
tbasicでのユニコード (Unicode) の取扱い

Ver. 1.6 用 tbasic.org ^{*1}

[2023 年 03 月版]

Tiny Basic for Windows(tbasic) は Ver.1.2 よりユニコード (Unicode) 対応になりました。また, Ver.1.6 より, ユニコード対応が強化されました。ここでは tbasic でのユニコード機能を活用したいと考える方へ, 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, 別により詳しい説明文書「ユニコードへ」がホームページにありますので, 必要な場合, そちらを参照してください。

^{*1} <https://www.tbasic.org>

目次

第 I 部 基礎知識編	3
1 ユニコードベースとは	3
2 テキストファイルとエンコーディング	3
3 漢字のエンコーディング	4
3.1 基本漢字	4
3.2 補助漢字・拡張漢字	4
4 ユニコード (Unicode)	5
4.1 始まり	5
4.2 Unicode Ver.1(1991)	6
4.3 Unicode Ver.2(1996)	6
4.4 Unicode Ver.3(2000)	7
4.5 Unicode Ver.4(2003) とその後	7
4.6 Unicode の成果	8
5 tbasic で扱える外部ファイル	9
第 II 部 操作編	10
6 プログラム編集	10
6.1 初期エンコーディング	10
6.2 新規プログラムを作成する	10
6.3 プログラムを開く	11
6.4 プログラムを保存する	12
7 実行環境	13
7.1 符号化指定コマンド	13
7.2 Encoding の確認	13
7.3 外部ファイルからのデータ入力	13
7.4 外部ファイルへのデータ出力	14
8 文字列	15
8.1 tbasic での文字列	15
8.2 tbasic での文字列への入力	16
8.3 文字列操作	18

第 I 部

基礎知識編

1 ユニコードベースとは

コンピューターでの処理は、外部からデータを入力し、それをコンピューター内部で処理し、その結果を出力するものです。tbasic がユニコードベースになったとは、tbasic 内部での文字処理がすべてユニコード形式で行われることを意味します*2。普通に計算処理を行う場合には、内部での文字処理がどのような形式で行われているか、意識する必要はありません。しかし、文書を扱う場合にはユニコードについて、ある程度意識して処理をする必要があります。

また、外部のデータとしては、ユニコード形式の他、種々のタイプのデータが実際に使われていますから、それらのことも考慮する必要があります。

tbasic では、外部データは内部にデータ取り込む時点で、ユニコード形式に変換して処理します。そして、それを必要に応じて、種々の形式に変換して出力します。このため、tbasic では、外部との入出力について種々の形式をサポートしています。

2 テキストファイルとエンコーディング

コンピューターで扱うファイルの分類として、テキストファイルとバイナリファイルがあります。テキストファイルは書式なし文書を保存するファイル、バイナリファイルはそれ以外のファイルと言われています。アルファベットだけを保存するのであれば、テキストファイルは単に改行付きアスキー文字の行の集まりで、単純な構造を持っています。

しかし、漢字等を保存する場合は、対応した形式に従う必要があり、単純ではありません。更に、漢字等を保存する形式は種々あり、それぞれに異なっています。ですから、そのようなテキストファイルを正しく読むためには、それに対応した形式の理解が必要です。この形式をエンコーディング（符号化方式、Encoding）と言います。

また、エンコーディングを規定するためには、対応する文字の集合の規定と、その各々の文字の番号付けが必要になります、これを符号化文字集合と言います。つまり、漢字の文書を扱う場合、

漢字の文書を扱う場合、

- ・符号化文字集合と
- ・それに対応するエンコーディング

が必要

となります。

*2 Windows では内部的に UTF-16LE で処理を行っていますが、tbasic を扱う際、このことは特に意識する必要はありません。

3 漢字のエンコーディング

3.1 基本漢字

日本のコンピューターでの漢字の利用は、漢字の符号化文字集合の規定から始まります。

1978年、日本漢字の符号化文字集合、JIS 基本漢字 (JIS X 208) が規定されました。基本漢字は漢字を 94(区) × 94(点) の表 (面) に対応付け、全体で 6879 文字が登録されています*3。これは、日常での使用にはほぼ十分なものです。1973年に規定された ISO/IEC 2022 の技術を基本漢字に適用すると、日本語のエンコーディングを構成することができます。

1970年代の漢字利用は、統一的な規格ではなく、個々のコンピューターでのエンコーディングの個別実装に限られていました。1980年代に入ると、パーソナルコンピューターや Unix の普及により、統一的な規格が求められようになりました。

それ中で規定された、基本漢字のエンコーディング規格としては次の3つがあります。

- Shift-JIS
- EUC
- JIS

Shift-JIS はパソコン用の符号化方式で、1982年に MSDOS 等を実装され、そして後に、Windows や Macintosh の内部エンコーディングとして、長く使われました。外部ファイル用のエンコーディングとしては現在でも広く使われています。

EUC は Unix 上で主に使われた符号化方式で 1985年に規定されました。

JIS は電子メール等で利用されている 7ビット符号化方式です。正式には 1993年 RFC1648 で規定されましたが、1986年頃から使われています。そして、電子メールの送受信のエンコーディング方式としては、現在でも広く使われています。

このように JIS 基本漢字のエンコーディングは、1980年代に標準化が進み、

1980年代後半のコンピューターでは、基本漢字が標準的に使える

ようになりました。

3.2 補助漢字・拡張漢字

1990年になると、補助漢字 (JIS X 0212) が規定されました。これは、基本漢字 (JIS X 208) に無い漢字等を補ったもので、アルファベット 245 文字、漢字 5801 文字が含まれます。基本漢字とは別な 94(区) × 94(点) の表で構成されています。

EUC は、補助漢字が使えるように拡張が行われ、EUC-JP として規定されましたが、広く使われるようにはなりません。また、Shift-JIS は、補助漢字を使うための拡張はなされませんでした。

更に、2000年には、拡張漢字 (JIS X0213) が規定されました。これは、基本漢字を拡張したもので、2面

*3 ここで現れる 94 と言う数は、元々は、アスキーコード表での印刷可能文字「!」から「~」までの個数に由来します。

94区94点（11223文字）で構成されています。第1面は基本漢字の表（面）を基として、基本漢字表で未使用の部分に文字を加えています。第2面は第1面と同じく94区94点で構成されていますが、実際に規定されているのは26点区です。拡張漢字は、基本漢字、補助漢字を含み、エンコーディングが構成しやすい構造をしていました。

これにより、拡張漢字については、Shift-JIS、EUCに対して、それぞれ拡張が行われ、それぞれ、Shift-JIS2004、EUC-JIS-2004として規定されました。しかし、これらについても広く使われるようにはなりませんでした。

実は現在、よく使われているエンコーディングは、Shift-JIS、EUC、JISで、いずれも基本漢字が基本です。コンピューター上で標準的に、補助漢字、拡張漢字が使用できるようになるには、規定から少し時間が必要でした。

4 ユニコード (Unicode)

4.1 始まり

符号化文字集合の拡張に伴って、その利用のためのエンコーディングが規定されても、そのエンコーディングが実際のコンピューターで広く使われるようになるとは限りません。

実際のコンピューターでの使用は、コンピューター内部でのエンコーディングでの使用を伴う必要があります。しかし、コンピューター内部でのエンコーディングの更新・変更は、OS内部でのテキストの取り扱い処理の更新・変更が必要になるからです。

JIS X 208、JIS X 0212、JIS X 0213はいずれも94×94の表として定義されています。そして、これらの規定は、ISO/IEC 2022の技術の利用を想定しています。これは複数の表を切り替えてそれぞれの表の文字を使用する方法です。多くの表を用意すれば、それを切り替えることで、多くの文字を使用することができず。しかし、それには煩雑な切り替え処理が必要です。更に、これらのエンコーディングは、1文字が1バイト或いは2バイト、また、場合によっては3バイト必要となるなど煩雑な処理が必要でした。

このような状況は日本に限らず、世界的な問題でした。一方で、世界のグローバル化に伴い、多国語を同時に利用したいとの要望はますます高まっています。しかし、この方法では、使用文字を増やすたびに、OS内部のエンコーディングを修正していく必要があり、そのたびに煩雑さが増し、速度の低下が起り、困難が伴います。

このことへの対応として、小規模の符号化文字集合を複数個使って、多くの文字を表すのではなく、「世界中の文字を含む大きな符号化文字集合を一つ定めて、それを使ってコンピューターで多国語の文字を扱えるようにする」方法が検討されるようになりました。

1980年代後半、このような目的への試みへのプロジェクトが始まり、このコードは、Universal, uniform, uniqueなcodeの意味からUnicode（ユニコード）と言われるようになりました。

1988年には「Unicode Standardへの提案（Unicode 88）」が公開されました。

そこでは、

Unicode は ASCII コードの拡張であり、次の性質を持つ。

- Unicode は固定 16 ビットコードである。
- 世界中の文字と固定幅で 1 対 1 に対応する。

と主張しています。

4.2 Unicode Ver.1(1991)

1991 年には、多くの組織、企業が参加して Unicode 制定のための非営利国際的組織 Unicode Consortium が発足し、1991 年 10 月にはその仕様書”The Unicode Standard Version 1.0” が発表されました。これは、1988 年の提案に沿って、それを具体化したものでした。

そこでは、

- Unicode 番号は、先頭に U+ を付けた 16 ビットの固定長 16 進数 U+0000 から U+FFFF で表される。
- Unicode は、現在、過去の世界中の文書を表す十分な文字を含んでいる。
- Unicode では、どのような言語であってもエスケープまたは制御記号による切り替えなしで利用できる
- 固定幅のコードは、種々のテキスト処理がより簡単に可能である。

と述べられています。

Ver. 1.0 では、全体で 34,338 の文字等^{*4}が登録され、日本語では、JIS X 208 (基本漢字) , JIS X 0212 (補助漢字) が含まれました。

また、内部コードとして 16 ビットを利用する際、Endian ^{*5}を区別するものとして BOM(Byte Order Mark) が定義されました。

4.3 Unicode Ver.2(1996)

1996 年には Ver. 2.0 が発表されました。Ver. 1 の発表後、各国より多くの文字の登録の要請があり、Ver. 2 では多くの文字が追加され、合計 47,400 文字が登録されています。

Unicode は 16 ビットで表すとされていますが、この範囲では最大 $2^{16} = 65536$ までしか表すことができません。Ver.1 当時の想定ではこれでも十分とされていましたが^{*6}、Ver. 2 でこれを超える文字を登録するためのサロゲートペア (Surrogate pair) とされる拡大法を定めました。

これは、 1024×1024 の表を新たに登録可能にするもので、2 個の 16 ビット (=32 ビット) を使って表現されます。この構成法により、Unicode 番号は 0 から 10FFFF となりました。これは 16 ビットを超えるものです。ですから、標準的な方法でこの文字を表すと、2 バイト必要になります。この文字を使用することは、Unicode の最初の目標「固定幅で文字を表す」を修正するものでした。この領域は将来的に極めてまれな文字を割り当てる可能性を残すものとして、文字は割り当てられませんでした。

^{*4} 割り当てられたコードで、必ずしも文字以外のものも含まれます。

^{*5} バイトを並べる順序のこと。メモリは普通バイト単位で、番号付けされます。例えば、1 番地のバイト数を A 、2 番地のバイト数を B とします。これらの 2 バイト (1,2 番地) で数を表す場合、 $A \times 256 + B$ とする方法と $A + B \times 256$ とする方法があります。前者を Big-Endian、後者を Little-Endian と言います。

^{*6} Ver. 2 の段階でも未割り当てが 18000 もあり、将来予想される必要数を超えると述べられています。

また、8ビットエンコーディングとして、UTF-8が規定されました*7。また、UTF-16についても言及されています。

纏めると、

- 登録文字数 47,400
- UTF-16 でサロゲートペアの導入
- Unicode 番号は 0 から 10FFFF
- UTF-32 の予感

となります。

4.4 Unicode Ver.3(2000)

2000年にはVer. 3.0が発表されました。Ver. 3では57,709文字が登録されています。

UTFについて、明示的に定義がなされ、16ビットEncoding、UTF-16BE、UTF-16LEの定義が述べられています。

日本語関係では、半角、全角(Halfwidth, Fullwidth)の性質が定義され、半角カタカナや全角アルファベットが登録されています*8。

- | |
|---|
| <ul style="list-style-type: none">• 半角文字： アイウエオ• 全角文字： アイウエオ |
|---|

纏めると、

- 登録文字数 57,709
- UTF-8, UTF-16 の明確な定義
- 半角, 全角の定義

となります。

4.5 Unicode Ver.4(2003) とその後

2003年にはVer. 4.0が発表されました。

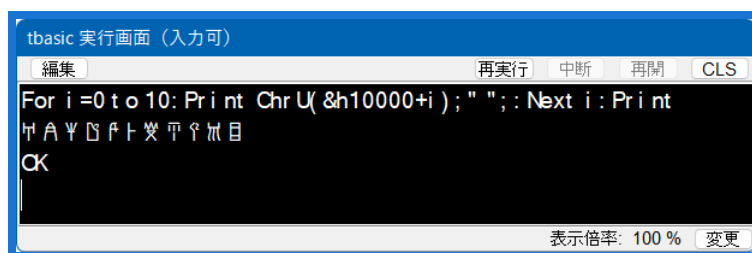
Ver. 4.0では、59,213文字が登録されています。特に、日本語関係では、JIS X 213(拡張漢字)が登録されています。

また、16ビットを超える領域のUnicode番号が実際に割り当てられています。例えば、U+10000からU+1005Dに古代ギリシアで使われた音節文字Linear B Syllabaryが登録されました。

*7 UTFはUnicode Transformation Formatに由来します。]

*8 Shift-JISでは、半角は1バイト、全角は2バイトとその文字を保存する際のメモリの量も意味しました。しかし、UnicodeではHalfwidth, Fullwidthは文字の幅を示す性質で、その文字を保存するためのメモリを意味するものではありません。

下の図は、tbasic でこの部分 10 文字*9を表示したものです。



このように、Ver. 4.0 では、16 ビットを超えた番号が実際に登録されました。これにより、「16 ビット固定幅で全ての文字を表すという」Unicode の当初の目標は果たされないことになりました。しかしその代わり、32 ビットエンコーディング UTF-32 が規定されました。UTF-32 では、すべての文字が Unicode 番号と直接対応付けられ、これを用いれば、固定幅での表現が得られることになります。

Ver.1.0 から Ver.4.0 を通じて次が規定されました。

- 登録文字数 59,213
- 8,16,32 ビット用 Encodeing UTF-8,UTF-16,UTF-32
- Endian 判定記号 Byte Order Mark BOM
- Unicode 番号を 0 から 10FFFF とする。
- 日本語関係では拡張漢字*10を含む

これらの、規定により、Ver. 4.0 で現在のユニコードの基本的枠組みが完成したと言えます。

その後更新が続けられ、最新版は Ver. 15(2022 September 13) です*11。149,186 文字が登録されています。Ver.1.0.0 は、全体で 700 ページ弱、Ver.15.0.0 は全体で 1000 ページ強にもなる文書です、

ユニコードが制定されると、コンピューター内部のコードとしてユニコードが使われるようになりました。現在では、Windows, Mac, Linux, Android 等が採用しています。

Windows で Unicode が採用される前は、内部コードは Shift-JIS を使っていました。この状況では、例えば、日本語の文章に「très bien」と言う言葉を含めることはできません。日本語に限ってみても、人名や地名などで使われる特殊な漢字、例えば鷗、鷗や吉や吉についてみると、Shift-JIS では鷗や吉を扱うことはできませんが、Unicode では扱うことができます*12。

4.6 Unicode の成果

ユニコードは世界中の文字を一つの文字集合で表し、それらを簡単に効率的に利用する方法の模索でした。Unicode Ver.1 が発表されて、約 30 年が経ち、その成果を纏めると次のようになります。

- ユニコードは世界中のコンピューターで使われている。⇒ Unicode は世界標準
- ユーザーは、一つのエンコーディングで世界中の文字を混在してコンピューターで使用できる。

*9 これらの特殊文字を表示するためには、それに対応したフォントが必要です。以下では、Noto Sans Linear B を使用しています。

*10 基本漢字、補助漢字を含みます。

*11 2023 年 3 月現在

*12 画面等に正しく表示するには対応するフォントが適切にインストールされている必要があります。

- 状況に応じて、UTF-8、UTF-16、UTF-32 を使うことができる。これらはすべてユニコードとして等価である。

例えば、Unicode を使うと、

パソコン, personal computer, persönlicher Rechner, Ordinateur personnel, 个人电脑, 개인용 컴퓨터

のように、外国語が混在する文章を扱うことができます。

5 tbasic で扱える外部ファイル

現在、コンピューターの世界では、広くユニコードが使われつつありますが、Windows では、外部ファイルの日本語形式として、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。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。しかし、Shift-JIS で使えない日本語の文字もありますし、可能なら、これからは、ユニコードを使うのが良いでしょう。

tbasic で扱える外部ファイルは上に挙げた通りですが、それらに対応する具体的エンコーディング (Encoding) は次の通りです。

表 1

tbasic での Encoding 名	CodePage	名前	説明
SJIS	932	shift_jis	日本語 (Shift-JIS)
EUC	20932	EUC-JP	日本語 EUC
JIS	50220	iso-2022-jp	日本語 JIS
UTF-8	65001	utf-8	Unicode(BOM 無し, UTF-8)
UTF-8BOM	65001	utf-8	Unicode(BOM 付, UTF-8)
UTF-16LE	1200	utf-16	Unicode(BOM 無し, UTF-16 Little Endian)
UTF-16BE	1201	unicodeFFFE	Unicode(BOM 無し, UTF-16 Big Endian)
UTF-16LEBOM	1200	utf-16	Unicode(BOM 付, UTF-16 Little Endian)
UTF-16BEBOM	1201	unicodeFFFE	Unicode(BOM 付, UTF-16 Big Endian)
UTF-32LE	12000	utf-8-32	Unicode(BOM 無し, UTF-16 Little Endian)
UTF-32BE	12001	utf-32	Unicode(BOM 無し, UTF-16 Big Endian)
UTF-32LEBOM	12000	utf-8-32	Unicode(BOM 付, UTF-16 Little Endian)
UTF-32BEBOM	12001	utf-32	Unicode(BOM 付, UTF-16 Big Endian)

第 II 部

操作編

6 プログラム編集

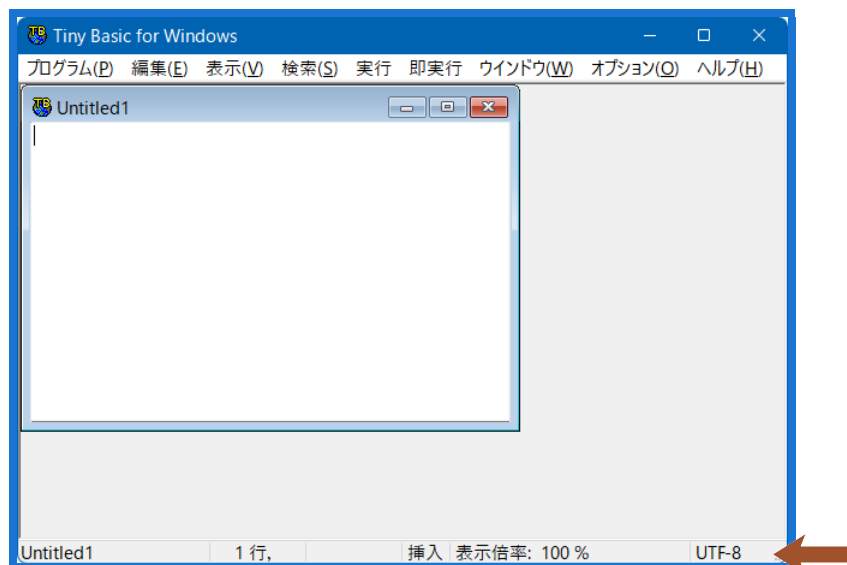
プログラムは編集画面で、記述・編集します。ここでは、プログラム編集でのエンコーディングの扱いについて説明します。

6.1 初期エンコーディング

初期エンコーディング (Default Encoding) はいくつかの場合で、自動的に利用されるエンコーディングです。初期エンコーディングは環境設定で指定しますが、推奨の値は UTF-8 です。UTF-8 以外を使いたい場合は、Shift-JIS でも良いかもしれません。

6.2 新規プログラムを作成する

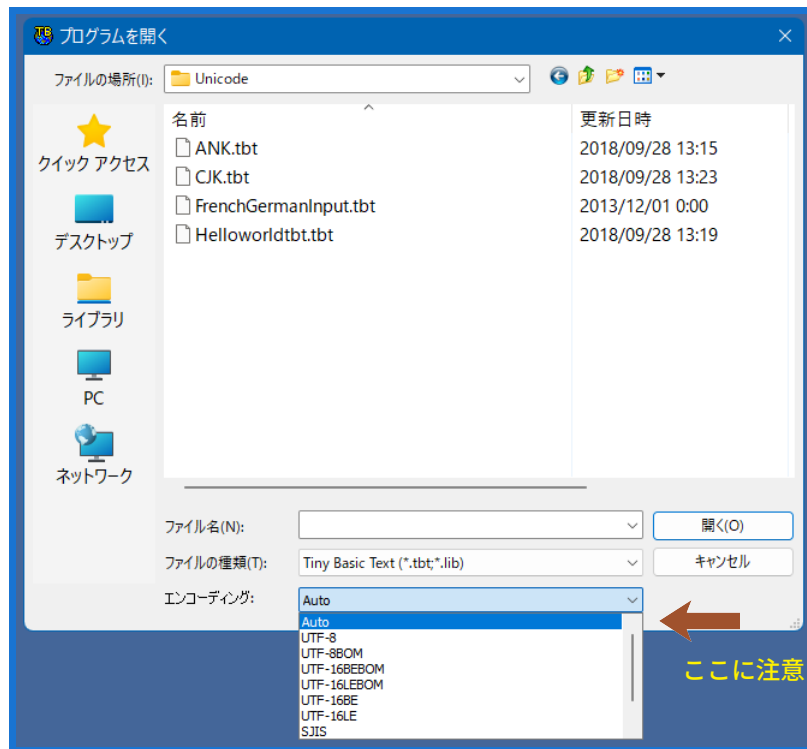
エディター画面から、プログラムの「新規作成」をすると、初期エンコーディングのテキストファイルとして作成されます。以下の画面にある例では、外部ファイルとして UTF-8 と保存するように作成されています。



ここに注意

6.3 プログラムを開く

エディター画面から、プログラムの「開く」をクリックし、ダイアログボックスを開き、開くプログラムを選択すると、同時にエンコーディングを選択できます。殆んどの場合、「Auto」で正しく開くことができますが、文字化けが起きる場合は、プルダウンメニューから、エンコーディングを具体的に指定して、開いてください。

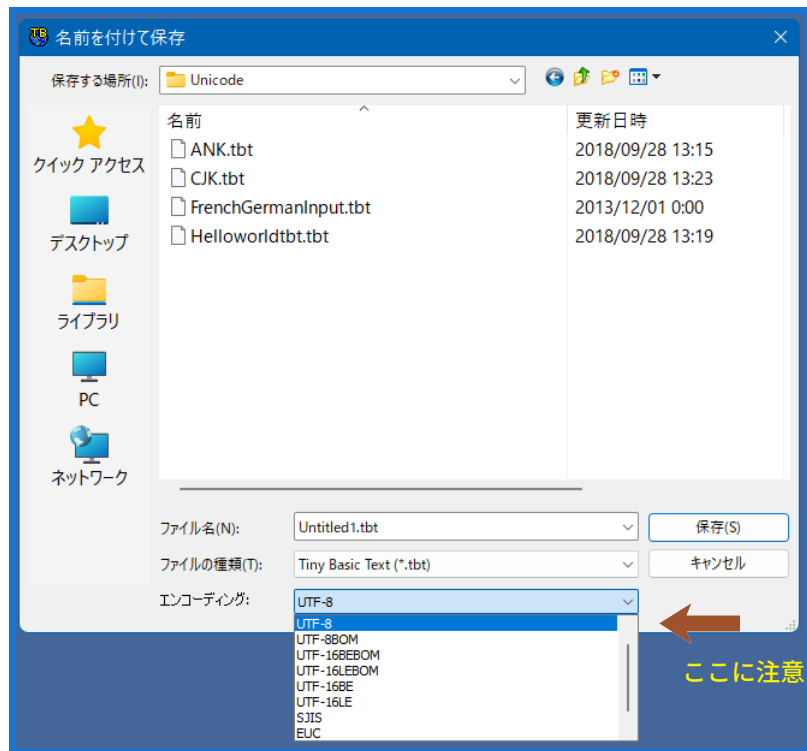


ファイルをエディター画面にドラッグアンドドロップして開く場合は、「Auto」のみになります。

6.4 プログラムを保存する

エディター画面から、「名前を付けて保存」をクリックし、ダイアログボックスを開き、保存するプログラムを選択と同時にエンコーディングを選択できます。

現在のエンコーディングがデフォルトはになりますが、プルダウンメニューから、エンコーディングを具体的に指定することもできます。



プログラムの「上書き保存」をした場合は、現在のエンコーディングで保存されます。

7 実行環境

7.1 符号化指定コマンド

初期エンコーディングは、環境設定で指定しますが、プログラムの中で初期エンコーディングを再設定することができます。

```
SetEncoding(Enc1)
```

SetEncoding は初期エンコーディングを Enc1 と再設定します。この再設定は、実行しているプログラムでのみ有効です。ここで Enc1 は、エンコーディング名または、コードページ番号です*13。

例 7.1. SetEncoding("SJIS")

初期エンコーディングを SJIS に指定します。

7.2 Encoding の確認

外部ファイルのエンコーディング名を確認したいことがあるかも知れません。このために、GetFileEncodingName 関数があります。

```
GetFileEncodingName(File1)
```

は、File1 の Encoding 名を返します。判定できない場合は、Unknown を返します。

例 7.2. GetFileEncodingName("file1.txt")

は、file1.txt のエンコーディング名を返します。

7.3 外部ファイルからのデータ入力

外部ファイルからの入力は、

```
ReadAllLines(file1)
ReadAllText(file1)
Open file1 for input as #1
Open file1 for append as #1
```

等で行いますが、この形での使い方では、file1 のエンコーディングを自動判別して、入力を行います。エンコーディングが判定できなかった場合は、初期エンコーディングで入力します。

多くの場合、この方法で期待通りの動作をしますが、場合によっては、文字化けが起こって上手く行かないこともあります*14。

*13 エンコーディング名は文字列、コードページ番号は数で指定します。コードページ番号は曖昧性があるため、できる限りエンコーディング名を指定することを推奨します。

*14 エンコーディングは、場合によっては間違った判定をすることがあります。例えば、Shift-JIS と EUC では、同じファイルで両方ともエンコーディングが正しいと見做せることがあります。この場合は、一方のエンコーディングでは正しく読めますが、他方のエンコーディングで読むと、不適な文になることがあります。

この場合は、いくつかのエンコーディングを指定して、試して正しいものを探す他ありません。

```
ReadAllLines(file1, Enc1)
ReadAllText(file1, Enc1)
```

の形式では、エンコーディングを Enc1 に指定して読み込みます*15。

例 7.3. `A$() = ReadAllLines(file1.txt, "EUC")`

`file1.txt` を文字配列 `A$()` にエンコーディング EUC で読み込む。

7.4 外部ファイルへのデータ出力

外部ファイルへのデータ出力は、

```
WriteAllLines(File1,A())
WriteAllText(File1,A())
Open file1 for output as #1
```

等で行いますが、この形での使い方では、`file1` の初期エンコーディングを利用して、文字配列 `A()` のデータを `File1` に書き込みます。

初期エンコーディングは `SetEncoding` を使っていつでも再設定できますから、どのようなエンコーディングでも利用できます。

```
WriteAllLines(File1,A(),Enc1)
WriteAllText(File1,A(),Enc1)
```

のように、直接エンコーディングを指定することもできます。

例 7.4. `WriteAllLines(file1.txt, A$(),"UTF-8BOM")`

配列 `A$()` を `file1.txt` にエンコーディング UTF-8BOM で書き込む。

*15 `Open file1 for ...` 形式の文ではこのような使い方はサポートしていません。

8 文字列

8.1 tbasic での文字列

tbody が内部的に扱う、文字・文字列はすべてユニコード文字・文字列です。ユニコードは、16 進数で、0 から 10FFFF の数で番号付けられています。ユニコードのコード番号を表すときは U+ を付けて表します。0 から FFFF は、元々 16 ビットでユニコードが始まったことから、U+0000 から U+FFFF と 16 進 4 桁で表すのが慣例となっています。この部分を基本領域と言います。それ以上 U+10000 から U+10FFFF を拡張領域と言います。

ユニコードは U+0000, . . . , U+10FFFF のように番号付けされた多国語文字の集まり。

U+0000 ~ U+FFFF を基本領域,
U+10000 ~ U+10FFFF を拡張領域と言う。

tbody では、これらのユニコード文字の並びが文字列です。

ユニコード文字は U+0000 から、U+10FFFF まで番号付けられていますが、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、多少注意が必要です。

現在でも、色々な場面で、半角文字、全角文字といった区別があり、使い方が違うことがあります。

全角・半角の違いは、元々は文字の幅を意味するものでしたが、コンピューターでは、ASCII で表される英数記号を半角、それ以外のいわゆる漢字を全角文字と言うこともあります。

エンコーディングの立場では、1 バイトで表される文字を半角、2 バイトで表される文字を全角とする場合もあります。Shift-JIS エンコーディングでは、JIS X 0201 部分を 1 バイト、JIS X 0208 部分を 2 バイトで表していました。これを内部コードとしたシステムでの文字列は、漢字は 2 バイト、英数記号は 1 バイトとして扱っていました^{*16}。

このようなシステムでは、文字列の長さ、それを表す内部メモリーの大きさが異なり、文字列の処理に煩雑さが伴いました。

一方ユニコードでは、半角文字と全角文字は文字としての数え方に区別はありません^{*17}。

これにより、それらの煩雑さが無く、簡単に扱うことができます。

つまり、

ユニコードでは文字を数える場合、半角・全角の区別はない。

これにより、文字列処理を、より簡単に行うことができる。

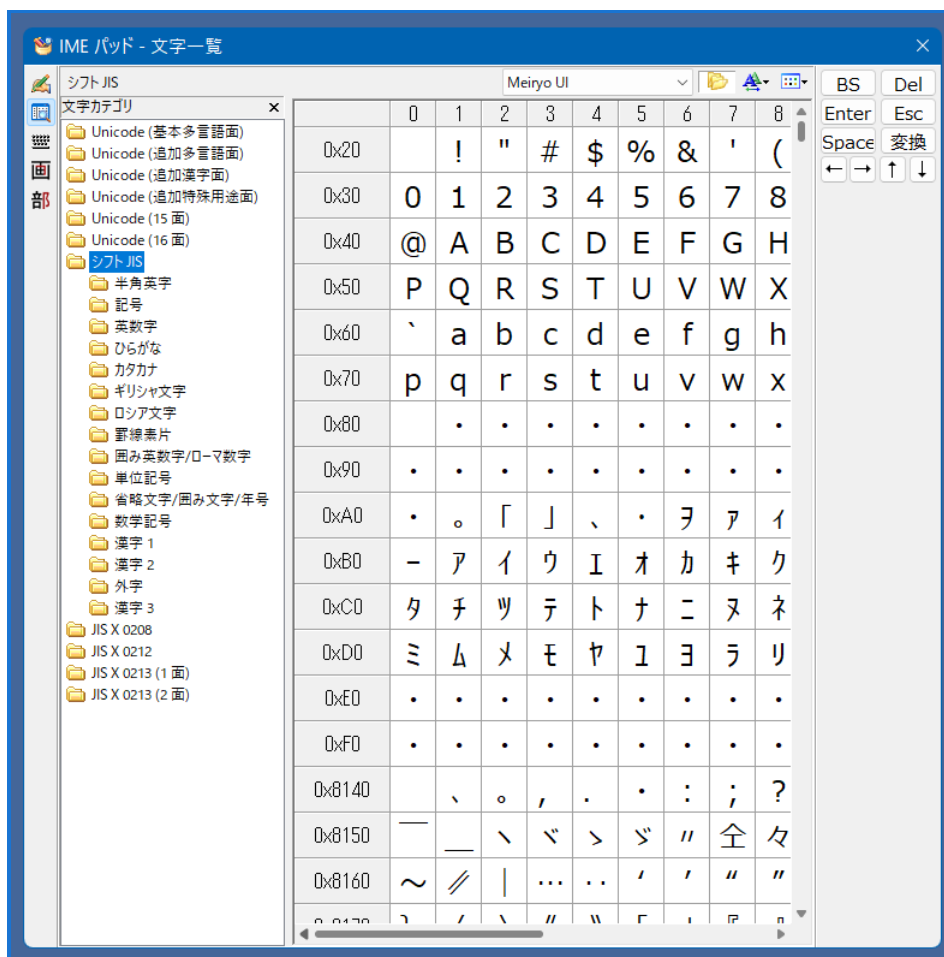
となります。

^{*16} tbasic も Ver.1.1 まではこの方式でした。

^{*17} ユニコードでの半角全角は文字のフォントの幅についての性質です。

8.2 tbasic での文字列への入力

tbasic のエディターはユニコード対応エディターですから、ユニコードをそのまま入力可能です。普通は、キーボードで IME を利用して入力しますが、IME パッドからも入力できます。



ユニコード番号が分かっている場合は、プログラムの中で、ユニコード番号から対応する文字を得ることができます。

tbasic ではコード番号から文字を得る関数として `AcsW`, `AscU`, `ChrW`, `ChrU` があります。

関数	引数の範囲	説明	逆関数
<code>Chr\$</code>	$0 \leq x < 128$	この範囲で ASC 文字 (ユニコード文字) を与えます。	<code>Asc</code>
<code>ChrW</code>	$0 \leq x < 65535$	この範囲でユニコード文字 (基本領域) を与えます。	<code>AscW</code>
<code>ChrU</code>	$0 \leq x < 1114111$	この範囲でユニコードを与えます。	<code>AscU</code>

`AscW`, `ChrW` は対応する Visual Basic の関数と同じもので、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。

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。

これらを使うと一覧等が簡単に作成可能です。

例 8.1.

```
For i=&h03B1 to &h03B1+24
```

```
  A$=A$+ChrU(i)
```

```
Next i
```

これを実行すると、

```
A$="αβγδεζηθικλμνξοπρστυφχψω"
```

となります。

例 8.2.

コマンド	実行結果
Print AscW("A")	65
Print Hex\$(AscW("叱"))	53F1
Print ChrW(&h53F1)	叱
Print Hex\$(AscW("眞"))	771E
Print ChrW(&h771E)	眞
Print Hex\$(AscW("眞"))	771F
Print ChrW(&h771F)	眞
Print Hex\$(AscW("瀧"))	7026
Print ChrW(&h7026)	瀧
Print Hex\$(AscU("叱"))	20B9F
Print ChrU(20B9F)	叱

8.3 文字列操作

文字列を操作するための関数やコマンドについて説明します。文字列の標準的な操作としては、文字列の長さや、部分文字列の取得、検索等があります。これらはそれぞれ、関数 Len, Left\$, Mid\$, Right\$, InStr で可能です。

関数名	使い方	意味
Len	Len(A\$)	A\$ の長さを返す
Left\$	Left\$(A\$,n)	A\$ の左から n 文字を返す
Mid\$	Mid\$(A\$,i,n)	A\$ の i 番目の文字から n 文字を返す
Right\$	Right\$(A\$,n)	A\$ の右から n 文字返す
InStr	InStr(n,A\$, B\$)	A\$ の n 番目の文字から、B\$を探し、検索開始位置からの文字数を返す

使い方は、標準的ですが、半角・全角の区別なく、1文字と数えることに注意してください。

例 8.3.

コマンド	実行結果
Len("半角 ABC と全角 A B C")	13
Left\$("123 1 2 3 半角全角",5)	123 1 2
Mid\$("日本語の文字",2,3)	本語の
Right\$("半角全角 ABC A B C",4)	C A B C
InStr(1,"全角 1 2 3 半角 123","12")	8

ユニコードでは、文字の種類に関わらず、1つの文字は1つと数えると説明しましたが、これには少し補足が必要です。

ユニコードには、結合文字と言う概念があります。これは、「が」¹⁸という文字は「か」と「ゝ」を結合した文字であるという考えです。「か」は U+304B, 「ゝ」は U+3099 として登録されていますが、一方で、「が」を表す文字は U+304C として登録されています。

「が」¹⁸と「が (=「か」と「ゝ」)」は結果として同じ文字を表しますが^{*18}, 前者は1文字、後者は2文字です。

例 8.4. A\$=ChrU(&h304C)

```
B$=ChrU(&h304B)+ChrU(&h3099)
```

```
Print A$,B$
```

```
Print Len(A$),Len(B$)
```

この実行結果は、

```
が          が
1          2
```

となります。

^{*18} 印刷すると全く同じに見えます。

この状況に対応する関数として `Normalize` があります。

`Normalize(x)` は文字列 `x` を分解して等価な合成文字で置換した文字列を返す。

例 8.5. 上の例での合成文字について、

`Normalize(ChrU(&h304B)+ChrU(&h3099))` は、`ChrU(&h304C)` である「が」

を返します。

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 エンコーディングは 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 コード (32 ビット) 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 コード文字の扱いになります。

例えば、「叱」(口へんに七) という文字は `U+20B9F` で指定される文字で、`U+FFFF` を超えるコード番号を持ちます^a。UTF-16 エンコーディングでは `U+FFFF` を超えるものは 1 コードで表すことはできません。今の場合、`叱=ChrW(&hD842) + ChrW(&hDF9F)` と表され、UTF-16 としては 2 コード (32 ビット) 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。

^a 「叱」(`U+53F1`) (口へんに匕) とは別字です。

「tbasic でのユニコード (Unicode) の取扱い」更新記録

- (2023 年 03 月版) Ver.1.6 用に全面的に修正・書き直し。ユニコードについての基礎知識を追加。
- (2013 年 10 月版) 初版公開