

Unicode 🐞🧘 過

maXbox Starter 120 - Code with Unicode in **UTF-32**.

"Language is a virus - Laurie Anderson.

Source: 1295_412_Zeosutils_sha64_uc2unicode.txt

Unicode is a character encoding system that assigns a code to every character and symbol in the world's languages.

Unicode is the only encoding system that ensures you may get or combine data using any combination of languages because no other encoding standard covers all languages with codepoints.

So today we deal with unicode codepoint code!

XML, Java, JavaScript, LDAP, OpenSSL and other web-based technologies all require Unicode like jap. 過勞死 karō shi.

UTF-8, a variable length encoding method in which one represents each written symbol to four-byte code, and UTF-16, a fixed width encoding scheme in which a two-byte code represents each written symbol, are the two most prevalent Unicode implementations for computer systems.

Be aware of the difference between Unicode and UTF:

UTF-8 is an encoding system - **Unicode** is a character set definition.

Lets start with a list of signs out from the album Improver:

BTB BeatTheBlues: 🐞
PC PlayChess: ♠️
CC CaryCopper: 🎧
EG EatGarlic: 🧄
EM ExerceMeditation: 🦋
PR PractiseRunning: 🏃
BNS BettNordSued: 🦋
OB OlemosBruja: 🦋
LWN ListenWhiteNoise: 🌈
surprise: 😲
🐞 ♠️ 🎧 🧄 🦋 🏃 🦋 🦋 🌈

And this is how we code it:

```
writeln('PC PlayChess:  '#$d83d#$dccc)
writeln('CC CaryCopper:  '#$d83c#$dfa7);
writeln('EG EatGarlic:  '#$d83d#$dcae);
writeln('EM ExerceMeditation:  '#$d83e#$dd8b);
writeln('PR PractiseRunning:  '#$d83d#$de80);
writeln('BNS BettNordSued:  '#$d83d#$dd2d);
writeln('OB OlemosBruja:  '#$d83e#$dda2);
writeln('LWN ListenWhiteNoise:  '#$d83c#$df08);
writeln('surprise sign:  '#$d83d#$de05); //code point : 01 F6 05
u+0001f605 UTF32
```

As you can see with the surprise sign, we have a code point: `u+0001f605` and this is also UTF-32, so each codepoint results in UTF-32. For example

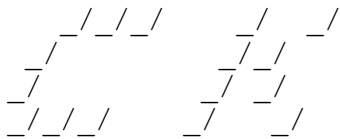
we can encode this sign as a QR-code and the surprise sign is: 🤪

```
aQRCode:= TDelphiZXingQRCode.Create;  
QRCodeBmp:= TBitmap.Create;  
form1:= getform2(700,500,123,'QR Draw PaintPerformPlatform PPP5');  
try  
  aQRCode.Data:= aTextline;  
  aQRCode.Encoding:= qrcAuto; //TQRCodeEncoding(cmbEncoding.ItemIndex);
```



But then we get ?? or ⚪ or [Qreader | Online QR Code Reader](#)
so what's wrong with this idea of 🤪 ? QR-code is text based!

Lets take another sign - the black star as unicode study:



```
https://www.compart.com/en/unicode/U+2605  
Unicode Character "★" (U+2605)  
Name:      Black Star[1]  
Unicode Version:      1.1 (June 1993)[2]  
Block:     Miscellaneous Symbols, U+2600 - U+26FF[3]  
Plane:     Basic Multilingual Plane, U+0000 - U+FFFF[3]  
Script:     Code for undetermined script (Zyyy) [4]
```

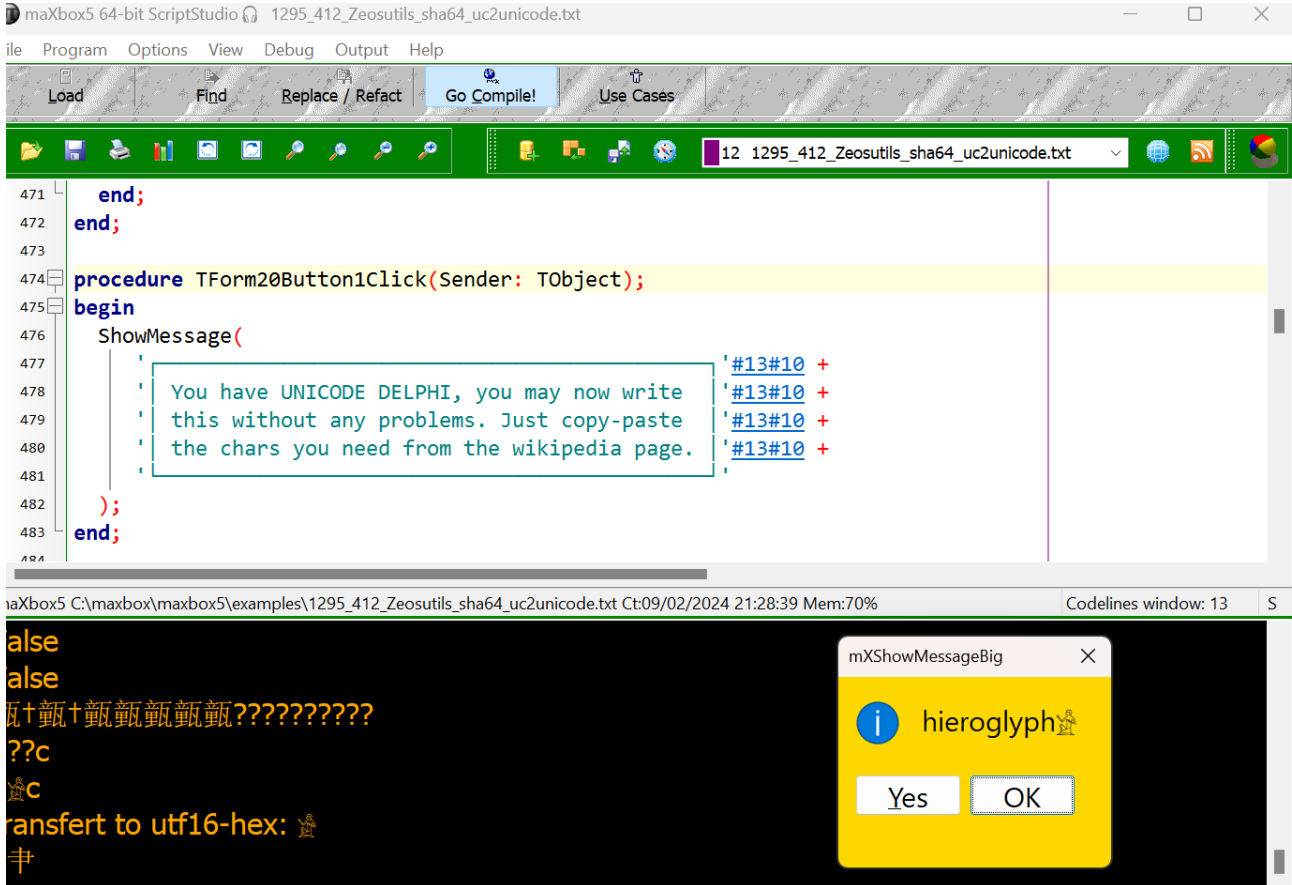
So where's the UTF-32 codepoint: u+00002605

UTF-32 (32-bit Unicode Transformation Format) is a fixed-length encoding used to encode Unicode code points. In UTF-32, each code point is represented by exactly 32 bits (four bytes). Until the formal balloting is concluded, the term UTF-32 can be used to refer to the subset of UCS-4 characters that are in the range of valid Unicode code points. UTF-32 is the only fixed-length encoding, in contrast to all other Unicode transformation formats, which are variable-length encodings so it represents one Unicode code point and is exactly equal to that code point's numerical value like u+00002605 as ★.

So what should I use? UTF8 or UTF16?

The main difference between them is that UTF8 is backwards compatible with ASCII. As long as you only use the first 128 characters, an application that is not Unicode aware can still process the data (which may be an advantage or disadvantage, depending on your scenario).

<https://stackoverflow.com/questions/9818617/what-should-i-use-utf8-or-utf16>



Pic1: 1295_Tut120_Unicdoeblock_Screenshot

Taking a skin-toned emoji, e.g. "👨" is seventeen bytes: `'\xf0\x9f\xa4\xa6\xf0\x9f\x8f\xbb\xe2\x80\x8d\xe2\x99\x82\xef\xb8\x8f'`.

We can write this as UTF-16:
`\ud83e\udd26\uud83c\uddfb\u200d\u2642\uufe0f`

or UTF-32:
`u+0001f926u+0001f3fbu+0000200du+00002642u+0000fe0f`

but wait those are more than four bytes! Yes those are 5 codepoints with fixed length:

```
writeln(#$d83e#$dd26#$d83c#$dfb#$200d#$2642#$fe0f) // 👨 = 👨 + ♂
>>> 👨
```

as a proof: [5 Codepunkte gefunden - Codepoints](https://codepoints.net/search?q=%F0%9F%A4%A6%F0%9F%8F%BB%E2%80%8D%E2%99%82%EF%B8%8F&na=)
[5 Codepunkte gefunden - Codepoints](https://codepoints.net/search?q=%F0%9F%A4%A6%F0%9F%8F%BB%E2%80%8D%E2%99%82%EF%B8%8F&na=)

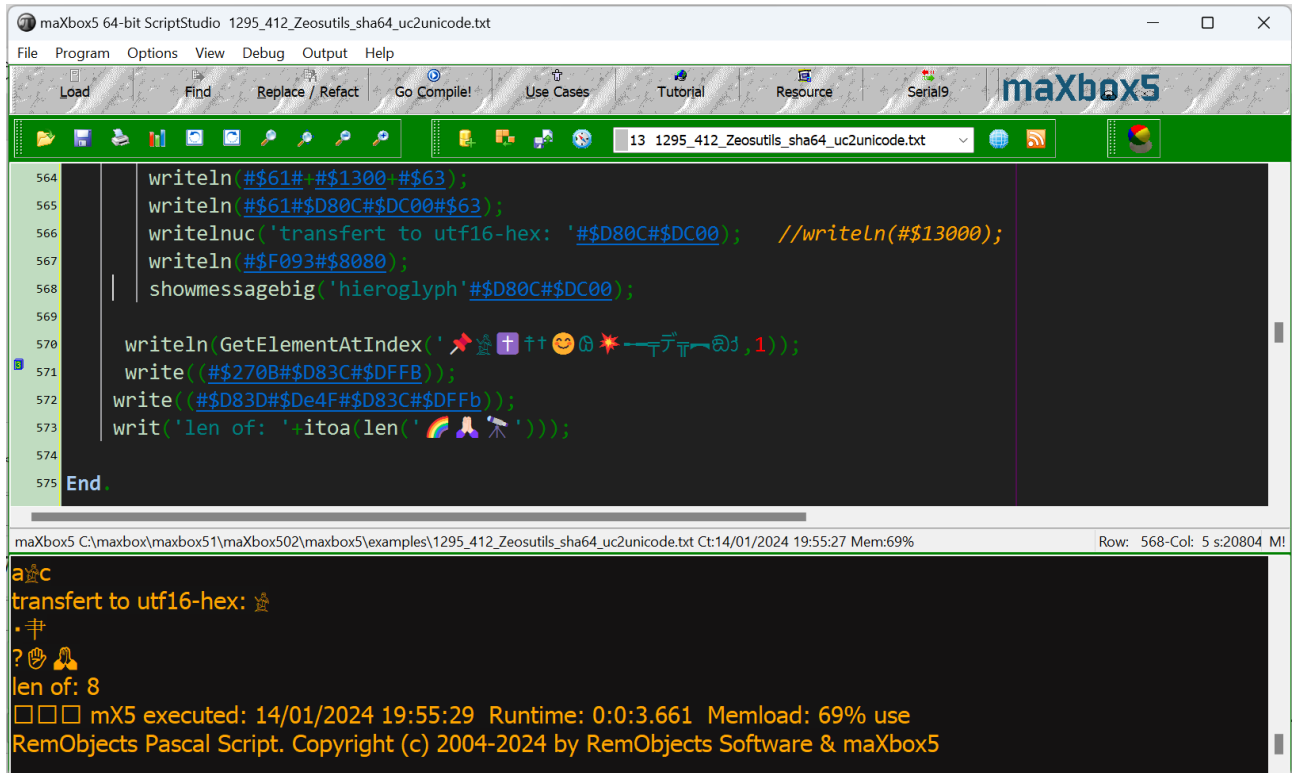
<https://codepoints.net/search?q=%F0%9F%A4%A6%F0%9F%8F%BB%E2%80%8D%E2%99%82%EF%B8%8F&na=>

Note that if you wish a file to be encoded as UTF-8 when you save it you need to specify that when you save it. Like this:

```
// const CP_UTF8= 65001;
```

UTF-8 consists of 1, 2, 3, or 4 bytes per character. The codepoint U+1F605 is correctly encoded as #F0#9F#98#85.

UTF-16 consists of 2 or 4 bytes per character. The 4 byte sequences are needed to encode codepoints beyond U+FFFF (such as most Emojis). Only UCS-2 is limited to codepoints U+0000 to U+FFFF (this applies to Windows NT versions before 2000).



Pic2: 1295_tutor120_grapheme_edit.png

<http://chart.apis.google.com/chart?chs=200x200&cht=qr&chld=M&chl=Text>

Unicode Character Examples

- 🌀🇨🇪🌞☀️☾♀
- 한국어
- 日本語 (｡◕｡)
- 中文
- ქართული
- ไทย
- বাংলা
- فارسی
- العربية
- עברית
- Українська
- Русский 🇷🇺
- Ελληνικά
- Čšâêçñà môt trò

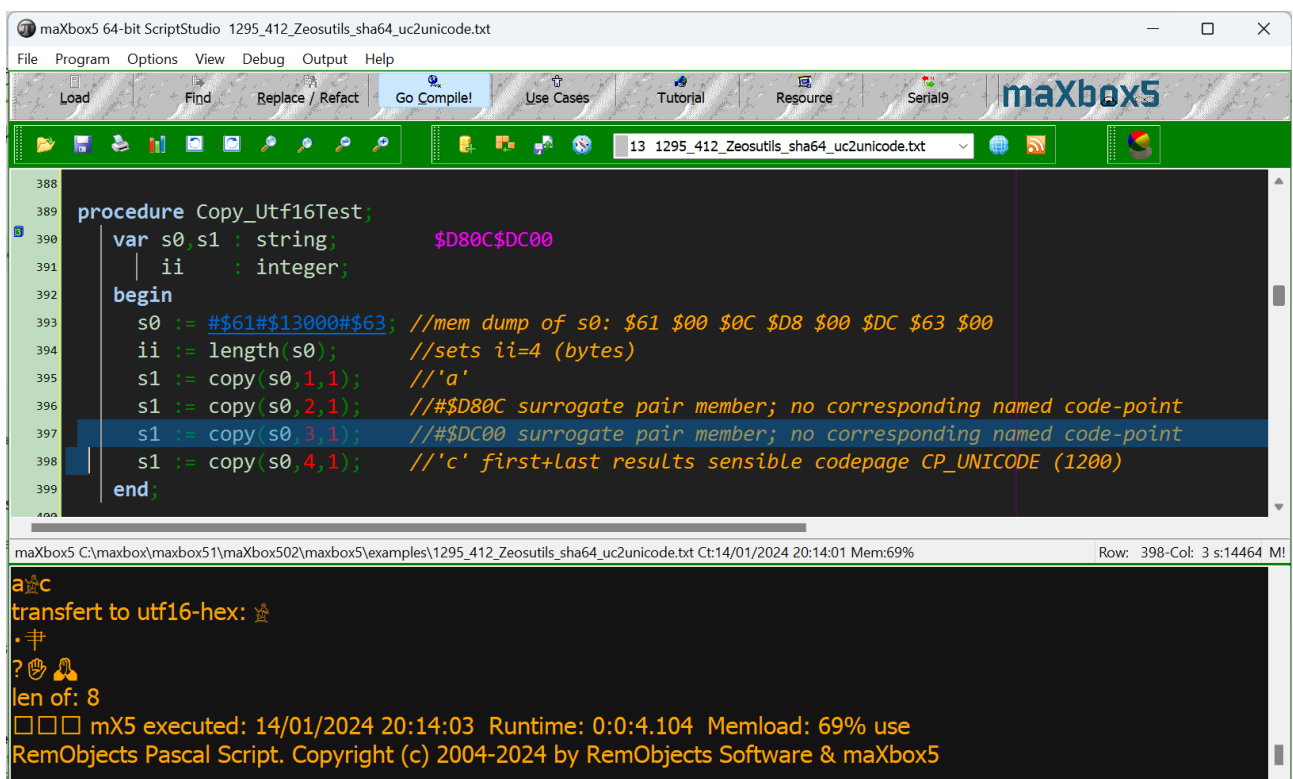
UC Source Representation

Why we cannot use single "\u" to represent smiley within a string? Because when \u escape was designed, all Unicode chars could be represented by 2 bytes or 4 hexadecimal digits. So there are always 4 hexadecimal digits after \u in a Java string literal.

To represent a larger value of Unicode you need a larger hexadecimal number but that will break existing Java strings. So there Java or Delphi uses same approach as utf-16 representation.

As an example, the letter A is represented in Unicode as U+0041 and in Ansi as just 41. So converting that would be pretty simple, but you must find out how the Unicode character is encoded. The most common are UTF-16 and UTF-8. UTF 16, is basically two bytes per character, but even that is an oversimplification, as a character may have more bytes. UTF-8 sounds as if it means 1 byte per character but can be 2 or 3. To further complicate matters, UTF-16 can be little endian or big endian. (U+0041 or U+4100).

Where it makes no sense in a language way is if you wanted to for example convert the arabic letter ain U+0639 ع to Ansi on an English locale machine. You can't. Because you probably missing the code page mapping letter for the code point!



```
388
389 procedure Copy_Utf16Test;
390     var s0,s1 : string;           $D80C$DC00
391         ii : integer;
392     begin
393         s0 := #{$61#13000#63}; //mem dump of s0: $61 $00 $0C $D8 $00 $DC $63 $00
394         ii := length(s0); //sets ii=4 (bytes)
395         s1 := copy(s0,1,1); // 'a'
396         s1 := copy(s0,2,1); //#$D80C surrogate pair member; no corresponding named code-point
397         s1 := copy(s0,3,1); //#$DC00 surrogate pair member; no corresponding named code-point
398         s1 := copy(s0,4,1); // 'c' first+last results sensible codepage CP_UNICODE (1200)
399     end;
```

maXbox5 C:\maxbox\maxbox51\maXbox502\maxbox5\examples\1295_412_Zeosutils_sha64_uc2unicode.txt Ct:14/01/2024 20:14:01 Mem:69% Row: 398-Col: 3 s:14464 M!

```
a c
transfert to utf16-hex:
. 卄
? 🧐 🧐
len of: 8
☐☐ mX5 executed: 14/01/2024 20:14:03 Runtime: 0:0:4.104 Memload: 69% use
RemObjects Pascal Script. Copyright (c) 2004-2024 by RemObjects Software & maXbox5
```

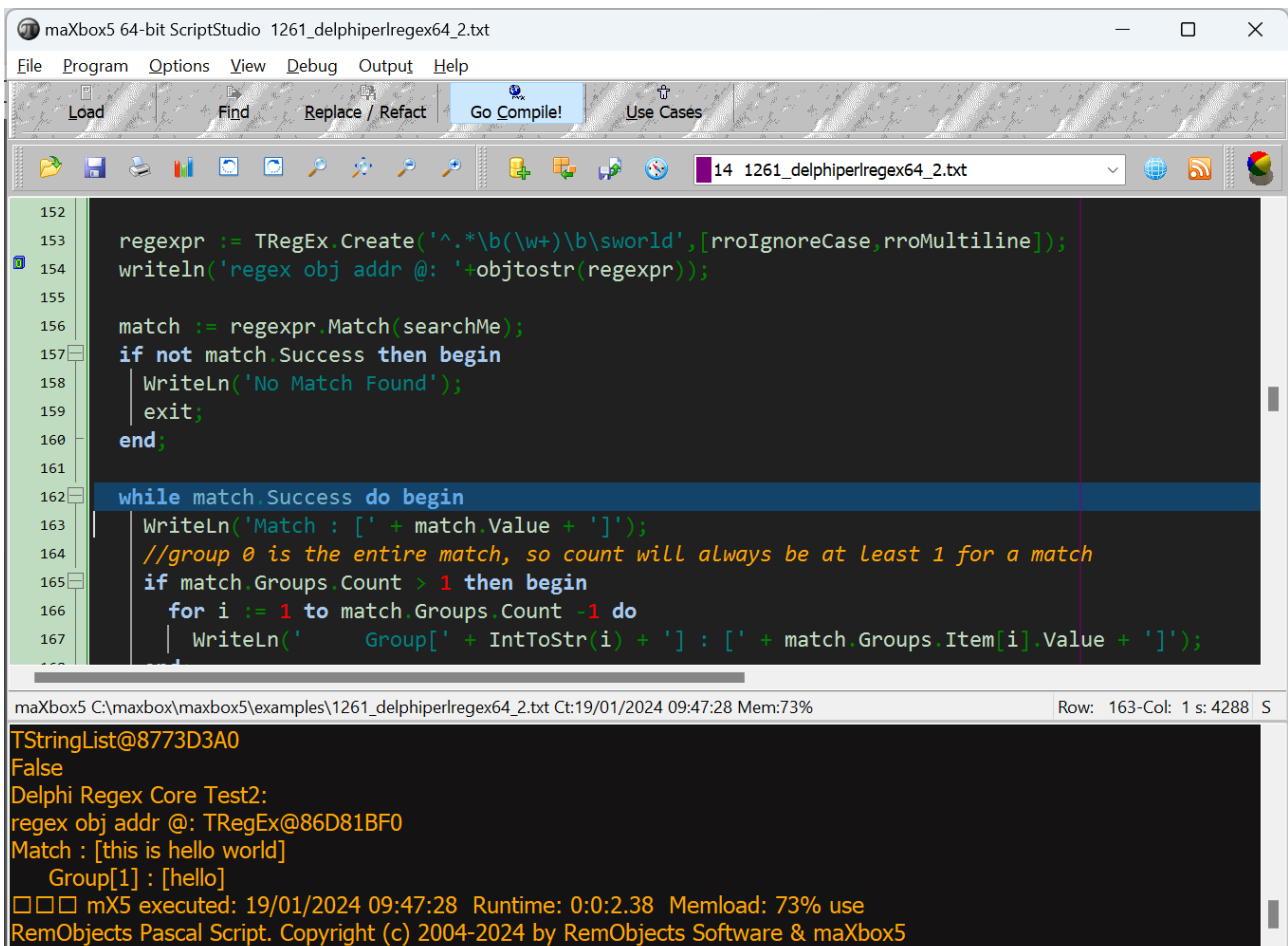
pic3: 1295_tutor120_grapheme_edit2.png

In general, character set of hundreds thousands entries cannot be converted to character set of 127 entries without some loss of information or encoding scheme.

If so, then you can use the Ord standard function to get the Unicode code-point value of whatever Unicode character you have.

An approach could be:

1. Encode the string to UTF-8 using the UTF8Encode function, which returns a UTF8String type.
2. Convert the UTF-8 string to a byte array using the BytesOf function, which returns a TBytes type.
3. Create a UTF-32 string using the TIdTextEncoding.UTF32.GetString method, which takes a TBytes parameter and returns a UnicodeString type.



The screenshot shows the ScriptStudio IDE interface. The main editor displays Delphi code for creating a regular expression and matching it. The code is as follows:

```
152
153  regexpr := TRegEx.Create('^.*\b(\w+)\b\sworld', [rroIgnoreCase, rroMultiline]);
154  writeln('regex obj addr @: '+objtostr(regexpr));
155
156  match := regexpr.Match(searchMe);
157  if not match.Success then begin
158    | WriteLn('No Match Found');
159    | exit;
160  end;
161
162  while match.Success do begin
163    | WriteLn('Match : [' + match.Value + ']');
164    | //group 0 is the entire match, so count will always be at least 1 for a match
165    | if match.Groups.Count > 1 then begin
166      | for i := 1 to match.Groups.Count - 1 do
167        | WriteLn('    Group[' + IntToStr(i) + '] : [' + match.Groups.Item[i].Value + ']');
```

The output window at the bottom shows the following text:

```
TStringList@8773D3A0
False
Delphi Regex Core Test2:
regex obj addr @: TRegEx@86D81BF0
Match : [this is hello world]
    Group[1] : [hello]
□□□ mX5 executed: 19/01/2024 09:47:28 Runtime: 0:0:2.38 Memload: 73% use
RemObjects Pascal Script. Copyright (c) 2004-2024 by RemObjects Software & maXbox5
```

Pic4: tutor119_regex_multicod.png

Last point in literally sense is surrogate as you can see in Pic3 above.

Surrogates are code points from two special ranges of Unicode values, reserved for use as the leading, and trailing values of paired code units in UTF-16. Leading surrogates, also called high surrogates, are encoded from D80016 to DBFF16, and trailing surrogates, or low surrogates, from DC0016 to DFFF16. They are called surrogates, since they do not represent characters directly, but only as a pair.

[FAQ - UTF-8, UTF-16, UTF-32 & BOM \(unicode.org\)](https://unicode.org)

By the way I never succeeded in receiving a Unicode sign from an Arduino board:

```
146 procedure COMPortCreate(Sender: TObject);
147 var Ini: TIniFile;
148     //inipath: string;
149 begin
150     comPort:= TComPort.Create(self);
151     with comPort do begin
152         BaudRate:= br9600;
153         //Port:= 'COM'+IntToStr(ACOMPORT);
154         Port:= (ACOMPORT);
155         Parity.Bits:= prNone;
156         StopBits:= sbOneStopBit;
157         DataBits:= dbEight;
158         Events:= [evRxChar, evTxEmpty, evRxFlag, evRing, evBreak,
159                 evCTS, evDSR, evError, evRLSD, evRx80Full]
160         FlowControl.OutCTSFlow:= False
161         FlowControl.OutDSRFlow:= False
162         FlowControl.ControlDTR:= dtrDisable
```

maxbox5 C:\maxbox\maxbox5\examples\611_Arduino_COMOutputs_64.txt Ct:20/01/2024 17:35:45 Mem:67% Rtime:0:0:2.398 Thr:23 S

```
maxbox5 611_Arduino_COMOutputs_64.txt Compiled done: 20/01/2024 17:35:45
-----
debug: 5- 4294967295 err:0
COM Inifile: C:\maxbox\maxbox5\examples\611_Arduino_COMOutputs_64.ini
Connected at Sat, 20 Jan 2024 17:35:45 +0100 COM3
☐☐☐ mX5 executed: 20/01/2024 17:35:45 Runtime: 0:0:2.398 Memload: 67% use
RemObjects Pascal Script. Copyright (c) 2004-2024 by RemObjects Software & maxbox5
```

Conclusion

UTF-32 (32-bit Unicode Transformation Format) is a fixed-length encoding used to encode Unicode code points that uses exactly 32 bits (four bytes) per code point (but a number of leading bits must be zero as there are far fewer than 2 Unicode code points, needing actually only 21 bits). UTF-32 is a fixed-length encoding, in contrast to all other Unicode transformation formats, which are variable-length encodings. A character set is a list of characters with unique numbers (these numbers are sometimes referred to as "code points"). For example, in the Unicode character set, the number for A is 41.

Script: softwareschule.ch/examples/unicode3.htm

References:

Compiled Project:

<https://github.com/maxkleiner/maxbox4/releases/download/V4.2.4.80/maxbox5.zip>

Topic and Tool:

<https://www.coderstool.com/unicode-text-converter>

<https://maxbox4.wordpress.com>

Max Kleiner 26/02/2024