳからＪＩＳ

```
//ShiftJIS->JIS
function ShiftJISToJIS(C: Word): Word;
var
  Hi,Lo: Word;
begin
  Hi := (C shr 8) and $FF;
  Lo := C and $FF;
  if Hi <= $9F then begin
    Dec(Hi,$71);
  end else begin
    Dec(Hi,$B1);
  end;
  Hi := Hi * 2 + 1;
  if Lo > $7F then begin
    Dec(Lo);
  end;
  if Lo >= $9E then begin
    Dec(Lo,$7D);
    Inc(Hi);
  end else begin
    Dec(Lo,$1F);
  end;
  Result := (Hi shl 8) or Lo;
end;
```

◆ＪＩＳから句点ＪＩＳ

```
//JIS->KutenJIS
function JisToKuten(N: WORD): WORD; assembler;
asm
  sub ax,2020h;
end;
```

◆ユニコードからＵＴＦ８

```
//Unicode -> UTF8へ変換
procedure UnitoUTF8(unich: pWidechar; var j: cardinal; UTFCh :Pchar;
  var i: cardinal);
type
  T_DoubleByte = record
    case integer of
      0: (LoByte,HiByte: byte);
      1: (WByte: word);
```

```
    end;
var
  dummy,uniw: T_DoubleByte;
begin
  //16ビットUnicode    ->  UTF8形式Unicode
  //00000000 0xxxxxxx  ->  0xxxxxxx
  uniw.WByte := word(uniCh[j]);
  dummy.WByte := Word(UniCh[j+1]);
  if (uniw.WByte=0) //and ( dummy.WByte =0)
    then begin
    j := 2147483647;
    exit;
  end;

  if not(bool((uniw.LoByte and $80)) or(uniw.HiByte<>0)) then begin
    byte(UTFCh[i]) := byte(uniw.LoByte) and $7F;
    // byte(UTFCh[i+1]) := 0;
    i := i+1;
    j := j+1;
    exit;
  end;

  //16ビットUnicode    ->  UTF8形式Unicode
  //00000xxx xxyyyyyy  ->  110xxxxx 10yyyyyy
  if ((uniw.HiByte) and $f8)=0 then begin
     dummy.WByte := (uniW.WByte*4); //xxxxx
     byte(UTFCh[i]) := (dummy.HiByte or $c0) and $DF;
     byte(UTFCh[i+1]):= (uniW.LoByte and $3F) or $80; //yyyyyyy
     i := i+2;
     j := j+1;
     exit;
   end;

  //16ビットUnicode    ->  UTF8形式Unicode
  //xxxxyyyy yyzzzzzz  ->  1110xxxx 10yyyyyy 10zzzzzz
  dummy.HiByte := uniw.HiByte; //uniCh[j+1]; //xxxxx
  byte(UTFCh[i]) := (((dummy.HiByte div 16))and $0f) or $E0;
  dummy.WByte := uniw.WByte; //yyyyyyy
  byte(UTFCh[i+1]) := ((dummy.WByte div 64) and $3f) or $80;
  // uniw.HiByte := 0;//uniCh[j+1]; //zzzzzz
  byte(UTFCh[i+2]) := (uniw.LoByte and $3f) or $80;
  i := i+3;
  j := j+1;
end;
//
function StringtoUTF8(wscd: Word): PChar;
var
  indexUni,IndexUTF: Cardinal;
  utf8: PChar;
  unich2: pWideChar;
  wc: WideChar;
begin
  UTF8 := StrAlloc(9);
```

```
    wc := WideChar(wscd);
    unich2 := @wc;
    indexUni := 0;
    indexUTF := 0;
    UnitoUTF8((unich2),indexUni,UTF8,indexUTF);
    //UnitoUTF8((unich2),indexUni,UTF8,indexUTF);
    byte(UTF8[indexUTF])   := 0;
    byte(UTF8[IndexUTF+1]) := 0;
    byte(UTF8[IndexUTF+2]) := 0;
    Result := utf8;
  end;
```

◆ＵＴＦ８(PChar)からユニコード(PChar)へ

```
  //UTF8 -> UniCode unichに変換
  procedure UTF8toUni(UTFCh: Pchar; var i: cardinal; unich: pchar;
    var j: cardinal);
  type
    T_DoubleByte = record
      case integer of
        0: (LoByte,HiByte: byte);
        1: (WByte: word);
      end;
  var
    UTFW,Dummy: T_DoubleByte;
    sw: integer;
  begin
    //    16ビットUnicode    <-  UTF8形式Unicode
    //    00000000 0xxxxxxx  <-  0xxxxxxx
    if (Byte(UTFCh[i])=0)and (Byte(UTFCh[i+1])=0) then begin
      j := 2147483647;
      exit;
    end;

    if not(((byte(UTFCh[i])and $80)=$80)) then begin
      byte(unich[j+1]) := 0;
      uniCh[j+0] := UTFCh[i];
      i := i+1;
      j := j+2;
      exit
    end;
    sw := integer(byte(UTFCh[i]) and $E0);
    case sw of
      //    16ビットUnicode    <-  UTF8形式Unicode
      //    00000xxx xxyyyyyy  <-  110xxxxx 10yyyyyy
      $c0: begin
        //yyyyyy
        UTFW.HiByte := 0 ;
        UTFW.LoByte := Byte(UTFCh[i+1]) and $3f;
        //xxxxxx
        dummy.HiByte := Byte(UTFch[i])  and $1f;
        dummy.LoByte := 0;
        dummy.WByte := (dummy.WByte div 4);
```

```
      UTFW.WByte := dummy.WByte or UTFW.WByte;

      Byte(unich[j+1]) := UTFW.HiByte;
      Byte(unich[j+0]) := UTFW.LoByte;
      i := I+2;
    end;
    //    16ビットUnicode    <-  UTF8形式Unicode
    //    xxxxyyyy yyzzzzzz  <-  1110xxxx 10yyyyyy 10zzzzzz
    $e0: begin
      UTFW.HiByte := 0; //zzzzzz
      UTFW.LoByte := byte(UTFCh[i+2]) and $3f;

      dummy.HiByte :=byte(UTFch[i+1]) and $3f; //yyyyyyy
      dummy.LoByte := 0;

      dummy.WByte := dummy.WByte div 4; //yyyyyyzzzzzzz
      UTFW.WByte  := dummy.WByte or UTFW.WByte;

      dummy.HiByte := byte(UTFch[i]) and $0f; //xxxxx
      dummy.HiByte := dummy.HiByte*16;
      dummy.LoByte := 0;
      UTFW.WByte   := dummy.WByte or UTFW.WByte;

      Byte(unich[j+1]) := UTFW.HiByte;
      Byte(unich[j+0]) := UTFW.LoByte;

      i := I+3;
    end;
  end;
  j := J+2;
end;
```

◆ＵＴＦ８(PChar)からStringへ

```
//UTF8文字列をストリングに文字列に変換する。
function UTF8toString(UTF8: pchar): string;
var
  indexUni,IndexUTF,len: cardinal;
  uniCh: pchar;
begin
  unich := '';
  len := strlen(UTF8);
  try
    uniCh := StrAlloc(len*2+2);

    indexUni := 0;
    indexUTF := 0;
    while (indexUTF< len) do begin
      UTF8toUni(UTF8,indexUTF , unich,indexUni);
    end;
    //ターミネータ
    byte(unich[indexUni])   := 0;
    byte(unich[indexUni+1]) := 0;
    Result := WideCharToString(PWideChar(uniCh));
```

```
  finally
    StrDispose(uniCh);
  end;
end;
```

つぎのようなコードで、上記のモジュールを利用している。

```
//コードの表示

var
 uc: Unsigned;
 ws; Widestring;
 pc: PChar;
 ss: string;
 int1: integer;
begin

 //Unicode=WideString
 uc := integer(WideChar(ws[1]));
 Caption  := Format('%4X',[uc]);

 //UTF8
 pc := StringtoUTF8(uc);
 ss := pc;
 Caption :=
   Format('%2X%2X%2X',[integer(ss[1]),integer(ss[2]),integer(ss[3])]);
 Caption := UTF8toString(pc);

 //ＪＩＳと句点
 ss := ws;
 int1 := integer(Char(ss[1]));
 if length(ss)>1 then begin
   int1 := int1*256+integer(Char(ss[2]));
 end;
 Caption := Format('%4X',[int1]);

 //ＪＩＳなら
 if  not (uc>$100) and (int1=$3f) then begin
   int1 := ShiftJISToJIS(Word(int1));
   Caption := Format('%4X',[int1]);
   Caption := JisToKuten(Word(int1));
   Caption := Format('%2d-%2d',[int2 div 256,int2 mod 256]);

   //EUC
   int1 := int2 or $8080;
   edtEUC.text := Format('%4X',[int1]);
 end;
```

使用説明書を表示する

　著者の場合の使用説明書は、htmの拡張子をもつＨＴＭＬファイルで用意するようにしている。これを、下記のように呼び出すのである。

mhtは、著者のPagemanで作成する。

```
//help
procedure TForm1.Button3Click(Sender: TObject);
begin
  if FileExists(GHelpFilename) then begin
    ExecuteFile(ExpandFileName(GHelpFileName));
  end;
end;
```

似たようなものに、外部のファイルを関連付けで起動したい場合もある。
下記のモジュールを使う。
また、あわせて、インターネット先（ローカルでもいいのだが）をインターネットエクスプローラで表示するときには、つぎのを使用する。

```
//関連付けでファイルを起動する
function ExecuteFile(FileName: string; ShowMode:
  integer=SW_SHOWNORMAL):integer;
var
  AppName: array [0..MAX_PATH] of Char;
begin
  if (Filename<>'') and (filename[1]<>'"') and
    (filename[length(filename)]<>'"') then begin
    filename := '"'+filename+'"';
  end;
  FindExecutable(PChar(FileName),
    PChar(ExtractFilePath(Application.ExeName)),AppName);
  Result := ShellExecute(Application.Handle,'open', AppName,
    PChar(FileName),nil,ShowMode);
end;

//Ｗｅｂサイト表示
procedure WebLink(WebUrl: string); overLoad;
var
  wStrs: WideString;
  UrlStr: PWideChar;
begin
  wStrs := WebUrl;
  UrlStr := PWideChar(wStrs);
  HlinkNavigateString(nil,UrlStr);
end;
```

余談になるが、ヘルプファイルについて少し触れたい。ウインドウズのXp以前の多くはいわゆるヘルプファイルを使うのが一般的であった。しかし、これを作成するのはそう容易ではない。
つぎに思いついたのが、いわゆるＨＴＭＬ形式のヘルプである。これも、ＨＴＭＬファイルとして１本程度ならさほど変でもないが、画面などの画像をともなう場合にヘルプだけでそうとう煩雑なファイル群ができる。ＨＴＭはこれらを１つのＨＴＭＬにしたものだ。
Pagemanであれば、ＷＹＳＷＹＧで容易にＨＴＭＬとＭＨＴがいきなり作成できる。
なお、Pagemanの本来というよりももともとの機能は実は、Microsoft Html Help Workshopと連動して、数ページから数百ページを持つ本格的なＨＴＭＬヘルプの作成ツールなのである。だが、マイクロソフトはＸｐのある段階（WidowsXpSP2あたり）からＨＴＭＬヘルプの仕組みが深刻な「基本ソフトの脆弱性」ということで、自由に使えなく

してしまった。
　そりゃそうである。ＨＴＭＬヘルプは、これから他のソフトが起動できる、インターネットルートで外部からファイルを得たりできるというのだから、セキュリティから見たら危険極まりない機能なのである。著者としてはこの機能の使用をやむなしとしてあきらめたものだ。

タイトルバーのアイコンにメニューを追加する

　デルファイで作る各種ツールは軽いものばかりで、インターフェイスも最小とするのが多い。ほとんどが使い捨て。その場で必要な処理をしてしまえば、それっきり。だが、クライアントからの依頼で作って提供する場合は、ある程度インターフェイスをきちんとしなければならない。
　インストールはないのがほとんどである。フォルダにExeとreadme、Mhtだけというのが多い。アンインストールはフォルダごと削除する。しかし、ある程度汎用ツールであることからせめてデスクトップにアイコン作成ぐらいは欲しいという。そこで、この機能をタイトルバーのアイコンで右クリックしたときのメニューから実行できるようにしてある。
　次のようにしている。

```
Unit1.pasのimplementationの前に、
const MyMenu_ID=100; を宣言する。

//デスクトップにアイコンを作る(uses: ActiveX,ShlObj,ComObj,Registry)
//Flag=True:設定 False:削除
procedure RegistDeskTopIcon(ExeFullname,ExeTitle: string; Flag: Boolean);
var
  Reg: TRegistry;
  Persist: IPersistFile;
  SLink: IShellLink;
  Path: String;
  LinkName: array [0..255] of WideChar;
  AP_ExeFileName: string;
  AP_Name: string;
begin
  // ディスクトップのパス取得
  Reg:= TRegistry.create;
  AP_ExeFileName := ExeFullName;
  AP_Name := ExeTitle;
  try
    Reg.Rootkey := HKEY_CURRENT_USER;
    Reg.Openkey('\Software\Microsoft\Windows\CurrentVersion\'+
      'Explorer\Shell Folders', false);
    Path := Reg.ReadString('Desktop') + '\' + AP_Name + '.lnk';
  finally
    Reg.Free;
  end;

  // ショートカットの作成
  SLink := CreateComObject(CLSID_ShellLink) as IShellLink;
  SLink.SetPath(PChar(AP_ExeFileName));
  SLink.SetWorkingDirectory(PChar(ExtractFileDir(AP_ExeFileName)));
  SLink.SetDescription(PChar(AP_Name));
  MultiByteToWideChar(0,0,PChar(Path),-1,LinkName,256);
  Persist := SLink as IPersistFile;
```

```
  if Flag=True then begin
    Persist.Save(LinkName, true);
  end else begin
    DeleteFile(LinkName);
  end;
end;

procedure TForm1.AppMessage(var Msg: TMsg; var Handled: Boolean);
const
  BufferLength = 255;
begin
  //システムメニュー追加
  if (Msg.Message=WM_SYSCOMMAND) and (Msg.wParam=MyMenu_ID)
then begin
    RegistDeskTopIcon(Application.Exename,Application.Title,True);
  end;
end;
```

全角スペースを含んだTrim

全角漢字のスペースも含めてTrimをしたいことは多い。

```
//漢字スペースも含めたtrim
function trimW(strs: widestring): widestring;
var
  sl,i,idx: integer;
begin
  Result := '';
  sl := length(strs);
  if trim(strs)='' then begin
    exit;
  end;
  idx := 0;
  for i:=1 to sl do begin
    if posW(strs[i],WideString('　'+TABS))=0 then begin
      idx := i;
      break;
    end;
  end;
  if idx>0 then begin
    strs := copy(strs,idx,length(strs));
  end;
  idx := 0;
  sl := length(strs);
  for i:=sl downto 1 do begin
    if posW(strs[i],WideString('　'+TABS))=0 then begin
      idx := i;
      break;
    end;
  end;
  if idx>0 then begin
    strs := copy(strs,1,idx);
  end;
```

```
 Result := strs;
end;
```

ユニコード対応の posW と PosWP

posWはposのWidestring版である。ターゲットがWidestringであればposでもそのままWidestringでの文字位置が返るのだが、同じことを明示的にしているものだ。

また、posWPは、idxを持ち、指定の文字位置以降から探す処理をする。

```
//Unicode対応pos
function posW(subStr,Str: WideString): integer;
 //文字列位置検索（インターネットで公開されていたモジュール）
 function StrPosW(Str, SubStr: PWideChar): PWideChar;
 asm
        PUSH EDI
        PUSH ESI
        PUSH EBX
        OR EAX, EAX
        JZ @@2
        OR EDX, EDX
        JZ @@2
        MOV EBX, EAX
        MOV EDI, EDX
        XOR AX, AX
        MOV ECX, 0FFFFFFFFH
        REPNE SCASW
        NOT ECX
        DEC ECX
        JZ @@2
        MOV ESI, ECX
        MOV EDI, EBX
        MOV ECX, 0FFFFFFFFH
        REPNE SCASW
        NOT ECX
        SUB ECX, ESI
        JBE @@2
        MOV EDI, EBX
        LEA EBX, [ESI - 1]
 @@1:    MOV ESI, EDX
        LODSW
        REPNE SCASW
        JNE @@2
        MOV EAX, ECX
        PUSH EDI
        MOV ECX, EBX
        REPE CMPSW
        POP EDI
        MOV ECX, EAX
        JNE @@1
        LEA EAX, [EDI - 2]
        JMP @@3

 @@2:    XOR EAX, EAX
```

```
  @@3:     POP EBX
           POP ESI
           POP EDI
  end;
var
  pwc1,pwc2,pw0: PWideChar;
  p: integer;
begin
  pwc1 := PWideChar(str);
  pwc2 := PWideChar(subStr);
  pw0 := StrPosW(pwc1,pwc2);
  p := integer(pw0-pwc1);
  if (p<0) or (p>length(str)) or (p+length(substr)>length(str)) then begin
    Result := 0;
  end else begin
    Result := p+1;
  end;
end;
//Unicode対応pos
//idxはStr[idx]を意味し１～
function posWP(subStr,Str: WideString; idx: integer): integer;
var
  wstr: WideString;
  pwc1,pwc2,pw0: PWideChar;
  p: integer;
begin
  Result := 0;
  if (Str='') or (subStr='') or (idx<0) or (length(str)<=idx) or
    (idx+length(subStr)>length(str)) then begin
    exit;
  end;
  if idx=0 then begin
    Result := posW(subStr,Str);
    exit;
  end;
  //
  wstr := copy(str,idx+1,length(str));
  p := posW(subStr,wstr);
  if (p<=0) or (p>length(wstr)) or (p+length(substr)>length(wstr)) then begin
    Result := 0;
  end else begin
    Result := p+idx;
  end;
end;
```

ＣＳＶ１行の操作 parse

データを処理するときに、ＣＳＶはたいへん便利だ。
カンマで区切ったデータ列で改行で１レコードだ。
この１レコードの任意の位置の項目を取り出したり、セットしたりする。
１レコードにいくつの項目があるのかを得ることもできる。
ＣＳＶといってもなかなかむずしい問題がともなう。文字列に数値のように、３桁ごとのカンマを含んでいたりしたら項目位置が完全に違ってしまう。エクセルや、ワードでは

半角スペースやタブも区切り記号として認識してしまう。文字列にホワイトスペースが含まれるとこれまた位置が違ってしまう。ダブルクォティションやシングルクォティションも不安だ。

一般的に紹介されているのは、StringListにセットして処理するやり方なのだが、これは思ったような動きをするとは限らない。（内部的にはきちんとしたルールで作動しているのだが…）

このあたりは、自分で動かしながらたんねんに作りこんでいくのが基本と思えるので、以下はあくまでも参考である。

◆CSVの項目数を得る

```
function getCsvItemsCount(const srcTxt: string): integer;
```

◆CSVのなかから、Noの位置の項目を得る

```
function Parse(const srcTxt: string; no: integer): string;
```

◆CSVのnoの位置に、strsをセットする

```
function setParse(const srcTxt: string; no: integer; const keys: string): string;

//SetParseで挿入するべき前と後を求める
function ParsePos(const srcTxt: string; no: integer; var smae: string;
  var sato: string): boolean;
var
  p,i,l: integer;
  src,mae2,mae: string;
  flag: boolean;
begin
  smae := '';
  sato := '';
  Result := False;

  //noの不正
  if no<1 then begin
    smae := srcTxt;
    exit;
  end;

  //ソースが''
  l := length(srcTxt);
  if l=0 then begin
    smae := StringOfChar(',',no-1);        //<--ソースが''
    exit;
  end;
  src := DelQ(srcTxt);
  if src='' then begin
    smae := StringOfChar(',',no-1);        //<--デルクオートの結果ソースが''
    exit;
  end;

  //２文字以上の長さがある
  p := pos(',',src);
  if p=0 then begin
    //カンマがない
    if No>1 then begin
      smae := src+StringOfChar(',',no-1); //<--Noが１以上なのにカンマが１つも
ない
      exit;
```

```
  end;
  exit; //<--Noが１で1つのItemが存在している
end;

//例[Edit1,2,"3 aa",401 abc,'522 223',6,7,8,, 9, 1 0]
//   1     2  3     4        5        6 7 8 9 10 11
//カンマ2つ目以上のケース
//-----------------------------------------------------
i := 1;
while p>0 do begin
  //最初の文字が"あるいは'なら、次の"or'をブロックの区切りとする
  flag := False;
  l := length(src);
  if l>1 then begin
    if src[1]='"' then begin
      mae2 := copy(src,2,l);
      p := pos('"',mae2); //ブロックの後ろの位置を得る
      if p>0 then begin
        mae := trim(copy(mae2,1,p-1));
        src := trim(copy(mae2,p+1,l));
        l := length(src);
        if l>1 then begin
          if src[1]=',' then begin
            src := trim(copy(src,2,l));
          end;
        end;
        flag := True;
      end;
    end else if src[1]='''' then begin
      mae2 := copy(src,2,l);
      p := pos('''',mae2); //ブロックの後ろの位置を得る
      if p>0 then begin
        mae := trim(copy(mae2,1,p-1));
        src := trim(copy(mae2,p+1,l));
        l := length(src);
        if l>0 then begin
          if src[1]=',' then begin
            src := trim(copy(src,2,l));
          end;
        end;
        flag := True;
      end;
    end;
  end;
  //"or'で得られないケース
  if flag=False then begin
    mae := trim(copy(src,1,p-1)); //maeは該当ブロック
    src := trim(copy(src,p+1,l));
  end;

  sato := src;
  if no=i then begin
    exit;              //<--NoでItemが見つかった
```

```
    end;

    inc(i);
    mae := DelQ(mae);
    smae := smae+AddQ(mae)+',';
    p := pos(',',src);
    if p=0 then begin //<--Noが最後またはそれ以降の項目
      l := no-i;
      if l<=0 then begin
        smae := smae+StringOfChar(',',l);
      end else begin
        smae := smae+sato+StringOfChar(',',l);
      end;
      sato := '';
      Result := True;
      exit;
    end;
  end;
end;
//
function ParseSub(const srcTxt: string; no: integer; var Er: integer): string;
  function DelQ(const srcTxt: string): string;
  var
    src: string;
    l: integer;
  begin
    src := srcTxt;
    Result := src;
    l := length(src);
    if l<2 then begin
      exit;
    end;
    //""と''の囲みをはずす
    if (src[1]='"') and (src[l]='"') then begin
      src := copy(src,2,l-2);
    end;
    l := length(src);
    if l<2 then begin
      exit;
    end;
    if (src[1]='''') and (src[l]='''') then begin
      src := copy(src,2,l-2);
    end;
    Result := src;
  end;
var
  i,p,l: integer;
  src,mae,mae2: string;
  flag: boolean;
begin
  Er := 0;
  Result := '';
```

```
//ソースが''
l := length(srcTxt);
if l=0 then begin
  Er := -2;          //<--ソースが''
  exit;
end;

src := DelQ(srcTxt);
if src='' then begin
  Er := -2;          //<--デルクオートの結果ソースが''
  exit;
end;
src := srcTxt;

//２文字以上の長さがある
p := pos(',',src);
if p=0 then begin
  //カンマがない
  if No>1 then begin
    Result := '';
    Er := -1;        //<--Noが１以上なのにカンマが１つもない
    exit;
  end;
  Result := src;     //<--Noが１で1つのItemが存在している
  exit;
end;

//例[Edit1,2,"3 aa",401 abc,'522 223',6,7,8,, 9, 1 0]
//   1    2 3    4       5       6 7 8 9 10 11
//カンマ2つ目以上のケース
//----------------------------------------------------
i := 1;
while p>0 do begin
  //最初の文字が"あるいは'なら、次の"or'をブロックの区切りとする
  flag := False;
  l := length(src);
  if l>1 then begin
    if src[1]='"' then begin
      mae2 := copy(src,2,l);
      p := pos('"',mae2); //ブロックの後ろの位置を得る
      if p>0 then begin
        mae := trim(copy(mae2,1,p-1));
        src := trim(copy(mae2,p+1,l));
        l := length(src);
        if l>1 then begin
          if src[1]=',' then begin
            src := trim(copy(src,2,l));
          end;
        end;
        flag := True;
      end;
    end else if src[1]='''' then begin
      mae2 := copy(src,2,l);
```

```
        p := pos('"',mae2); //ブロックの後ろの位置を得る
        if p>0 then begin
          mae := trim(copy(mae2,1,p-1));
          src := trim(copy(mae2,p+1,l));
          l := length(src);
          if l>0 then begin
            if src[1]=',' then begin
              src := trim(copy(src,2,l));
            end;
          end;
          flag := True;
        end;
      end;
    end;
    //"or'で得られないケース
    if flag=False then begin
      mae := trim(copy(src,1,p-1)); //maeは該当ブロック
      src := trim(copy(src,p+1,l));
    end;

    if no=i then begin
      Result := DelQ(mae);
      exit;             //<--NoでItemが見つかった
    end;
    inc(i);
    p := pos(',',src);
    if (p=0) and (no=i) then begin
      Result := DelQ(src);
      if src='' then begin
        Er := -1;
      end;
      exit;             //<--Noが最後の項目
    end;
  end;
  Er := -1;
  Result := '';         //<--Noまでのカンマがなかった
end;
//CSVのなかから、Noの位置の項目を返す
function Parse(const srcTxt: string; no: integer): string;
var
  Er: integer;
begin
  Result := ParseSub(srcTxt,no,Er);
end;
function ParseCount(const srcTxt: string): integer;
begin
  Result := getCsvItemsCount(srcTxt);
end;
//CSVの項目数を返す
function getCsvItemsCount(const srcTxt: string): integer;
var
  Er,i: integer;
begin
```

```
  if Trim(srcTxt)='' then begin
    Result := 0;
    exit;
  end;
  i := 1;
  Er := 0;
  while Er=0 do begin
    ParseSub(srcTxt,i,Er);
    inc(i);
  end;
  if srcTxt[length(srcTxt)]=',' then begin
    Result := i-1;
  end else begin
    Result := i-2;
  end;
end;
//CSVのnoの位置に、strsをセットする
function setParse(const srcTxt: string; no: integer; const keys: string): string;
var
  mae,ato: string;
  sts: boolean;
begin
  sts := ParsePos(srcTxt,no,mae,ato);
  if sts then begin
    Result := mae+AddQ(keys);
  end else begin
    Result := mae+AddQ(keys)+','+ato;
  end;
end;
```

Boolean と string の交換

Iniファイルに Booleanを保存・読み出しするときに使用している。

```
//Boolを文字列で返す
function strOfBool(flag: boolean): string;
begin
  if flag=true then begin
    Result := 'True';
  end else begin
    Result := 'False';
  end;
end;

//文字列からBoolにする
function strToBool(strs: string): boolean;
begin
  if UpperCase(strs)='TRUE' then begin
    Result := true;
  end else begin
    Result := false;
  end;
end;
```

文字列リストからダブりを除く

元のリストがあらかじめ整列している（ソートされている）のかどうかで方法が違ってくる。

また、整列されていない場合に、結果もソートされていない状態を望むのか、ということでも違ってくる。

元の並びを復元したい場合は、元の位置(index)を保持した上での処理が必要になる。

stringlistを使う場合、stringlist.Duplicates := dupIgnore; とダブりを禁止する状態で add していくことでダブりをとりさることができる。

indexを保持する場合は、文字列を固定長にして、その文字列の後ろに元のindexを文字列として結合して、ソートする。その後順に固定長の長さだけを次の行と比較しながらチェックをすればよい。次行の固定長の長さが現在行と同じであればダブりなので行削除する。最後までループしたところで、indexの箇所で並べ替える。最後にindex箇所と固定長にしたときの無駄な文字列を削除して終わればよい。

謝辞など

WindowsXp上のBorland社製Delphi5ProUP1で開発しました。
使用コンポーネントは、下記のとおりです。有用なツールのご提供に敬意を表します。

- XBiff180: by M&Iさま
  エクセルのデータファイルを読み込むのに使っています。
- TntUnicodeComponent: by TntWareさま
  ユニコードを保持するために使っています。
- ObjStreamV1.03: by VRAMの魔術師さま
  設定状態の保存と復帰で使用しています。
- LightReport251: by Ohtakaさま
  表印刷で使っています。
- 拙作のTateEdit（縦書きEdit）
- 拙作のi-JapanPost.ttf （カスタマーバーコード用フォント）

また、インターネットサイトでDelphiの貴重な情報を公開されている方がた、さらにメールでの相談にこころよく応じていただいた先輩方に心から感謝します。

最新版の取得

筆者のサイトを確認してみてください。
http://hp.vector.co.jp/authors/VA041673/

著者　矢萩光也

1947年生まれ
デザイン編集事務所を経て業務ソフト会社のサポートを17年担当
現在文字データと画像の入力加工専門会社に勤務
実践的なツールの作成は20余年間で多数

縦書きエディトとユニコード

2006年6月15日　初版発行

著　者　矢萩光也
発行者　YahagiOffice
発行所　有限会社インプット
184-0025 東京都新宿区住吉町6-10 小松ビル3F
TEL: 03-5363-4868
http://www2.odn.ne.jp/~cao16090/
印刷所　有限会社インプット

ⒸYahagi Mitsuya 2006 Printed in Japan