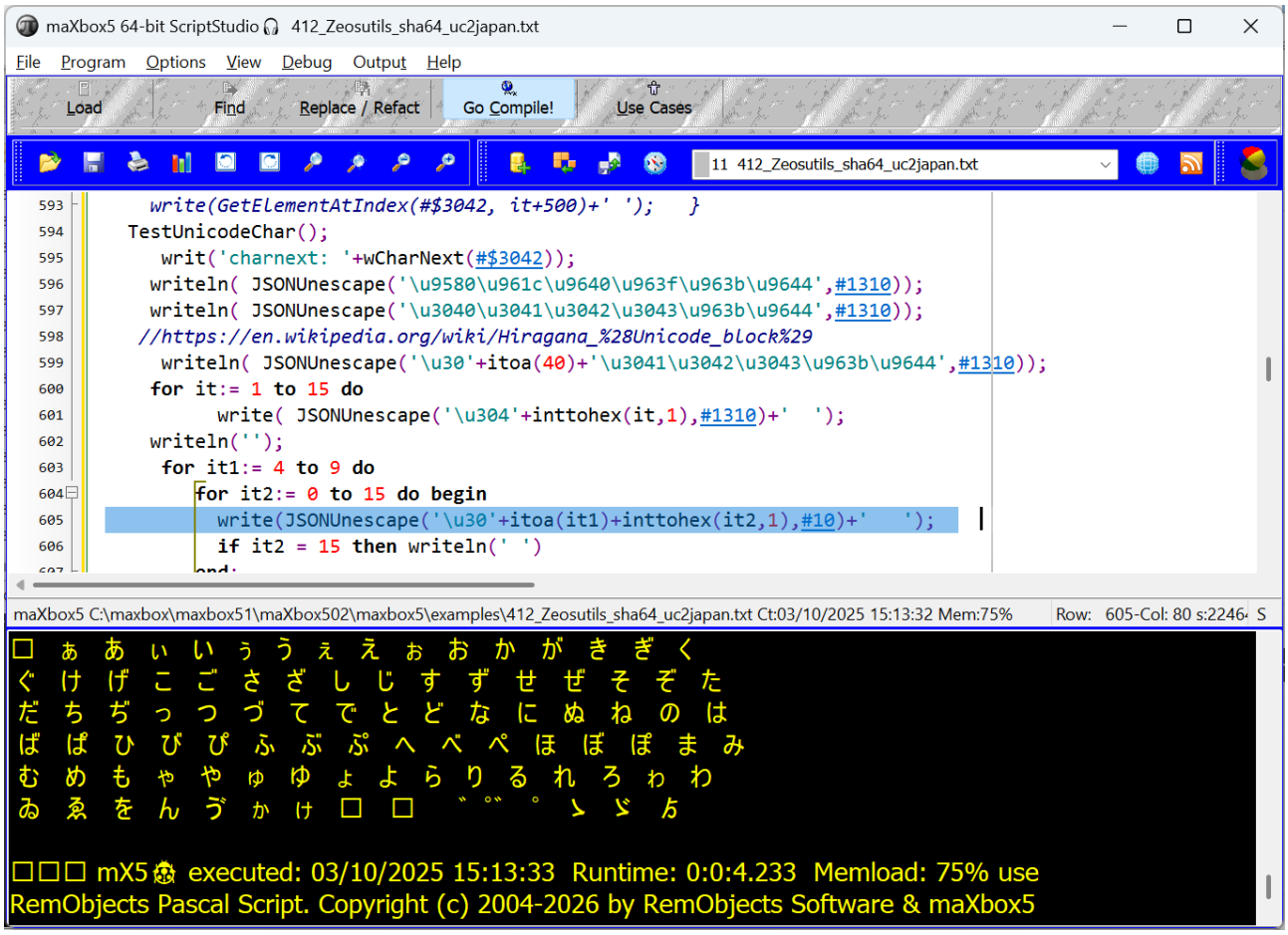
Unicode enumerator ユニコード

maXbox Starter 158 - Code in Unicode.

Source:
Hiragana (Unicode block) - Wikipedia

Script_: Download 412_Zeosutils_sha64_uc2japan.txt (maXbox5)

I have a multi-byte string containing a mixture of japanese and latin characters. I'm trying to copy parts of this string to a separate memory location. Since it's a multi-byte string, some of the characters uses one byte and other characters uses two. When copying parts of the string, I must not copy "half" japanese characters. To be able to do this properly, I need to be able to determine where in the multi-byte string characters starts and ends.



412_mX5GuiScreenshot2025-10-27100346.png

As an example, if the string contains 3 characters which requires [2 byte][2 byte][1 byte], I must copy either 2, 4 or 5 bytes to the other location and not 3, since if I were copying 3 I would copy only half the second character.

The advantages of multibyte (or multi-byte) are:

- Try using 932 for the code page. I don't think CP_UTF8 is a real codepage, and it may only work for WideCharToMultibyte() and back. You can also try isleadByte(), but that requires either setting the locale correctly, or setting the default codepage correctly. I have successfully used IsDBCSLeadByteEx(), but never with CP_UTF8.

A grapheme or extended grapheme cluster is a single user-perceived character that may be made up of multiple Unicode code points. For example, the string containing the Thai character "kam" (กำ) consists of the following two characters:

ก (= '\u0e01') THAI CHARACTER KO KAI

ำ (= '\u0e33') THAI CHARACTER SARA AM

When displayed to the user, the operating system combines the two characters to form the single display character (or grapheme) "kam" or กำ. Emoji can also consist of multiple characters that are combined for display in a similar way.

We start with the **Pascal** solution:

```
writeln( JSONUnescape('\u3040\u3041\u3042\u3043\u963b\u9644',#
    //https://en.wikipedia.org/wiki/Hiragana_%28Unicode_block%
for it1:= 4 to 9 do
 for it2:= 0 to 15 do begin
   write(JSONUnescape('\u30'+itoa(it1)+inttohex(it2,1),#10)+'
   if it2 = 15 then writeln(' ')
 end;
```

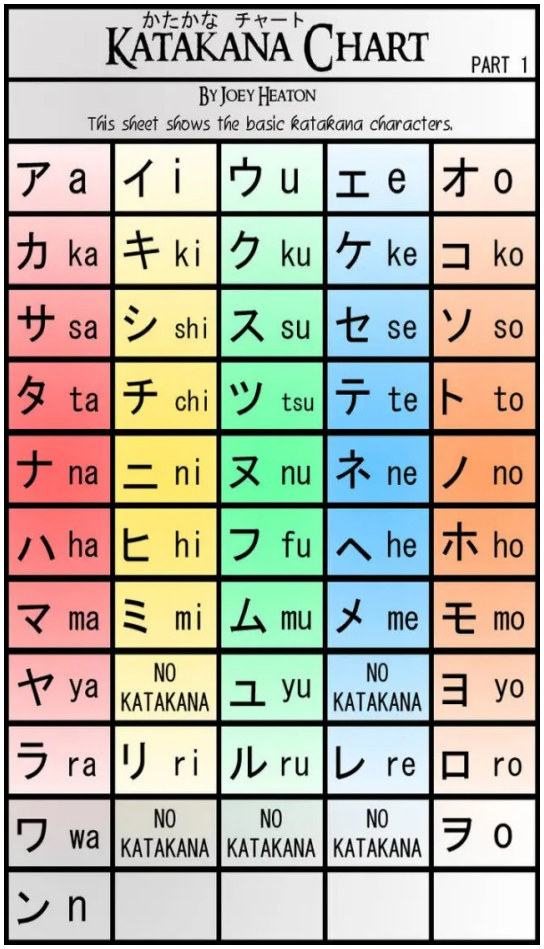
Katakana is a Unicode block containing katakana characters for the Japanese and Ainu languages.
The second, the **Katakana** one has a form to configure the output:

```
//https://en.wikipedia.org/wiki/Hiragana_(Unicode_block)
for i:= 10 to 15 do
  for j:= 0 to 15 do begin
   write(JSONUnescape('\u30'+inttohex(i,1)+inttohex(j,1),#10)+
   if j = 15 then writeln(' ')
end;
```

It works with: Delphi version > 7.0 and needs the Library SysUtils, Characters and more classes like TWideStrings or TEncoding and StdCtrls, in maXbox there are precompiled built on board.

The third try is to built the alphabet as a chart. All manipulations of strings received by user input should consider each element and not each individual character. This is basically why delphi TEdit does not work properly with these kind of emojis (just try adding one and pressing backspace that you will see part of problem). In short, when you don't consider this, you risk breaking a string into a position in the middle of a Grapheme, creating a malformed string.
Delphi itself only concerns itself with the encoding/decoding of UTF-16 itself (especially when converting that data to other encodings, like ANSI, UTF-8, etc). Delphi does not care what the UTF-16 data represents. Graphemes are handled only by application code that needs to be do text processing, glyph rendering, etc. Things that are outside of Delphi's scope as a general programming language.
Most of the time, you should just let the OS deal with them. Unless you are writing your own engines that need to be Grapheme-aware.

Katakana is a Japanese syllabary system comprising 46 basic characters, just like Hiragana. It’s primarily used for writing foreign loanwords, names, and sometimes even for emphasis in text (similar to italics in English). You’ll see it a lot when reading words like “coffee” (コーヒー) or names like “Michael” (マイケル).



While both Hiragana and Katakana are used to represent the same set of sounds (syllables), they are used for different purposes. Hiragana is for native Japanese words and grammatical functions, while Katakana is for loan words and foreign names.

There is no such thing as a "normal graphic character" in Unicode. What you are thinking of as a "character" is officially referred to as a "grapheme", which consists of 1 or more Unicode codepoints linked together to make up 1 human-readable glyph. Individual Unicode codepoints are encoded as 1 or 2 codeunits in UTF-16, which is what each 2-byte Char represent. When a codepoint is encoded into 2 UTF-16 codeunits, that is also known as a "surrogate pair".
We first save the alphabet in a unicode string:

```
 writeln('aiueo:'+CRLF);
 //https://en.wikipedia.org/wiki/Hiragana_(Unicode_block)
   t:= 0;
   //for i:= 10 to 15 do begin
   ucstring:='';
   for i:= 10 to 15 do begin
      for j:= 1 to 15 do
         ucstring:= ucstring+ JSONUnescape('\
                              u30'+inttohex(i,1)+inttohex(j,1),#10);
   end;


//https://my6.code.blog/wp-content/uploads/2025/09/
katakana_chart_part_1_by_treacherouschevalier_d2kiy5y-pre.jpg?w=642
    writ(itoa(length(ucstring)));
     for k:= 1 to 90 do
        if k <= 10 then begin
          if k mod 2 = 0 then
             write(ucstring[k]+' ');
          if k= 10 then writeln(' ')
        end else begin
           if k mod 2 = 1 then
           write(ucstring[k+1]+' ') ;
           if k mod 10 = 0 then writeln(' ')
        end;
     //end
```

```
ア イ ウ エ オ
ガ ギ ケ コ サ
シ ス セ ソ タ
ヂ ツ テ ト ナ
ヌ ノ パ ビ フ
プ ベ ホ ポ ミ
モ ヤ ユ ヨ リ
レ ヮ ヲ ン ヵ
ヷ ヸ ・ ヽ ヿ
```

The Japanese language has three types of characters: Hiragana, Katakana, and Kanji from Chinese.
Hiragana and Katakana are phonetic symbols, each representing one syllable while Kanji is ideogram, each stand for certain meaning.

The destination folder can contain the script, or in the script itself as a **const** or you can invoke scripts as an **URL** from a server with:

```
Navigate('https://maxbox4.wordpress.com/2025/07/01/ekon-29/jsdemo.js');
```

```
const JSCRIPT =
'(() => {                                                          '+lf+
'    "use strict";                                                 '+lf+
'                                                                  '+lf+
'    // ----------- SPLIT ON CHARACTER CHANGES ------------       '+lf+
'    const main = () =>                                            '+lf+
'        group("ア イ ウ エ オ++///\\")                            '+lf+
'        .map(x => x.join(""))                                     '+lf+
'        .join(", ");                                              '+lf+
```

The above is the basic usage of the WebView2 handling. You can customize the parameters and error handling based on specific needs. There's a tutorial about the topic: Tutorial 129 Using WebView2, October 2024.

The last solution is the Python one procedure:

```
Const PYFUNC =
  'def splitter(text):  '+lf+
  '  return ", ".join(""".join(group) for key, group in groupby(text)) ';
```

Conclusion

Set the font property of the components that will display the Japanese characters to a font that supports Japanese characters. Common fonts that support Japanese include "MS UI Gothic," "MS Gothic," and "Meiryo." You can set the font property either at design time or programmatically during runtime.
If you are reading the Japanese characters from a file, make sure to use the appropriate encoding when reading the file. The most common encoding for Japanese is UTF-8. Delphi provides the TStreamReader class, which allows you to specify the encoding when reading from a file.

The challenges are:

- Note that the 932 code page used in the MultiByteToWideChar and WideCharToMultiByte functions refers to Shift-JIS encoding.
- If you are fetching Japanese characters from a database, make sure the database connection and component you are using support Unicode.
- Make sure the font used supports Japanese characters. Choose a font that includes Japanese characters, such as "MS Gothic" or "Meiryo."

References:

Katakana alphabet - basic Japanese writing system for beginner
Hiragana (Unicode block) - Wikipedia
Katakana (Unicode block) - Wikipedia

Doc and Tool: maXbox5 - Manage Files at SourceForge.net
Release maXbox V5.2.9 · maxkleiner/maXbox5

Max Kleiner 04/10/2025

maXbox4exe Digitally signed by maXbox4exe
Date: 2025.10.03 21:04:56 +02'00'