Majerus.net ANSI/VT automation library

Contents

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 *(used as pause in ActiveScript Shell)*
`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
`NewLine`	Platform-dependent newline character(s)

Note on Windows, NewLine 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 NewLine 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

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 (bright) *Note the Windows console does not currently support unsetting bold, use Reset() instead.*
`Underline (bUnderline=false)`	Set or unset underline
`Inverted (bInverted=false)`	Set or unset inverted (negative)
`ForeAnsiColor (foreground=-1)`	Set or unset the foreground color using an ANSI color index (0..15 or -1 to unset)

`BackAnsiColor (background=-1)`	Set or unset the background color using an ANSI color index (0..15 or -1 to unset)
`AnsiColors (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=Palletized, 24=RGB)
ForeAnsiColor	Get or set the rendering foreground color for 4-bit
BackAnsiColor	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.

ForeAnsiColor and BackAnsiColor 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.

`LoadFont (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 LoadFont 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.

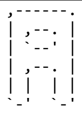
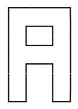
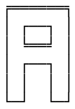
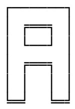
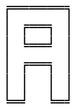
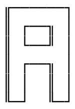
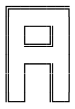
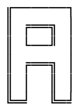
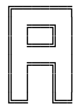
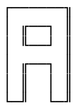
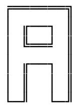
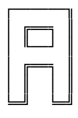
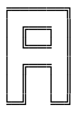
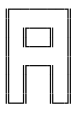
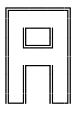
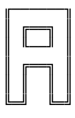
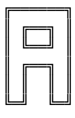
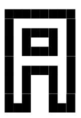
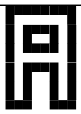
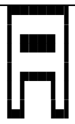
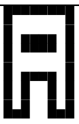
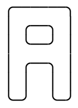
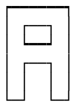
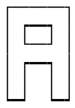
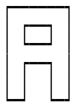
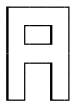
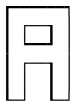
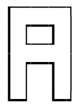
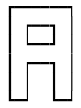
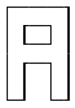
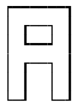
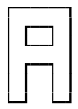
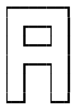
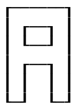
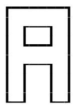
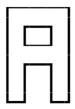
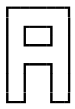
`Size`	Get the font size (For TheDraw fonts, this is the height in characters)
`AvailableCharacters`	Get characters included in this font (a string containing all available characters)
`Type`	Get the font type (0=Outline, 1=Block, 2=Color)

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)*
`LeftAnsiColor`	Get or set the outline left color
`TopAnsiColor`	Get or set the outline top color
`RightAnsiColor`	Get or set the outline right color
`BottomAnsiColor`	Get or set the outline bottom color
`FaceAnsiColor`	Get or set the outline face color
`BackAnsiColor`	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	 0									
1 to 20 are Extended ASCII / ANSI. They work on any terminal compatible with original US codepage.	 1	 2	 3	 4	 5	 6	 7	 8	 9	 10
	 11	 12	 13	 14	 15	 16	 17	 18	 19	 20
21 to 36 are Unicode. They typically require a UTF terminal and a modern font (like Consolas)	 21	 22	 23	 24	 25	 26	 27	 28	 29	 30
	 31	 32	 33	 34	 35	 36	Styles 22 to 36 are similar to styles 2 to 16 respectively, but using a heavy line in place of a double line.			

(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 two half-blocks, 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, a 24-bit bitmap will both create an ANSI image with RGB control sequences for exact colors matching and support for 16M colors.

Using icons

`LoadIcon (strFileName, Width=16, Height=16, BitDepth=4)`
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.
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,…)

`LoadImage (strFileName, Width=auto, Height=auto, BitDepth=24, Dither=False)`
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, .wdp), PNG (.png), TIFF (.tiff, .tif), Digital Negative (.dng) and DirectDraw Surface (.dds).
On Windows 10 Version 1803, and later, the free “HEIF Image Extensions” can be installed from the Microsoft Store to add support for High Efficiency Image (.heif, .heic) files.
(Get it from the Store at : https://www.microsoft.com/store/productId/9PMMSR1CGPWG)

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 8×8 pixels per half blocks). If either Width or Height is specified, the other is automatically set to keep the aspect-ratio of the image. If both are set, the image is stretched to fit the requested half-blocks dimensions.

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.

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 a specified number of columns (taLeft=0, taRight=1, taCenter=2, taJustify=3)
`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)`	Convert from 16 colors to fixed RGB colors (Uses CGA palette by default)
`RemoveEscapeSequences (strText)`	Remove escape sequences from a string
`LoadAnsi (strFileName, Codepage=437)`	Load an ANSI .ans/.asc file as a string
`SaveAnsi (strFileName, strAnsi, Codepage=437)`	Save a string as an ANSI .ans/.asc file
`ConvertCodepage(strText, FromCodepage, ToCodepage=437)`	Convert from one codepage to another

Annex – Codepages

LoadBinary, LoadAnsi, SaveAnsi and ConvertCodepage methods use the original hardware codepage 437 by default, but other codepages can be specified when working with text that uses some other encoding such as files created on localized versions of MS-DOS and Windows.

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

Country/Region	**MS-DOS / CUI (console/terminal)**		**Windows / GUI**	
Original IBM PC hardware United States, United Kingdom, Western Europe	437	OEM - United States	1252	ANSI - Latin I
Western Europe, Latin America and Canada	850	OEM - Multilingual Latin I		
	858	OEM - Multilingual Latin I + Euro		
Portugal (but not Brazil)	860	OEM - Portuguese		
French Canada	863	OEM - Canadian French		
Eastern Europe: Croatia, Czech Republic, Hungary, Poland, Serbia, Slovakia, Slovenia	852	OEM - Latin II	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)	737	OEM - Greek 437G		
Greece (modern)	869	OEM - Modern Greek	1253	ANSI – 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		
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		
7-bit Unicode (all languages) (UTF-8 is preferred)	65000	UTF-7		
8-bit Unicode (all languages) (Modern MBCS codepage)	65001	UTF-8		

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.