SAUCE – Standard Architecture for Universal Comment Extensions

by Olivier "Tasmaniac" Reubens / ACiD

The Standard Architecture for Universal Comment Extensions or SAUCE as it is more commonly known, is an architecture or protocol for attaching meta data or comments to files. Mainly intended for ANSI art files, SAUCE has always had provisions for many different file types.

Why SAUCE?

In the early 1990s there was a growing popularity in ANSI artwork. The ANSI art groups regularly released the works of their members over a certain period. Some of the bigger groups also included specialised viewers in each 'artpack'. One of the problems with these artpacks was a lack of standardized way to provide meta data to the art, such as the title of the artwork, the author, the group, ... Some of the specialised viewers provided such information for a specific artpack either by encoding it as part of the executable, or by having some sort of database or list. However every viewer did it their own way. This meant you either had to use the viewer included with the artpack, or had to make do without the extra info. SAUCE was created to address that need. So if you wanted to, you could use your prefered viewer to view the art in a certain artpack, or even store the art files you liked in a separate folder while retaining the meta data.

The goal was simple, but the way to get there certainly was not. Logistically, we wanted as many art groups as possible to support it. Technically, we wanted a system that was easy to implement and – if at all possible – manage to provide this meta data while still being compatible with all the existing software such as ANSI viewers, and Bulletin Board Software.

The SAUCE Legacy

SAUCE was created in 1994 and it continues to be in use today as a de facto standard within the ANSI art community. However, being created so many years ago, some of the common assumptions made back then are now cause for confusion. This document is a revised edition of the SAUCE specifications; slightly more formal and in a fresh HTML format. This specification is still compatible with the original, some of the legacy confusions are explicitly addressed, and a few new features have been added.

The 'born from DOS' nature explains some of the limits of SAUCE. The Title, Author and Group fields are so short because part of the original design idea was that the DOS filename, title and author needed to fit on a single line in text mode while still leaving some space so you could create a decent UI to select the files in the ANSI viewers. Limitations were also designed around what video cards could do in text mode.
The specialised viewers in the various ANSI artpacks also explain why it was possible to add SAUCE to some files even though the file format really does not like it when you just add a load of extra bytes at the end. The SAUCE-aware viewer could account for the SAUCE and still render the file properly, even if the applications used to edit those files regarded the file as corrupt. In such an event, it was easy enough to remove the SAUCE.

SAUCE is not a perfect solution, but at the time – and with a bit of friendly pressure from the right folks – it managed to fulfill what it was designed to do.

SAUCE applied to a file

Applied to a file, SAUCE has 4 parts: The original file contents, an End-Of-File or EOF character (Decimal 26, Hex 1A, Ctrl+Z), an optional comment block and the SAUCE record.

The EOF character is an important part of what makes SAUCE work for text files. When processing a text file, the DOS / Windows TYPE command, most BBS terminal software and ANSI-viewers will all stop processing at the end of the file or when a EOF character is encountered with the EOF character itself not being rendered.

Writing SAUCE to a file

Adding SAUCE to a file that does not already contain SAUCE is straightforward. First, you prepare the SAUCE record and optionally the comment block. You open the file you want to add SAUCE to, you append the EOF character, if needed you append the comment block and you append the SAUCE record.

Updating a file that already has SAUCE, can be easy as well if you only want to change some fields in the SAUCE record or change existing comment lines. Things become slightly tricky if you want to add a comment block when there was no comment block before, or if the new comment block has a different number of lines than the old one. The simple solution in such a case is to remove the old SAUCE then add the new SAUCE you want. This involves truncating a file, which depending on the language / library / OS can be a matter of a simple function call, or might involve copying the file contents without the SAUCE to a new file.

Reading SAUCE from a file

Reading SAUCE is equally easy. Read the last 128 bytes of the file into a SAUCE record structure. If the ID is set to "SAUCE", then you can assume you have a file that has SAUCE otherwise it does not. If you need the comment block as well, then once you have the SAUCE record, you can check the Comment field and work your way back to read the comment block.

Below you find some pseudo code to explain this. This code assumes a language that can not handle variable records, so the comment block is read one line at a time. In a language like C or C++ this can be handled with a single read.

```
Variables:
    Byte : Count;
    Long : FileSize;
    file : F;

Code:
    Open_File(F);                              | Open the file for read access
    FileSize = Size_of_file(F);                | Determine file size
    Seek_file (F, FileSize-128);               | Seek to start of SAUCE (Eof-128)
    Read_File (F, SAUCE);                      | Read the SAUCE record
    IF SAUCE.ID="SAUCE" THEN                   | ID bytes match "SAUCE" ?
        IF SAUCE.Comments>0 THEN               | Is there a comment block ?
            Seek_File(F, FileSize-128-(SAUCE.Comments*64)-5);
                                               | Seek to start of Comment block.
            Read_File(F, CommentID);           | Read Comment ID.
            IF CommentID="COMNT" THEN          | Comment ID matches "COMNT" ?
                FOR Count=1 to SAUCE.Comments  | \ Read all comment lines.
                    Read_File(F, CommentLine)  | /
                ENDFOR
            ELSE
                Invalid_Comment;               | Non fatal, No comment present.
            ENDIF
        ENDIF
    ELSE
        Invalid_SAUCE;                         | No valid SAUCE record was found.
    ENDIF
```

Warning!

SAUCE was initially created for supporting only the ANSi and RIP formats. Since those formats are ‘technically’ processed one character at a time, SAUCE almost never interferes with how a program treats the files, reading and processing should stop at the EOF character. Files that are treated in a similar way such as ASCII, PCBoard, Avatar and ANSiMations equally tend to work unaffected even in programs that are not SAUCE aware.

This is **not** true for all the other types of files SAUCE has DataType / FileType values for however. Adding SAUCE to some of the other types of files may have serious consequences on programs using those files. For this reason I would currently advise against adding SAUCE to such files, unless there is a pressing reason to do so anyway.

If there are any requests for extending SAUCE to support additional file types, such requests will only be honored if the file does not already have its own meta data feature and it can be proven that SAUCE can safely be added from the typical programs editing such files.

SAUCE Layout

A SAUCE record is a structure of 128 bytes having the following layout:

Field name	Type	Size	Description	Required [1]	Revision [2]
ID	Character [3]	5	SAUCE identification. This should be equal to "SAUCE".	yes	00.0
Version [6]	Character [3]	2	SAUCE version number, should be "00".	yes	00.0
Title	Character [3]	35	The title of the file.	no	00.0
Author	Character [3]	20	The (nick)name or handle of the creator of the file.	no	00.0
Group	Character [3]	20	The name of the group or company the creator is employed by.	no	00.0
Date	Character [3]	8	The date the file was created. The format for the date is CCYYMMDD (century, year, month, day). Example: 4 may 2013 would be stored as "20130504".	no	00.0
FileSize	Unsigned [4]	4	The original file size not including the SAUCE information.	no [8]	00.0
DataType	Unsigned [4]	1	Type of data.	yes	00.0
FileType	Unsigned [4]	1	Type of file.	yes	00.0
TInfo1 [7]	Unsigned [4]	2	Type dependant numeric information field 1.	no	00.0
TInfo2 [7]	Unsigned [4]	2	Type dependant numeric information field 2.	no	00.0
TInfo3 [7]	Unsigned [4]	2	Type dependant numeric information field 3.	no	00.0
TInfo4 [7]	Unsigned [4]	2	Type dependant numeric information field 4.	no	00.0
Comments	Unsigned [4]	1	Number of lines in the extra SAUCE comment block. 0 indicates no comment block is present.	no	00.0
TFlags [7]	Unsigned [4]	1	Type dependant flags.	no	00.1
TInfoS [7]	ZString [5]	22	Type dependant string information field	no	00.5

A SAUCE comment block is an optional, variable sized structure that holds up to 255 lines of additional information, each line 64 characters wide. There are as many comment lines as is mentioned in the Comments field of the SAUCE record. If the Comments field is set to 0, there should not be a comment block at all. The comment block has the following layout:

Field name	Type	Size	Description	Required [1]	Revision [2]
ID	Character [3]	5	SAUCE comment block identification. This should be equal to "COMNT".	yes	00.0
Comment Line 1	Character [3]	64	Line of text.	yes	00.0
...					
Comment Line x	Character [3]	64	Line of text.	yes	00.0

Notes:

1. You need to provide a correct value for all required fields. A non-required field should either be set to the correct value, or when not used should be filled with spaces for the Character fields, set to 0 for the Unsigned fields or filled with binary zeroes for the ZString field.
2. The revision this field was introduced. Before its introduction, it was a filler and should have been set to binary 0.
3. The Character type is a string of characters encoded according to code page 437 (IBM PC / OEM ASCII). It is neither a pascal type string with a leading length byte nor is it a C-style string with a trailing terminator character. Any value shorter than the available length should be padded with spaces.
   - A Character field should be filled with spaces to indicate it is empty / unused.
   - Example: the ID field is 5 bytes long and contains the string "SAUCE".
   - I have seen SAUCE where Character fields were terminated with a binary 0 with the remainder containing garbage. When making a SAUCE reader, it is a good idea to properly handle this case. When writing SAUCE, stick with space padding.
   - Character fields should contain only plain text, do not insert formatting codes such as ansi escape codes, pcboard color codes, html tags, ...
   - Do not assume that a viewer will display this in a particular font or even if the font is fixed width or not.
   - Prior to revision 5, the SAUCE specifications specified ‘ASCII characters’ which implies code page 437. There are ANSI files out there however where the artist assumed an ANSI according to his native code page and also has SAUCE assumed as such.
4. An unsigned binary value of 1 byte (0 to 255), 2 bytes (0 to 65535) or 4 bytes (0 to 4294967295) stored in intel little-endian format.
   - Note that the FileSize field prior to revision 5 was listed as signed. This was an artefact of Turbo Pascal – a dominant programming language in the early 1990s – supporting a signed

integer of 4 bytes but not having an unsigned integer of 4 bytes. Since a file size can never be negative, it is safe to assume unsigned.
    - SAUCE prior to revision 5 only allowed SAUCE on files up to 2Gb in size, which back then was hardly a problem since hard disks were typically smaller.
5. A C-style zero terminated string of characters encoded according to code page 437 (IBM PC / OEM ASCII). The part of the string not used should be filled with binary zero. Prior to revision 00.5 this field was a filler and was expected to be set to all binary zero, the different approach compared to the other Character fields serves as extra backwards compatibility.
6. If you read a SAUCE record with a version number your software does not support, do not try to interpret the rest of the SAUCE record according to the 00 version spec.
7. The interpretation of the TInfo1, TInfo2, TInfo3, TInfo4, TFlags and TInfoS fields are dependant on the DataType and FileType fields.
8. FileSize was required in SAUCE prior to revision 5, it is now optional to be more in line with established reality. Many SAUCEd files have an incorrectly set FileSize. If you need to add SAUCE to a file larger than 4Gb, set FileSize equal to 0.

An Example C or C++ SAUCE record looks like this:

```
struct SAUCE
{
    char           ID[5];
    char           Version[2];
    char           Title[35];
    char           Author[20];
    char           Group[20];
    char           Date[8];
    unsigned long  FileSize;
    unsigned char  DataType;
    unsigned char  FileType;
    unsigned short TInfo1;
    unsigned short TInfo2;
    unsigned short TInfo3;
    unsigned short TInfo4;
    unsigned char  Comments;
    unsigned char  TFlags;
    char           TInfoS[22];
};
```

SAUCE DataType

SAUCE currently supports the following values for DataType:

DataType	Name	Description	Revision
0	None	Undefined filetype. You could use this to add SAUCE to a custom or proprietary file, without giving it any particular meaning or interpretation.	00.0
1	Character	A character based file. These files are typically interpreted sequentially. Also known as streams.	00.0
2	Bitmap [1]	Bitmap graphic and animation files.	00.0
3	Vector	A vector graphic file.	00.0
4	Audio [2]	An audio file.	00.0
5	BinaryText	This is a raw memory copy of a text mode screen. Also known as a .BIN file. This is essentially a collection of character and attribute pairs.	00.0
6	XBin	An XBin or eXtended BIN file.	00.2
7	Archive	An archive file.	00.2
8	Executable	A executable file.	00.2

Notes:

1. Prior to revision 5 this was called Graphics.
2. Prior to revision 5 this was called Sound.

SAUCE FileType

Each DataType has 1 or more possible values for FileType, and associated with each FileType value

there is also an interpretation of the TInfo and Flags fields:

DataType	FileType	Name	Description	TInfo1 [1]	TInfo2 [1]	TInfo3 [1]	TInfo4 [1]	Flags [1]	TInfoS [2]
None	0	-	Undefined filetype.	0	0	0	0	0	0
Character	0	ASCII	Plain ASCII text file with no formatting codes or color codes.	Character width [3]	Number of lines [4]	0	0	ANSiFlags	FontName
Character	1	ANSi	A file with ANSi coloring codes and cursor positioning.	Character width [3]	Number of lines [4]	0	0	ANSiFlags	FontName
Character	2	ANSiMation	Like an ANSi file, but it relies on a fixed screen size.	Character width [3]	Character screen height [5]	0	0	ANSiFlags	FontName
Character	3	RIP script	Remote Imaging Protocol graphics.	Pixel width (640)	Pixel height (350)	Number of colors (16)	0	0	0
Character	4	PCBoard	A file with PCBoard color codes and macros, and ANSi codes.	Character width [3]	Number of lines [4]	0	0	0	0
Character	5	Avatar	A file with Avatar color codes, and ANSi codes.	Character width [3]	Number of lines [4]	0	0	0	0
Character	6	HTML	HyperText Markup Language	0	0	0	0	0	0
Character	7	Source	Source code for some programming language. The file extension should determine the programming language.	0	0	0	0	0	0
Character	8	TundraDraw	A TundraDraw file. Like ANSI, but with a custom palette.	Character width [3]	Number of lines [4]	0	0	0	0
Bitmap	0	GIF	CompuServe Graphics Interchange Format	Pixel width	Pixel height	Pixel depth [6]	0	0	0
Bitmap	1	PCX	ZSoft Paintbrush PCX	Pixel width	Pixel height	Pixel depth [6]	0	0	0
Bitmap	2	LBM/IFF	DeluxePaint LBM/IFF	Pixel width	Pixel height	Pixel depth [6]	0	0	0
Bitmap	3	TGA	Targa Truecolor	Pixel width	Pixel height	Pixel depth [6]	0	0	0
Bitmap	4	FLI	Autodesk FLI animation	Pixel width	Pixel height	Pixel depth [6]	0	0	0
Bitmap	5	FLC	Autodesk FLC animation	Pixel width	Pixel height	Pixel depth [6]	0	0	0
Bitmap	6	BMP	Windows or OS/2 Bitmap	Pixel width	Pixel height	Pixel depth [6]	0	0	0
Bitmap	7	GL	Grasp GL Animation	Pixel width	Pixel height	Pixel depth [6]	0	0	0
Bitmap	8	DL	DL Animation	Pixel width	Pixel height	Pixel depth [6]	0	0	0

DataType	FileType	Name	Description	TInfo1 [1]	TInfo2 [1]	TInfo3 [1]	TInfo4 [1]	Flags [1]	TInfoS [2]
Bitmap	9	WPG	Wordperfect Bitmap	Pixel width	Pixel height	Pixel depth [6]	0	0	0
Bitmap	10	PNG	Portable Network Graphics	Pixel width	Pixel height	Pixel depth [6]	0	0	0
Bitmap	11	JPG/JPeg	JPeg image (any subformat)	Pixel width	Pixel height	Pixel depth [6]	0	0	0
Bitmap	12	MPG	MPeg video (any subformat)	Pixel width	Pixel height	Pixel depth [6]	0	0	0
Bitmap	13	AVI	Audio Video Interleave (any subformat)	Pixel width	Pixel height	Pixel depth [6]	0	0	0
Vector	0	DXF	CAD Drawing eXchange Format	0	0	0	0	0	0
Vector	1	DWG	AutoCAD Drawing File	0	0	0	0	0	0
Vector	2	WPG	WordPerfect or DrawPerfect vector graphics	0	0	0	0	0	0
Vector	3	3DS	3D Studio	0	0	0	0	0	0
Audio	0	MOD	4, 6 or 8 channel MOD (NoiseTracker)	0	0	0	0	0	0
Audio	1	669	Renaissance 8 channel 669	0	0	0	0	0	0
Audio	2	STM	Future Crew 4 channel ScreamTracker	0	0	0	0	0	0
Audio	3	S3M	Future Crew variable channel ScreamTracker 3	0	0	0	0	0	0
Audio	4	MTM	Renaissance variable channel MultiTracker	0	0	0	0	0	0
Audio	5	FAR	Farandole composer	0	0	0	0	0	0
Audio	6	ULT	UltraTracker	0	0	0	0	0	0
Audio	7	AMF	DMP/DSMI Advanced Module Format	0	0	0	0	0	0
Audio	8	DMF	Delusion Digital Music Format (XTracker)	0	0	0	0	0	0
Audio	9	OKT	Oktalyser	0	0	0	0	0	0
Audio	10	ROL	AdLib ROL file (FM audio)	0	0	0	0	0	0
Audio	11	CMF	Creative Music File (FM Audio)	0	0	0	0	0	0
Audio	12	MID	MIDI (Musical Instrument Digital Interface)	0	0	0	0	0	0
Audio	13	SADT	SAdT composer (FM Audio)	0	0	0	0	0	0
Audio	14	VOC	Creative Voice File	0	0	0	0	0	0
Audio	15	WAV	Waveform Audio File Format	0	0	0	0	0	0
Audio	16	SMP8	Raw, single channel 8bit sample	Sample rate [7]	0	0	0	0	0

DataType	FileType	Name	Description	TInfo1 [1]	TInfo2 [1]	TInfo3 [1]	TInfo4 [1]	Flags [1]	TInfoS [2]
Audio	17	SMP8S	Raw, stereo 8 bit sample	Sample rate [7]	0	0	0	0	0
Audio	18	SMP16	Raw, single-channel 16 bit sample	Sample rate [7]	0	0	0	0	0
Audio	19	SMP16S	Raw, stereo 16 bit sample	Sample rate [7]	0	0	0	0	0
Audio	20	PATCH8	8 Bit patch file	0	0	0	0	0	0
Audio	21	PATCH16	16 bit patch file	0	0	0	0	0	0
Audio	22	XM	FastTracker][module	0	0	0	0	0	0
Audio	23	HSC	HSC Tracker (FM Audio)	0	0	0	0	0	0
Audio	24	IT	Impulse Tracker	0	0	0	0	0	0
BinaryText	Variable [8]	-	Binary screen image	0	0	0	0	ANSiFlags	FontName
XBin	0	-	eXtended Bin	Character width [3]	Number of lines [4]	0	0	0	0
Archive	0	ZIP	PKWare Zip.	0	0	0	0	0	0
Archive	1	ARJ	Archive Robert K. Jung	0	0	0	0	0	0
Archive	2	LZH	Haruyasu Yoshizaki (Yoshi)	0	0	0	0	0	0
Archive	3	ARC	S.E.A.	0	0	0	0	0	0
Archive	4	TAR	Unix TAR	0	0	0	0	0	0
Archive	5	ZOO	ZOO	0	0	0	0	0	0
Archive	6	RAR	RAR	0	0	0	0	0	0
Archive	7	UC2	UC2	0	0	0	0	0	0
Archive	8	PAK	PAK	0	0	0	0	0	0
Archive	9	SQZ	SQZ	0	0	0	0	0	0
Executable	0	-	Any executable file. .exe, .dll, .bat, ... Executable scripts such as .vbs should be tagged as Source.	0	0	0	0	0	0

Notes:

1. A 0 here means the value should always be set to zero, otherwise it can optionally contain the matching value.
2. A 0 here means the field should be set to all binary zeroes, otherwise it can optionally contain a a set of flags.
3. The width in characters the screen or window should be set to in order to properly display the file. A value of 0 means no width was provided and you should use an appropriate default (usually 80).
4. Either 0 or the number of lines the file occupies when fully rendered. Some ANSI files have incorrect values for this, having the screen height instead. You could use the value as a guide for preallocating a rendering buffer. Do not assume the value is correct, the actual number of screen lines could be lower or higher.
5. For ANSI-animations to render and animate properly, they require the rendering window to be set to a specific size. Without this, the scrolling and absolute positioning may not work as intended and cause the animation to be garbled. When omitted, it is probably a good idea to assume a 80 by 25 character window.
6. The number of bits for each pixel. A monochrome image has 1 bit per pixel. 16 and 256 color images have 4 and 8 bits per pixel respectively, possibly assisted by a palette to select colors from a larger color set. Truecolor images are 24 bit. A 32 bit image typically means a truecolor image with an alpha transparency component. While there are currently bitmap formats that allow for even larger pixel depths. None of them are currently provided for via SAUCE. Some image formats have a fixed number of colors, others are variable.
7. The sampling rate the sample was recorded at.
8. The BinaryText datatype does not have a file type, instead the FileType field is used to encode the width of the image. In order to maximize the possible range of widths, the width is halved

allowing the FileType to specify any even width up to 510 (255*2) characters. Odd sizes are not supported, but this was never a real problem because video cards also only allowed even widths. Set the FileType to half the character width of the image, the height of the image can then be inferred by the formula:

Character Height = (File size - SAUCE information$_{(SAUCE\ record,\ comment\ block\ and\ EOF\ byte)}$ *) / (FileType * 2 * 2*$_{(Character\ /\ attribute\ pairs)}$*)*

ANSiFlags

ANSiFlags allow an author of ANSi and similar files to provide a clue to a viewer / editor how to render the image. The 8 bits in the ANSiFlags contain the following information:

0	0	0	A	R	L	S	B

These bits are interpreted as:

- B: Non-blink mode (iCE Color).
  When 0, only the 8 low intensity colors are supported for the character background. The high bit set to 1 in each attribute byte results in the foreground color blinking repeatedly.
  When 1, all 16 colors are supported for the character background. The high bit set to 1 in each attribute byte selects the high intensity color instead of blinking.
- LS: Letter-spacing (a.k.a. 8/9 pixel font selection). Introduced in Version 00.5
  Fixed-width text mode as used in DOS and early graphics based computers such as the Amiga used bitmap fonts to display text. Letter-spacing and line spacing is a part of the font bitmap, so the font box inside each bitmap was a bit smaller than the font bitmap.
  For the VGA, IBM wanted a smoother font. 2 more lines were added, and the letter-spacing was removed from the font and instead inserted by the VGA hardware during display. All 8 pixels in the font could thus be used, and still have a 1 pixel letter spacing. For the line graphics characters, this was a problem because they needed to match up with their neighbours on both sides. The VGA hardware was wired to duplicate the 8th column of the font bitmap for the character range C0h to DFh. In some code pages was undesired, so this feature could be turned off to get an empty letter spacing on all characters as well (via the ELG field (Enable Line Graphics Character Codes) in the Mode Control register (10h) of the VGA Attribute Controller (3C0h). While the VGA allows you to enable or disable the 8th column duplication, there is no way to specify which range of characters this needs to be done for, the C0h to DFh range is hardwired into the VGA.
  These 2 bits can be used to select the 8 pixel or 9 pixel variant of a particular font:
  - 00: Legacy value. No preference.
  - 01: Select 8 pixel font.
  - 10: Select 9 pixel font.
  - 11: Not currently a valid value.

  Changing the font width and wanting to remain at 80 characters per row means that you need to adjust for a change in horizontal resolution (from 720 pixels to 640 or vice versa). When you are trying to match the original aspect ratio (see the AR bits), you will need to adjust the vertical stretching accordingly.
  Only the VGA (and the Hercules) video cards actually supported fonts 9 pixels wide. SAUCE does not prevent you from specifying you want a 9 pixel wide font for a font that technically was never used that way. Note that duplicating the 8th column on non-IBM fonts (and even some code pages for IBM fonts) may not give the desired effect.
  Some people like the 9 pixel fonts, some do not because it causes a visible break in the 3 ‘shadow blocks’ (B0h, B1h and B2h)
- AR: Aspect Ratio. Introduced in Version 00.5
  Most modern display devices have square pixels, but that has not always been the case. Displaying an ANSI file that was created for one of the older devices on a device with square pixels will vertically compress the image slightly. This can be compensated for by either taking a font that is slightly taller than was used on the legacy device, or by stretching the rendered image.
  These 2 bits can be used to signal that the image was created with square pixels in mind, or that it was created for a legacy device with the elongated pixels:
  - 00: Legacy value. No preference.
  - 01: Image was created for a legacy device. When displayed on a device with square pixels, either the font or the image needs to be stretched.
  - 10: Image was created for a modern device with square pixels. No stretching is desired on a device with square pixels.
  - 11: Not currently a valid value.

FontName

The FontName field allows an author of ANSi and similar files to provide a clue to the viewer / editor which font to use to render the image.

Prior to revision 00.5, this field was a filler, so this field will be empty for legacy files. Supplying a font name is not required.

Font name [1]	Font size	Resolution [2]	Aspect ratio [3]		Vertical stretch [6]	Description
			Display [4]	Pixel [5]		
`IBM VGA`	9×16 [7]	720×400	4:3	20:27 (1:1.35)	35%	Standard hardware font on VGA cards for 80×25 text mode (code page 437)
	8×16	640×400	4:3	6:5 (1:1.2)	20%	Modified stats when using an 8 pixel wide version of `"IBM VGA"` or code page variant.
`IBM VGA50`	9×8 [7]	720×400	4:3	20:27 (1:1.35)	35%	Standard hardware font on VGA cards for condensed 80×50 text mode (code page 437)
	8×8	640×400	4:3	5:6 (1:1.2)	20%	Modified stats when using an 8 pixel wide version of `"IBM VGA50"` or code page variant.
`IBM VGA25G`	8×19	640×480	4:3	1:1	0%	Custom font for emulating 80×25 in VGA graphics mode 12 (640×480 16 color) (code page 437).
`IBM EGA`	8×14	640×350	4:3	35:48 (1:1.3714)	37.14%	Standard hardware font on EGA cards for 80×25 text mode (code page 437)
`IBM EGA43`	8×8	640×350	4:3	35:48 (1:1.3714)	37.14%	Standard hardware font on EGA cards for condensed 80×43 text mode (code page 437)
`IBM VGA` *`###`* [8]	9×16 [7]	720×400	4:3	20:27 (1:1.35)	35%	Software installed code page font for VGA 80×25 text mode
`IBM VGA50` *`###`* [8]	9×8 [7]	720×400	4:3	20:27 (1:1.35)	35%	Software installed code page font for VGA condensed 80×50 text mode
`IBM VGA25G` *`###`* [8]	8×19	640×480	4:3	1:1	0%	Custom font for emulating 80×25 in VGA graphics mode 12 (640×480 16 color).
`IBM EGA` *`###`* [8]	8×14	640×350	4:3	35:48 (1:1.3714)	37.14%	Software installed code page font for EGA 80×25 text mode
`IBM EGA43` *`###`* [8]	8×8	640×350	4:3	35:48 (1:1.3714)	37.14%	Software installed code page font for EGA condensed 80×43 text mode
`Amiga Topaz 1`	8×8 [9]	640×200	4:3	5:12 (1:2.4)	140%	Original Amiga Topaz Kickstart 1.x font. (A500, A1000, A2000)
`Amiga Topaz 1+`	8×8 [9]	640×200	4:3	5:12 (1:2.4)	140%	Modified Amiga Topaz Kickstart 1.x font. (A500, A1000, A2000)
`Amiga Topaz 2`	8×8 [9]	640×200	4:3	5:12 (1:2.4)	140%	Original Amiga Topaz Kickstart 2.x font (A600, A1200, A4000)
`Amiga Topaz 2+`	8×8 [9]	640×200	4:3	5:12 (1:2.4)	140%	Modified Amiga Topaz Kickstart 2.x font (A600, A1200, A4000)
`Amiga P0T-NOoDLE`	8×8 [9]	640×200	4:3	5:12 (1:2.4)	140%	Original P0T-NOoDLE font.
`Amiga MicroKnight`	8×8 [9]	640×200	4:3	5:12 (1:2.4)	140%	Original MicroKnight font.
`Amiga MicroKnight+`	8×8 [9]	640×200	4:3	5:12 (1:2.4)	140%	Modified MicroKnight font.
`Amiga mOsOul`	8×8 [9]	640×200	4:3	5:12 (1:2.4)	140%	Original mOsOul font.
`C64 PETSCII unshifted`	8×8 [10]	320×200	4:3	5:6 (1:1.2)	20%	Original Commodore PETSCII font (PET, VIC-20, C64, CBM-II, Plus/4, C16, C116 and C128) in the unshifted mode. Unshifted mode (graphics) only has uppercase letters and additional graphic characters. This is the normal boot font.
`C64 PETSCII shifted`	8×8 [10]	320×200	4:3	5:6 (1:1.2)	20%	Original PETSCII font in shifted mode. Shifted mode (text) has both uppercase and lowercase letters. This mode is actuated by pressing

Atari ATASCII	8×8 [11]	320×192	4:3	4:5 (1:1.25)	25%	Shift+Commodore key. Original ATASCII font (Atari 400, 800, XL, XE)

Notes:

1. The font name the author designed the artwork for. Since these are typically bitmapped fonts that only support 256 characters, these fonts have a fixed size and are designed with a specific use in mind. Characters in the range 32-127 tend to follow the ASCII encoding which is now also the basis of the same range in Unicode. Control characters in the range 0 to 31 are typically not displayable in text mode formats like ANSI files and require more direct access to the hardware to get them displayed. The visual appearance of these control characters is not part of the ASCII standard. Characters in the range 128-255 are different for each font. For the IBM PC, IBM designed their own encoding which later became known as code page 437. Some other systems (Amiga, C64, ...) used their own encoding for the high characters that did not follow the IBM code page convention. If you do not support a particular font in this list, either fallback to an appropriate less specific definition of the same font, or fallback to whatever default font your program supports.
   As a simple but effective fallback strategy for the IBM family of fonts:
   - If you do not support the `VGA25G`, `VGA50` or `EGA43` variant of a font, fallback to the `VGA` or `EGA` variant. (string-replace `"VGA50"` and `"VGA25G"` with `"VGA"` and `"EGA43"` with `"EGA"`)
   - If you do not support `EGA` fonts, fallback to the matching `VGA` variant and vice versa. (string-replace `"EGA"` with `"VGA"` or vice versa).
   - Code page `872` and `855` are mostly interchangeable (only different for the euro sign).
   - Code page `858` and `850` are mostly interchangeable (only different for the euro sign).
   - Code page `865` and `437` are mostly interchangeable (9Bh ‘ř’ replacing ‘ťť’ and 9Dh ‘Ř’ replacing ‘Ą’).
   - If you do not support the specific code page, fallback to default variant. (remove `" ###"` from the end of the string)
   - Fallback to whatever font your program uses as default.

   For the Amiga family of fonts:
   - If you do not support the `"+"` version of a font, fallback to the normal version. (remove the `"+"` from the string)
   - If you do not support the custom font, fallback to `Amiga Topaz 1`
   - Fallback to whatever font your program uses as default.
2. The screen resolution the font was intended for.
3. The aspect ratio is the ratio of the width of a shape to its height. The aspect ratio is expressed as two numbers separated by a colon (width:height). These values do not express an actual measurement, they represent the relation between width and height. If the width is larger than the height the shape has a “landscape” orientation. Width equal to height gives a square. Width smaller than height gives a “portrait” orientation.
4. The aspect ratio of the display device the font was intended for. Up until around 2003 the common display device was either a CRT computer monitor or a CRT TV which usually had a display aspect ratio of 4:3, and occasionally 5:4. Those formats are being replaced by “widescreen” displays commonly having a 16:10 or 16:9 aspect ratio.
5. Modern display devices (LCD, LED and plasma) tend to have square pixels or at least as near square as is technically feasible, because square pixels make it easy to draw squares and circles. For various technical reasons square pixels have not always been the norm however.
   The downside of this is that if one tries to display an image or font on a display with a different aspect ratio than it was intended for, the image will appear to be stretched or compressed. All the old display modes tended to have a pixels that were taler than they were wide, so they will appear compressed vertically on a display with a 1:1 pixel aspect ratio.
   You can compensate for this compression by stretching the image, but this will introduce pixelation when you just duplicate pixel lines or blurring when you apply anti-aliassing or interpolation.
   Similar to trying to display a 4:3 TV signal on a widescreen display, some people will prefer the compressed but ‘native pixel’ sharpness, and others will prefer the native dimension and accept the loss in sharpness.
   The pixel aspect ratio is obtained by dividing the display aspect ratio width by the horizontal resolution and dividing the display aspect ratio height by the vertical resolution and then reducing both sides of the ratio.
6. The stretching percentage that needs to be applied to the composed bitmap so that its relative size matches the original device it was intended for. The actual calculation for this stretch percentage would normally involve knowing the current display aspect ratio as well as the resolution. However since most modern screens have a 1:1 aspect ratio, this can simplified to:
   *((pixel aspect ratio height / pixel aspect ratio width) - 1) * 100*

Or alternatively:
*(((display aspect height / y-resolution) / (display aspect width / x-resolution)) -1) * 100*

7. 9 pixel fonts. to be completed
8. The "###" is a placeholder, these 3 characters should be replaced by one of the code pages IBM/Microsoft defined for use in DOS. The possible values for code page ### are:
   - 437: The character set of the original IBM PC. Also known as 'MS-DOS Latin US'.
   - 720: Arabic. Also known as 'Windows-1256'.
   - 737: Greek. Also known as 'MS-DOS Greek'.
   - 775: Baltic Rim (Estonian, Lithuanian and Latvian). Also known as 'MS-DOS Baltic Rim'.
   - 819: Latin-1 Supplemental. Also known as 'Windows-28591' and 'ISO/IEC 8859-1'.
     It is commonly mistaken for 'Windows-1252' which it resembles except for the 80h-9fh range (C1 control codes).
   - 850: Western Europe. Also known as 'MS-DOS Latin 1'.
     Designed to include all the language glyphs, part of the line graphics characters were sacrificed, which made some DOS programs appear strange. Because of this most systems in these countries stuck with CP437 instead.
   - 852: Central Europe (Bosnian, Croatian, Czech, Hungarian, Polish, Romanian, Serbian and Slovak). Also known as 'MS-DOS Latin 2'.
     Designed to include all the language glyphs, part of the line graphics characters were sacrificed, which made some DOS programs appear strange. Because of this, several countries adopted their own unofficial encoding system instead.
   - 855: Cyrillic (Serbian, Macedonian Bulgarian, Russian). Also known as 'MS-DOS Cyrillic'.
     Used in Serbia, Macedonia and Bulgaria, but eventually replaced by CP866. Never caught on in Russia.
   - 857: Turkish. Also known as 'MS-DOS Turkish'.
     Based on CP850 and designed to include all characters from ISO 8859-9 (with a different encoding).
   - 858: Western Europe.
     Identical to CP850 but has the euro sign at D5h instead. Also known as 'Modified code page 850'.
   - 860: Portuguese. Also known as 'MS-DOS Portuguese'.
     Even though this code page preserves all the line graphics characters, Brazil has mainly adopted CP850 which does not.
   - 861: Icelandic. Also known as 'MS-DOS Icelandic'.
   - 862: Hebrew. Also known as 'MS-DOS Hebrew'.
     Obsolete on modern operating systems, replaced by Unicode which preserves the logical bidirectional order (which DOS could not).
   - 863: French Canada. Also known as 'MS-DOS French Canada'.
   - 864: Arabic.
     Sacrifices all of the line graphics characters (encoding the single line box characters differently) in order to get more Arabic symbols. CP720 does preserve the line graphics characters.
   - 865: Nordic.
   - 866: Cyrillic.
   - 869: Greek 2. Also known as 'MS-DOS Greek 2'.
     Designed to include all the glyphs from ISO 8859-7 (with a different encoding), part of the line graphics characters were sacrificed, which made some DOS programs appear strange. This made CP869 unpopular and most Greek systems used CP737 instead.
   - 872: Cyrillic.
     Identical to CP855 but has the euro sign at CFh instead.
   - KAM: 'Kamenický' encoding. Also known as 'KEYBCS2'.
     Used for Czech, this code page was custom designed as an alternative for CP852 to preserved the line graphics characters while providing the characters needed for the Czech language. Supported in some DOS clones under the unofficial code page number 867 or 895.
   - MAZ: 'Mazovia' encoding.
     Used for Polish, this code page was custom designed as an alternative for CP852 to preserved the line graphics characters while providing the characters needed for the Polish language. Supported in some DOS clones under the unofficial code page number 667 or 790.
   - MIK: Cyrillic.
     Most widespread codepage in Bulgaria. Supported in some DOS clones under the unofficial code page number 866.

   So for example "IBM VGA 855" would select the VGA 9×16 font for code page 855 for 80×25 text mode.
   Unsupported code pages:

- `667`: Unofficial code page number for ‘Mazovia’ encoding, use `"MAZ"` instead.
- `790`: Unofficial code page number for ‘Mazovia’ encoding on FreeDOS, use `"MAZ"` instead.
- `866`: Unofficial code page number for ‘MIK’ encoding, use `"MIK"` instead.
- `867`: Unofficial code page number for ‘Kamenický’ encoding, use `"KAM"` instead.
- `895`: Unofficial code page number for ‘Kamenický’ encoding, use `"KAM"` instead.
- `991`: Unofficial code page number, depending on the Operating System or device its meaning could differ. Probably refers to `"MAZ"`.

9. The Amiga 500 typically works at a 640×200 resolution or 640×400 in interlaced mode on NTSC monitor (USA and Japan), and at 640×256 (640×512 interlaced) on PAL monitors (most of Europe). In interlaced mode, rather than condensing the font to achieve double the amount of lines of text, the font instead was stretched to double its size.
   It is not uncommon to now find programs using a pre-stretched 8×16 font for displaying Amiga ANSI files. In that case, the pixel ratio mentioned here should be adjusted accordingly to 5:6 (1:1.2) and the vertical stretching to 20%
10. This font was used in a 40×25 character mode. It may be necessary to double the size of the font or image to get the "chunky" look.
11. This font was used in a 40×24 character mode. It may be necessary to double the size of the font or image to get the "chunky" look.

Revision History

Version.Revision	Date	Description
00.0	March 1, 1994	SAUCE is released as part of the monthly ACiD acquisition artpack.
00.1	June 30, 1994	Added the Flags field in the SAUCE record.
00.2	July 13, 1996	Added support for: • Character/Html • Bitmap/PNG (replacing Bitmap/SBM) • Vector/3DS (replacing Vector/SVI) • Audio/XM • Audio/HSC • XBin • Archive/*various formats* • Executable
00.3	July 29, 1996	Character/Avatar extended to use TInfo1 and TInfo2. Added support for Audio/IT
00.4	May 31, 1997	Technical cleanup
00.5	November 12, 2013	Complete rewrite of the spec – this document. • Filler is now used as a type dependant string. • Flags for ANSI files and similar has been changed to include letter-spacing selection, as well as aspect ratio selection. • ANSI files and similar can now specifiy a font.