Tiny Basic for Windows でのユニコードの取扱い

tbasic.org [*1]

[2013 年 10 月版]

Ver. 1.2 より Tiny Basic for Windows がユニコード（Unicode）対応になりました。ここではその考え方，使い方について説明します。ユニコードについては，別に簡単な説明文書がありますので，必要な場合，そちらを参照してください。

1　ユニコード対応とは

Tiny Basic for Windows（以下 tbasic と表します。）がユニコード対応になったとは，tbasic の処理がすべてユニコードを基本として行われることを意味します。外部的にはユニコード形式のファイルを扱うことになりますし，内部的にもテキスト処理の基本が。ユニコードをもとに行われることになります。

1.1　扱う外部ファイル

tbasic で扱うファイルは，プログラムやプログラムで作成するテキストファイルですが，この保存形式の基本がユニコードとなります。世界には色々な文字符号化方式がありまが，ユニコードは世界中の文字を一つの体系で扱おうとするものです。ですから，ユニコード形式のファイルでは，日本語に限らず，世界中の多くの言語を混在して使うことができます。ですから tbasic では色々な言語を扱うことができることになります。

ユニコードでのファイル形式は主として次の 3 つがあります。

- UTF-8
- UTF-16
- UTF-32

これらの違いは具体的なファイルの保存形式の違いで，それによって保存される内容はいずれも等価です。ですから，実質的にはどの形式で保存しても同じ内容が保存されますし，また，互いに変換することもできます。tbasic でのプログラムや外部ファイルはこれらのどの形式でも扱うことができます。

これに対して，従来から日本のコンピュータで使われてきた漢字などの，ファイルへの保存形式として次の 3 つがあります。

- Shift-JIS
- JIS
- EUC

このうち，Shift-JIS は Windows の PC で使われている形式です。JIS はもっとも古い日本語の保存形式ですが，メールの保存形式として現在も使われています。また，EUC は Unix や Linux で以前は広く用いられていたものですが，最近ではそれに代わりユニコードの形式の一つである UTF-8 が使われることが多くなっています。

[*1] http://www.tbasic.org

ユニコード形式のファイルでは，これらの形式で保存される文字は含まれますが，実は漢字に限った場合でも，ユニコードの方が多くの文字を扱うことができます。

例えば，土に口を重ねた「吉」は Shif-JIS などでは扱うことができませんが，ユニコードでは扱うことができます。実際，Shif-JIS では，士に口を重ねた「吉」しか扱えません。このように 日本語を使う目的だけであってもユニコードを使う意義はあります。

現在，コンピュータの世界では，広くユニコードが使われつつありますが，Windows のコンピュータ では，外部ファイルの日本語形式としては，現在も Shift-JIS が広く使われています。また，tbasic も Ver.1.2 より前のバージョンでは Shift-JIS しか扱えませんでした。しかし実は，Windows の PC での内部的処理には比較的古くからユニコード（UTF-16）が使われていました。[*2]

今後も tbasic は，従来との互換性の維持のため，Shift-JIS を使うことができますし，普通は Shift-JIS を使います。しかし，ユニコードやそれ以外の方式も使えます。そして場合によってはユニコードを使うべき状況もあるでしょう。

1.2 内部処理

tbasic では色々な符号化方式の外部ファイルを扱うことができますが，内部的には，ファイルを読み込んだ時点で，ユニコード（UTF-16）に変換して，内部処理をします。

これに対して，従来は Shift-JIS を基本として内部処理をしていました。ですから，従来のプログラムと Ver.1.2 以降のプログラムでは違いがあり，同じ動作をさせるにはいくつか修正が必要となることもあります。

この修正法については，以下の節でより具体的に説明しますが，関係するところは漢字や半角文字のなどの文字列の扱いだけです。文字列処理以外は従来のものとの違いはありません。ですから，殆んどのプログラムがそのまま動作するでしょう。

まとめると，

- ユニコード対応とは，外部ファイル，内部処理の基本がユニコードとなること。
- 外部ファイルの読み書きでは，ユニコードを含めて，色々な符号化方式が使える。
  従来からの Shift-JIS の他，UTF-16, UTF-8, UTF-32, JIS, EUC などが使える。
- 文字の内部処理はユニコードとして処理される。
- 漢字処理のプログラムは，ユニコード対応と Shift-JIS 対応では動作が異なり，
  Shift-JIS 対応プログラムは同じ動作をさせるには，一部変更する必要があることもある。

となります。

[*2] 実際，1993 年に発表された Windows NT 3.1 でユニコードが採用されました。

2　ユニコードでの文字

ユニコード文字は U+0000 から，U+10FFFF まで番号付けられていて，その各々が 1 文字です。１文字ということで言えばそれらの各々文字について区別はありません。これはある意味で当然な感じですが，従来の日本語処理と比較すると，実はかなり大きな違いです。

従来方式で日本語を扱う場合，半角 1 文字，全角 1 文字といった区別があります。そしてそれらを文字として数える場合，全角，半角によって扱いが異なりました。例えば，「あいうえお」という文字列は全角 5 文字ですが，文字列の長さを量るとき半角を基準として量れば，半角１０文字として数えられます。これは実際にコンピューターのメモリーの使用の量でもありました。実際半角 1 文字は 1 バイトを使い，全角 1 文字は 2 バイトを使って内部的にも表現されていました。ですから，「Ａ A」（全角Ａと半角 A）の文字数は，半角基準で 3 文字となります。

しかし，ユニコード対応の場合，このような内部的な区別が無くなります。実際 UTF-16 で表現する場合，半角文字の「A」と全角文字「Ａ」が内部的に使うメモリーは共に，２バイトつまり１６ビットです。つまり，ユニコードでは半角・全角の区別は文字の違いであり，それを格納するメモリーの大きさに違いはありません。

従来から全角文字と半角文字を含んだ文字列を処理する場合，この問題に関連してが煩雑な処理を行わなければなりませんでした。例えば文字列 A$の長さを調べるとき，Len(A$) は半角として数えた場合の値を返しました。全角半角まで考慮した文字数を返すには別の関数 KLen(A$) を使う必要がありました。ただ勿論これは日本語の漢字のみに対応するもので，他の文字，中国語やハングルなどには対応していません。

これに対して，ユニコード対応の場合，文字列に半角・全角の区別は元々ありませんから，いわゆる半角文字と全角文字を混在した文字列を使う場合，それらはすべて一文字として数えられます。例えば次のようになります。

文字列	従来方式：Len(A$)	ユニコード対応：Len(A$)
abc ＡＢＣ	9	6
1 月 1 日	6	4

Mid$, Left$, Right$ などの文字列処理もすべてこの方式になります。ですからこれに関連する処理を行っているプログラムは修正する必要があります。しかし，文字列処理の考えはむしろ単純になります。

まとめると

- ユニコード対応では文字を数える場合，半角・全角の区別はない。

となります。

3　ユニコード関連の関数

今回ユニコード対応になったことに関連して新たにサポートされた関数，仕様が変更になった関数をあげます。詳細は対応するヘルプをご覧ください。

3.1　仕様変更になった関数

次の関数は仕様が変更になりました。

Len，Left$，Mid$，Right$，InStr

これらはユニコード対応になったことにより全角半角等の区別なくユニコード文字としての対応した処理をします。

例 **3.1.**

- Print Len("半角 ABC と全角文字ＡＢＣ") の実行結果:　13
- Print Left$("123 １２３半角全角",5) の実行結果:　123 １２
- Print Mid$("日本語の文字",2,3) の実行結果:　本語の
- Print Right$("半角全角 ABC ＡＢＣ",4) の実行結果:　ＣＡＢＣ
- Print InStr(1,"全角１２３半角 123","12") の実行結果:　8

3.2　新たにサポートされた関数

新たにサポートされた関数として以下があります。

AcsW，AscU，ChrW，ChrU，Normalize

AscW，ChrW は対応する Visual Basic の関数と同じものです。AscW は，文字のユニコード番号を返します。ChrW ユニコード番号に対応するユニコード文字を返します。ここで，AscW，ChrW は 16 ビット文字の範囲で処理をします。例えば ChrW で扱える番号は U0000+〜UFFFF までです。また AscW もそれに対応した文字しか番号を返しません。

AscW などが返す数値は 10 進法で返しますが，Hex$を使えば，ユニコード番号で良く使われる 16 進表示で表すこともできます。同様に，&h を使うと，16 進表示で ChrW の入力も可能です。

例 **3.2.**

- Print AscW("A") の実行結果:　65
- Print Hex$(AscW("叱")) の実行結果:　53F1
- Print ChrW(&h53F1) の実行結果:　叱
- Print Hex$(AscW("眞")) の実行結果:　771E
- Print ChrW(&h771E) の実行結果:　眞
- Print Hex$(AscW("真")) の実行結果:　771F

- Print ChrW(&h771F) の実行結果: 真
- Print Hex$(AscW("瀦")) の実行結果: 7026

AscU と ChrU は AscW，ChrW と似たものですが，tbasic 独自の拡張関数です。AscU と ChrU は 16 ビットを超える番号にも対応します。16 ビットの範囲では，AscU と ChrU はそれぞれ，AscW，ChrW と同じ動作をします。

AscU(A$) は文字列 A$の先頭文字のユニコード番号を返します。ChrU(N) はユニコード番号 N の文字を返します。

例 **3.3.**

- Print Hex$(AscU("叱")) の実行結果: 20B9F
- Print ChrU(20B9F) の実行結果: 叱

Normalize(A$) は A$ の C 型正規化形式を返します。これは，結合文字を含んだ文章を結合した結果の文章に変換します。

例 **3.4.**

- Normalize("「にっほ゜んし゛ん」") は「にっぽんじん」を返します。

4 符号化方式関係

色々な符号化方式をサポートするようになりました。関連する変更事項等をまとめます。詳細は対応するヘルプをご覧ください。

4.1 プログラム

プログラムは色々な符号化方式が使えます。日本語の場合，Shift-JIS，JIS，EUC，UTF-16，UTF-8 などが使えます。「開く」は自動判別ができますが，失敗した場合，指定して下さい。「保存」は指定しなければ Default 符号化方式（Shift-JIS）ですが，指定すれば色々な符号化方式で保存が可能になります。

入力出力の符号化方式に関わらず，エディター上では，ユニコードで処理されます。ですから，ユニコードで表現できても Shift-JIS で表せない文字を含むプログラムは，ユニコードで保存する必要があります。tbasic 同梱の Sample プログラムにある，Helloworld.tbt は 色々な言語による hello world ですが，Shift-JIS では表せない文字が多く含まれています。これらの文字を正しく保存するために，このプログラムは UTF-16 で保存されています。

4.2 ファイル

ファイルは指定しないと Default 符号化方式になりますが，指定すれば色々な符号化方式のファイルが読み書きできます。

4.3 符号化指定コマンド

SetEncoding

SetEncoding は種々の符号化方式を指定するコマンドです。

4.4 サロゲートペアについての扱い

ユニコードは世界中の文字を一つの体系として表示するプロジェクトですが，最初はすべての文字を 2 バイト（１６ビット）で表すことを想定して規定されました。しかし，その後拡張され現在では２１ビットを使って表現されています。UTF-16 は１６ビットを基本とする内部処理の方法ですが，２１ビットで番号付けされた文字をすべて１６ビットでは表現できません。そこで，UTF-16 では，いくつかの文字は 2 文字で表現しています。このような扱いの文字は **サロゲートペア**と言われています。つまり，サロゲートペアで表される文字 1 文字は，UTF-16 の内部表現では 2 文字の扱いになります。

例えば，「叱」（口へんに七）という文字[*3]は，U+20B9F で指定される文字です。「叱」（口へんに匕）とは別字です。UTF-16 では U+FFFF を超えるものは 1 文字で表すことはできません。実際はコンピューター内部では

```
叱=ChrW(&hD842) + ChrW(&hDF9F)
```

と 2 文字で表現され 4 バイトになります。現在の .NET 系の言語での取扱いは Len や Mid 関数は内部表現で量るのが基本になっていますから，Len("叱") =2 と返すことになります。これはある意味大変煩雑なことになります。

そこで，tbasic では，ユニコードとしての 1 文字は 1 文字と扱うことにしました。つまりサロゲートペアは 1 文字として扱う処理を特別に内部的に行います。

従って，tbasic では，

```
Len("叱") =1
```

と返します。このような処理は，文字列処理を行う，Len，Left$，Mid$，Right$，InStr などでも対応しています。ですから，ユーザーは，サロゲートペアの存在を気にせずにプログラムを書くことができます。

まとめると

- Tiny Basic for Windows での文字列処理ではサロゲートペアについて考えなくて良い。

となります。

5 まとめ

Tiny Basic for Windows がユニコード対応になったことにより，色々な言語の文字列の処理が可能になりました。まだまだ不完全なところもありますが，色々なところで活用できれば良いと思っています。

[*3] この文字「叱」は新常用漢字にある文字ですが，次にある「叱」は常用漢字表にはありません。