MSAgent Character Data Specification

Spec Version:	1.6
Last updated:	November 6, 2022
Written by:	Remy Lebeau
Copyright:	© 2003-2022 Lebeau Software. All Rights Reserved.

DISCLAIMER

Permission to copy and distribute this document is granted so long as the contents of this document are not altered. Any additions or modifications to this document shall be submitted to the author for review and inclusion in future releases by the author. Contributors will be credited accordingly.

In no event shall Lebeau Software, Microsoft Corporation, or any individuals or other organizations listed in this document, be held liable for any use of the information contained herein. This information is provided as-is for educational purposes.

The information contained in this specification was gathered through observation of Microsoft Agent data files, and may not be entirely accurate or complete. Use it at your own risk.

This specification is in no way sponsored, supported, or condoned by Microsoft Corporation, and is the sole work of Lebeau Software and credited individuals and organizations.

Microsoft Agent is Copyright ©1996-1998 Microsoft Corporation. All Rights Reserved.

Table of Contents

General

- Introduction
- Data Types
- Win32 API Data Structures
- Data Structure Lists
- Common Data Structures
- Compression Algorithm

Data Formats

- ACS Format
- ACF Format
- ACA Format

Introduction

The following specification outlines the ACS, ACF, and ACA file formats of Microsoft Agent version 2. At this time, this specification does not explore the file formats for Microsoft Agent version 1. I hope to include that information in a future version of this specification.

I have been a long-time supporter of the MSAgent technology, and became curious about its inner workings. This specification outlines my efforts to learn the format of the data files used for the animated Characters, in preparation for future projects that can utilize existing and future MSAgent animated Characters without use of the MSAgent engine produced by Microsoft. Primarily, I want to do this to help to streamline the technology, improve upon it if possible, and then eventually port it to other platforms which Microsoft does not support MSAgent on.

Why do this? Because I think that MSAgent is a fascinating technology. It provides a relatively simple way to add animated, interactive personalities to software applications and web pages. Sure, other technologies exist that accomplish the same thing. But many of them are more complicated to develop with than MSAgent. I think MSAgent still has untapped potential. On the other hand, it has been on a downward spiral the past few years. Partially due to lack of new versions released by Microsoft. Partially due to lack of much visibility in the marketplace. I'd hate to see the technology die altogether. So I try to do my part to help it in any way I can.

Data Types

All structures mentioned in this specification are based on the following C/C++ data types:

DATA TYPE	DESCRIPTION	VALUE RANGE (inclusive)
BOOL	8-bit boolean	false (0) or true (non-zero)
BYTE	8-bit unsigned char	0 to 255
WCHAR	16-bit signed wchar_t	-32,768 to 32,767
SHORT	16-bit signed short	-32,768 to 32,767
USHORT	16-bit unsigned short	0 to 65,535
LONG	32-bit signed long	-2,147,483,648 to 2,147,483,647
ULONG	32-bit unsigned long	0 to 4,294,967,295

The high-order BYTE of a value is the most significant BYTE. The high-order bit of a BYTE is the most significant bit.

For example, a ULONG value of decimal 16909060 (hex 0x01020304) is stored as follows:

```
(hex)     0x04 0x03 0x02 0x01
(binary)  00000100 00000011 00000010 00000001
```

Win32 API Data Structures

The following structures are provided by the Windows API and are listed here for reference purposes. These structures are publicly documented by Microsoft, and as such their full descriptions are omitted from this specification.

GUID (MSDN reference)

A GUID is a globally unique identifier for an object.

DATA TYPE	QUANTITY	DESCRIPTION
ULONG	1	Data 1
USHORT	1	Data 2
USHORT	1	Data 3
BYTE	8	Data 4

BITMAPINFOHEADER (MSDN reference)

The BITMAPINFOHEADER structure contains information about the dimensions and color format of a device-independent bitmap (DIB).

DATA TYPE	QUANTITY	DESCRIPTION
ULONG	1	Size of this structure (in BYTEs)
LONG	1	Width (in pixels)
LONG	1	Height (in pixels)
USHORT	1	# of planes (always 0x01)
USHORT	1	# of bits per pixel
ULONG	1	Type of compression
ULONG	1	Size of image data (in BYTEs)
LONG	1	Horizontal resolution, pixels per meter
LONG	1	Vertical resolution, pixels per meter
ULONG	1	# of color indices in color table
ULONG	1	# of important color indices

RGBQUAD (MSDN reference)

The RGBQUAD structure describes a color consisting of relative intensities of red, green, and blue.

DATA TYPE	QUANTITY	DESCRIPTION
BYTE	1	Red value
BYTE	1	Green value
BYTE	1	Blue value
BYTE	1	Reserved (always 0x00)

ICONIMAGE

The ICONIMAGE structure contains information about the dimensions and color format of an icon.

DATA TYPE	QUANTITY	DESCRIPTION
BITMAPINFOHEADER	1	Icon Header
RGBQUAD	variable	Color Table
BYTE	variable	XOR Mask Bits
BYTE	variable	AND Mask Bits

LANGID (MSDN reference)

The LangID structure is used to specify a language locale, including primary and sublanguage identifiers. Refer to the Windows API documentation for the MAKELANGID() macro for a list of possible values.

DATA TYPE	QUANTITY	DESCRIPTION
USHORT	1	Primary/Secondary language ID

RECT (MSDN reference)

The RECT structure defines the coordinates of the upper-left and lower-right corners of a rectangle.

DATA TYPE	QUANTITY	DESCRIPTION
LONG	1	X-coordinate of the upper-left corner of the rectangle
LONG	1	Y-coordinate of the upper-left corner of the rectangle
LONG	1	X-coordinate of the lower-right corner of the rectangle
LONG	1	Y-coordinate of the lower-right corner of the rectangle

RGNDATAHEADER (MSDN reference)

The RGNDATAHEADER structure describes the data returned by the GetRegionData() function.

DATA TYPE	QUANTITY	DESCRIPTION
ULONG	1	Size of this structure (in BYTEs)
ULONG	1	Specifies the type of region
ULONG	1	# of rectangles that make up the region
ULONG	1	Size of the buffer required to receive the RECT structure that specifies the coordinates of the rectangles that make up the re
RECT	1	Specifies a bounding rectangle for the region in logical units

RGNDATA (MSDN reference)

The RGNDATA structure contains a header and an array of rectangles that compose a region. The rectangles are sorted top to bottom, left to right. They do not overlap.

DATA TYPE	QUANTITY	DESCRIPTION
RGNDATAHEADER	1	Region Header
BYTE	variable	Contains the RECT structures that make up the region

RIFF AUDIO (MSDN reference)

All Audio Data in Microsoft Agent is formatted as standardized Resource Interchange File Format (RIFF) chunks, as described in the WAV Format Specification. The RIFF and WAV specifications are provided by Microsoft separately.

The Audio Data can be used as-is with any RIFF-enabled output, such as DirectSound, the Win32 API PlaySound() function, etc. The Audio Data is never compressed in Microsoft Agent, although it may be compressed and/or encoded separately before the Character data files are compiled. The appropriate codec(s) need to be installed on the target machine beforehand in order to process the Audio Data for playback.

Data Structure Lists

Many of the structures described in this specification appear as standalone items, as well as lists of multiple items of the same structure type. When a list of items of a given structure type is present, the list items are immediately preceeded by a numeric value specifying how many items are in the list.

If a structure described in this specification is available in a list format, the definition of the structure will include the data type used for the list count, in parenthesis. When the list is used in another structure, the keyword "**LIST**" will be specified as the Quantity. For example:

SAMPLE (List Count: USHORT)

DATA TYPE	QUANTITY	DESCRIPTION
ULONG	1	Value 1
USHORT	2	Values 2-3
BYTE	4	Values 4-7

SOMETHINGELSE

DATA TYPE	QUANTITY	DESCRIPTION
SAMPLE	LIST	List of SAMPLE Items

In the example above, a list of 4 items of type SAMPLE would consist of 50 BYTEs total, as follows:

DATA TYPE	QUANTITY	DESCRIPTION
USHORT	1	# of Items in the List
SAMPLE	4	Items of SAMPLE type

For the rest of this specification, the general definition of a structure list is as follows:

<STRUCTURE> (List Count: <COUNTTYPE>)

DATA TYPE	QUANTITY	DESCRIPTION
<COUNTTYPE>	1	# of Items in the List
<STRUCTURE>	variable	Items of <STRUCTURE> type

Common Data Structures

The structures in this section describe data that is commonly used in the ACS, ACF, and ACA formats.

STRING (List Count: USHORT)

The STRING structure contains printable characters, usually for human-readible display.

DATA TYPE	QUANTITY	DESCRIPTION
ULONG	1	# of Characters
WCHAR	variable	String Characters

The character data in the ACS format includes a Null Terminator (0x0000). However, the number of characters does not include the terminator. In other words, a STRING with a count of 5 will contain 5 characters and a 6th character for the terminator. If the count is 0, then no character data is present at all, and no terminator is present.

The character data in the ACF format never includes a Null Terminator (0x0000).

The STRING structure is not used in the ACA format.

STATEINFO (List Count: USHORT)

The STATEINFO structure specifies the supported animations that are assigned to a particular Character State.

DATA TYPE	QUANTITY	DESCRIPTION
STRING	1	State Name
STRING	LIST	Animations in State

Refer to the "Designing Characters for Microsoft Agent" documentation provided by Microsoft for more information regarding how Character States work.

LOCALIZEDINFO (List Count: USHORT)

The LOCALIZEDINFO structure contains Character information that is localized to a particular language locale. A Character can support multiple locales.

DATA TYPE	QUANTITY	DESCRIPTION
LANGID	1	Language ID
STRING	1	Character Name
STRING	1	Character Description
STRING	1	Character Extra Data

PALETTECOLOR (List Count: ULONG)

The PALETTECOLOR structure contains an entry for the color palette that is used when rendering images. All images in a character must use the same color palette.

DATA TYPE	QUANTITY	DESCRIPTION
RGBQUAD	1	Color value

VOICEINFO

The VOICEINFO structure contains the default settings for the Character's spoken TTS output. Once a Character has been loaded, an application or web page can override these settings prior to performing TTS operations.

DATA TYPE	QUANTITY	DESCRIPTION
GUID	1	TTS Engine ID
GUID	1	TTS Mode ID
ULONG	1	Speed
USHORT	1	Pitch
BOOL	1	Extra Data is Present

If Extra Data is present, the following values immediately follow the above:

DATA TYPE	QUANTITY	DESCRIPTION
LANGID	1	Language ID
STRING	1	Language Dialect
USHORT	1	Gender
USHORT	1	Age
STRING	1	Style

BALLOONINFO

The BALLOONINFO structure contains the default settings for the Character's word balloon that is displayed during TTS output. Once a Character has been loaded, an application or web page can override these settings prior to performing TTS operations.

DATA TYPE	QUANTITY	DESCRIPTION
BYTE	1	# of Text Lines
BYTE	1	Characters per Line
RGBQUAD	1	Foreground Color
RGBQUAD	1	Background Color
RGBQUAD	1	Border Color
STRING	1	Font Name
LONG	1	Font Height (in logical units)
LONG	1	Font Weight (in the range 0 - 1000)
BOOL	1	Italicized
BYTE	1	unknown (possibly underline/strikeout flag?)

The Font Weight has been observed as having the following values:

```
0x0190 = normal
0x02BC = bold
```

The Unknown field has been observed as usually being 0.

TRAYICON

The TRAYICON structure describes the icon that appears in the Windows System Tray while the Character is loaded in the MSAgent engine (if the icon is available).

DATA TYPE	QUANTITY	DESCRIPTION
ULONG	1	Size of Monochrome Bitmap Data (in BYTEs)
ICONIMAGE	1	Monochrome Bitmap Data
ULONG	1	Size of Color Bitmap Data (in BYTEs)
ICONIMAGE	1	Color Bitmap Data

The icon is comprised of two separate images - a monochrome bitmap and a color bitmap. To produce the final image, first the colors of the monochrome bitmap are combined with the colors of the target device context using the Boolean AND operator. Then the colors of the color bitmap are combined with the colors of the target device context using the Boolean XOR operator.

In Win32 programming, DDBs (device-dependant bitmaps) can be created from the two ICONIMAGE structures by using CreateBitmap() and CreateDIBitmap(), and then the resulting HBITMAP handles can be passed to CreateIconIndirect() to create an HICON handle.

BRANCHINFO (List Count: BYTE)

The BRANCHINFO structure describes a particular branch for an animation frame. Branches are used to specify the order in which frames are displayed when playing an animation.

DATA TYPE	QUANTITY	DESCRIPTION
USHORT	1	0-based Index of Frame to jump to
USHORT	1	Probability %

Refer to the "Designing Characters for Microsoft Agent" documentation provided by Microsoft for more information regarding how Branches work.

DATABLOCK

The DATABLOCK structure contains an arbitrary-sized block of information.

DATA TYPE	QUANTITY	DESCRIPTION
ULONG	1	Size of Data (in BYTEs)
BYTE	variable	Data

COMPRESSED

The COMPRESSED structure contains an arbitrary-sized block of information that may or may not be compressed.

DATA TYPE	QUANTITY	DESCRIPTION
ULONG	1	Size of compressed Data (in BYTEs)
ULONG	1	Size of uncompressed Data (in BYTEs)
BYTE	variable	Data

If the Compressed Size is 0, the data is not compressed, and the size of the data is the specified Uncompressed Size.

If the Compressed Size is not 0, the data is compressed, and the size of the data after decompression should match the specified Uncompressed Size.

Refer to the Compression Algorithm section for details about the format used for compressing data.

Compression Algorithm

Microsoft Agent implements a custom scheme for compressing data. Below are direct quotes from Microsoft's own postings in the microsoft.public.msagent newsgroup describing the algorithm used for compressing and storing data.

February 3, 1999
Message ID: <eXJ1aJ#T#GA.160@uppssnewspub04.moswest.msn.net>

"The compression that Agent uses is a proprietary, lossless, bit level compressor....The images are stored only once...the compressor does run length encoding...the image is compressed as a contiguous buffer."

May 30, 2000
Message ID: <#OMEQ4ky$GA.240@cppssbbsa02.microsoft.com>#1/1

"Agent does NOT compress audio files."

November 1, 2001
Message ID: <#OFR1bvYBHA.1456@tkmsftngp07>

"Frames accept multiple images to enable a developer to create an animation frame out of multiple pieces...all the overlays and base image are composited for the frame."

February 21, 2002
Message ID: <3c751da1$1@news.microsoft.com>

"The transparent area around the a character's image and mouth image is compressed out."

February 25, 2002
Message ID: <3c7a78d3$1@news.microsoft.com>

"...the editor uses its own compression which is more efficient than GIF to compile the character."

COMPRESSION DETAILS

Compressed data is formatted as follows:

DATA TYPE	QUANTITY	DESCRIPTION
BYTE	1	Always 0x00
BYTE	variable	Compressed Bit Stream
BYTE	6	Always 0xFF

The Compressed Bit Stream contains a series of bit sequences, where each bit sequence describes either a single uncompressed BYTE, or a range of compressed BYTEs.

The high-order bit of a BYTE in the stream is the most significant bit. The bits of each BYTE are processed in order from least-significant bit to most-significant bit.

Compressed BYTEs are decompressed inside the destination buffer as the buffer is being written to. Meaning that after one or more BYTEs has been decompressed into the buffer, later bit sequences can make copies of those BYTEs in order to fill in other areas of the same buffer. If a range of BYTEs is duplicated multiple times in the original data, this approach cuts down on the number of bits needed within the bit stream to duplicate the same uncompressed BYTEs over and over in the buffer. In other words, the more repetition there is in the original data, the smaller the data will compress.

The first bit of a sequence specifies the type of data that follows the bit:

VALUE	DESCRIPTION
0	a single uncompressed BYTE
1	compressed range of at least 2 BYTEs

If a sequence represents an uncompressed BYTE, the 8 bits following the above bit are the uncompressed BYTE. Write this BYTE into the destination buffer at the current insertion point, and then increment the insertion point.

If a sequence represents a compressed BYTE range, the remaining bits of the sequence are processed as follows:

- The initial count of the number of BYTEs that will be decoded from the sequence is 2.
- Count the number of sequential bits which have a value of 1. The maximum number of bits is 3. This count specifies the number of bits that make up the next value in the sequence, as follows:

# SEQ BITS	VALUE BIT COUNT
0	6
1	9
2	12
3	20

  - If the count of sequential bits is less than 3, skip past the bit whose value of 0 ends the sequential bits. The next value in the sequence follows this skipped bit.
  - Otherwise, do not skip the 4th bit. It is part of the next value in the sequence.
- Using the bit count from the previous step, take the numeric value that is made up of the specified number of bits, and incremeent its value as follows:

BIT COUNT	VALUE TO ADD
6	1 (0x0001)
9	65 (0x0041)
12	577 (0x0241)
20	4673 (0x1241)

This final numeric value is an offset, in BYTEs, within the destination buffer, subtracted from the buffer's current insertion point.

If the bit count is 20:

  - If the numeric value is 1048575 (0x000FFFFF) before adding 4673, the end of the bit stream has been reached.
  - Otherwise, increment the count of BYTEs to be decoded by 1, and continue with the next steps.
- Following the offset bits, count the number of sequential bits which have a value of 1. The maximum number of bits is 11.
  - If the 12th bit is 1, an error occured.
  - Otherwise, continue with the next steps.
- Add the numeric value that is made up of the bits in the previous step, if any, to the count of BYTEs to be decoded.

- Do not count the bit whose value of 0 ends the sequential bits. Skip past this bit. The next value in the sequence follows this skipped bit.
- The count of sequential bits is also the number of remaining bits in the sequence.

- Add the numeric value that is made up of the remaining bits, if any, to the count of BYTEs to be decoded.

This final count is the number of BYTEs to copy from the above offset to the current insertion point. Copy the BYTEs one at a time, incrementing the insertion point after each BYTE, as the copying may overlap past the original insertion point.

COMPRESSION EXAMPLE

Binary values are typically written in order from most-significant bit to least-significant bit. In this example, they are written from least-significant bit to most-significant bit instead, to display the bits in processing order.

For example, a BYTE value of decimal 64 (hex 0x40) is typically written as **01000000** (MSB to LSB), but is written below as **00000010** (LSB to MSB) instead.

Here is an example of decoding a compressed block of Region Data from the ACSIMAGEINFO structure:

Compressed Data

```
00 40 00 04 10 D0 90 80
42 ED 98 01 B7 FF FF FF
FF FF FF
```

Bit Stream

```
00000010000000000010000000001000
00001011000010010000000101000010
10110111000110011000000011101101
11111111111111111111
```

Bit Stream Breakdown

```
000000100 000000000 100000000
010000000 10110000100 10000000101
000010101 101110001100l 100000001110110
11111111111111111111111
```

SEQUENCE BITS	DESCRIPTION
0	single uncompressed BYTE
00000100	value = 0x20
0	single uncompressed BYTE
00000000	value = 0x00
1	compressed BYTE range
0	offset bit count = 6, decoded BYTE count = 2
000000	offset = (0+1) BYTE
0	increment decoded count by 0
-	increment decoded count by 0
	copy (2+0+0) BYTEs from offset into buffer
0	single uncompressed BYTE
10000000	value = 0x01
1	compressed BYTE range
0	offset bit count = 6, decoded BYTE count = 2
110000	offset = (3+1) BYTEs
10	increment decoded count by 1
0	increment decoded count by 0
	copy (2+1+0) BYTEs from offset into buffer
1	compressed BYTE range
0	offset bit count = 6, decoded BYTE count = 2
000000	offset = (0+1) BYTE
10	increment decoded count by 1
1	increment decoded count by 1
	copy (2+1+1) BYTEs from offset into buffer
0	single uncompressed BYTE
00010101	value = 0xA8
1	compressed BYTE range
0	offset bit count = 6, decoded BYTE count = 2
111000	offset = (7+1) BYTEs
110	increment decoded count by 3
01	increment decoded count by 2
	copy (2+3+2) BYTEs from offset into buffer
1	compressed BYTE range
0	offset bit count = 6, decoded BYTE count = 2
000000	offset = (0+1) BYTE
1110	increment decoded count by 7
110	increment decoded count by 3
	copy (2+7+3) BYTEs from offset into buffer
1	compressed BYTE range
111	offset bit count = 20, decoded BYTE count = 3
1111111111111111111	0x000FFFFF (end of bit stream)

Decoded Data

```
20 00 00 00 01 00 00 00
00 00 00 00 A8 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

RGNDATAHEADER
    dwSize = 32
    iType = RDH_RECTANGLES
```

```
nCount = 0
nRgnSize = 168
rcBound = {0, 0, 0, 0}
```

ACS Format

The ACS data format contains everything about the Character in a single file. An ACS file is designed for high performance, since all of the relavant data is stored in one place.

An ACS file should be installed on the machine that will be accessing it. On Windows machines, ACS files are installed in the "**%WINDIR%\MSAgent\Chars**\" folder by default, where "%WINDIR%" is the folder that Windows itself is installed in. ACS files can also be loaded into the MSAgent engine programmably from any path on the machine or local network.

In addition to the Common Data Structures, the ACS format includes the following structures:

ACSLOCATOR

The ACSLOCATOR structure is used to specify the exact location within the data file where a particular block of information can be found.

DATA TYPE	QUANTITY	DESCRIPTION
ULONG	1	0-based BYTE Offset from the Beginning of File
ULONG	1	Size of Data (in BYTEs)

ACSHEADER

The ACS data format begins with a header structure which identifies the data as being the ACS format. It contains the locations of the rest of the ACS data.

DATA TYPE	QUANTITY	DESCRIPTION
ULONG	1	Signature, Always 0xABCDABC3
ACSLOCATOR	1	Location of ACSCHARACTERINFO Structure
ACSLOCATOR	1	Location of ACSANIMATIONINFO List
ACSLOCATOR	1	Location of ACSIMAGEINFO List
ACSLOCATOR	1	Location of ACSAUDIOINFO List

ACSCHARACTERINFO

The ACSCHARACTERINFO structure describes the dimensions and characteristics of the Character.

DATA TYPE	QUANTITY	DESCRIPTION
USHORT	1	Minor Version
USHORT	1	Major Version
ACSLOCATOR	1	Location of LOCALIZEDINFO List
GUID	1	Unique Identifier for the Character
USHORT	1	Character Width (in pixels)
USHORT	1	Character Height (in pixels)
BYTE	1	0-based Index of Transparent Color in Color Table
ULONG	1	Flags
USHORT	1	Animation Set Major Version?
USHORT	1	Animation Set Minor Version?
VOICEINFO	1	Voice Output Info (if enabled)
BALLOONINFO	1	Word Balloon Info (if enabled)
PALETTECOLOR	LIST	Color Table
BOOL	1	System Tray Icon is Enabled
TRAYICON	1	System Tray Icon (if enabled)
STATEINFO	LIST	Animation States

The Flags value is comprised of the following bits:

BITS	DESCRIPTION
0-3	unknown
4	Voice Output 0 = disabled 1 = enabled
5-7	unknown
8, 9	Word Balloon disabled/enabled (respectively) (I don't know why Microsoft separates these bits)
10-15	unknown
16-18	Word Balloon Styles Can contain any combination of the following: 0x01 = size to text enabled 0x02 = auto hide disabled 0x04 = auto pace disabled
19	unknown
20	Standard Animation Set 0 = not supported 1 = supported
21-31	unknown

The Animation Set Major Version has been observed as always being 2, and the Minor Version as begin 0.

ACSOVERLAYINFO (List Count: BYTE)

The ACSOVERLAYINFO structure describes information about a particular mouth overlay for an animation frame. Mouth overlays are displayed only during the Character's spoken TTS output.

DATA TYPE	QUANTITY	DESCRIPTION
BYTE	1	Overlay Type
BOOL	1	Replace the Top Image of the Frame
USHORT	1	0-based Index of Image in ACSIMAGEINFO List from the ACSHEADER structure
BYTE	1	unknown
BOOL	1	Region Data is Present
SHORT	1	X-offset from top of Frame (in pixels)
SHORT	1	Y-offset from top of Frame (in pixels)
USHORT	1	Width (in pixels)
USHORT	1	Height (in pixels)
DATABLOCK	1	Region Data (if present)

The Overlay Type can be one of the following:

```
0x00 = mouth closed
0x01 = mouth wide open 1
0x02 = mouth wide open 2
0x03 = mouth wide open 3
0x04 = mouth wide open 4
0x05 = mouth medium
0x06 = mouth narrow
```

The Unknown value has been observed as usually being 0.

The Region data is a RGNDATA structure.

A frame can be composited using multiple images. If the "Replace Top Image" flag is enabled, the top-most image defined for the frame is substituted with the overlay image. Otherwise, the overlay image is placed over the frame's top-most image instead.

Note: The Width and Height are expressed as values that have been divided by 2. I do not know why Microsoft is doing that.

ACSFRAMEIMAGE (List Count: USHORT)

The ACSFRAMEIMAGE structure describes a particular image used for an animation frame. A frame can be composited using multiple images, and a single image can be shared by multiple frames.

DATA TYPE	QUANTITY	DESCRIPTION
ULONG	1	0-based Index of Image in ACSIMAGEINFO List from the ACSHEADER structure
SHORT	1	X-offset from Top of Frame (in pixels)
SHORT	1	Y-offset from Top of Frame (in pixels)

ACSFRAMEINFO (List Count: USHORT)

The ACSFRAMEINFO structure describes a particular frame in an animation sequence.

DATA TYPE	QUANTITY	DESCRIPTION
ACSFRAMEIMAGE	LIST	Images used to Composite the Frame
USHORT	1	0-based Index in ACSAUDIOINFO List from the ACSHEADER structure
USHORT	1	Frame Duration (in 1/100 seconds)
SHORT	1	0-based Index of Frame to Exit to in Animation
BRANCHINFO	LIST	Frame Branches
ACSOVERLAYINFO	LIST	Mouth Overlays

The Frame Images are composited in reverse order from last to first.

Refer to the "Designing Characters for Microsoft Agent" documentation provided by Microsoft for more information regarding how Exit Frames work.

Note: the Exit Frame index may be -2. I am not sure the significance of this value yet, but it seems to only appear in the last frame(s) of an animation

ACSANIMATIONINFO (List Count: ULONG)

The ACSANIMATIONINFO structure describes information about a particular animation.

DATA TYPE	QUANTITY	DESCRIPTION
STRING	1	Animation Name
ACSLOCATOR	1	Location of Animation Information

The Animation Information is as follows:

DATA TYPE	QUANTITY	DESCRIPTION
STRING	1	Animation Name (in uppercase)
BYTE	1	Transition Type
STRING	1	Return Animation (in uppercase)
ACSFRAMEINFO	LIST	Animation Frames

The Transition Type can be one of the following values:

```
0x00 = use return animation
0x01 = use exit branches
0x02 = no transition
```

Refer to the "Designing Characters for Microsoft Agent" documentation provided by Microsoft for more information regarding how Animation Transitions work.

ACSIMAGEINFO (List Count: ULONG)

The ACSIMAGEINFO structure describes information about a particular frame image.

DATA TYPE	QUANTITY	DESCRIPTION
ACSLOCATOR	1	Location of Image Information
ULONG	1	Checksum?

The Image Information is as follows:

DATA TYPE	QUANTITY	DESCRIPTION
BYTE	1	unknown
USHORT	1	Width (in pixels)
USHORT	1	Height (in pixels)
BOOL	1	Image Data is Compressed
DATABLOCK	1	Image Data
COMPRESSED	1	Region Data

The Unknown value has been observed as begin either 0 or 1.

The Image Data is the raw bitmap bits. Refer to the Win32 API documentation for the BITMAPINFOHEADER structure for details about how the bits are formatted. The Image Data has 1 Plane and a Bit Count of 8, and does not use any pixel compression.

If the Image Data is compressed, the number of BYTEs needed for allocating a suitable buffer prior to decompressing the data can be calculated by rounding the Image Width up to a ULONG boundary and then multiplying it by the Image Height. For example:

((Width + 3) & 0xFC) * Height)

The Region Data is a RGNDATA structure.

ACSAUDIOINFO (List Count: ULONG)

The ACSAUDIOINFO structure describes information about a particular block of waveform audio.

DATA TYPE	QUANTITY	DESCRIPTION
ACSLOCATOR	1	Audio Data
ULONG	1	Checksum?

Refer to the RIFF Audio section for details about the format used for audio data.

ACF Format

The ACF data format contains just the Character's core information. The animation data is stored in separate ACA files, one for each animation. ACF/ACA files are designed to be download from a remote location, such as an HTTP server, and thus are relatively small to reduce network traffic. Because the animations are stored separately, they must be downloaded before they can be played. On the other hand, the benefit is that they can be downloaded individually on an as-needed basis without having to download all of the Character's data beforehand, unlike with ACS files which have to be loaded in full.

In addition to the Common Data Structures, the ACF format includes the following structures:

ACFHEADER

The ACF data format begins with a header structure which identifies the data as being the ACF format.

DATA TYPE	QUANTITY	DESCRIPTION
ULONG	1	Signature, Always 0xABCDABC4
COMPRESSED	1	Character Data

The Character Data is an ACFCHARACTERINFO structure.

ACFANIMATIONINFO (List Count: USHORT)

The ACFANIMATIONINFO structure specifies the name of the ACA file for a particular animation.

DATA TYPE	QUANTITY	DESCRIPTION
STRING	1	Animation Name
STRING	1	ACA Filename
STRING	1	Return Animation (in uppercase)
ULONG	1	ACA Checksum

The ACA filename has no folder path included in it. ACA files must be stored in the same folder as the ACF file.

The Checksum value matches the checksum in the ACAHEADER structure of the ACA file.

- At the time of this writing, the algorithm used for calculating the Checksum has not been determined yet.

ACFCHARACTERINFO

The ACFCHARACTERINFO structure describes the dimensions and characteristics of the Character.

DATA TYPE	QUANTITY	DESCRIPTION
USHORT	1	Minor Version
USHORT	1	Major Version
ACFANIMATIONINFO	LIST	Animations
GUID	1	Unique Identifier for the Character
LOCALIZEDINFO	LIST	Character Localized Info
USHORT	1	Character Width (in pixels)
USHORT	1	Character Height (in pixels)
BYTE	1	0-based Index of Transparent Color in Color Table
ULONG	1	Flags
USHORT	1	Animation Set Major Version?
USHORT	1	Animation Set Minor Version?
VOICEINFO	1	Voice Output Info (if enabled)
BALLOONINFO	1	Word Balloon Info (if enabled)
PALETTECOLOR	LIST	Color Table
BOOL	1	System Tray Icon is Enabled
TRAYICON	1	System Tray Icon (if enabled)
STATEINFO	LIST	Animation States

The Flags are comprised of the following bits:

BITS	DESCRIPTION
0-3	unknown
4	Voice Output 0 = disabled 1 = enabled
5-7	unknown
8, 9	Word Balloon disabled/enabled (respectively) (I don't know why Microsoft separates these bits)
10-15	unknown
16-18	Word Balloon Styles Can contain any combination of the following: 0x01 = size to text enabled 0x02 = auto hide disabled 0x04 = auto pace disabled
19	unknown
20	Standard Animation Set

```
        |    0 = not supported
        |    1 = supported
   21-31 | unknown
```

The Animation Set Major Version has been observed as always being 2, and the Minor Version as being 0.

ACA Format

The ACA data format contains information about a specific animation for a Character that is described in a separate ACF file.

In addition to the Common Data Structures, the ACA format includes the following structures:

ACAHEADER

The ACA data format begins with a header structure which identifies the data as being the ACA format.

DATA TYPE	QUANTITY	DESCRIPTION
USHORT	1	Minor Version
USHORT	1	Major Version
ULONG	1	Checksum
BOOL	1	Animation Data is Compressed
BYTE	variable	Animation Data

The Checksum value matches the checksum in the ACFANIMATIONINFO structure of the ACF file.

- At the time of this writing, the algorithm used for calculating the Checksum has not been determined yet.

The Animation Data is an ACAANIMATIONINFO structure. If it is compressed, it is inside of a COMPRESSED structure. Otherwise, it is just the ACAANIMATIONINFO structure by itself.

ACAIMAGEINFO (List Count: USHORT)

The ACAIMAGEINFO structure describes information about a particular frame image.

DATA TYPE	QUANTITY	DESCRIPTION
ULONG	1	Size of Image Data (in BYTEs)
BYTE	1	unknown
BYTE	variable	Image Data
DATABLOCK	1	Region Data

The Unknown value has been observed as always being 0.

The Image Data is the raw bitmap bits. Refer to the Win32 API documentation for the BITMAPINFOHEADER structure for details about how the bits are formatted. The Image Data has 1 Plane and a Bit Count of 8, and does not use any pixel compression.

The Region Data is a RGNDATA structure.

ACAAUDIOINFO (List Count: USHORT)

The ACAAUDIOINFO structure describes information about a particular block of waveform audio.

DATA TYPE	QUANTITY	DESCRIPTION
DATABLOCK	1	Audio Data

Refer to the RIFF Audio section for details about the format used for audio data.

ACAOVERLAYINFO (List Count: BYTE)

The ACAOVERLAYINFO structure describes information about a particular mouth overlay for an animation frame. Mouth overlays are displayed only during the Character's spoken TTS output.

DATA TYPE	QUANTITY	DESCRIPTION
BYTE	1	Overlay Type
BYTE	variable	Base Image Info (present only if Overlay Type is 0x00)
ULONG	1	Size of Image Data (in BYTEs)
BYTE	1	unknown
BOOL	1	Region Data is Present
SHORT	1	X-offset from top of Frame (in pixels)
SHORT	1	Y-offset from top of Frame (in pixels)
USHORT	1	Width (in pixels)
USHORT	1	Height (in pixels)
BYTE	variable	Image Data
DATABLOCK	1	Region Data (if present)

The Overlay Type can be one of the following:

```
0x00 = mouth closed
0x01 = mouth wide open 1
0x02 = mouth wide open 2
0x03 = mouth wide open 3
0x04 = mouth wide open 4
0x05 = mouth medium
0x06 = mouth narrow
```

If the Overlay Type is 0x00, the Base Image Info is present, as is defined as the following

DATA TYPE	QUANTITY	DESCRIPTION
BOOL	1	Replace the Top Image of the Frame
BYTE	variable	Base Image Data

If the "Replace Top Image" is 0x00, the Base Image Data is not present. Otherwise, it is defined as the following:

DATA TYPE	QUANTITY	DESCRIPTION
DATABLOCK	1	Base Image Data
DATABLOCK	1	Base Region Data

Unlike frames in ACS files, frames in ACA files do not contain a list of multiple images. If multiple images are specified when the frame is defined by the character author, they are composited into a single image when the ACA file is compiled. If the "Replace Top Image" flag is 0x00, the overlay image is placed on top of the frame image specified by the ACAFRAMEINFO structure. If the "Replace Top Image" flag is not 0x00, the Base Image for the overlay is the complete frame image without the top image present. The Overlay is placed on top of this Base Image rather than the frame image specified by the ACAFRAMEINFO structure.

The Unknown value has been observed as always being 0.

The Image Data is the raw bitmap bits. Refer to the Win32 API documentation for the BITMAPINFOHEADER structure for details about how the bits are formatted. The Image Data has 1 Plane and a Bit Count of 8, and does not use any pixel compression.

Note: The Width and Height are expressed as values that have been divided by 2. I do not know why Microsoft is doing that.

ACAFRAMEINFO (List Count: USHORT)

The ACAFRAMEINFO structure describes a particular frame in an animation sequence.

DATA TYPE	QUANTITY	DESCRIPTION
USHORT	1	0-based Index of Image in ACAIMAGEINFO List from the ACAANIMATIONINFO structure
USHORT	1	0-based Index of Audio in ACAAUDIOINFO List from the ACAANIMATIONINFO structure
USHORT	1	Frame Duration (in 1/100 seconds)
ULONG	1	unknown
SHORT	1	0-based Index of Frame to Exit to in Animation
BRANCHINFO	LIST	Frame Branches
ACAOVERLAYINFO	LIST	Mouth Overlays

The Unknown value has been observed as always being 0.

ACAANIMATIONINFO

The ACAANIMATIONINFO structure describes the characteristics of the Animation.

DATA TYPE	QUANTITY	DESCRIPTION
ACAAUDIOINFO	List	Frame Sounds
ACAIMAGEINFO	List	Frame Images
BYTE	1	Transition Type
ACAFRAMEINFO	LIST	Animation Frames

The Transition Type can be one of the following values:

```
0x00 = use return animation
0x01 = use exit branches
0x02 = no transition
```

Specification Change History

November 6, 2022 - Version 1.6 released publicly

- Updating Copyright.
- Updated links to MSDN documentation.
- Updated Data Types section to add BOOL.
- Fixed a typo in GUID definition.
- Updated Common Data Structures section to add COMPRESSED structure.
- Added some clarifications in the Compression Algorithm section.
- Miscellaneous formatting changes.

February 25, 2021 - Version 1.5 released publicly

- Updating Copyright.

December 14, 2020 - Version 1.4 released publicly

May 10, 2017 - Version 1.4 (not released yet)

- Added links to MSDN documentation for Win32 API structures.
- ACFCHARACTERINFO structure:
  Added fields for VOICEINFO, BALLOONINFO, and TRAYICON.
- ACAANIMATIONINFO structure:
  The Quantity of Animation Frames has been corrected to be LIST instead of 1.
- ACAOVERLAYINFO structure:
  Correcting the offset of the Overlay Type field.

June 10, 2009 - Version 1.3 released publicly

November 15, 2006 - Version 1.3 (not released yet)

- Specification re-written in HTML format.
- Updated Data Types section to include a note about the endian format used.
- ACS Format
  - Added additional structures to describe Image and Audio data.
  - Added new section to describe the Compression Algorithm.
  - Header structure:
    The unknown fields have been identified as ACSLOCATORs to ACSIMAGEINFO and ACSAUDIOINFO Lists.

- ExtraCharacterInfo structure:
  Renamed to LOCALIZEDINFO.
- ExitBranchInfo structure:
  Renamed to BRANCHINFO. This is no longer specific to an exit branch.
- OverlayInfo structure:
  The index field has been corrected to be an index into the ACSIMAGEINFO List from the ACSHEADER structure.
- FrameInfo structure:
  The unknown fields have been identified as an ACSFRAMEIMAGE List, and an index into the ACSAUDIOINFO List from the ACSHEADER structure.

- ACF and ACA Formats
  - Added to the specification.

June 18, 2004 - Version 1.2 (not released yet)

- Added quantity column to structure definitions.
- AnimationInfo structure:
  The unknown fields have been identified as an ACSLOCATOR to extended animation info.

March 14, 2004 - Version 1.1 released publicly

- VoiceInfo structure:
  The unknown field has been identified as a Flag indicating whether the remaining Voice data is present.
- CharacterInfo structure:
  The unknown fields surrounding the Color Table flag have been identified as being the actual number of colors in the color table.
- Footer structure:
  Renamed structure to ExtraCharacterInfo and added ExtraCharacterInfoList structure.

March 31, 2003 - Version 1.0 released publicly