

Majerus.net ANSI/VT automation library

Contents

Introduction	2
Control Characters (C0).....	2
Virtual Terminal Sequences (VTS).....	3
Select Graphic Rendition (SGR).....	4
ANSI Fonts	5
Using installed Windows fonts.....	5
Using raster font files (.fon)	6
Using TheDraw font files (.tdf).....	7
Using OLE Fonts	8
ANSI Bitmaps and Icons	8
Using bitmaps	8
Using icons	9
ANSI pictures.....	9
Using image files (.gif, .png, .jpeg,...)	9
Using OLE Pictures	10
ANSI Binary files	11
Using binary files (.bin)	11
Other methods.....	11
Annexes.....	13
Annex 1 – Codepages.....	13
Annex 2 – Colors Palettes	14

Introduction

Majerus.ANSI is a COM/OLE Automation component designed to make working with ANSI and VT terminals easier. It is bundled with Majerus.net ActiveScript Shell (axsh) and with Majerus.net PowerShell Tools.

The library is implemented as a single object containing all the core functionality, but some methods will create and return other objects when used. The main object is created by instantiating "Majerus.ANSI".

```
JScript: var ansi = new ActiveXObject("Majerus.ANSI");  
VBScript: Set Ansi = CreateObject("Majerus.ANSI")  
PowerShell: $ansi = New-Object -ComObject Majerus.ANSI
```

Control Characters (C0)

ASCII contains some control characters that, instead of displaying some glyph in the console, are used as in-band control. To make it easier to get these characters to build strings with embedded control codes, the ANSI object exposes them as properties using their standard human-readable acronyms.

The properties are as follows:

NUL	Null character
BEL	Bell (audible alert) character
BS	Backspace character
HT	Tabulation (horizontal tabulation) character
LF	Line Feed / End of Line character
VT	Line tabulation (vertical tabulation) character
FF	Form Feed character (<i>used as pause in ActiveScript Shell</i>)
CR	Carriage Return character
SUB	Substitute character / End of File character (Ctrl-Z)
ESC	Escape character
CRLF	Carriage Return + Line Feed characters
DEL	Delete character
NL	Platform-dependent NewLine character(s)

Note on Windows, NL is identical to CRLF, but if we build for a Unix platform, this would return LF instead. You should use CRLF when you want CR + LF regardless of the platform, and NL when you want the appropriate end-of-line marker for the current platform.

The SOH, STX, ETX, EOT, ENQ, ACK, SS, SI, DLE, DC1..4, NAK, SYN, ETB, CAN, EM, FS, GS, RS and US control codes are not provided as they are typically used for serial communication or structured files, not for console outputs.

Virtual Terminal Sequences (VTS)

ANSI escape sequences are a set of characters within a string interpreted by a terminal as commands instead of character codes. The first character of these sequences is the control character ESC, which gives them the common name of “escape sequences”.

Support for these sequences depends on the terminal used. Windows added support for these late with Windows 10 Version 1607 “Anniversary Update”.

The ANSI object provides string properties for most standardized control sequences. Many of them can be used as it to get their default behavior, or with parameters to get custom behaviors.

CUU (nCells=1)	Cursor Up
CUD (nCells=1)	Cursor Down
CUF (nCells=1)	Cursor Forward (Right)
CUB (nCells=1)	Cursor Backward (Left)
CNL (nLines=1)	Cursor Next Line
CPL (nLines=1)	Cursor Previous Line
CHA (nColumns=1)	Cursor Horizontal Absolute
VPA (nRow=1)	Cursor Line Position Absolute
CUP (nRow=1, nColumn=1)	Cursor Position
HVP (nRow=1, nColumn=1)	Horizontal Vertical Position
RI	Reverse Index (performs the reverse operation of Line Feed)
SCP	Save Cursor Position
RCP	Restore Cursor Position
SU (nLines=1)	Scroll Up
SD (nLines=1)	Scroll Down
ICH (nSpaces=1)	Insert Character
DCH (nCharacters=1)	Delete Character
ECH (nCharacters=1)	Erase Character
IL (nLines=1)	Insert Line
DL (nLines=1)	Delete Line
ED (nMode=0)	Erase in Display (0=to end, 1=to beginning, 2=entire screen)
EL (nMode=0)	Erase in Line (0=to end, 1=to beginning, 2=entire line)

Control sequences that are behaving more like commands than in-band control are provided as methods instead of properties.

ShowCursor (bVisible=true)	Show or hide the cursor
CursorBlinking (bBlinking=true)	Enable or disable cursor blinking
WindowTitle (strTitle)	Set window title
Link ([strURI], [strID])	Create a hyperlink
SetClipboard (strText)	Set clipboard data
SetPalette (ColorsArray)	Change the terminal colors palette (0 to 256 colors array)
SetPaletteColor (ClrIndex, Color)	Change a single color in the terminal colors palette

Note these methods do not take effect when called, instead they return strings to be used as in-band control to perform the specified action.

Select Graphic Rendition (SGR)

Select Graphic Rendition are a group of control sequences to change colors and text attributes.

Most terminals support bold, but usually simply by switching to a brighter color instead of changing the font weight, underline, and inverted, which is sometimes called “negative”, but does not generate a negative of the foreground and background colors, instead it swaps foreground and background colors.

Reset ()	Reset all SGR attributes
Bold (bBold=false)	Set or unset bold (and/or bright)
Faint (bFaint=false)	Set or unset faint/dim (lower intensity) <i>(works in Windows Terminal 1.2 and later)</i>
Italic (bItalic=false)	Set or unset italic <i>(works in Windows Terminal 1.6 and later)</i>
Underline (bUnderline=false)	Set or unset underline
Overline (bOverline=false)	Set or unset overline <i>(works in Windows Terminal 1.2 and later)</i>
Strikethrough (bStrikethrough=false)	Set or unset strikethrough (crossed out) <i>(works in Windows Terminal 1.2 and later)</i>
Blink (bBlink=false)	Set or unset blinking <i>(works in Windows Terminal 1.4 and later)</i>
Conceal (bConceal=false)	Set or unset conceal (hide) <i>(works in Windows Terminal 1.2 and later)</i>
Invert (bInvert=false)	Set or unset inverted (negative)
ForeColorIndex (foreground=-1)	Set or unset the foreground color using an ANSI color index (0..15 or -1 to unset)
BackColorIndex (background=-1)	Set or unset the background color using an ANSI color index (0..15 or -1 to unset)
ColorsIndexes (foreground=-1, background=-1)	Set or unset both foreground and background colors using ANSI colors indexes. (0..15 or -1 to unset)
ForeColor (foreground=clrDefault)	Set or unset the foreground color using an OLE Color
BackColor (background=clrDefault)	Set or unset the background color using an OLE Color
Colors (foreground=clrDefault, background=clrDefault)	Set or unset both foreground and background colors using OLE Colors

Note the RGB (16M colors) control sequences are only fully supported in Windows 10 Version 1704 “Creators Update”. When used in Version 1607 “Anniversary Update”, the console will pick the closest color from the current 16 colors palette instead.

ANSI Fonts

The ANSI object provides several methods to create “ANSI fonts”. These are objects that can be used to create ANSI-Art from standard text.

ANSI fonts can be created from installed Windows Fonts, Windows Raster font files (.fon), TheDraw font files (.tdf), or from an OLE Font object (IFontDisp). FIGlet fonts are not currently supported but could be added if there is enough interest.

In all cases, the resulting ANSI font object will provide a common set of core features and is interchangeable. In most cases you don’t need to change anything when changing between types of fonts.

Name	Get the font name
Width	Get the average width of the font in ANSI image characters
Height	Get the height of the font in ANSI image characters
Bold	Get whether font is a bold font
Italic	Get whether font is an italic font
Underline	Get whether font is an underline font
Strikethrough	Get whether font is a strikeout font
Weight	Get the weight of the font

A single method is used to generate an ANSI image string using a font:

GetText (strText)	Get text as an ANSI image string using this font
-------------------	--

Using installed Windows fonts

This is used to get a font object from any font installed in Windows.

CreateFont (FaceName, Size=16, Weight=400, Italic=false, Underline=false, Strikethrough=false, Antialiased=true)
Get an ANSI font from a font installed on the system

FaceName can be any font family currently installed on the system (OpenType, TrueType or Raster). Size is the height (sometimes referred to as “em”).

Weight is the numerical “boldness” of the font from 0 to 1000. 400 is normal, 700 is bold.

Italic, Underline and Strikethrough are Booleans to set whether the font should be italic, underlined or struck-out.

Antialiased can be set to false to prevent grayscale anti-aliasing. ClearType is always disabled as it is only intended to be used to control subpixels brightness when fonts are rendered at pixel-perfect size.

The method returns an ANSI font object with some features specifically designed to control Windows fonts.

Size	Get the font size (For Windows fonts, this is the “em” height)
------	---

BitDepth	Get or set the rendering bit depth (1=mono, 4=16 colors, 8=Paletized, 24=RGB)
ForeColorIndex	Get or set the rendering foreground color for 4-bit
BackColorIndex	Get or set the rendering background color for 4-bit
ForeColor	Get or set the rendering foreground color for 8-bit and 24-bit
BackColor	Get or set the rendering background color for 8-bit and 24-bit

Note there is no property to set foreground and background colors for 1-bit, as in monochromatic mode, no color control sequence is embedded in the generated ANSI image string. You can control colors for a 1-bit image by prefixing it with the appropriate control sequence generated using the Colors(foreground, background) method provided by the main ANSI object.

ForeColorIndex and BackColorIndex are numbers between 0 and 15 corresponding to the ANSI colors palette.

ForeColor and BackColor are OLE Colors, these are numbers in the 0x00BBGGRR format.

When using 8-bit rendering, the ForeColor and BackColor will be changed to their closest available matches from the fixed 256 colors palette.

Using raster font files (.fon)

This is used to get a collection of fonts directly from a Windows Raster Font (.fon) file.

LoadFonts (strFileName)
Load a collection of ANSI fonts from a .fon or .tdf file

Windows raster font files are files containing individual bitmaps for the set of supported characters. A single .fon file typically contains several sizes variations of the same font, and separate files are usually used for bold and italic variations of the same font family.

Using the LoadFonts method with a .fon file will provide a collection object containing all the fonts variations found in the .fon file. Automation collections have a Count property to get the number of items, and an Item(iIndex) indexed property to retrieve an individual item. Be careful iIndex is zero-based, so a font collection with Count=5 will let you retrieve fonts Item(0) to Item(4).

When you retrieve an individual font variation from the collection, you get an ANSI font object with some features specifically designed for Windows Raster fonts.

Size	Get the font size (For raster fonts, this is the height in pixels)
AvailableCharacters	Get characters included in this font (a string containing all available characters)
Copyright	Get the copyright information (a string specified by the font designer)

No properties are available to change the font rendering, as they always create monochrome ANSI without any color control sequence.

Using TheDraw font files (.tdf)

Using TheDraw font files is the same as using raster font files, but providing a .tdf file instead of a .fon file.

Again, a collection of the font variations found in the file is returned. Many fonts designers use a .tdf file to store color-variations of the same font, but they can also sometimes be completely different fonts stored together for convenience.

When you retrieve an individual font variation from the collection, you get an ANSI font object with some features specifically designed for TheDraw fonts.

Size	Get the font size (For TheDraw fonts, this is the height in characters)
AvailableCharacters	Get characters included in this font (a string containing all available characters)
Type	Get the font type (0=Outline, 1=Block, 2=Color)

Type = Block (1) are fonts made of ANSI characters. Type = Color (2) are fonts made of ANSI characters and 16-colors foreground and background for each of these characters. Finally, Type = Outline (0) are fonts designed as bevels that can take many different styles depending on an extra set of properties. Note these properties are only available if Type=0.

OutlineStyle	Get or set the outline rendering style (0 to 36, default is 1)
LeftColorIndex	Get or set the outline left color
TopColorIndex	Get or set the outline top color
RightColorIndex	Get or set the outline right color
BottomColorIndex	Get or set the outline bottom color
FaceColorIndex	Get or set the outline face color
BackColorIndex	Get or set the outline background color

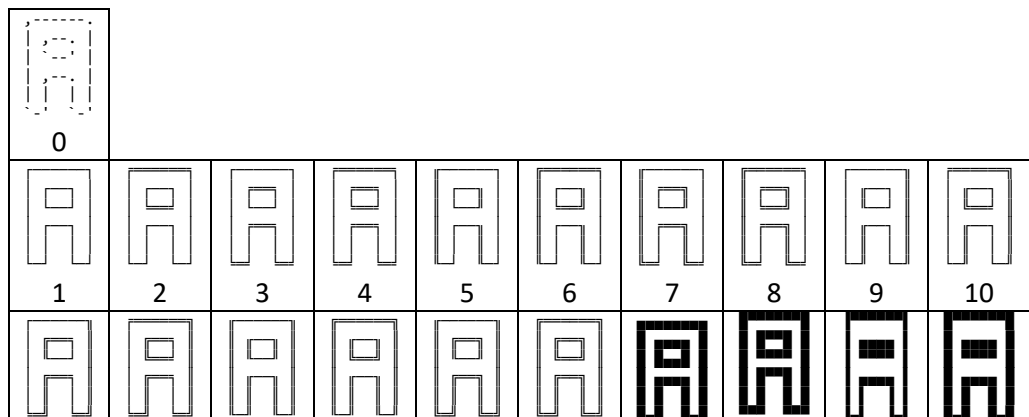
OutlineStyle is used to change the visual style. A single outline font can be displayed using single box drawing lines, double box drawing lines, a combination of both to make a 3D bevel effect, block elements, rounded lines, ...


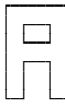
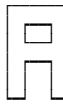
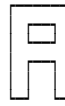
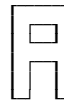
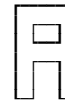
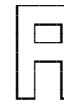



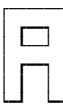
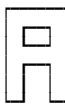
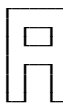
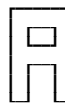
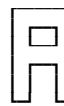

Here are the supported values for OutlineStyle:

Style 0 is pure
ASCII.

It is safe on
any terminal.

1 to 20 are
Extended
ASCII / ANSI.
They work on
any terminal
compatible



with original US codepage. 21 to 36 are Unicode. They typically require a UTF terminal and a modern font (like Consolas).	11	12	13	14	15	16	17	18	19	20
										
	21	22	23	24	25	26	27	28	29	30
							<div> <div>Styles 22 to 36 are similar to styles 2 to 16 respectively, but using a heavy line in place of a double line.</div> </div>			
	31	32	33	34	35	36				

(If you used TheDraw, styles 1 to 19 are equivalent to the original style types A to S.)

You can find over a thousand TheDraw fonts at <http://www.roysac.com/thedrawfonts-tdf.html>.

Using OLE Fonts

You can convert an OLE IFontDisp object to an ANSI font using the GetFont(Font) method.

The returned ANSI font object does not provide much customization as the OLE Font is already fixed size, style, and smoothing mode.

Support for OLE Fonts is only provided if no other mean to get a font is available, as they are less flexible than the other methods. As they are often set to use ClearType, which does not work well to generate ANSI images, they are always rendered as monochrome to avoid artifacts.

ANSI Bitmaps and Icons

Bitmaps and icons can be used to create ANSI images with exact pixel to half-block character mapping. It is the recommended way to create colorful ANSI-Art with precise control. These conversions are very fast as there is no codec or image processing involved, the file is directly converted to an ANSI representation.

A 160×50 characters console gives you a 160×100 half-blocks graphics mode, which is the same resolution as the 16 colors mode of the original CGA. Blocks ANSI-art in the console is very similar to pixel-art on the original graphics adapter of the PC.

Using bitmaps

LoadBitmap (strFileName)
Load an ANSI image string from a .bmp/.dib file

Using LoadBitmap with a .bmp or .dib file returns an ANSI image string.

There is absolutely nothing more to it, the resulting ANSI image depends only on the format of the bitmap file provided, and only uncompressed Bitmap files are supported (no RLE-encoding).

Each pixel of the bitmap will map to a half-block, so the ANSI image has a characters-width matching the pixels-width of the bitmap, and a characters-height half of the pixels-height of the bitmap.

The type of color control sequences depends on the color depth of the bitmap. A Monochrome bitmap (1bpp) will create an ANSI image without any control sequences, using the current foreground and background colors. A 16-color bitmap (4bpp) will create an ANSI image using the terminal color palette, it could display incorrectly if the terminal palette does not match the ANSI standard palette. A 256-colors (8bpp) bitmap will be adapted to fit the terminal's 256 colors palette as closely as possible. Finally, 24-bit and 32-bit bitmaps will both create ANSI images with RGB control sequences for exact colors matching and support for 16M colors.

Using icons

<code>LoadIcon (strFileName, Width=16, Height=16, BitDepth=auto)</code>
Load an ANSI image string from a .ico file

Using `LoadIcon` with a .ico file returns an ANSI image string corresponding to the icon variation found in the file.

The Width, Height and BitDepth must match an icon format available in the .ico file, no conversion will be performed.

Any dimensions and bit-depths of 1, 4, 8, 24 (with transparency masks) and 32 (with alpha channel) are supported. In case of 32bpp, the alpha channel is simplified to an image mask, so if available, 24bpp will provide better results as it contains the same colors depth with an image mask designed to match perfectly. If BitDepth is not specified, the best available match will be used, in the following preference order: 24bpp, 32bpp, 8bpp, 4bpp, 1bpp.

It can also load icons embedded in .exe and .dll files by specifying the filename, and optionally, the icon index (as zero-based integer) or a resource identifier (prefixed by a '-' after separated from the filename by a comma. For example, "C:\path\filename.exe,-100" for the icon with resource identifier 100.

Just like when using the `LoadBitmap` method, the resulting ANSI image depends only on the format of the icon requested.

Transparent pixels of the icon will use the default background color, while inverted pixels will use the default foreground color.

ANSI pictures

Using image files (.gif, .png, .jpeg,...)

<code>LoadImage (strFileName, Width=auto, Height=auto, BitDepth=24, Dither=False)</code>
Generate an ANSI image string from an image or picture file

Any image file can be used to generate an ANSI image using the `LoadImage` method. The image file can be any image supported by Windows Imaging Component (WIC). New formats can be added by installing the appropriate WIC image codec.

Out of the box, Windows includes WIC image codecs for Bitmap (.bmp, .dib), Icon (.ico), GIF 89a (.gif), JPEG (.jpeg, .jpg, .jpe), JPEG XR (.jxr, .hdp, .wdp), PNG (.png), TIFF (.tiff, .tif), Digital Negative (.dng) and DirectDraw Surface (.dds).

Several extensions for Windows 10 can be installed from the Microsoft Store to add support for other image formats, note they require some minimum version of Windows 10:

Extension	Description	Min. ver.	Microsoft Store page
HEIF Image Extensions	Support for High Efficiency Image (.heif, .heic) files.	1803	https://www.microsoft.com/store/productId/9PMMSR1CGPWG
Webp Image Extensions	Support for WebP image (.webp) files.	1809	https://www.microsoft.com/store/productId/9PG2DK419DRG
Raw Image Extension	Support for many RAW image files. (LibRaw-based)	1903	https://www.microsoft.com/store/productId/9NCTDW2W1BH8

Unlike the LoadBitmap and LoadIcon methods, LoadImage is designed to scale and resample any picture during conversion. Width and Height will automatically be set to the appropriate number of half-blocks for the image to show at about the same dimensions as if it was shown as pixels (assuming 8x8 pixels per half blocks), except if value 0 is used Width and/or Height, which then uses 1:1 pixels to half-blocks size. If either Width or Height is specified, the other is automatically set to keep the aspect-ratio of the image (and the ExactPixelScale argument is ignored). If both are set, the image is stretched to fit the requested half-blocks dimensions instead of keeping its aspect-ratio.

BitDepth defaults to 24 to use RGB control sequences, but can be set to 4 to use the console colors palette, 1 to create a monochrome image without any control sequence, 8 to use the console 256 colors palette, or 32 to use RGB with transparency. In case of 32bpp, the alpha channel is simplified to an image mask using empty half-blocks with the default background color. The result really is a 25bpp image, same as the 24bpp RGB with an extra 1bpp for transparency.

No dithering will be used by default as photos are typically rendered using 32-bit, and lower bit depths are typically used for infographics or pixel-art and look better with solid colors. However, if rendering a photo or another picture with many colors at lower bit depths, you can set Dither to True to enable it.

Using OLE Pictures

GetPicture (Picture, Width=auto, Height=auto, BitDepth=24)
Generate an ANSI image string from an OLE Picture object

OLE Pictures works similarly to image files, the GetPicture will generate an ANSI image from an OLE IPictureDisp object. Value 0 for Width and/or Height requests original 1:1 pixel size.

ANSI Binary files

Some files are distributed as a raw copy of text mode video memory, often for 160 columns ANSI-art files. These contain characters and colors for each character of the screen.

Using binary files (.bin)

LoadBinary (strFileName, Columns=160, Codepage=437)
Load a binary (raw text mode video memory copy) .bin file

This method will convert the memory dump file into an ANSI image string, with colors attributes converted to corresponding SGR control sequences.

If the memory copy is for another video mode, the Columns argument can be used to specify the number of columns per row.

Other methods

GetDisplayWidth (strText, nColumnOffset=0)	Get the number of columns used to display text in a console
GetDisplayHeight (strText, nLineOffset=0)	Get the number of rows used to display text in a console
Band (strText, iStartColumn, nColumns)	Get a vertical band from ANSI text
Crop (strText, nColumns, Alignment=taLeft, strEllipsis="")	Crop ANSI text to use at most a specified number of columns (taLeft=0, taRight=1, taCenter=2)
Pad (strText, nColumns, Alignment=taLeft, strPaddingCharacter=" ")	Pad ANSI text to use at least a specified number of columns (taLeft=0, taRight=1, taCenter=2)
Fit (strText, nColumns, Alignment=taLeft, strEllipsis="", strPaddingCharacter=" ")	Fit ANSI text into a specified number of columns (cropping and padding) (taLeft=0, taRight=1, taCenter=2)
Flip (strText)	Reverse ANSI text horizontally (currently skips control sequences)
ExpandTabs (strText, nTabStopColumns=8, nTabStopLines=6)	Expand horizontal tabs to spaces and vertical tabs to newlines
ConvertAsciiToLines(strText, strCharactersToConvert="/\ ")	Convert from ASCII slashes and lines to Unicode box drawing lines
FixColorsPalette (strText, ColorsArray=CGA, foreground=clrDefault, background=clrDefault)	Convert from 16 colors to fixed colors (Uses CGA palette by default, see Annex 2)
RemoveEscapeSequences (strText)	Remove escape sequences from a string
LoadAnsi (strFileName, Codepage=437, bStopAtSUB=True, bReplaceNUL=True)	Load an ANSI .ans/.asc file as a string (By default, following CP/M and DOS convention of handling any SUB character as

	an end-of-file marker, and replacing all NUL characters by spaces)
SaveAnsi (strFileName, strAnsi, Codepage=437)	Save a string as an ANSI .ans/.asc file
ReinterpretCodepage(strText, FromCodepage, ToCodepage=437)	Convert from one codepage encoding interpretation to another

LoadBinary, LoadAnsi, SaveAnsi and ReinterpretCodepage methods use the original hardware codepage 437 by default. The original IBM PC has that character set in ROM and since then, it has been considered the default. Another codepage can be specified when working with text that uses some other encoding such as files created on localized versions of MS-DOS and Windows.

You should provide the codepage explicitly when working with the load and save related methods if the files are not using codepage 437.

ReinterpretCodepage should only be used in a specific scenario: If you use some other object that handles encodings, for example an external tool, and it converts from an 8-bit character set into Unicode to return a string, or the other way around, but does not assume the correct code page when doing so.

For example, if you use the “WScript.Shell” object’s Exec method to execute an external utility, it will assume the input and output of that utility to be in the system default Windows ANSI code page, which is usually not the case:

```
var wshell = new ActiveXObject("WScript.Shell");
var p = wshell.exec("cmd /c echo Тһіs іs à тєѕт");
var s = p.stdout.readAll();
echo(s);
```

This will output “ÈšŃ’Î•Ń• Ä-Åÿ Ã İ,,Ń‘Å>İ,,” on a console using UTF-8. The reason is that the WScript.Shell converts the 8-bit text received from cmd.exe from the default ANSI codepage into Unicode. But cmd.exe checked the codepage of the console to format its output, so it was sending back UTF-8, not default ANSI.

At that point, you cannot do much, since WScript.Shell does not let you specify the codepages to use. This is where the ReinterpretCodepage method can be useful, as it can take a Unicode string, convert it back to 8-bit according to the FromCodepage, effectively reversing the wrong 8-bit conversion performed by the other object, then convert it from 8-bit to Unicode again using the proper encoding specified by ToCodepage.

```
var ansi = new ActiveXObject("Majerus.ANSI");
s = ansi.reinterpretCodepage(s, 0, 65001);
echo(s);
```

This will now output the correct “Тһіs іs à тєѕт” string.

Annexes

Annex 1 – Codepages

This table provides a non-exhaustive reference of commonly used codepages.

Country, region or language	CUI (MS-DOS, console, terminal)		GUI (Windows)	
United States, United Kingdom, Western Europe	437	OEM - United States <i>(Original IBM PC hardware codepage)</i>	1252	ANSI - Latin I <i>(Original Windows 1.x codepage)</i> (superset of ISO-8859-1 / “Latin-1” excluding C1 controls)
Western Europe, Latin America and Canada	850	OEM - Multilingual Latin I		
	858	OEM - Multilingual Latin I + Euro		
Portugal <i>(but not Brazil)</i>	860	OEM - Portuguese		
French Canada	863	OEM - Canadian French		
Eastern Europe: Albania, Bosnia/Herzegovina, Croatia, Czech Republic, Hungary, Poland, Romania, Slovakia, Slovenia, Yugoslavia (Latin)	852	OEM - Latin II (Slavic/Eastern European)	1250	ANSI - Central Europe
Yugoslavia (Serbia/Montenegro, Macedonia), Bulgaria	855	OEM – Cyrillic	1251	ANSI - Cyrillic
Russia (aka Cyrillic II / CIS 1)	866	OEM - Russian		
Greece (legacy, aka Greek I)	737	OEM - Greek 437G	1253	ANSI - Greek
Greece (modern, aka Greek II)	869	OEM - Modern Greek		
Pan Europe / Baltic Rim countries: Estonian, Lithuanian, Latvian	775	OEM - Baltic	1257	ANSI - Baltic
Denmark, Norway	865	OEM - Nordic		
Icelandic	861	OEM - Icelandic		
Turkey	857	OEM - Turkish	1254	ANSI - Turkish
Hebrew	862	OEM - Hebrew	1255	ANSI - Hebrew
Arabic	864	OEM - Arabic	1256	ANSI - Arabic
Japanese	932	ANSI/OEM - Japanese Shift-JIS (superset of JIS X 0201)		
China (PRC), Singapore	936	ANSI/OEM - Simplified Chinese GBK		
Taiwan, Hong Kong	950	ANSI/OEM - Traditional Chinese Big5		
Korean	949	ANSI/OEM - Korean		
Thai	874	ANSI/OEM - Thai		
Vietnamese	1258	ANSI/OEM - Viet Nam		
8-bit Unicode: Worldwide	65001	UTF-8 <i>(Modern MBCS codepage for all languages)</i>		

The following special values can also be used, they are system-dependent:

















System default Windows ANSI code page (for GUI apps)	0	Default ANSI codepage
System default OEM code page (for CUI apps)	1	Default OEM codepage
System default Macintosh code page (classic Mac OS)	2	Default MAC codepage

Note strings are handled as Unicode internally. Codepages are only relevant when loading from or saving to files or correcting text which encoding has been improperly handled.

Annex 2 – Colors Palettes

Palettes colors are in ANSI/VT order, not Windows Console or CGA hardware order.

CGA palette

















0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
															

This is the default palette used by the FixColorsPalette method when the ColorsArray argument is not specified.

VBScript: Array(RGB(0,0,0), RGB(170,0,0), RGB(0,170,0), RGB(170,85,0), RGB(0,0,170), RGB(170,0,170), RGB(0,170,170), RGB(170,170,170), RGB(85,85,85), RGB(255,85,85), RGB(85,255,85), RGB(255,255,85), RGB(85,85,255), RGB(255,85,255), RGB(85,255,255), RGB(255,255,255))

JScript: [0x000000, 0x0000AA, 0x00AA00, 0x0055AA, 0xAA0000, 0xAA00AA, 0AAAAA00, 0xAAAAAA, 0x555555, 0x5555FF, 0x55FF55, 0x55FFFF, 0xFF5555, 0xFF55FF, 0xFFFF55, 0xFFFFFF].toVBArray()

RGBI palette

















0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
															

This palette is similar to the CGA palette, but with simple dark yellow (ochre) instead of brown.

VBScript: Array(RGB(0,0,0), RGB(170,0,0), RGB(0,170,0), RGB(170,170,0), RGB(0,0,170), RGB(170,0,170), RGB(0,170,170), RGB(170,170,170), RGB(85,85,85), RGB(255,85,85), RGB(85,255,85), RGB(255,255,85), RGB(85,85,255), RGB(255,85,255), RGB(85,255,255), RGB(255,255,255))

JScript: [0x000000, 0x0000AA, 0x00AA00, 0x00AAAA, 0xAA0000, 0xAA00AA, 0AAAAA00, 0xAAAAAA, 0x555555, 0x5555FF, 0x55FF55, 0x55FFFF, 0xFF5555, 0xFF55FF, 0xFFFF55, 0xFFFFFF].toVBArray()

Windows default 16-colors palette

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
															

This palette was the standard Windows Console palette until Windows 10 version 1709.

VBScript: Array(RGB(0,0,0), RGB(128,0,0), RGB(0,128,0), RGB(128,128,0), RGB(0,0,128), RGB(128,0,128), RGB(0,128,128), RGB(192,192,192), RGB(128,128,128), RGB(255,0,0), RGB(0,255,0), RGB(255,255,0), RGB(0,0,255), RGB(255,0,255), RGB(0,255,255), RGB(255,255,255))

JScript: [0x000000, 0x000080, 0x008000, 0x008080, 0x800000, 0x800080, 0x808000, 0xC0C0C0, 0x808080, 0x0000FF, 0x00FF00, 0x00FFFF, 0xFF0000, 0xFF00FF, 0xFFFF00, 0xFFFFFF].toVBArray()