To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Introduction

Macromedia Flash (SWF) file format specification

The Macromedia Flash file format (SWF) (pronounced “swiff”) delivers vector graphics, text, video and sound over the Internet to Macromedia Flash Player. The SWF file format is designed to be an efficient delivery format, not a format for exchanging graphics between graphics editors. It is designed to meet the following goals:

On-screen display

The format is primarily intended for on-screen display and supports anti-aliasing, fast rendering to a bitmap of any color format, animation, and interactive buttons.

Extensibility

The format is a tagged format, so it can be evolved with new features while maintaining backward compatibility with earlier versions of Flash Player.

Network delivery

The format can travel over a network with limited and unpredictable bandwidth. The files are compressed to be small and support incremental rendering through streaming. The SWF file format is a binary format and is not human readable like HTML. The SWF file format uses techniques such as bit-packing and structures with optional fields to minimize file size.

Simplicity

The format is simple so that Flash Player is small and easily ported. Also, Flash Player depends upon a limited set of operating system features only.

File independence

The files display without dependence on external resources such as fonts.

Scalability

The files work well on limited hardware, and can take advantage of better hardware when it is available. This ability is important because computers have different monitor resolutions and bit depths.

Speed

The files render with quick, high quality.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Scriptability
The format includes tags that provide sequences of byte codes to be interpreted by a stack machine. The byte codes support the ActionScript language. Flash Player provides a runtime ActionScript object model that allows interaction with drawing primitives, servers, and features of Flash Player.

SWF files have the extension .swf and a MIME type of application/x-shockwave-flash. The SWF format has evolved through several versions. Through SWF 5, substantial additions were made to the SWF tag set. Starting with SWF 6 and later, the SWF format changes less, as more new Flash features are implemented partly or entirely at the ActionScript level. For this reason, anyone planning to generate SWF file content that uses newer features should become familiar with the ActionScript object model that Flash Player exposes. The best references for this information are Learning ActionScript 2.0 in Flash (see http://livedocs.macromedia.com/flash/8/main/00001303.html) and Essential ActionScript 2.0 by Colin Moock.

Macromedia seriously considers all feedback to the SWF format specification. In addition, any known errors from previously released specifications are corrected in this document. If you encounter unclear or potentially erroneous information within the specification, please e-mail Macromedia at Flash_Format@macromedia.com, and Macromedia will investigate an improvement to the document.

The SWF header
All SWF files begin with the following header. The types are defined in Basic Data Types:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature</td>
<td>UI8</td>
<td>Signature byte:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“F” indicates uncompressed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“C” indicates compressed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(SWF 6 and later only)</td>
</tr>
<tr>
<td>Signature</td>
<td>UI8</td>
<td>Signature byte always “W”</td>
</tr>
<tr>
<td>Version</td>
<td>UI8</td>
<td>Single byte file version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(for example, 0x06 for SWF 6)</td>
</tr>
<tr>
<td>FileLength</td>
<td>UI32</td>
<td>Length of entire file in bytes</td>
</tr>
<tr>
<td>FrameSize</td>
<td>RECT</td>
<td>Frame size in twips</td>
</tr>
<tr>
<td>FrameRate</td>
<td>UI16</td>
<td>Frame delay in 8.8 fixed number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of frames per second</td>
</tr>
<tr>
<td>FrameCount</td>
<td>UI16</td>
<td>Total number of frames in file</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
The header begins with a three-byte signature of either 0x46, 0x57, 0x53 ("FWS"); or 0x43, 0x57, 0x53 ("CWS"). An FWS signature indicates an uncompressed SWF file; CWS indicates that the entire file after the first 8 bytes (that is, after the FileLength field) was compressed by using the ZLIB open standard. The data format that the ZLIB library uses is described by Request for Comments (RFCs) documents 1950 to 1952. CWS file compression is permitted in SWF 6 or later only.

A one-byte version number follows the signature. The version number is not an ASCII character, but an 8-bit number. For example, for SWF 4, the version byte is 0x04, not the ASCII character "4" (0x35).

The FileLength field is the total length of the SWF file, including the header. If this is an uncompressed SWF file (FWS signature), the FileLength field should exactly match the file size. If this is a compressed SWF file (CWS signature), the FileLength field indicates the total length of the file after decompression, and thus generally does not match the file size. Having the uncompressed size available can make the decompression process more efficient.

The FrameSize field defines the width and height of the on-screen display. This field is stored as a RECT structure, meaning that its size may vary according to the number of bits needed to encode the coordinates. The FrameSize RECT always has Xmin and Ymin of 0; the Xmax and Ymax members define the width and height (see Using bit values).

The FrameRate is the desired playback rate in frames per second. This rate is not guaranteed if, for example, Flash Player is running on a slow or busy CPU.

The FrameCount is the total number of frames in this SWF file.

SWF file structure
Following the header is a series of tagged data blocks. All tags share a common format, so any program parsing a SWF file can skip over blocks it does not understand. Data inside the block can point to offsets within the block, but can never point to an offset in another block. This ability enables tags to be removed, inserted, or modified by tools that process a SWF file.

The FileAttributes tag is only required for SWF 8 and later.
Introduction

Tag format
Each tag begins with a tag type and a length. The tag-header formats can be either short or long. Short tag headers are used for tags with 62 bytes of data or less. Long tag headers can be used for any tag size up to 4 GB, far larger than is presently practical.

The length specified in the TagCodeAndLength field does not include the RECORDHEADER that starts a tag.

If the tag is 63 bytes or longer, it is stored in a long tag header. The long tag header consists of a short tag header with a length of 0x3f, followed by a 32-bit length.

Definition and control tags
The two categories of tags in a SWF file are as follows:

Definition tags
define the content of the SWF file—the shapes, text, bitmaps, sounds, and so on. Each definition tag assigns a unique ID called a character ID to the content it defines. Flash Player then stores the character in a repository called the dictionary. Definition tags, by themselves, do not cause anything to be rendered.

Control tags
create and manipulate rendered instances of characters in the dictionary, and control the flow of the file.

---

RECORDHEADER (short)

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TagCodeAndLength</td>
<td>0x0</td>
</tr>
</tbody>
</table>

RECORDHEADER (long)

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TagCodeAndLength</td>
<td>0x0</td>
</tr>
<tr>
<td>Length</td>
<td>0x0</td>
</tr>
</tbody>
</table>

---

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Generally speaking, tags in a SWF can occur in any order. However, you must observe the following rules:

■ The FileAttributes tag must be the first tag in the SWF file for SWF 8 and later.
■ A tag should only depend on tags that come before it. A tag should never depend on a tag that comes later in the file.
■ A definition tag that defines a character must occur before any control tag that refers to that character.
■ Streaming sound tags must be in order. Out-of-order streaming sound tags result in the sound being played out of order.
■ The End tag is always the last tag in the SWF file.

The dictionary

The dictionary is a repository of characters that are defined, and are available for control tags to use. The process of building and using the dictionary is as follows:

1. The definition tag defines some content, such as a shape, font, bitmap, or sound.
2. The definition tag assigns a unique CharacterId to the content.
3. The content is saved in the dictionary under the CharacterId.
4. A control tag uses the CharacterId to retrieve the content from the dictionary, and performs some action on the content, such as displaying a shape, or playing a sound.

Every definition tag must specify a unique ID. Duplicate IDs are not allowed. Typically, the first CharacterId is 1, the second CharacterId is 2, and so on. The number zero (0) is special and is considered a null character.

Control tags are not the only tags that reference the dictionary. Definition tags can use characters from the dictionary to define more complex characters. For example, the DefineButton and DefineSprite tags refer to other characters to define their contents. The DefineText tag can refer to font characters to select different fonts for the text.
Introduction

The following diagram illustrates a typical interaction between definition tags, control tags, and the dictionary:

- See The Display List.

Processing a SWF file

Flash Player processes all of the tags in a SWF file until a ShowFrame tag is encountered. At this point, the display list is copied to the screen and Flash Player is idle until it is time to process the next frame. The contents of the first frame are the cumulative effect of performing all of the control tag operations before the first ShowFrame tag. The contents of the second frame are the cumulative effect of performing all of the control tag operations from the beginning of the file to the second ShowFrame tag, and so on.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Since SWF files are frequently delivered over a network connection, they should be as compact as possible. Several techniques are used to accomplish this, including the following items:

- **Reuse**
  The structure of the character dictionary makes it easy to reuse elements in a SWF file. For example, a shape, button, sound, font, or bitmap can be stored in a file once and referenced many times.

- **Compression**
  Shapes are compressed by using an efficient delta encoding scheme; often the first coordinate of a line is assumed to be the last coordinate of the previous line. Distances are also often expressed relative to the last position.

- **Default values**
  Some structures, like matrixes and color transforms, have common fields that are used more often than others. For example, for a matrix, the most common field is the translation field. Scaling and rotation are less common. Therefore, if the scaling field is not present, it is assumed to be 100%. If the rotation field is not present, it is assumed that there is no rotation. This use of default values helps to minimize file sizes.

- **Change Encoding**
  As a rule, SWF files only store the changes between states. This is reflected in shape data structures and in the place-move-remove model that the display list uses.

**Shape Data Structure**
The shape data structure uses a unique structure to minimize the size of shapes and to render anti-aliased shapes efficiently on the screen.

**Summary**
A SWF file is made up of a header, followed by a number of tags. The two types of tags are definition tags and control tags. Definition tags define the objects known as characters, which are stored in the dictionary. Control tags manipulate characters, and control the flow of the file.
CHAPTER 1

Macromedia Flash (SWF) File Format 8

This chapter describes the new features and tags in the Macromedia Flash SWF 8 file format specification.

Bitmap filters

SWF 8 introduces expressive graphic effects at runtime without significantly increasing the SWF file size. New bitmap filters including blur, drop shadow, glow, bevel, gradient glow, gradient bevel, convolution, color matrix, and others can be applied to a display object with the use of the PlaceObject3 tag along with the FILTERLIST and FILTER records.

Blend modes

Compositing effects are now possible in SWF 8 or later by using blend modes to change the way the image of a display object or button combines with the images of objects at lower depths. With the PlaceObject3 tag and Button record, blend modes can now be selected as an alternative to normal alpha compositing.

Bitmap caching

Playback performance can be optimized with bitmap caching. By specifying that a static movie clip or button symbol be cached as a bitmap at runtime with the PlaceObject3 tag, Flash Player internally caches the display object instead of continually redrawing the image, which can provide a significant improvement in playback performance in certain scenarios.
Enhanced strokes

Tags DefineShape4 and DefineMorphShape2, along with the LINESTYLE2 and MORPHLINESTYLE2 records, build upon previous SWF file capabilities by allowing new types of joins and caps as well as the ability to fill a stroke. While earlier line-style records permitted only rounded joins and round caps, LINESTYLE2 and MORPHLINESTYLE2 also support miter and bevel joins, and square caps and no caps. The new line styles also include the option for pixel hinting to correct blurry vertical or horizontal lines.

Enhanced gradients

SWF 8 presents increased control over gradients. FILLSTYLE and the new FOCALGRADIENT record add a new radial gradient type that permits the focal point to be set. In addition, the GRADIENT record can now have up to fifteen control points, up from eight in SWF 7 and earlier.

Advanced text rendering with FlashType

With the new DefineFont3, DefineFontAlignZones, and CSMT extSettings tags, FlashType brings a new anti-aliasing technology to Flash Player. FlashType enables high-quality font rendering, especially at small point sizes, while maintaining the aesthetic look and feel of the fonts.

9-slice scaling

The DefineScalingGrid tag introduces the concept of 9-slice scaling, which allows component-style scaling to be applied to a sprite or button character, better maintaining visual integrity.

On2 Truemotion VP6

For SWF 6 or later, Flash Player 8 supports the On2 Truemotion VP6 bitstream format, a leading-edge video compression algorithm. Additionally, Flash Player 8 adds support for an extra alpha channel that is used to simulate transparency in video.
Screen Video V2 bitstream codec

SWF 6 supports Screen Video V2 bitstream format, which is an extension of the Screen video bitstream format and is supported in Flash Player 8 and later. Screen Video V2 uses several techniques to reduce the amount of data required to represent the on-screen content.

SWF file attributes and metadata

Starting with SWF 8, SWF files must always have the new FileAttributes tag as the first tag. The FileAttributes tag identifies characteristics of the SWF file itself.

In SWF 8, the FileAttributes tag serves two purposes. The tag identifies whether the SWF file contains the Metadata tag, which uses properties such as title and description to describe the SWF file to an external process, such as a search engine. In addition, for local playback of a SWF file, FileAttributes specifies whether Flash Player should grant the SWF file local or network file access.

FLV data tag format

The format of FLV Data Tags is specified in the FLV File Format chapter, along with descriptions of Audio Tags and Video Tags.

ImportAssets2

The ImportAssets2 tag replaces the ImportAssets tag for SWF 8 and later. ImportAssets2 mirrors the ImportAssets functionality.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
CHAPTER 2

Macromedia Flash (SWF) File Format 7

This chapter describes the new features in the Macromedia SWF 7 file format specification.

ActionScript extensions

The new ActionScript bytecode ActionDefineFunction2 expands upon the existing ActionDefineFunction, now allowing a function to store parameters and local variables in registers. (The ActionDefineFunction bytecode is rarely used in SWF 7 and later and was superseded by ActionDefineFunction2 in SWF8.) ActionDefineFunction2 also provides control over the creation and storage of the this, arguments, super, _root, _parent and _global common variables.

To support these improvements, ActionStoreRegister can now access up to 256 registers with the use of bytecode ActionDefineFunction2.

To improve the compliance of ActionScript with the ECMA-262 standard, and to provide greater support of object-oriented programming, SWF 7 introduces the ActionExtends, ActionCastOp, and ActionImplementsOp bytecodes, the only file format changes made to support ActionScript 2.0. ActionExtends offers the ability to create a relationship between two classes, called the subclass and the superclass. With ActionCastOp, Flash Player 7 or later converts an object of one type to another type. ActionImplementsOp specifies the interfaces that a class or interface can implement.

With the SWF 7 format, ActionInstanceOf now reports whether an object implements an interface.

ActionScript now employs exception handling with the ActionTry and ActionThrow bytecodes. ActionTry declares handlers for exceptional conditions, and ActionThrow pops an error value to be thrown.
New video format
Flash Player 7 supports a simple new lossless video codec called Screen video bitstream format, optimized for captures of computer screens in motion. ScreenVideo, like Sorenson H.263 Bitstream Format, can appear in both SWF files and FLV files.

Flash Player 7 can play back FLV files directly, without the use of the RTMP protocol or Flash Communication Server MX.

Runtime ActionScript controls
The new ScriptLimits tag provides control over the maximum recursion depth and the number of seconds before possible script time out.

SetTabIndex
The new SetTabIndex tag sets the index of an object within the Flash Player tab order.

ClipEventConstruct
The ClipEventFlags sequence now includes the ClipEventConstruct tag to signal the construct event, in addition to the already existing ClipEventInitialize.

Small text rendering
Previously, in certain cases, small anti-aliased text would appear blurry in Flash Player. With the new flag FontFlagsSmallText in the DefineFontInfo, DefineFontInfo2, and DefineFont2 tags, Flash Player 7 and later aligns character glyphs on pixel boundaries for dynamic and input text.
CHAPTER 3

Macromedia Flash (SWF) File Format 6

This chapter describes the new features in the Macromedia SWF 6 file format specification.

File compression

SWF 6 or later files can be compressed to reduce their size. A different file-header signature (CWS instead of FWS) signals this choice. The compression used is the popular ZLIB standard.

Unicode support

SWF 6 or later files support Unicode text.

SWF 6 adds the DefineFontInfo2 tag. This is a minor extension to the DefineFontInfo tag; DefineFontInfo2 adds a language code field. Similarly, the DefineFont2 tag uses a previously reserved byte to store a language code. Language codes are used for line-breaking considerations, and for choosing fallback fonts when a specified device font is unavailable.

The DefineFontInfo, DefineFont2, and DefineFontInfo2 tags have different usage rules in SWF 6 or later files. The ANSI and shift-JIS encoding options for character tables are deprecated, and all character tables in these tags are encoded by using UCS-2.

Device font names in SWF 6 or later files are specified by using UTF-8 encoding rather than the locale-specific encodings previously used.

The common STRING type in SWF 6 or later files uses UTF-8 encoding rather than the ANSI or shift-JIS encodings previously used.
Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Improved documentation

The existing DefineSound and SoundStreamBlock tags support a new codec option in SWF 6 or later files: NellyMoser Asao, optimized for low bitrates (see Nellymoser compression).

Two tags, DefineVideoStream and VideoFrame, allow video to be embedded in SWF 6 or later files. For SWF 6, a single video codec, Sorenson Spark, is available (see Sorenson H.263 Bitstream Format).

The FLV file format SWF content can perform dynamic two-way audio, video, and data interaction with Flash Communication Server MX. In one form of this interaction, Flash Communication Server can serve prerecorded or streaming files of the FLV format, which encodes synchronized audio, video, and data. The audio and video formats used in FLV are the same as those used in SWF. The FLV format, like the SWF format, is an open standard documented by Macromedia.

Improved documentation

With the release of the SWF 6 specification, the documentation of the following SWF file format chapters was extensively revised to improve clarity and detail:

- Sounds
- Fonts and Text
- Bitmaps
- Clip actions in PlaceObject2
- Button actions in DefineButton2

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
CHAPTER 4
Basic Data Types

This section describes the basic data types that make up the more complex data structures in the Macromedia Flash (SWF) file format. All other structures in the SWF file format are built on these fundamental types.

Coordinates and twips

The SWF file format stores all x-y coordinates as integers, usually in a unit of measurement called a twip. In the SWF format, a twip is 1/20th of a logical pixel. A logical pixel is the same as a screen pixel when the file is played at 100%—that is, without scaling.

For example, a rectangle 800 twips wide by 400 twips high is rendered as 40 by 20 logical pixels. Fractional pixel sizes are approximated with anti-aliasing. A rectangle 790 by 390 twips (39.5 by 19.5 pixels) appears to have slightly blurred edges.

Twips are a good compromise between size and precision. They provide sub-pixel accuracy for zooming and precise placement of objects, while consuming very few bits per coordinate.

Coordinates in the SWF file format use the traditional graphics axes: x proceeds from minimum values at the left to maximum values at the right, and y is vertical and proceeds from minimum values at the top to maximum values at the bottom.

Integer types and byte order

The SWF file format uses 8-bit, 16-bit, 32-bit, 64-bit, signed, and unsigned integer types. All integer values are stored in the SWF file by using little-endian byte order: the least significant byte is stored first, and the most significant byte is stored last, in the same way as the Intel x86 architecture. The bit order within bytes in the SWF file format is big-endian: the most significant bit is stored first, and the least significant bit is stored last.

For example:

The 32-bit value 0x456e7120 is stored as 20 71 6e 45.

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.
The 16-bit value \(0xe712\) is stored as \(12 e7\).

All integer types must be byte aligned. That is, the first bit of an integer value must be stored in the first bit of a byte in the SWF file.

Signed integers are represented by using traditional 2's-complement bit patterns. These are the signed integer representations used on most modern computer platforms. In the 2's complement system, negative numbers have 1 as the first bit, and zero and positive numbers have 0 as the first bit. A negative number, \(-n\), is represented as the bitwise opposite of the positive-zero number \(n\) - 1.

Fixed-point numbers

The SWF file format supports two types of fixed-point numbers: 32 bit and 16 bit. The 32-bit fixed-point numbers are 16.16. That is, the high 16 bits represent the number before the decimal point, and the low 16 bits represent the number after the decimal point.

**FIXED values** are stored like 32-bit integers in the SWF file (using little-endian byte order) and must be byte aligned.

For example:

The real value 7.5 is equivalent to: \(0x0007.8000\)

<table>
<thead>
<tr>
<th>Integer Types</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SI8</td>
<td>Signed 8-bit integer value</td>
</tr>
<tr>
<td></td>
<td>SI16</td>
<td>Signed 16-bit integer value</td>
</tr>
<tr>
<td></td>
<td>SI32</td>
<td>Signed 32-bit integer value</td>
</tr>
<tr>
<td></td>
<td>SI8[n]</td>
<td>Signed 8-bit array— (n) is the number of array elements</td>
</tr>
<tr>
<td></td>
<td>SI16[n]</td>
<td>Signed 16-bit array— (n) is the number of array elements</td>
</tr>
<tr>
<td></td>
<td>UI8</td>
<td>Unsigned 8-bit integer value</td>
</tr>
<tr>
<td></td>
<td>UI16</td>
<td>Unsigned 16-bit integer value</td>
</tr>
<tr>
<td></td>
<td>UI32</td>
<td>Unsigned 32-bit integer value</td>
</tr>
<tr>
<td></td>
<td>UI8[n]</td>
<td>Unsigned 8-bit array— (n) is the number of array elements</td>
</tr>
<tr>
<td></td>
<td>UI16[n]</td>
<td>Unsigned 16-bit array— (n) is the number of array elements</td>
</tr>
<tr>
<td></td>
<td>UI24[n]</td>
<td>Unsigned 24-bit array— (n) is the number of array elements</td>
</tr>
<tr>
<td></td>
<td>UI32[n]</td>
<td>Unsigned 32-bit array— (n) is the number of array elements</td>
</tr>
<tr>
<td></td>
<td>UI64[n]</td>
<td>Unsigned 64-bit array— (n) is the number of array elements</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
This value is stored in the SWF file as: 00 80 07 00

SWF 8 and later supports 16-bit 8.8 signed, fixed-point numbers. The high 8 bits represent the number before the decimal point, and the low 8 bits represent the number after the decimal point. FIXED8 values are stored like 16-bit integers in the SWF file (using little-endian byte order) and must be byte aligned.

Floating-point numbers

SWF 8 and later supports the use of IEEE Standard 754 compatible floating-point types. Three types of floating-point numbers are supported.

FLOAT16 is identical to the characteristics of FLOAT except for changes to the number of bits allocated to the exponent and mantissa:

- 1 bit for the sign
- 5 bits for the exponent, with an exponent bias of 16
- 10 bits for the mantissa

Fixed-Point Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIXED</td>
<td>32-bit 16.16 fixed-point number</td>
</tr>
<tr>
<td>FIXED8</td>
<td>16-bit 8.8 fixed-point number</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Floating-Point Types</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOAT16</td>
<td>FLOAT16</td>
<td>Half-precision (16-bit) floating-point number</td>
</tr>
<tr>
<td>FLOAT</td>
<td>FLOAT</td>
<td>Single-precision (32-bit) IEEE Standard 754 compatible</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>Double-precision (64-bit) IEEE Standard 754 compatible</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Bit values do not have to be byte aligned. Other types (such as UI8 and UI16) are always byte aligned. If a byte-aligned type follows a bit value, the last byte that contains the bit value is padded with zeros.

The following example is a stream of 64 bits, made up of 9-bit values of varying length, followed by a UI16 value:

The bit stream begins with a 6-bit value (BV1), followed by a 5-bit value (BV2) that is spread across Byte1 and Byte2. BV3 is spread across Byte2 and Byte3, while BV4 is wholly contained within Byte3. Byte 5 contains two bit values: BV7 and BV8. BV9 is followed by a byte-aligned type (UI16), so the last four bits of Byte 6 are padded with zeros.

When an unsigned-bit value is expanded into a larger word size, the leftmost bits are filled with zeros. When a signed-bit value is expanded into a larger word size, the high bit is copied to the leftmost bits. This expansion is called sign extension.

For example, the 4-bit unsigned value \( \text{UB}[4] = 1110 \) would be expanded to a 16-bit value like this:

\[
0000000000001110 = 14
\]

The same value interpreted as a signed value, \( \text{SB}[4] = 1110 \), would be expanded to

\[
1111111111111110 = -2
\]

Signed-bit values are similar but must take account of the sign bit. The signed value of 35 is represented as \( \text{SB}[7] = 0100011 \).

Fixed-point bit values are 32-bit 16.16 signed, fixed-point numbers. That is, the high 16 bits represent the number before the decimal point, and the low 16 bits represent the number after the decimal point. A fixed-point bit value is identical to a signed-bit value, but the interpretation is different. For example, a 19-bit, signed-bit value of \( 0x30000 \) is interpreted as 196608 decimal. The 19-bit, fixed-point bit value \( 0x30000 \) is interpreted as 3.0. The format of this value is effectively 3.16 rather than 16.16.
Using bit values

Bit values are stored by using the minimum number of bits possible for the range needed. Most bit value fields use a fixed number of bits. Some use a variable number of bits, but in all such cases, the number of bits to be used is explicitly stated in another field in the same structure. In these variable-length cases, SWF-file generating applications must determine the minimum number of bits necessary to represent the actual values that will be specified. For signed-bit values, if the number to be encoded is positive, an extra bit is necessary to preserve the leading 0; otherwise sign extension changes the bit value into a negative number.

As an example of variable-sized bit values, consider the RECT structure:

The Nbits field determines the number of bits used to store the coordinate values Xmin, Xmax, Ymin, and Ymax. Say the coordinates of the rectangle are as follows:

Xmin = 127 decimal = 1111111 binary
Xmax = 260 decimal = 100000100 binary
Ymin = 15 decimal = 1111 binary
Ymax = 514 decimal = 1000000010 binary

Nbits is calculated by finding the coordinate that requires the most bits to represent. In this case, that value is 514 (01000000010 binary) which requires 11 bits to represent. The rectangle is stored as the following table shows:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nbits</td>
<td>UB[5]</td>
<td>Bits in each rect value field</td>
</tr>
<tr>
<td>Xmin</td>
<td>SB[Nbits]</td>
<td>x minimum position for rect</td>
</tr>
<tr>
<td>Xmax</td>
<td>SB[Nbits]</td>
<td>x maximum position for rect</td>
</tr>
<tr>
<td>Ymin</td>
<td>SB[Nbits]</td>
<td>y minimum position for rect</td>
</tr>
<tr>
<td>Ymax</td>
<td>SB[Nbits]</td>
<td>y maximum position for rect</td>
</tr>
</tbody>
</table>

RECT

Field Type and Value Comment

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nbits</td>
<td>UB[5]</td>
<td>1011</td>
</tr>
<tr>
<td>Xmin</td>
<td>SB[11]</td>
<td>00001111111</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
String values

A string value represents a null terminated character string. The format for a string value is a sequential list of bytes terminated by the null character byte.

In SWF 5 or earlier, STRING values are encoded using either ANSI (which is a superset of ASCII) or shift-JIS (a Japanese encoding). You cannot indicate the encoding that is used; instead, the decoding choice is made according to the locale in which Flash Player is running. This means that text content in SWF 5 or earlier can only be encoded in ANSI or shift-JIS, and the target audience must be known during authoring—otherwise garbled text results.

In SWF 6 or later, STRING values are always encoded by using the Unicode UTF-8 standard. This is a multibyte encoding; each character is composed of between one and four bytes.

UTF-8 is a superset of ASCII; the byte range 0 to 127 in UTF-8 exactly matches the ASCII mapping, and all ASCII characters 0 to 127 are represented by just one byte. UTF-8 guarantees that whenever a character other than character 0 (the null character) is encoded by using more than one byte, none of those bytes are zero. This avoids the appearance of internal null characters in UTF-8 strings, meaning that it remains safe to treat null bytes as string terminators, just as for ASCII strings.

Language code

A language code identifies a spoken language that applies to text. Language codes are associated with font specifications in the SWF file format.

<table>
<thead>
<tr>
<th>STRING Field Type Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
</tr>
<tr>
<td>Non-null string character data</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>StringEnd</td>
</tr>
<tr>
<td>Marks end of string; always zero</td>
</tr>
</tbody>
</table>

NOTE

A language code does not specify a text encoding; it specifies a spoken language.

<table>
<thead>
<tr>
<th>LANGCODE Field Type Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LanguageCode</td>
</tr>
<tr>
<td>Language code (see following)</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
Flash Player uses language codes to determine line-breaking rules for dynamic text, and to choose backup fonts when a specified device font is unavailable. Other uses for language codes may be found in the future.

A language code of zero means no language. This code results in behavior that is dependent on the locale in which Flash Player is running.

At the time of writing, the following language codes are recognized by Flash Player:

- Latin (the western languages covered by Latin-1: English, French, German, and so on)
- Japanese
- Korean
- Simplified Chinese
- Traditional Chinese

RGB color record

The RGB record represents a color as a 24-bit red, green, and blue value.

RGBA color with alpha record

The RGBA record represents a color as 32-bit red, green, blue, and alpha value. An RGBA color with an alpha value of 255 is completely opaque. An RGBA color with an alpha value of zero is completely transparent. Alpha values between zero and 255 are partially transparent.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>UI8</td>
<td>Red color value</td>
</tr>
<tr>
<td>Green</td>
<td>UI8</td>
<td>Green color value</td>
</tr>
<tr>
<td>Blue</td>
<td>UI8</td>
<td>Blue color value</td>
</tr>
<tr>
<td>Alpha</td>
<td>UI8</td>
<td>Transparency color value</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The ARGB record behaves exactly like the RGBA record, but the alpha value for the ARGB record is in the first byte.

A rectangle value represents a rectangular region defined by a minimum x- and y-coordinate position and a maximum x- and y-coordinate position. The RECT record must be byte aligned.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha</td>
<td>UI8</td>
<td>Transparency color value</td>
</tr>
<tr>
<td>Red</td>
<td>UI8</td>
<td>Red color value</td>
</tr>
<tr>
<td>Green</td>
<td>UI8</td>
<td>Green color value</td>
</tr>
<tr>
<td>Blue</td>
<td>UI8</td>
<td>Blue color value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nbits</td>
<td>UB[5]</td>
<td>Bits used for each subsequent field</td>
</tr>
<tr>
<td>Xmin</td>
<td>SB[Nbits]</td>
<td>x minimum position for rectangle in twips</td>
</tr>
<tr>
<td>Xmax</td>
<td>SB[Nbits]</td>
<td>x maximum position for rectangle in twips</td>
</tr>
<tr>
<td>Ymin</td>
<td>SB[Nbits]</td>
<td>y minimum position for rectangle in twips</td>
</tr>
<tr>
<td>Ymax</td>
<td>SB[Nbits]</td>
<td>y maximum position for rectangle in twips</td>
</tr>
</tbody>
</table>
The MATRIX record represents a standard 2x3 transformation matrix of the sort commonly used in 2D graphics. It is used to describe the scale, rotation, and translation of a graphic object. The MATRIX record must be byte aligned.

The ScaleX, ScaleY, RotateSkew0 and RotateSkew1 fields are stored as 16.16 fixed-point values. The TranslateX and TranslateY values are stored as signed values in twips.

The MATRIX record is optimized for common cases such as a matrix that performs a translation only. In this case, the HasScale and HasRotate flags are zero, and the matrix only contains the TranslateX and TranslateY fields.

The mapping from the MATRIX fields to the 2x3 matrix is as follows:

For any coordinates \((x, y)\), the transformed coordinates \((x', y')\) are calculated as follows:

\[
x' = x \times \text{ScaleX} + y \times \text{RotateSkew1} + \text{TranslateX}
\]
\[
y' = x \times \text{RotateSkew0} + y \times \text{ScaleY} + \text{TranslateY}
\]

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasScale</td>
<td>UB[1]</td>
<td>Has scale values if equal to 1</td>
</tr>
<tr>
<td>NScaleBits</td>
<td>UB[5]</td>
<td>Bits in each scale value field</td>
</tr>
<tr>
<td>ScaleX</td>
<td>FB[NScaleBits]</td>
<td>x scale value</td>
</tr>
<tr>
<td>ScaleY</td>
<td>FB[NScaleBits]</td>
<td>y scale value</td>
</tr>
<tr>
<td>HasRotate</td>
<td>UB[1]</td>
<td>Has rotate and skew values if equal to 1</td>
</tr>
<tr>
<td>NRotateBits</td>
<td>UB[5]</td>
<td>Bits in each rotate value field</td>
</tr>
<tr>
<td>RotateSkew0</td>
<td>FB[NRotateBits]</td>
<td>First rotate and skew value</td>
</tr>
<tr>
<td>RotateSkew1</td>
<td>FB[NRotateBits]</td>
<td>Second rotate and skew value</td>
</tr>
<tr>
<td>NTranslateBits</td>
<td>UB[5]</td>
<td>Bits in each translate value field</td>
</tr>
<tr>
<td>TranslateX</td>
<td>SB[NTranslateBits]</td>
<td>x translate value in twips</td>
</tr>
<tr>
<td>TranslateY</td>
<td>SB[NTranslateBits]</td>
<td>y translate value in twips</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader®or Adobe®Acrobat® version 6.0 or above.
42 Basic Data Types

The following table describes how the members of the matrix are used for each type of operation:

**Color transform record**

The CXFORM record defines a simple transform that can be applied to the color space of a graphic object. The following are the two types of transform possible:

- **Multiplication transforms**
- **Addition transforms**

**Multiplication transforms** multiply the red, green, and blue components by an 8.8 fixed-point multiplication term. The fixed-point representation of 1.0 is \(0x100\) or \(256\) decimal.

For any color \((R,G,B)\), the transformed color \((R', G', B')\) is calculated as follows:

\[
R' = \frac{R \times \text{RedMultTerm}}{256} \\
G' = \frac{G \times \text{GreenMultTerm}}{256} \\
B' = \frac{B \times \text{BlueMultTerm}}{256}
\]

**Addition transforms** add an addition term (positive or negative) to the red, green, and blue components of the object being displayed. If the result is greater than 255, the result is clamped to 255. If the result is less than zero, the result is clamped to zero.

For any color \((R,G,B)\), the transformed color \((R', G', B')\) is calculated as follows:

\[
R' = \max(0, \min(R + \text{RedAddTerm}, 255)) \\
G' = \max(0, \min(G + \text{GreenAddTerm}, 255)) \\
B' = \max(0, \min(B + \text{BlueAddTerm}, 255))
\]

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Addition and multiplication transforms can be combined as follows. The multiplication operation is performed first:

\[
R' = \max(0, \min((R \times \text{RedMultTerm}) / 256 + \text{RedAddTerm}, 255))
\]

\[
G' = \max(0, \min((G \times \text{GreenMultTerm}) / 256 + \text{GreenAddTerm}, 255))
\]

\[
B' = \max(0, \min((B \times \text{BlueMultTerm}) / 256 + \text{BlueAddTerm}, 255))
\]

The CXFORM record must be byte aligned.

Color transform with alpha record

The CXFORMWITHALPHA record extends the functionality of CXFORM by allowing color transforms to be applied to the alpha channel, as well as the red, green, and blue channels.

The following are the two types of transform possible:

- Multiplication Transforms
- Addition Transforms

Multiplication transforms multiply the red, green, blue, and alpha components by an 8.8 fixed-point value. The fixed-point representation of 1.0 is 0x100 or 256 decimal. Therefore, the result of a multiplication operation should be divided by 256.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasAddTerms</td>
<td>UB[1]</td>
</tr>
<tr>
<td>HasMultTerms</td>
<td>UB[1]</td>
</tr>
<tr>
<td>Nbits</td>
<td>UB[4]</td>
</tr>
<tr>
<td>RedMultTerm</td>
<td>If HasMultTerms = 1, SB[Nbits]</td>
</tr>
<tr>
<td>GreenMultTerm</td>
<td>If HasMultTerms = 1, SB[Nbits]</td>
</tr>
<tr>
<td>BlueMultTerm</td>
<td>If HasMultTerms = 1, SB[Nbits]</td>
</tr>
<tr>
<td>RedAddTerm</td>
<td>If HasAddTerms = 1, SB[Nbits]</td>
</tr>
<tr>
<td>GreenAddTerm</td>
<td>If HasAddTerms = 1, SB[Nbits]</td>
</tr>
<tr>
<td>BlueAddTerm</td>
<td>If HasAddTerms = 1, SB[Nbits]</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
For any color (R,G,B,A), the transformed color (R’, G’, B’, A’) is calculated as follows:

\[ R' = \frac{R \times \text{RedMultTerm}}{256} \]
\[ G' = \frac{G \times \text{GreenMultTerm}}{256} \]
\[ B' = \frac{B \times \text{BlueMultTerm}}{256} \]
\[ A' = \frac{A \times \text{AlphaMultTerm}}{256} \]

The CXFORMWITHALPHA record is most commonly used to display objects as partially transparent, achieved by multiplying the alpha channel by some value between zero and 256.

Addition transforms add a fixed value (positive or negative) to the red, green, blue, and alpha components of the object being displayed. If the result is greater than 255, the result is clamped to 255. If the result is less than zero, the result is clamped to zero.

For any color (R,G,B,A), the transformed color (R’, G’, B’, A’) is calculated as follows:

\[ R' = \max(0, \min(R + \text{RedAddTerm}, 255)) \]
\[ G' = \max(0, \min(G + \text{GreenAddTerm}, 255)) \]
\[ B' = \max(0, \min(B + \text{BlueAddTerm}, 255)) \]
\[ A' = \max(0, \min(A + \text{AlphaAddTerm}, 255)) \]

Addition and multiplication transforms can be combined as follows. The multiplication operation is performed first:

\[ R' = \max(0, \min((\frac{R \times \text{RedMultTerm}}{256}) + \text{RedAddTerm}, 255)) \]
\[ G' = \max(0, \min((\frac{G \times \text{GreenMultTerm}}{256}) + \text{GreenAddTerm}, 255)) \]
\[ B' = \max(0, \min((\frac{B \times \text{BlueMultTerm}}{256}) + \text{BlueAddTerm}, 255)) \]
\[ A' = \max(0, \min((\frac{A \times \text{AlphaMultTerm}}{256}) + \text{AlphaAddTerm}, 255)) \]

Like the CXFORM record, the CXFORMWITHALPHA record is byte aligned.

**CXFORMWITHALPHA**

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasAddTerms</td>
<td>Has color addition values if equal to 1</td>
</tr>
<tr>
<td>HasMultTerms</td>
<td>Has color multiply values if equal to 1</td>
</tr>
<tr>
<td>Nbits</td>
<td>Bits in each value field</td>
</tr>
<tr>
<td>RedMultTerm</td>
<td>If HasMultTerms = 1, SB[Nbits] Red multiply value</td>
</tr>
<tr>
<td>GreenMultTerm</td>
<td>If HasMultTerms = 1, SB[Nbits] Green multiply value</td>
</tr>
<tr>
<td>BlueMultTerm</td>
<td>If HasMultTerms = 1, SB[Nbits] Blue multiply value</td>
</tr>
<tr>
<td>AlphaMultTerm</td>
<td>If HasMultTerms = 1, SB[Nbits] Alpha multiply value</td>
</tr>
<tr>
<td>RedAddTerm</td>
<td>If HasAddTerms = 1, SB[Nbits] Red addition value</td>
</tr>
<tr>
<td>GreenAddTerm</td>
<td>If HasAddTerms = 1, SB[Nbits] Green addition value</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
CHAPTER 5

The Display List

Displaying a frame of a Macromedia Flash (SWF) file is a three-stage process:

1. Objects are defined with definition tags such as DefineShape, DefineSprite and so on. Each object is given a unique ID called a character, and stored in a repository called the dictionary.

2. Selected characters are copied from the dictionary and placed on the display list, which is the list of the characters that will be displayed in the next frame.

3. Once complete, the contents of the display list are rendered to the screen with ShowFrame. A depth value is assigned to each character on the display list. The depth determines the stacking order of the character. Characters with lower depth values are displayed underneath characters with higher depth values. A character with a depth value of 1 is displayed at the bottom of the stack. A character can appear more than once in the display list, but at different depths. Only one character can be at any given depth.

In SWF 1 and 2, the display list was a flat list of the objects that are present on the screen at any given time. In SWF 3 and later versions, the display list is a hierarchical list where an element on the display can have a list of child elements. For more information, see DefineSprite.

The following six tags are used to control the display list:

- PlaceObject Adds a character to the display list.
- PlaceObject2 & PlaceObject3 Adds a character to the display list, or modifies the character at the specified depth.
- RemoveObject Removes the specified character from the display list.
- RemoveObject2 Removes the character at the specified depth.
- ShowFrame Renders the contents of the display list to the display.

NOTE
The older tags, PlaceObject and RemoveObject, are rarely used in SWF 3 and later versions.
The Display List

The following diagram illustrates the display process. First, three objects are defined: a shape, a text object, and a sprite. These objects are given character IDs and stored in the dictionary. Character 1 (the shape) is then placed at depth 1, the bottom of the stack, and will be obscured by all other characters when the frame is rendered. Character 2 (the text) is placed twice; once at depth 2, and once at depth 4, the top of the stack. Character 3 (the sprite) is placed at depth 3.

Clipping layers

Flash supports a special kind of object in the display list called a clipping layer. A character placed as a clipping layer is not displayed; rather it clips (or masks) the characters placed above it. The ClipDepth field in PlaceObject2 specifies the top-most depth that the clipping layer masks.

DefineShape
Character ID = 1

DefineText
Character ID = 1

DefineSprite
Character ID = 1

Definition
Character ID = 1
Character ID = 2
Character ID = 3

Dictionary
Character ID = 2
Depth = 4
Character ID = 3
Depth = 3
Character ID = 2
Depth = 2
Character ID = 1
Depth = 1

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
Using the display list

For example, if a shape was placed at depth 1 with a ClipDepth of 4, all depths above 1, up to and including depth 4, are masked by the shape placed at depth 1. Characters placed at depths above 4 are not masked.

Using the display list

The following procedure creates and displays a Flash animation:

1. Define each character with a definition tag. Each character is given a unique character ID, and added to the dictionary.
2. Add each character to the display list with a PlaceObject2 tag. Each PlaceObject2 tag specifies the character to be displayed, plus the following attributes:
   - A depth value, which controls the stacking order of the character being placed. Characters with lower depth values appear to be underneath characters with higher depth values. A depth value of 1 means the character is displayed at the bottom of the stack. Only one character can be at any given depth.
   - A transformation matrix, which determines the position, scale, factor, and angle of rotation of the character being placed. The same character can be placed more than once (at different depths) with a different transformation matrix.
   - An optional color transform, which specifies the color effect applied to the character being placed. Color effects include transparency and color shifts.
   - An optional name string, which identifies the character being placed for SetTarget actions. SetTarget is used to perform actions inside sprite objects.

Character ID = 4
Depth = 5
Character ID = 3
Depth = 4
Character ID = 3
Depth = 3
Character ID = 2
Depth = 2
Character ID = 1
Depth = 1
ClipDepth = 4
3. Use a ShowFrame tag to render the contents of the display list to the screen.

4. Use a PlaceObject2 tag to modify each character on the display List. Each PlaceObject2 assigns a new transformation matrix to the character at a given depth. The character ID is not specified because each depth can have only one character.

5. Use a ShowFrame tag to display the characters in their new positions. Repeat steps 4 and 5 for each frame of the animation.

6. Use a RemoveObject2 tag to Remove each character from the display list. Only the depth value is required to identify the character being removed.

Display list tags

Display list tags are used to add character and character attributes to a display list.

PlaceObject

The PlaceObject tag adds a character to the display list. The CharacterId identifies the character to be added. The Depth field specifies the stacking order of the character. The Matrix field species the position, scale, and rotation of the character. If the size of the PlaceObject tag exceeds the end of the transformation matrix, it is assumed that a ColorTransform field is appended to the record. The ColorTransform field specifies a color effect (such as transparency) that is applied to the character. The same character can be added more than once to the display list with a different depth and transformation matrix.

NOTE

If a character does not change from frame to frame, you do not need to replace the unchanged character after each frame.

NOTE

PlaceObject is rarely used in SWF 3 and later versions; it is superseded by PlaceObject2 and PlaceObject3.
The minimum file format version is SWF 1.

The PlaceObject2 tag extends the functionality of the PlaceObject tag. The PlaceObject2 tag can both add a character to the display list, and modify the attributes of a character that is already on the display list. The PlaceObject2 tag changed slightly from Flash 4 to Flash 5. In Flash 5, clip actions were added.

The tag begins with a group of flags that indicate which fields are present in the tag. The optional fields are CharacterId, Matrix, ColorTransform, Ratio, ClipDepth, Name, and ClipActions. The Depth field is the only field that is always required.

The depth value determines the stacking order of the character. Characters with lower depth values are displayed underneath characters with higher depth values. A depth value of 1 means the character is displayed at the bottom of the stack. Any given depth can have only one character. This means a character that is already on the display list can be identified by its depth alone (that is, a CharacterId is not required).

The PlaceFlagMove and PlaceFlagHasCharacter tags indicate whether a new character is being added to the display list, or a character already on the display list is being modified. The meaning of the flags is as follows:

- PlaceFlagMove = 0 and PlaceFlagHasCharacter = 1 A new character (with ID of CharacterId) is placed on the display list at the specified depth. Other fields set the attributes of this new character.
- PlaceFlagMove = 1 and PlaceFlagHasCharacter = 0 The character at the specified depth is modified. Other fields modify the attributes of this character. Because any given depth can have only one character, no CharacterId is required.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORDHEADER</td>
<td>Tag type = 4</td>
</tr>
<tr>
<td>CharacterId</td>
<td>UI16</td>
<td>ID of character to place</td>
</tr>
<tr>
<td>Depth</td>
<td>UI16</td>
<td>Depth of character</td>
</tr>
<tr>
<td>Matrix</td>
<td>MATRIX</td>
<td>Transform matrix data</td>
</tr>
<tr>
<td>ColorTransform</td>
<td>CXFORM</td>
<td>Color transform data</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
The character at the specified Depth is removed, and a new character (with ID of CharacterId) is placed at that depth. Other fields set the attributes of this new character.

For example, a character that is moved over a series of frames has PlaceFlagHasCharacter set in the first frame, and PlaceFlagMove set in subsequent frames. The first frame places the new character at the desired depth, and sets the initial transformation matrix. Subsequent frames replace the transformation matrix of the character at the desired depth.

The optional fields in PlaceObject2 have the following meaning:

- The CharacterId field specifies the character to be added to the display list. CharacterId is used only when a new character is being added. If a character that is already on the display list is being modified, the CharacterId field is absent.
- The Matrix field specifies the position, scale and rotation of the character being added or modified.
- The ColorTransform field specifies the color effect applied to the character being added or modified.
- The Ratio field specifies a morph ratio for the character being added or modified. This field applies only to characters defined with DefineMorphShape, and controls how far the morph has progressed. A ratio of zero displays the character at the start of the morph. A ratio of 65535 displays the character at the end of the morph. For values between zero and 65535 Flash Player interpolates between the start and end shapes, and displays an in-between shape.
- The ClipDepth field specifies the top-most depth that will be masked by the character being added. A ClipDepth of zero indicates that this is not a clipping character.
- The Name field specifies a name for the character being added or modified. This field is typically used with sprite characters, and is used to identify the sprite for SetTarget actions. It allows the main file (or other sprites) to perform actions inside the sprite (see Sprites and Movie Clips).
- The ClipActions field, which is valid only for placing sprite characters, defines one or more event handlers to be invoked when certain events occur.
### Field Type Comment
- **Header**
  - RECORDHEADER
  - Tag type = 26
- **PlaceFlagHasClipActions**
  - UB[1]
  - SWF 5 and later: has clip actions (sprite characters only)
  - Otherwise: always 0
- **PlaceFlagHasClipDepth**
  - UB[1]
  - Has clip depth
- **PlaceFlagHasName**
  - UB[1]
  - Has name
- **PlaceFlagHasRatio**
  - UB[1]
  - Has ratio
- **PlaceFlagHasColorTransform**
  - UB[1]
  - Has color transform
- **PlaceFlagHasMatrix**
  - UB[1]
  - Has matrix
- **PlaceFlagHasCharacter**
  - UB[1]
  - Places a character
- **PlaceFlagMove**
  - UB[1]
  - Defines a character to be moved
- **Depth**
  - UI16
  - Depth of character
- **CharacterId**
  - If PlaceFlagHasCharacter
  - UI16
  - ID of character to place
- **Matrix**
  - If PlaceFlagHasMatrix
  - MATRIX
  - Transform matrix data
- **ColorTransform**
  - If PlaceFlagHasColorTransform
  - CXFORMWITHALPHA
  - Color transform data
- **Ratio**
  - If PlaceFlagHasRatio
  - UI16
- **Name**
  - If PlaceFlagHasName
  - STRING
  - Name of character
- **ClipDepth**
  - If PlaceFlagHasClipDepth
  - UI16
  - Clip depth
  - (see Clipping layers)
- **ClipActions**
  - If PlaceFlagHasClipActions
  - CLIPACTIONS
  - SWF 5 and later:
  - Clip Actions Data

---

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.

**Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8**
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
A number of different blend modes can be specified as an alternative to normal alpha compositing. The following blend modes are supported:

- Add
- Subtract
- Lighten
- Darken
- Overlay
- Difference
- Multiply
- Erase
- Screen
- Hardlight
- Invert

A number of bitmap filters can be applied to the display object. Adding filters implies that the display object will be cached as a bitmap. The following bitmap filters are supported:

- Bevel
- Drop shadow
- Blur
- Glow
- Color matrix
- Gradient bevel
- Convolution
- Gradient glow

The minimum file format version is SWF 8.
56 The Display List

Reserved  
UB[5]  
Must be 0

PlaceFlagHasCacheAsBitmap  
UB[1]  
Enables bitmap caching

PlaceFlagHasBlendMode  
UB[1]  
Has blend mode

PlaceFlagHasFilterList  
UB[1]  
Has filter list

Depth  
UI16  
Depth of character

CharacterId  
If PlaceFlagHasCharacter  
UI16  
ID of character to place

Matrix  
If PlaceFlagHasMatrix  
MATRIX  
Transform matrix data

ColorTransform  
If PlaceFlagHasColorTransform  
CXFORMWITHALPHA  
Color transform data

Ratio  
If PlaceFlagHasRatio  
UI16  
Name of character

ClipDepth  
If PlaceFlagHasClipDepth  
UI16  
Clip depth  
(see Clipping layers)

SurfaceFilterList  
If PlaceFlagHasFilterList  
FILTERLIST  
List of filters on this object

Blend Mode  
If PlaceFlagHasBlendMode  
UI8  
0 or 1 = normal  
2 = layer  
3 = multiply  
4 = screen  
5 = lighten  
6 = darken  
7 = add  
8 = subtract  
9 = difference  
10 = invert  
11 = alpha  
12 = erase  
13 = overlay  
14 = hardlight  
Values 15 to 255 are reserved.

ClipActions  
If PlaceFlagHasClipActions  
CLIPACTIONS  
SWF 5 and later:  
Clip Actions Data

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The Display List

Color Matrix filter

A Color Matrix filter applies a color transformation on the pixels of a display list object. Given an input RGBA pixel in a display list object, the color transformation is calculated in the following way:

The resulting RGBA values are saturated.

The matrix values are stored from left to right and each row from top to bottom. The last row is always assumed to be (0,0,0,0,1) and does not need to be stored.

Convolution filter

The Convolution filter is a two-dimensional discrete convolution. It is applied on each pixel of a display object. In the following mathematical representation, F is the input pixel plane, G is the input matrix, and H is the output pixel plane:

\[
R' = \begin{bmatrix}
  r0 & r1 & r2 & r3 & r4 \\
  g0 & g1 & g2 & g3 & g4 \\
  b0 & b1 & b2 & b3 & b4 \\
  a0 & a1 & a2 & a3 & a4 \\
  0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

\[
G = \begin{bmatrix}
  G_{i,j} \\
\end{bmatrix}
\]

\[
H = F \ast G + B
\]

Divisor

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The convolution is applied on each of the RGBA color components and then saturated, except when the PreserveAlpha flag is set; in this case, the alpha channel value is not modified.

The clamping flag specifies how pixels outside of the input pixel plane are handled. If set to false, the DefaultColor value is used, and otherwise, the pixel is clamped to the closest valid input pixel.

Blur filter

The blur filter is based on a sub-pixel precise median filter (also known as a box filter). The filter is applied on each of the RGBA color channels.

The general mathematical representation of a simple non-sub-pixel precise median filter is as follows, and can be easily extended to support sub-pixel precision.

\[
\begin{align*}
H_x & = \left[ \sum_{y=0}^{\text{BlurY}} \left( \sum_{i=0}^{\text{BlurX}} \frac{\text{Matrix}(i,y)}{\text{Divisor}} + \text{Bias} \right) \right] / \text{Divisor} \times \text{Divisor} \\
& = \left[ \sum_{y=0}^{\text{BlurY}} \left( \sum_{i=0}^{\text{BlurX}} \text{Matrix}(i,y) \right) \right] / \text{Divisor} \\
& = \left[ \sum_{y=0}^{\text{BlurY}} \left( \sum_{i=0}^{\text{BlurX}} \text{Matrix}(i,y) \right) - \text{Bias} \right] / \text{Divisor}
\end{align*}
\]

NOTE

This representation assumes that BlurY and BlurX are odd integers in order to get the same result as Flash Player. The filter window is always centered on a pixel in Flash Player.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.
When the number of passes is set to three, it closely approximates a Gaussian Blur filter. A higher number of passes is possible, but for performance reasons, Macromedia does not recommend it.

The Drop Shadow filter is based on the same median filter as the blur filter, but the filter is applied only on the alpha color channel to obtain a shadow pixel plane.

The angle parameter is in radians. With angle set to 0, the shadow shows on the right side of the object. The distance is measured in pixels. The shadow pixel plane values are interpolated bilinearly if sub-pixel values are used.

The strength of the shadow normalized is 1.0 in fixed point. The strength value is applied by multiplying each value in the shadow pixel plane.

Various compositing options are available for the drop shadow to support both inner and outer shadows in regular or knockout modes.

The resulting color value of each pixel is obtained by multiplying the color channel of the provided RGBA color value by the associated value in the shadow pixel plane. The resulting pixel value is composited on the original input pixel plane by using one of the specified compositing modes.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlurX</td>
<td>FIXED</td>
<td>Horizontal blur amount</td>
</tr>
<tr>
<td>BlurY</td>
<td>FIXED</td>
<td>Vertical blur amount</td>
</tr>
<tr>
<td>Passes</td>
<td>UB[5]</td>
<td>Number of blur passes</td>
</tr>
<tr>
<td>Reserved</td>
<td>UB[3]</td>
<td>Must be 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>DropShadowColor</td>
<td>RGBA</td>
<td>Color of the shadow</td>
</tr>
<tr>
<td>BlurX</td>
<td>FIXED</td>
<td>Horizontal blur amount</td>
</tr>
<tr>
<td>BlurY</td>
<td>FIXED</td>
<td>Vertical blur amount</td>
</tr>
<tr>
<td>Angle</td>
<td>FIXED</td>
<td>Radian angle of the drop</td>
</tr>
<tr>
<td>Distance</td>
<td>FIXED</td>
<td>Distance of the drop</td>
</tr>
</tbody>
</table>
Glow filter

The Glow filter works in the same way as the Drop Shadow filter, except that it does not have a distance and angle parameter. Therefore, it can run slightly faster.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>GlowColor</td>
<td>RGBA</td>
</tr>
<tr>
<td>BlurX</td>
<td>FIXED</td>
</tr>
<tr>
<td>BlurY</td>
<td>FIXED</td>
</tr>
<tr>
<td>Strength</td>
<td>FIXED8</td>
</tr>
<tr>
<td>InnerGlow</td>
<td>UB[1]</td>
</tr>
<tr>
<td>Knockout</td>
<td>UB[1]</td>
</tr>
<tr>
<td>CompositeSource</td>
<td>UB[1]</td>
</tr>
<tr>
<td>Passes</td>
<td>UB[5]</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The Display List

Bevel filter
The Bevel filter creates a smooth bevel on display list objects.

Gradient Glow and Gradient Bevel filters
The Gradient Glow and Gradient Bevel filters are extensions of the normal Glow and Bevel Filters and allow a gradient to be specified instead of a single color. Instead of multiplying a single color value by the shadow-pixel plane value, the shadow-pixel plane value is mapped directly into the gradient ramp to obtain the resulting color pixel value, which is then composited by using one of the specified compositing modes.

BEVELFILTER

Field Type Comment

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShadowColor</td>
<td>RGBA</td>
<td>Color of the shadow</td>
</tr>
<tr>
<td>HighlightColor</td>
<td>RGBA</td>
<td>Color of the highlight</td>
</tr>
<tr>
<td>BlurX</td>
<td>FIXED</td>
<td>Horizontal blur amount</td>
</tr>
<tr>
<td>BlurY</td>
<td>FIXED</td>
<td>Vertical blur amount</td>
</tr>
<tr>
<td>Angle</td>
<td>FIXED</td>
<td>Radian angle of the drop shadow</td>
</tr>
<tr>
<td>Distance</td>
<td>FIXED</td>
<td>Distance of the drop shadow</td>
</tr>
<tr>
<td>Strength</td>
<td>FIXED8</td>
<td>Strength of the drop shadow</td>
</tr>
<tr>
<td>InnerShadow</td>
<td>UB[1]</td>
<td>Inner shadow mode</td>
</tr>
<tr>
<td>Knockout</td>
<td>UB[1]</td>
<td>Knockout mode</td>
</tr>
<tr>
<td>CompositeSource</td>
<td>UB[1]</td>
<td>Composite source</td>
</tr>
<tr>
<td>OnTop</td>
<td>UB[1]</td>
<td>OnTop mode</td>
</tr>
<tr>
<td>Passes</td>
<td>UB[4]</td>
<td>Number of blur passes</td>
</tr>
</tbody>
</table>

GRADIENTGLOWFILTER

Field Type Comment

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumColors</td>
<td>UI8</td>
<td>Number of colors in the gradient</td>
</tr>
<tr>
<td>GradientColors</td>
<td>RGBA[NumColors]</td>
<td>Gradient colors</td>
</tr>
<tr>
<td>GradientRatio</td>
<td>UI8[NumColors]</td>
<td>Gradient ratios</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The extra events added in SWF 6 correspond to the Flash button movie clips, which are sprites that can be scripted in the same way as buttons (see BUTTONCONDACTION). The DragOut through Press events correspond to the button state transition events in button action conditions; the correspondence between them is shown in the description of Button events (see Events, state transitions, and actions).

The KeyDown and KeyUp events are not specific to a particular key; handlers for these events are executed whenever any key on the keyboard (with the possible exception of certain special keys) transitions to the down state or up state, respectively. To find out what key made the transition, actions within a handler should call methods of the ActionScript Key object.

The KeyPress event works differently from KeyDown and KeyUp. KeyPress is specific to a particular key or ASCII character (which is specified in the CLIPACTIONRECORD). KeyPress events work in an identical way (see BUTTONCONDACTION).
The Display List

RemoveObject

The RemoveObject tag removes the specified character (at the specified depth) from the display list.

The minimum file format version is SWF 1.

RemoveObject2

The RemoveObject2 tag removes the character at the specified depth from the display list.

The minimum file format version is SWF 3.

ShowFrame

The ShowFrame tag instructs Flash Player to display the contents of the display list. The file is paused for the duration of a single frame.

The minimum file format version is SWF 1.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
CHAPTER 6
Control Tags

Control tags manage some overall aspects of files, frames, and playback in Macromedia Flash (SWF) files.

SetBackgroundColor
The SetBackgroundColor tag sets the background color of the display.
The minimum file format version is SWF 1.

FrameLabel
The FrameLabel tag gives the specified Name to the current frame. ActionGoToLabel uses this name to identify the frame.
The minimum file format version is SWF 3.

---

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.
In SWF files of version 6 or later, an extension to the FrameLabel tag called named anchors is available. A named anchor is a special kind of frame label that, in addition to labeling a frame for seeking using ActionGoToLabel, labels the frame for seeking using HTML anchor syntax. The browser plug-in versions of the Macromedia Flash Player, in version 6 and later, will inspect the URL in the browser's Location bar for an anchor specification (a trailing phrase of the form #anchorname). If an anchor specification is present in the Location bar, Flash Player will begin playback starting at the frame that contains a FrameLabel tag that specifies a named anchor of the same name, if one exists; otherwise playback will begin at Frame 1 as usual. In addition, when Flash Player arrives at a frame that contains a named anchor, it will add an anchor specification with the given anchor name to the URL in the browser's Location bar. This ensures that when users create a bookmark at such a time, they can later return to the same point in the Flash file, subject to the granularity at which named anchors are present within the file.

To create a named anchor, insert one additional non-null byte after the null terminator of the anchor name. This is valid only for SWF 6 or later.

### Protect

The Protect tag marks a file as not importable for editing in an authoring environment. If the Protect tag contains no data (tag length = 0), the SWF file cannot be imported. If this tag is present in the file, any authoring tool should prevent the file from loading for editing.

If the Protect tag does contain data (tag length is not 0), the SWF file can be imported if the correct password is specified. The data in the tag is a null-terminated string that specifies an MD5 encrypted password. Specifying a password is only supported in SWF 5 or later.

The MD5 password encryption algorithm used was written by Poul-Henning Kamp and is freely distributable. It resides in the FreeBSD tree at src/lib/libcrypt/crypt-md5.c. The EnableDebugger tag also uses MD5 password encryption algorithm.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The ImportAssets tag imports characters from another SWF file. The importing SWF file references the exporting SWF file by the URL where it can be found. Imported assets are added to the dictionary just like characters defined within a SWF file.

The URL of the exporting SWF file can be absolute or relative. If it is relative, it will be resolved relative to the location of the importing SWF file.

The ImportAssets tag must be earlier in the frame than any later tags that rely on the imported assets.

The ImportAssets tag was deprecated in SWF 8; Flash Player 8 or later ignores this tag. In SWF 8 or later, use the ImportAssets2 tag instead.

The minimum file format version is SWF 5, and the maximum file format version is SWF 7.
The EnableDebugger tag enables debugging. The password in the EnableDebugger tag is encrypted by using the MD5 algorithm, in the same way as the Protect tag. The EnableDebugger tag was deprecated in SWF 6; Flash Player 6 or later ignores this tag because the format of the debugging information required in the ActionScript debugger was changed in SWF 6. In SWF 6 or later, use the EnableDebugger2 tag instead. The minimum and maximum file format version is SWF 5.

The EnableDebugger2 tag enables debugging. The Password field is encrypted by using the MD5 algorithm, in the same way as the Protect tag. The minimum file format version is SWF 6.

### Field Types

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORDHEADER</td>
<td>Tag type = 58</td>
</tr>
<tr>
<td>Password</td>
<td>Null-terminated STRING</td>
<td>MD5-encrypted password</td>
</tr>
</tbody>
</table>

### Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The ScriptLimits tag includes two fields that can be used to override the default settings for maximum recursion depth and ActionScript time-out: MaxRecursionDepth and ScriptTimeoutSeconds.

The MaxRecursionDepth field sets the ActionScript maximum recursion limit. The default setting is 256 at the time of this writing. This default can be changed to any value greater than zero (0).

The ScriptTimeoutSeconds field sets the maximum number of seconds the player should process ActionScript before displaying a dialog box asking if the script should be stopped. The default value for ScriptTimeoutSeconds varies by platform and is between 15 to 20 seconds. This default value is subject to change.

The minimum file format version is SWF 7.

Flash Player maintains a concept of tab order of the interactive and textual objects displayed. Tab order is used both for actual tabbing and, in SWF 6 and later, to determine the order in which objects are exposed to accessibility aids (such as screen readers). The SWF 7 SetTabIndex tag sets the index of an object within the tab order.

If no character is currently placed at the specified depth, this tag is ignored.

You can also use the ActionScript tabIndex property to establish tab ordering, but this does not provide a way to set a tab index for a static text object, because the player does not provide a scripting reflection of static text objects. Fortunately, this is not a problem for the purpose of tabbing, because static text objects are never actually tab stops. However, this is a problem for the purpose of accessibility ordering, because static text objects are exposed to accessibility aids. When generating SWF content that is intended to be accessible and contains static text objects, the SetTabIndex tag is more useful than the tabIndex property.
The minimum file format version is SWF 7.

FileAttributes

The FileAttributes tag defines characteristics of the SWF file. This tag is required for SWF 8 and later and must be the first tag in the SWF file. Additionally, the FileAttributes tag can optionally be included in all SWF file versions.

The HasMetadata flag identifies whether the SWF file contains the Metadata tag.

The UseNetwork flag signifies whether Flash Player should grant the SWF local or network file access if the SWF file is loaded locally. Flash Player 8 introduced security restrictions on the privileges of SWF files loaded from local file systems. The default behavior is to allow local SWF files to interact with local files only, and not with the network. However, by setting the UseNetwork flag, the local SWF can forfeit its local file system access in exchange for access to the network. Any version of SWF can use the UseNetwork flag to set the file access for locally loaded SWF files for Flash Player 8 or later. For further detail about Flash Player 8 security changes, refer to http://www.macromedia.com/devnet/flash/articles/fplayer8_security.html.

<table>
<thead>
<tr>
<th>SetTabIndex</th>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORDHEADER</td>
<td>Tag type = 66</td>
</tr>
<tr>
<td>Depth</td>
<td>UI16</td>
<td>Depth of character</td>
</tr>
<tr>
<td>TabIndex</td>
<td>UI16</td>
<td>Tab order value</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The minimum file format version is SWF 1.

ImportAssets2

The ImportAssets2 tag replaces the ImportAssets tag for SWF 8 and later. ImportAssets2 currently mirrors the ImportAssets tag’s functionality.

The ImportAssets2 tag imports characters from another SWF file. The importing SWF file references the exporting SWF file by the URL where it can be found. Imported assets are added to the dictionary just like characters defined within a SWF file.

The URL of the exporting SWF file can be absolute or relative. If it is relative, it is resolved relative to the location of the importing SWF file.

The ImportAssets2 tag must be earlier in the frame than any later tags that rely on the imported assets.

FileAttributes

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECORDHEADER</td>
<td>Tag type = 69</td>
</tr>
<tr>
<td>Reserved UB[3]</td>
<td>Must be 0</td>
</tr>
<tr>
<td>HasMetadata UB[1]</td>
<td>If 1, the SWF file contains the Metadata tag. If 0, the SWF file does not contain the Metadata tag</td>
</tr>
<tr>
<td>Reserved UB[3]</td>
<td>Must be 0</td>
</tr>
<tr>
<td>UseNetwork UB[1]</td>
<td>If 1, the SWF file is given network file access when loaded locally. If 0, this SWF file is given local file access when loaded locally</td>
</tr>
<tr>
<td>Reserved UB[24]</td>
<td>Must be 0</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
The format of the metadata is RDF that is compliant with the XMP specification. For more information about RDF and XMP, see the following sources:

- The RDF Primer at http://www.w3.org/TR/rdf-primer/
- The RDF Specification at http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

The following examples show two of many acceptable ways to represent the Metadata string in the SWF file. The first example provides basic information about the SWF file, the title and description:

```xml
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
  <rdf:Description rdf:about='' xmlns:dc='http://purl.org/dc/1.1'>
    <dc:title>Simple Title</dc:title>
    <dc:description>Simple Description</dc:description>
  </rdf:Description>
</rdf:RDF>
```

In the second example, the title is described for multiple languages:

```xml
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
  <rdf:Description rdf:about='' xmlns:dc='http://purl.org/dc/1.1'>
    <dc:title>
      <rdf:Alt>
        <rdf:li xml:lang='x-default'>Default Title</rdf:li>
        <rdf:li xml:lang='it-it'>Titolo Italiano</rdf:li>
      </rdf:Alt>
    </dc:title>
    <dc:description>Simple Description</dc:description>
  </rdf:Description>
</rdf:RDF>
```

The Metadata string is stored in the SWF file with all unnecessary white space removed. The minimum file format version is SWF 1.
The DefineScalingGrid tag introduces the concept of 9-slice scaling, which allows component-style scaling to be applied to a sprite or button character.

When the DefineScalingGrid tag associates a character with a 9-slice grid, Flash Player conceptually divides the sprite or button into nine sections with a grid-like overlay. When the character is scaled, each of the nine areas is scaled independently. To maintain the visual integrity of the character, corners are not scaled, while the remaining areas of the image are scaled larger or smaller, as needed.

The Splitter rectangle specifies the center portion of the nine regions of the scaling grid, and from this rectangle Flash Player derives the 9-slice grid. The width and height of the rectangle must be at least one twip each (1/20 pixel), or Flash Player ignores the DefineScalingGrid tag.

When a sprite or button with a DefineScalingGrid association is scaled, the nine regions of the character scale according to the following table:

<table>
<thead>
<tr>
<th>9-slice scaling</th>
<th>No scale</th>
<th>No scale</th>
<th>Vertical scale</th>
<th>Vertical scale</th>
<th>No scale</th>
<th>No scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal scale</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Horizontal scale</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9-slice scaling does not affect the children of, or any text within, the specified character. These objects transform normally.

The sprite or button with a DefineScalingGrid association cannot be rotated or skewed, and doing so disables 9-slice behavior. However, this limitation does not apply to parents or children of the 9-slice object, and parent rotation or skew is applied to the 9-slice objects in the normal manner.

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Flash Player stretches any fills in the character to fit the shape. 9-slice scaling does not affect the bounds or origin of any object. If a 9-slice character is scaled below its original size, the five scaling regions are consumed until they become very small. Once the minimum size is reached, Flash Player reverts to normal, non-9-slice scaling.

The minimum file format version is SWF 8.
CHAPTER 7

Actions

Actions are an essential part of an interactive Macromedia Flash (SWF) file. Actions allow a file to react to events such as mouse movements or mouse clicks. The SWF 3 action model and earlier supports a simple action model. The SWF 4 action model supports a greatly enhanced action model that includes an expression evaluator, variables, and conditional branching and looping. The SWF 5 action model adds a JavaScript-style object model, data types and functions.

SWF 3 action model

The SWF 3 action model consists of eleven instructions for Flash Player:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Play</td>
<td>ActionPlay</td>
</tr>
<tr>
<td>Stop</td>
<td>ActionStop</td>
</tr>
<tr>
<td>NextFrame</td>
<td>ActionNextFrame</td>
</tr>
<tr>
<td>PreviousFrame</td>
<td>ActionPreviousFrame</td>
</tr>
<tr>
<td>GotoFrame</td>
<td>ActionGotoFrame</td>
</tr>
<tr>
<td>GotoLabel</td>
<td>ActionGoToLabel</td>
</tr>
<tr>
<td>WaitForFrame</td>
<td>ActionWaitForFrame</td>
</tr>
<tr>
<td>GetURL</td>
<td>ActionGetURL</td>
</tr>
<tr>
<td>StopSounds</td>
<td>ActionStopSounds</td>
</tr>
<tr>
<td>ToggleQuality</td>
<td>ActionToggleQuality</td>
</tr>
<tr>
<td>SetTarget</td>
<td>ActionSetTarget</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
An action (or list of actions) can be triggered by a button state transition, or by SWF 3 actions. The action is not executed immediately, but is added to a list of actions to be processed. The list is executed on a ShowFrame tag, or after the button state has changed. An action can cause other actions to be triggered, in which case, the action is added to the list of actions to be processed. Actions are processed until the action list is empty.

By default, Timeline actions such as Stop (see ActionStop), Play (see ActionPlay), and GoToFrame (see ActionGotoFrame) apply to files that contain them. However, the SetTarget action (see ActionSetTarget), which is called TellTarget in the Macromedia Flash user interface, can be used to send an action command to another file or sprite (see DefineSprite).

SWF 3 actions

The actions in this section are available in SWF 3.

DoAction

DoAction instructs Flash Player to perform a list of actions when the current frame is complete. The actions are performed when the ShowFrame tag is encountered, regardless of where in the frame the DoAction tag appears.

---

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORDHEADER</td>
</tr>
<tr>
<td></td>
<td>Tag type = 12</td>
</tr>
<tr>
<td>Actions</td>
<td>ACTIONRECORD [zero or more]</td>
</tr>
<tr>
<td></td>
<td>List of actions to perform (see following table, ActionRecord)</td>
</tr>
<tr>
<td>ActionEndFlag</td>
<td>UI8 = 0</td>
</tr>
<tr>
<td></td>
<td>Always set to 0</td>
</tr>
<tr>
<td>ActionCode</td>
<td>code = UI8</td>
</tr>
<tr>
<td></td>
<td>An action code (see following)</td>
</tr>
<tr>
<td>Length</td>
<td>If code &gt;= 0x80 UI16</td>
</tr>
<tr>
<td></td>
<td>The number of bytes (after this) in the ACTIONRECORD.</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
ActionGotoFrame
ActionGotoFrame instructs Flash Player to go to the specified frame in the current file.

ActionGetURL
ActionGetURL instructs Flash Player to get the URL that UrlString specifies. The URL can be of any type, including an HTML file, an image or another SWF file. If the file is playing in a browser, the URL is displayed in the frame that TargetString specifies. The "_level0" and "_level1" special target names are used to load another SWF file into levels 0 and 1 respectively.

ActionNextFrame
ActionNextFrame instructs Flash Player to go to the next frame in the current file.

ActionPreviousFrame
ActionPreviousFrame instructs Flash Player to go to the previous frame of the current file.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
ActionSetTarget

ActionSetTarget instructs Flash Player to change the context of subsequent actions, so they apply to a named object (TargetName) rather than the current file. For example, the SetTarget action can be used to control the Timeline of a sprite object. The following sequence of actions sends a sprite called "spinner" to the first frame in its Timeline:

1. SetTarget "spinner"
2. GotoFrame zero
3. SetTarget " " (empty string)
4. End of actions. (Action code = 0)

All actions following SetTarget "spinner" apply to the spinner object until SetTarget "", which sets the action context back to the current file. For a complete discussion of target names see DefineSprite.

ActionGoToLabel

ActionGoToLabel instructs Flash Player to go to the frame associated with the specified label. You can attach a label to a frame with the FrameLabel tag.

SWF 4 action model

The SWF 4 file format supports a greatly enhanced action model that includes an expression evaluator, variables, conditional branching and looping.

Macromedia Flash Player 4 incorporates a stack machine that interprets and executes SWF 4 actions. The key SWF 4 action is ActionPush. This action is used to push one or more parameters onto the stack. Unlike SWF 3 actions, SWF 4 actions do not have parameters embedded in the tag, rather they push parameters onto the stack, and pop results off the stack.
The expression evaluator is also stack based. Arithmetic operators include ActionAdd, ActionSubtract, ActionMultiply, and ActionDivide. The Flash authoring tool converts expressions to a series of stack operations. For example, the expression 1+x*3 is represented as the following action sequence:

ActionPush "x"
ActionGetVariable
ActionPush "3"
ActionMultiply
ActionPush "1"
ActionAdd

The result of this expression is on the stack.

The program counter

The current point of execution of Flash Player is called the program counter (PC). The value of the PC is defined as the address of the action that follows the action currently being executed. Control flow actions such as ActionJump change the value of the PC. These actions are similar to branch instructions in assembler, or the goto instruction in other languages. For example, ActionJump tells Flash Player to jump to a new position in the action sequence. The new PC is specified as an offset from the current PC. Both positive and negative offsets can occur, so Flash Player can jump forward and backward in the action sequence.

SWF 4 actions

The following actions are available in SWF 4:

<table>
<thead>
<tr>
<th>Type of action</th>
<th>Name of action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic operators</td>
<td>ActionAdd, ActionDivide, ActionMultiply, ActionSubtract</td>
</tr>
<tr>
<td>Numerical comparison</td>
<td>ActionEquals, ActionLess</td>
</tr>
<tr>
<td>Logical operators</td>
<td>ActionAnd, ActionNot, ActionOr</td>
</tr>
</tbody>
</table>

NOTE

All values on the stack, including numeric values, are stored as strings. In the preceding example, the numeric values 3 and 1 are pushed onto the stack as the strings "3" and "1".
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
This section lists stack operations.

**ActionPush**

ActionPush pushes one or more values to the stack. The Type field specifies the type of the value to be pushed.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>UI8</td>
<td>0 = string literal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = floating-point literal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = null</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 = undefined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 = register</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 = Boolean</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 = double</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 = integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 = constant 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9 = constant 16</td>
</tr>
</tbody>
</table>

**String**

If Type = 0, STRING

Null-terminated character string

**Float**

If Type = 1, FLOAT

32-bit IEEE single-precision little-endian floating-point value

**RegisterNumber**

If Type = 4, UI8

Register number

**Boolean**

If Type = 5, UI8

Boolean value

**Double**

If Type = 6, DOUBLE

64-bit IEEE double-precision little-endian double value

**Integer**

If Type = 7, UI32

32-bit little-endian integer

**Constant8**

If Type = 8, UI8

Constant pool index (for indexes < 256) (see ActionConstantPool)

**Constant16**

If Type = 9, UI16

Constant pool index (for indexes >= 256) (see ActionConstantPool)
If Type = 1, the value to be pushed is specified as a 32-bit IEEE single-precision little-endian floating-point value. PropertyIds are pushed as FLOATs. ActionGetProperty and ActionSetProperty use PropertyIds to access the properties of named objects.

If Type = 4, the value to be pushed is a register number. Flash Player supports up to 4 registers. With the use of ActionDefineFunction2, up to 256 registers can be used.

In the first field of ActionPush, the length in ActionRecord defines the total number of Type and value bytes that follow the ACTIONRECORD itself. More than one set of Type and value fields may follow the first field, depending on the number of bytes that the length in ACTIONRECORD specifies.

ActionPop pops a value from the stack and discards it.

ActionPop pops a value off the stack and discards the value.

Arithmetic operators

The following sections describe arithmetic operators.

ActionAdd

ActionAdd adds two numbers and pushes the result back to the stack.

ActionAdd does the following:

1. Pops value A off the stack.
2. Pops value B off the stack.
3. Converts A and B to floating-point; non-numeric values evaluate to 0.
4. Adds the numbers A and B.
5. Pushes the result, A+B, to the stack.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
ActionOr does the following:
1. Pops value A off the stack.
2. Pops value B off the stack.
3. Converts A and B to floating-point; non-numeric values evaluate to 0.
4. If either of the numbers is nonzero, \textit{true} is pushed to the stack for SWF 5 and later (1 is pushed for SWF 4); otherwise, \textit{false} is pushed to the stack for SWF 5 and later (0 is pushed for SWF 4).

ActionNot performs a logical NOT of a number.

ActionNot does the following:
1. Pops a value off the stack.
2. Converts the value to floating point; non-numeric values evaluate to 0.
3. If the value is zero, \textit{true} is pushed on the stack for SWF 5 and later (1 is pushed for SWF 4).
4. If the value is nonzero, \textit{false} is pushed on the stack for SWF 5 and later (0 is pushed for SWF 4).

String manipulation
ActionStringEquals
ActionStringEquals tests two strings for equality.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
\textbf{Field} & \textbf{Type} & \textbf{Comment} \\
\hline
ActionNot & UI8 & Action = 0x12 \\
\hline
Result & 0x0 & Boolean \\
\hline
ActionStringEquals & UI8 & Action = 0x13 \\
\hline
\end{tabular}
\end{table}
92 Actions

ActionStringEquals does the following:
1. Pops value A off the stack.
2. Pops value B off the stack.
3. Compares A and B as strings.
   The comparison is case-sensitive.
4. If the strings are equal, \texttt{true} is pushed to the stack for SWF 5 and later (SWF 4 pushes \texttt{1}).
5. Otherwise, \texttt{false} is pushed to the stack for SWF 5 and later (SWF 4 pushes \texttt{0}).

ActionStringLength

ActionStringLength computes the length of a string.

ActionStringLength does the following:
1. Pops a string off the stack.
2. Calculates the length of the string and pushes it to the stack.

ActionStringAdd

ActionStringAdd concatenates two strings.

ActionStringAdd does the following:
1. Pops value A off the stack.
2. Pops value B off the stack.
3. Pushes the concatenation BA to the stack.

ActionStringExtract

ActionStringExtract extracts a substring from a string.

Field Type Comment
---
ActionStringLength UI8 Action = 0x14
ActionStringAdd UI8 Action = 0x21
ActionStringExtract UI8 Action = 0x15

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
ActionStringExtract does the following:

1. Pops number \textit{count} off the stack.
2. Pops number \textit{index} off the stack.
3. Pops string \textit{string} off the stack.
4. Pushes the substring of the string starting at the indexed character and \textit{count} characters in length to the stack.
5. If either \textit{index} or \textit{count} do not evaluate to integers, the result is the empty string.

ActionStringLess

ActionStringLess tests to see if a string is less than another string.

ActionStringLess does the following:

1. Pops value \textit{A} off the stack.
2. Pops value \textit{B} off the stack.
3. If \textit{B} < \textit{A} using a byte-by-byte comparison, \textit{true} is pushed to the stack for SWF 5 and later (SWF 4 pushes 1); otherwise, \textit{false} is pushed to the stack for SWF 5 and later (SWF 4 pushes 0).

ActionMBStringLength

ActionMBStringLength computes the length of a string and is multi-byte aware.

ActionMBStringLength does the following:

1. Pops a string off the stack.
2. Calculates the length of the string in characters and pushes it to the stack.

This is a multi-byte aware version of ActionStringLength. On systems with double-byte support, a double-byte character is counted as a single character.
ActionMBStringExtract extracts a substring from a string and is multi-byte aware.

It does the following:
1. Pops the number \( \text{count} \) off the stack.
2. Pops the number \( \text{index} \) off the stack.
3. Pops the string \( \text{string} \) off the stack.
4. Pushes the substring of string starting at the index'th character and count characters in length to the stack.

This is a multi-byte aware version of ActionStringExtract. Index and count are treated as character indexes, counting double-byte characters as single characters.

ActionToInteger converts a value to an integer.

ActionToInteger does the following:
1. Pops a value off the stack.
2. Converts the value to a number.
3. Discards any digits after the decimal point, resulting in an integer.
4. Pushes the resulting integer to the stack.

**Field Type Comment**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionMBStringExtract</td>
<td>UI8</td>
<td>Action = 0x35</td>
</tr>
<tr>
<td>ActionToInteger</td>
<td>UI8</td>
<td>Action = 0x18</td>
</tr>
</tbody>
</table>

**NOTE**

If either index or count do not evaluate to integers, the result is the empty string.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
96 Actions

ActionMBAsciiToChar
converts ASCII to character code and is multi-byte aware.
ActionMBAsciiToChar does the following:
1. Pops a value off the stack.
2. Converts the value from a number to the corresponding character.
3. If the character is a 16-bit value (>= 256), a double-byte character is constructed with the first byte containing the high-order byte, and the second byte containing the low-order byte.
4. Pushes the resulting character to the stack.

Control flow

ActionJump
creates an unconditional branch.
ActionJump adds BranchOffset bytes to the instruction pointer in the execution stream.
The offset is a signed quantity, enabling branches from –32,768 bytes to 32,767 bytes. An offset of 0 points to the action directly after the ActionJump action.

ActionIf
creates a conditional test and branch.

Field Type Comment
ActionMBAsciiToChar UI8 Action = 0x37
ActionJump UI8 Action = 0x99
BranchOffset SI16
ActionIf UI8 Action = 0x9D
BranchOffset SI16

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
ActionIf does the following:

1. Pops Condition, a number, off the stack.
2. Converts Condition to a Boolean value.
3. Tests if Condition is true. If Condition is true, BranchOffset bytes are added to the instruction pointer in the execution stream.

The offset is a signed quantity, enabling branches from –32768 bytes to 32767 bytes. An offset of 0 points to the action directly after the ActionIf action.

ActionCall calls a subroutine.

ActionCall does the following:

1. Pops a value off the stack. This value should be either a string that matches a frame label, or a number that indicates a frame number. The value can be prefixed by a target string that identifies the movie clip that contains the frame being called.
2. If the frame is successfully located, the actions in the target frame are executed. After the actions in the target frame are executed, execution resumes at the instruction after the ActionCall instruction.
3. If the frame cannot be found, nothing happens.

NOTE When playing a Flash 4 SWF file, Condition is not converted to a Boolean value and is instead compared to 0, not true.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Movie control
ActionGetURL2
gets a URL and is stack based.
ActionGetURL2 does the following:

1. Pops target off the stack.
   - A LoadTargetFlag value of **0** indicates that the target is a window. The target can be an empty string to indicate the current window.
   - A LoadTargetFlag value of **1** indicates that the target is a path to a sprite. The target path can be in slash or dot syntax.

2. Pops a URL off the stack; the URL specifies the URL to be retrieved.

3. SendVarsMethod specifies the method to use for the HTTP request.
   - A SendVarsMethod value of **0** indicates that this is not a form request, so the movie clip’s variables should not be encoded and submitted.
   - A SendVarsMethod value of **1** specifies a HTTP GET request.
   - A SendVarsMethod value of **2** specifies a HTTP POST request.

4. If the SendVarsMethod value is **1** (GET) or **2** (POST), the variables in the current movie clip are submitted to the URL by using the standard x-www-form-urlencoded encoding and the HTTP request method specified by method.

If the LoadVariablesFlag is set, the server is expected to respond with a MIME type of application/x-www-form-urlencoded and a body in the format var1=value1&var2=value2&...&varx=valuex. This response is used to populate ActionScript variables rather than display a document. The variables populated can be in a timeline (if LoadTargetFlag is 0) or in the specified sprite (if LoadTargetFlag is 1).

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionGetURL2</td>
<td>UI8</td>
</tr>
<tr>
<td>SendVarsMethod</td>
<td>UB[2]</td>
</tr>
<tr>
<td>Reserved</td>
<td>UB[4]</td>
</tr>
<tr>
<td>LoadTargetFlag</td>
<td>UB[1]</td>
</tr>
<tr>
<td>LoadVariablesFlag</td>
<td>UB[1]</td>
</tr>
</tbody>
</table>
If the LoadTargetFlag is specified without the LoadVariablesFlag, the server is expected to respond with a MIME type of application/x-shockwave-flash and a body that consists of a SWF file. This response is used to load a subfile into the specified sprite rather than to display an HTML document.

**ActionGotoFrame2**

**ActionGotoFrame2** goes to a frame and is stack based.

**ActionGotoFrame2** does the following:

1. Pops a frame off the stack.
   - If the frame is a number, \(n\), the next frame of the movie to be displayed is the \(n\)th frame in the current movie clip.
   - If the frame is a string, frame is treated as a frame label. If the specified label exists in the current movie clip, the labeled frame will become the current frame. Otherwise, the action is ignored.

2. Either a frame or a number can be prefixed by a target path, for example, /MovieClip:3 or /MovieClip:FrameLabel.

3. If the Play flag is set, the action goes to the specified frame and begins playing the enclosing movie clip. Otherwise, the action goes to the specified frame and stops.

**ActionSetTarget2**

**ActionSetTarget2** sets the current context and is stack based.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionGotoFrame2</td>
<td>UI8</td>
</tr>
<tr>
<td>Reserved</td>
<td>UB[6]</td>
</tr>
<tr>
<td>SceneBiasFlag</td>
<td>UB[1]</td>
</tr>
<tr>
<td>Play flag</td>
<td>UB[1]</td>
</tr>
<tr>
<td>SceneBias</td>
<td>UI16</td>
</tr>
</tbody>
</table>

**ActionSetTarget2** pops the target off the stack and makes it the current active context.

---

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.
This action behaves exactly like ActionSetTarget but is stack based to enable the target path to be the result of expression evaluation.

**ActionGetProperty**

gets a file property. ActionGetProperty does the following:

1. Pops index off the stack.
2. Pops target off the stack.
3. Retrieves the value of the property enumerated as index from the movie clip with target path target and pushes the value to the stack.

The following table lists property index values. The _quality, _xmouse and _ymouse properties are available in SWF 5 and later.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionGetProperty</td>
<td>UI8</td>
<td>Action = 0x22</td>
</tr>
<tr>
<td>_X</td>
<td>0x0</td>
<td>0</td>
</tr>
<tr>
<td>_Y</td>
<td>1x0</td>
<td>1</td>
</tr>
<tr>
<td>_xscale</td>
<td>2x0</td>
<td>2</td>
</tr>
<tr>
<td>_yscale</td>
<td>3x0</td>
<td>3</td>
</tr>
<tr>
<td>_currentframe</td>
<td>4x0</td>
<td>4</td>
</tr>
<tr>
<td>_totalframes</td>
<td>5x0</td>
<td>5</td>
</tr>
<tr>
<td>_alpha</td>
<td>6x0</td>
<td>6</td>
</tr>
<tr>
<td>_visible</td>
<td>7x0</td>
<td>7</td>
</tr>
<tr>
<td>_width</td>
<td>8x0</td>
<td>8</td>
</tr>
<tr>
<td>_height</td>
<td>9x0</td>
<td>9</td>
</tr>
<tr>
<td>_rotation</td>
<td>10x0</td>
<td>10</td>
</tr>
<tr>
<td>_target</td>
<td>11x0</td>
<td>11</td>
</tr>
<tr>
<td>_framesloaded</td>
<td>12x0</td>
<td>12</td>
</tr>
<tr>
<td>_name</td>
<td>13x0</td>
<td>13</td>
</tr>
<tr>
<td>_droptarget</td>
<td>14x0</td>
<td>14</td>
</tr>
<tr>
<td>_url</td>
<td>15x0</td>
<td>15</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
**102 Actions**

**ActionSetProperty**

*ActionSetProperty* does the following:

1. Pops a value off the stack.
2. Pops an index off the stack.
3. Pops a target off the stack.
4. Sets the property enumerated as index in the movie clip with the target path `target` to the value `value`.

**ActionCloneSprite**

*ActionCloneSprite* clones a sprite.

*ActionCloneSprite* does the following:

1. Pops a depth off the stack.
2. Pops a target off the stack.
3. Pops a source off the stack.
4.Duplicates the movie clip source, giving the new instance the name `target`, at z-order depth `depth`.

<table>
<thead>
<tr>
<th>Property Value</th>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>highquality</em></td>
<td>UI8</td>
<td>Action = 0x23</td>
</tr>
<tr>
<td><em>focusrect</em></td>
<td>UI8</td>
<td>Action = 0x24</td>
</tr>
<tr>
<td><em>soundbuftime</em></td>
<td>UI8</td>
<td>Action = 0x25</td>
</tr>
<tr>
<td><em>quality</em></td>
<td>UI8</td>
<td>Action = 0x26</td>
</tr>
<tr>
<td><em>xmouse</em></td>
<td>UI8</td>
<td>Action = 0x27</td>
</tr>
<tr>
<td><em>ymouse</em></td>
<td>UI8</td>
<td>Action = 0x28</td>
</tr>
</tbody>
</table>
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
ActionWaitForFrame2

waits for a frame to be loaded and is stack based.

1. Pops a frame off the stack.
2. If the frame is loaded, SkipCount actions that follow the current action are skipped.
3. The frame is evaluated in the same way as ActionGotoFrame2.

ActionTrace

sends a debugging output string.

1. Pops a value off the stack.
2. In the Test Movie mode of the Macromedia Flash editor, ActionTrace appends a value to the output window if the debugging level is not set to None. In the Macromedia Flash Player, nothing happens.

Field Type Comment

ActionWaitForFrame2

UI8 Action = 0x8D

SkipCount

UI8

Field Type Comment

ActionTrace

UI8 Action = 0x26

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader®or Adobe®Acrobat® version 6.0 or above.
ActionGetTime reports the milliseconds since the Macromedia Flash Player started.

ActionGetTime does the following:
1. Calculates as an integer the number of milliseconds since Flash Player was started.
2. Pushes the number to the stack.

ActionRandomNumber calculates a random number.

ActionRandomNumber does the following:
1. Pops the maximum off the stack.
2. Calculates a random number as an integer in the range 0…(maximum-1).
3. Pushes the random number to the stack.

SWF 5 action model
SWF 5 is similar to SWF 4. New actions greatly expand ActionScript functionality. There are also new type conversion, math and stack operator actions.

Field Type Comment
---
ActionGetTime UI8 Action = 0x34

Field Type Comment
---
ActionRandomNumber UI8 Action = 0x30
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

---

### SWF 5 actions

<table>
<thead>
<tr>
<th>Type of action</th>
<th>Name of action</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScriptObject actions</td>
<td>ActionCallFunction</td>
</tr>
<tr>
<td></td>
<td>ActionCallMethod</td>
</tr>
<tr>
<td></td>
<td>ActionConstantPool</td>
</tr>
<tr>
<td></td>
<td>ActionDefineFunction</td>
</tr>
<tr>
<td></td>
<td>ActionDefineLocal</td>
</tr>
<tr>
<td></td>
<td>ActionDefineLocal2</td>
</tr>
<tr>
<td></td>
<td>ActionDelete</td>
</tr>
<tr>
<td></td>
<td>ActionDelete2</td>
</tr>
<tr>
<td></td>
<td>ActionEnumerate</td>
</tr>
<tr>
<td></td>
<td>ActionEquals2</td>
</tr>
<tr>
<td></td>
<td>ActionGetMember</td>
</tr>
<tr>
<td></td>
<td>ActionInitArray</td>
</tr>
<tr>
<td></td>
<td>ActionInitObject</td>
</tr>
<tr>
<td></td>
<td>ActionNewMethod</td>
</tr>
<tr>
<td></td>
<td>ActionNewObject</td>
</tr>
<tr>
<td></td>
<td>ActionSetMember</td>
</tr>
<tr>
<td></td>
<td>ActionTargetPath</td>
</tr>
<tr>
<td></td>
<td>ActionWith</td>
</tr>
<tr>
<td>Type actions</td>
<td>ActionToNumber</td>
</tr>
<tr>
<td></td>
<td>ActionToString</td>
</tr>
<tr>
<td></td>
<td>ActionTypeOf</td>
</tr>
<tr>
<td>Math actions</td>
<td>ActionAdd2</td>
</tr>
<tr>
<td></td>
<td>ActionLess2</td>
</tr>
<tr>
<td></td>
<td>ActionModulo</td>
</tr>
<tr>
<td>Stack operator actions</td>
<td>ActionBitAnd</td>
</tr>
<tr>
<td></td>
<td>ActionBitLShift</td>
</tr>
<tr>
<td></td>
<td>ActionBitOr</td>
</tr>
<tr>
<td></td>
<td>ActionBitRShift</td>
</tr>
<tr>
<td></td>
<td>ActionBitURShift</td>
</tr>
<tr>
<td></td>
<td>ActionBitXor</td>
</tr>
<tr>
<td></td>
<td>ActionDecrement</td>
</tr>
<tr>
<td></td>
<td>ActionIncrement</td>
</tr>
<tr>
<td></td>
<td>ActionPush</td>
</tr>
<tr>
<td></td>
<td>ActionPushDuplicate</td>
</tr>
<tr>
<td></td>
<td>ActionReturn</td>
</tr>
<tr>
<td></td>
<td>ActionStackSwap</td>
</tr>
<tr>
<td></td>
<td>ActionStoreRegister</td>
</tr>
</tbody>
</table>

---

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
If the named method exists, ActionCallMethod does the following:

1. Pops the name of the method from the stack. If the method name is blank or undefined, the object is taken to be a function object that should be invoked, rather than the container object of a method. For example, if CallMethod is invoked with object obj and method name blank, it’s equivalent to using the syntax: `obj();` If a method’s name is foo, it’s equivalent to: `obj.foo();`

2. Pops the ScriptObject, object, from the stack.

3. Pops the number of arguments, args, from the stack.

4. Pops the arguments off the stack.

5. Executes the method call with the specified arguments.

6. Pushes the return value of the method or function to the stack. If no appropriate return value is present (the function does not have a return statement), a push undefined is generated by the compiler and is pushed to the stack. The undefined return value should be popped off the stack.

For all of the call actions (ActionCallMethod, ActionNewMethod, ActionNewObject, and ActionCallFunction) and initialization actions (ActionInitObject and ActionInitArray), the arguments of the function are pushed onto the stack in reverse order, with the rightmost argument first and the leftmost argument last. The arguments are subsequently popped off in order (first to last).
ActionDefineFunction defines a function with a given name and body size.

ActionDefineFunction parses (in order) FunctionName, NumParams, [param1, param2, …, param N] and then code size.

After the DefineFunction tag, the next codeSize bytes of action data are considered to be the body of the function.

4. Gets the code for the function.

ActionDefineFunction can be used in the following ways:

Usage 1
Pushes an anonymous function on the stack that does not persist. This function is a function literal that is declared in an expression instead of a statement. An anonymous function can be used to define a function, return its value, and assign it to a variable in one expression, as in the following ActionScript:

area = (function () {return Math.PI * radius * radius;})(5);

NOTE ActionDefineFunction is rarely used as of SWF 7 and later; it was superseded by ActionDefineFunction2.
**Function Definition**

Sets a variable with a given `FunctionName` and a given function definition. This is the more conventional function definition. For example, in ActionScript:

```javascript
function Circle(radius) {
    this.radius = radius;
    this.area = Math.PI * radius * radius;
}
```

### ActionDefineLocal

**Definition:**

defines a local variable and sets its value. If the variable already exists, the value is set to the newly specified value.

**Action:**

```
UI8 Action = 0x3C
```

### ActionDefineLocal2

**Definition:**

defines a local variable without setting its value. If the variable already exists, nothing happens. The initial value of the local variable is `undefined`.

**Action:**

```
UI8 Action = 0x41
```

### ActionDelete

**Definition:**

deletes a named property from a ScriptObject.

**Action:**

```
UI8 Action = 0x3A
```

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI8</td>
<td>Action = 0x3C</td>
</tr>
<tr>
<td>UI8</td>
<td>Action = 0x41</td>
</tr>
<tr>
<td>UI8</td>
<td>Action = 0x3A</td>
</tr>
</tbody>
</table>

---

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

---

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
ActionDelete2 deletes a named property. Flash Player first looks for the property in the current scope, and if the property cannot be found, continues to search in the encompassing scopes.

ActionDelete2 pops the name of the property to delete off the stack.

ActionEnumerate obtains the names of all "slots" in use in an ActionScript object—that is, for an object `obj`, all names `X` that could be retrieved with the syntax `obj.X`. ActionEnumerate is used to implement the `for..in` statement in ActionScript.

ActionEnumerate does the following:
1. Pops the name of the object variable (which can include slash-path or dot-path syntax) off of the stack.
2. Pushes a null value onto the stack to indicate the end of the slot names.
3. Pushes each slot name (a string) onto the stack.

ActionEquals2 is similar to ActionEquals, but ActionEquals2 knows about types. The equality comparison algorithm from ECMA-262 Section 11.9.3 is applied.

### Field Type Comment

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionDelete2</td>
<td>UI8</td>
<td>Action = 0x3B</td>
</tr>
<tr>
<td>ActionEnumerate</td>
<td>UI8</td>
<td>Action = 0x46</td>
</tr>
<tr>
<td>ActionEquals2</td>
<td>UI8</td>
<td>Action = 0x49</td>
</tr>
</tbody>
</table>

**NOTE**

Certain special slot names are omitted; for a list of these, search for the term `DontEnum` in the ECMA-262 standard.
112 Actions

ActionEquals2 does the following:
1. Pops arg1 off the stack.
2. Pops arg2 off the stack.
3. Pushes the return value to the stack.

ActionGetMember

ActionGetMember retrieves a named property from an object, and pushes the value of the property onto the stack.

ActionGetMember does the following:
1. Pops the name of the member function.
2. Pops the ScriptObject object off of the stack.
3. Pushes the value of the property on to the stack.

For example, assume obj is an object, and it is assigned a property, foo, as follows:

```javascript
obj.foo = 3;
```

Then, ActionGetMember with object set to obj and name set to foo pushes 3 onto the stack.

If the specified property does not exist, undefined is pushed to the stack.

The object parameter cannot actually be of type Object. If the object parameter is a primitive type such as number, Boolean, or string, it is converted automatically to a temporary wrapper object of class Number, Boolean, or String. Thus, methods of wrapper objects can be invoked on values of primitive types.

For example, the following correctly prints 5:

```javascript
var x = "Hello";
trace (x.length);
```

In this case, the variable, x, contains the primitive string, "Hello". When x.length is retrieved, a temporary wrapper object for x is created by using the String type, which has a length property.
ActionInitArray initializes an array in a ScriptObject and is similar to ActionInitObject. The newly created object is pushed to the stack. The stack is the only existing reference to the newly created object. A subsequent SetVariable or SetMember action can store the newly created object in a variable.

ActionInitArray pops elems and then \[\text{arg1, arg2, …, argn}\] off the stack.

ActionInitArray does the following:
1. Gets the number of arguments (or elements) from the stack.
2. If arguments are present, ActionInitArray initializes an array object with the right number of elements.
3. Initializes the array as a ScriptObject.
4. Sets the object type to \text{Array}.
5. Populates the array with initial elements by popping the values off of the stack.

For all of the call actions (ActionCallMethod, ActionNewMethod, ActionNewObject, and ActionCallFunction) and initialization actions (ActionInitObject and ActionInitArray), the arguments of the function are pushed onto the stack in reverse order, with the rightmost argument first and the leftmost argument last. The arguments are subsequently popped off in order (first to last).

ActionInitObject initializes an object and is similar to ActionInitArray. The newly created object is pushed to the stack. The stack is the only existing reference to the newly created object. A subsequent SetVariable or SetMember action can store the newly created object in a variable.

ActionInitObject pops \text{elems} off of the stack.

### Field Type Comment

**ActionInitArray**
- **Field Type**: UI8
- **Comment**: Action = 0x42

**ActionInitObject**
- **Field Type**: UI8
- **Comment**: Action = 0x43

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
114 Actions

ActionInitObject does the following:

1. Pops the number of initial properties from the stack.
2. Initializes the object as a ScriptObject.
3. Sets the object type to Object.
4. Pops each initial property off the stack.

For each initial property, the value of the property is popped off the stack, then the name of the property is popped off the stack. The name of the property is converted to a string. The value can be of any type.

For all of the call actions (ActionCallMethod, ActionNewMethod, ActionNewObject, and ActionCallFunction) and initialization actions (ActionInitObject and ActionInitArray), the arguments of the function are pushed onto the stack in reverse order, with the rightmost argument first and the leftmost argument last. The arguments are subsequently popped off in order (first to last).

ActionNewMethod

ActionNewMethod invokes a constructor function to create a new object. A new object is constructed and passed to the constructor function as the value of the this keyword. Arguments can be specified to the constructor function. The return value from the constructor function is discarded. The newly constructed object is pushed to the stack, similar to ActionCallMethod and ActionNewObject.

ActionNewMethod does the following:

1. Pops the name of the method from the stack.
2. Pops the ScriptObject from the stack.

If the name of the method is blank, the ScriptObject is treated as a function object that is invoked as the constructor function. If the method name is not blank, the named method of the ScriptObject is invoked.

3. Pops the number of arguments from the stack.
4. Executes the method call.
5. Pushes the newly constructed object to the stack.

If no appropriate return value occurs (for instance, the function does not have a return statement), the compiler generates a push undefined and pushes it to the stack. The undefined return value should be popped off the stack.

Field Type Comment

ActionNewMethod UI8 Action = 0x53

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
For all of the call actions (ActionCallMethod, ActionNewMethod, ActionNewObject, and ActionCallFunction) and initialization actions (ActionInitObject and ActionInitArray), the arguments of the function are pushed onto the stack in reverse order, with the rightmost argument first and the leftmost argument last. The arguments are subsequently popped off in order (first to last).

**ActionNewObject**

invokes a constructor function. A new object is created and passed to the **this** keyword. In addition, arguments can optionally be specified to the constructor function on the stack. The return value of the constructor function is discarded. The newly constructed object is pushed to the stack. ActionNewObject is similar to ActionCallFunction and ActionNewMethod.

ActionNewObject does the following:

1. Pops the object name (STRING) **this** from the stack.
2. Pops numArgs (int) from the stack.
3. Pops the arguments off the stack.
4. Invokes the named object as a constructor function, passing it the specified arguments and a newly constructed object as the **this** keyword.
5. The return value of the constructor function is discarded.
6. The newly constructed object is pushed to the stack.
116 Actions

ActionSetMember does the following:

1. Pops the new value off the stack.
2. Pops the object name off the stack.
3. Pops the object off of the stack.

ActionTargetPath

If the object in the stack is of type *MovieClip*, the object's target path is pushed on the stack in dot notation. If the object is not a *MovieClip*, the result is *undefined* rather than the movie clip target path.

ActionTargetPath does the following:

1. Pops the object off the stack.
2. Pushes the target path onto the stack.

ActionWith

Defines a With block of script.

ActionWith does the following:

1. Pops the object involved with the With.
2. Parses the size (body length) of the With block from the ActionWith tag.
3. Checks to see if the depth of calls exceeds the maximum depth, which is 16 for SWF 6 and later, and 8 for SWF 5.
   
   If the With depth exceeds the maximum depth, the next Size bytes of data are skipped rather than executed.

4. After the ActionWith tag, the next Size bytes of action codes are considered to be the body of the With block.
5. Adds the With block to the scope chain.

Field Type Comment

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI8</td>
<td>Action = 0x45</td>
</tr>
<tr>
<td>UI8</td>
<td>Action = 0x94</td>
</tr>
<tr>
<td>UI16</td>
<td># of bytes of code that follow</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Type actions

ActionToNumber

Converts the object on the top of the stack into a number, and pushes the number back to the stack.

For the Object type, the ActionScript valueOf() method is invoked to convert the object to a Number type for ActionToNumber. Conversions between primitive types, such as from String to Number, are built-in.

ActionToNumber does the following:

1. Pops the object off of the stack.
2. Pushes the number on to the stack.

ActionToString

ActionToString converts the object on the top of the stack into a String, and pushes the string back to the stack.

For the Object type, the ActionScript toString() method is invoked to convert the object to the String type for ActionToString.

ActionToString does the following:

1. Pops the object off of the stack.
2. Pushes the string on to the stack.

Field Type Comment

<table>
<thead>
<tr>
<th>ActionToNumber</th>
<th>UI8</th>
<th>Action = 0x4A</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>ActionToString</th>
<th>UI8</th>
<th>Action = 0x4B</th>
</tr>
</thead>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
118 Actions

ActionTypeOf

ActionTypeOf

pushes the object type to the stack, which is equivalent to the ActionScript TypeOf() method. The possible types are:

"number"
"boolean"
"string"
"object"
"movieclip"
"null"
"undefined"
"function"

ActionTypeOf does the following:

1. Pops the value to determine the type of off the stack.
2. Pushes a string with the type of the object on to the stack.

Math actions

ActionAdd2

ActionAdd2 is similar to ActionAdd, but performs the addition differently, according to the data types of the arguments. The addition operator algorithm in ECMA-262 Section 11.6.1 is used. If string concatenation is applied, the concatenated string is arg2 followed by arg1.

ActionAdd2 does the following:

1. Pops arg1 off of the stack.
2. Pops arg2 off of the stack.
3. Pushes the result back to the stack.
ActionLess2 calculates whether arg1 is less than arg2 and pushes a Boolean return value to the stack. This action is similar to ActionLess, but performs the comparison differently according to the data types of the arguments. The abstract relational comparison algorithm in ECMA-262 Section 11.8.5 is used.

ActionLess2 does the following:
1. Pops arg1 off of the stack.
2. Pops arg2 off of the stack.
4. Pushes the return value (a Boolean value) onto the stack.

ActionModulo calculates x modulo y. If y is 0, then NaN (0x7FC00000) is pushed to the stack.

ActionModulo does the following:
1. Pops x then y off of the stack.
2. Pushes the value \(x \% y\) on to the stack.

Stack operator actions
ActionBitAnd pops two numbers off of the stack, performs a bitwise AND, and pushes an S32 number to the stack. The arguments are converted to 32-bit unsigned integers before performing the bitwise operation. The result is a SIGNED 32-bit integer.
120 Actions

ActionBitAnd does the following:
1. Pops arg1 then arg2 off of the stack.
2. Pushes the result to the stack.

ActionBitLShift pops the shift count arg and then value off of the stack. The value argument is converted to 32-bit signed integer and only the least significant 5 bits are used as the shift count. The bits in the value arg are shifted to the left by the shift count. ActionBitLShift pushes an S32 number to the stack.

ActionBitOr does the following:
1. Pops arg1 then arg2 off of the stack.
2. Pushes the result to the stack.

ActionBitOr pops two numbers off of the stack, performs a bitwise OR, and pushes an S32 number to the stack. The arguments are converted to 32-bit unsigned integers before performing the bitwise operation. The result is a SIGNED 32-bit integer.

Field Type Comment

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
With Flash 5, eight new types were added to ActionPush. For more on ActionPush, see the SWF 4 actions.

**ActionPushDuplicate**

ActionPushDuplicate pushes a duplicate of top of stack (the current return value) to the stack.

**ActionReturn**

ActionReturn forces the return item to be pushed off the stack and returned. If a return is not appropriate, the return item is discarded.

**ActionStackSwap**

ActionStackSwap does the following:

1. Pops Item1 and then Item2 off of the stack.
2. Pushes Item2 and then Item1 back to the stack.

**ActionStoreRegister**

ActionStoreRegister reads the next object from the stack (without popping it) and stores it in one of four registers. If ActionDefineFunction2 is used, up to 256 registers are available.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionPushDuplicate</td>
<td>UI8</td>
</tr>
<tr>
<td>ActionReturn</td>
<td>UI8</td>
</tr>
<tr>
<td>ActionStackSwap</td>
<td>UI8</td>
</tr>
<tr>
<td>ActionStoreRegister</td>
<td>UI8</td>
</tr>
</tbody>
</table>

**RegisterNumber**

A separate section in the document provides a table comparing the different types of ActionPush.

---

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

---

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
ActionStoreRegister parses register number from the StoreRegister tag.

SWF 6 action model

SWF 6 adds the DoInitAction action-definition tag, and a few new action bytecodes.

SWF 6 actions

The following actions are available in SWF 6:

- DoInitAction
- ActionInstanceOf
- ActionEnumerate2
- ActionStrictEquals
- ActionGreater
- ActionStringGreater

DoInitAction

The DoInitAction tag is similar to the DoAction tag: it defines a series of bytecodes to be executed. However, the actions defined with DoInitAction are executed earlier than the usual DoAction actions, and are executed only once.

In some situations, actions must be executed before the ActionScript representation of the first instance of a particular sprite is created. The most common such action is calling Object.registerClass to associate an ActionScript class with a sprite. Such a call is generally found within the #initclip pragma in the ActionScript language. DoInitAction is used to implement the #initclip pragma.

A DoInitAction tag specifies a particular sprite to which its actions apply. A single frame can contain multiple DoInitAction tags; their actions are executed in the order in which the tags appear. However, the SWF file can contain only one DoInitAction tag for any particular sprite.
The specified actions are executed immediately before the normal actions of the frame in which the DoInitAction tag appears. This only occurs the first time that this frame is encountered; playback reaches the same frame again later, actions provided in DoInitAction are skipped.

ActionInstanceOf implements the ActionScript `instanceof()` operator. This is a Boolean operator that indicates whether the left operand (typically an object) is an instance of the class represented by a constructor function passed as the right operand. Additionally, with SWF 7 or later, ActionInstanceOf also supports with interfaces. If the right operand constructor is a reference to an interface object, and the left operand implements this interface, ActionInstanceOf accurately reports that the left operand is an instance of the right operand.

ActionInstanceOf does the following:

1. Pops constr then obj off of the stack.
2. Determines if obj is an instance of constr.
3. Pushes the return value (a Boolean value) onto the stack.

NOTE

Specifying actions at the beginning of a DoAction tag is not the same as specifying them in a DoInitAction tag. Flash Player takes steps before the first action in a DoAction tag, most relevantly the creation of ActionScript objects that represent sprites. The actions in DoInitAction occur before these implicit steps are performed.
ActionEnumerate2 is similar to ActionEnumerate, but uses a stack argument of object type rather than using a string to specify its name.

ActionEnumerate2 does the following:
1. Pops obj off of the stack.
2. Pushes a null value onto the stack to indicate the end of the slot names.
3. Pushes each slot name (a string) from obj onto the stack.

ActionStrictEquals is similar to ActionEquals2, but the two arguments must be of the same type in order to be considered equal. Implements the '===' operator from the ActionScript language.

ActionStrictEquals does the following:
1. Pops arg1 then arg2 off the stack.
2. Pushes the return value, a Boolean value, to the stack.

ActionGreater is the exact opposite of ActionLess2. Originally there was no ActionGreater, because it can be emulated by reversing the order of argument pushing, then performing an ActionLess followed by an ActionNot. However, this argument reversal resulted in a reversal of the usual order of evaluation of arguments, which in a few cases led to surprises.

Field Type Comment
ActionEnumerate2 UI8 Action = 0x55
ActionStrictEquals UI8 Action = 0x66
ActionGreater UI8 Action = 0x67

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.
ActionGreater does the following:
1. Pops arg1 and then arg2 off of the stack.
2. Compares if arg2 > arg1.
3. Pushes the return value, a Boolean value, onto the stack.

ActionStringGreater
ActionStringGreater is the exact opposite of ActionGreater. This action code was added for the same reasons as ActionGreater.

ActionStringGreater does the following:
1. Pops arg1 and then arg2 off of the stack.
2. Compares if arg2 > arg1, using byte-by-byte comparison.
3. Pushes the return value, a Boolean value, onto the stack.

SWF 7 actions
The following actions are available in SWF 7:
■ ActionDefineFunction2
■ ActionExtends
■ ActionCastOp
■ ActionImplementsOp
■ ActionTry
■ ActionThrow
ActionDefineFunction2 is similar to ActionDefineFunction, with additional features that can help speed up the execution of function calls by preventing the creation of unused variables in the function's activation object and by enabling the replacement of local variables with a variable number of registers. With ActionDefineFunction2, a function can allocate its own private set of up to 256 registers. Parameters or local variables can be replaced with a register, which is loaded with the value instead of the value being stored in the function's activation object. (The activation object is an implicit local scope that contains named arguments and local variables. For further description of the activation object, see the ECMA-262 standard.)

ActionDefineFunction2 also includes six flags to instruct Flash Player to preload variables, and three flags to suppress variables. By setting PreloadParentFlag, PreloadRootFlag, PreloadSuperFlag, PreloadArgumentsFlag, PreloadThisFlag, or PreloadGlobalFlag, common variables can be preloaded into registers before the function executes (\_parent, \_root, super, arguments, this, or \_global, respectively). With flags SuppressSuper, SuppressArguments, and SuppressThis, common variables super, arguments, and this are not created. By using suppress flags, Flash Player avoids pre-evaluating variables, thus saving time and improving performance.

No suppress flags are provided for \_parent, \_root, or \_global because Flash Player always evaluates these variables as needed; no time is ever wasted on pre-evaluating these variables. Specifying both the preload flag and the suppress flag for any variable is not allowed.

The body of the function that ActionDefineFunction2 specifies should use ActionPush and ActionStoreRegister for local variables that are assigned to registers. ActionGetVariable and ActionSetVariable cannot be used for variables assigned to registers.

Flash Player 6 release 65 and later supports ActionDefineFunction2.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.
REGISTERPARAM is defined as follows:

The function body following an ActionDefineFunction2 consists of further action codes, just as for ActionDefineFunction.

Flash Player selects register numbers by first copying each argument into the register specified in the corresponding REGISTERPARAM record. Next, the preloaded variables are copied into registers starting at 1, and in the order this, arguments, super, _root, _parent, and _global, skipping any that are not to be preloaded. (The SWF file must accurately specify which registers will be used by preloaded variables and ensure that no parameter uses a register number that falls within this range, or else that parameter is overwritten by a preloaded variable.)

The value of NumParams should equal the number of parameter registers. The value of RegisterCount should equal NumParams plus the number of preloaded variables and the number of local variable registers desired.

For example, if NumParams is 2, RegisterCount is 6, PreloadThisFlag is 1, and PreloadRootFlag is 1, the REGISTERPARAM records will probably specify registers 3 and 4. Register 1 will be this, register 2 will be _root, registers 3 and 4 will be the first and second parameters, and registers 5 and 6 will be for local variables.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>UI8</td>
<td>For each parameter to the function, a register can be specified. If the register specified is zero, the parameter is created as a variable named ParamName in the activation object, which can be referenced with ActionGetVariable and ActionSetVariable. If the register specified is nonzero, the parameter is copied into the register, and it can be referenced with ActionPush and ActionStoreRegister, and no variable is created in the activation object.</td>
</tr>
<tr>
<td>ParamName</td>
<td>STRING</td>
<td>Parameter name</td>
</tr>
</tbody>
</table>
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
132 Actions

ActionCastOp does the following:
1. Pops the ScriptObject to cast off the stack.
2. Pops the constructor function off the stack.
3. Determines if object is an instance of constructor (doing the same comparison as ActionInstanceOf).
4. If the object is an instance of constructor, the popped ScriptObject is pushed onto the stack.
   If the object is not an instance of constructor, a null value is pushed onto the stack.

ActionImplementsOp

ActionImplementsOp implements the ActionScript `implements` keyword. The ActionImplementsOp action specifies the interfaces that a class implements, for use by ActionCastOp. ActionImplementsOp can also specify the interfaces that an interface implements, as interfaces can extend other interfaces.

ActionImplementsOp does the following:
1. Pops the constructor function off the stack.
   - The constructor function represents the class that will implement the interfaces. The constructor function must have a prototype property.
2. Pops the count of implemented interfaces off the stack.
3. For each interface count, pops a constructor function off of the stack.
   - The constructor function represents an interface.
4. Sets the constructor function's list of interfaces to the array collected in the previous step, and sets the count of interfaces to the count popped in step 2.
ActionTry defines handlers for exceptional conditions, implementing the ActionScript `try`, `catch`, and `finally` keywords.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionTry</td>
<td>UI8</td>
<td>Action = 0x8F</td>
</tr>
<tr>
<td>CatchInRegisterFlag</td>
<td>UB[1]</td>
<td>0 - Do not put caught object into register (instead, store in named variable)</td>
</tr>
<tr>
<td>1 - Put caught object into register (do not store in named variable)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FinallyBlockFlag</td>
<td>UB[1]</td>
<td>0 - No finally block</td>
</tr>
<tr>
<td>1 - Has finally block</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CatchBlockFlag</td>
<td>UB[1]</td>
<td>0 - No catch block</td>
</tr>
<tr>
<td>1 - Has catch block</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TrySize</td>
<td>UI16</td>
<td>Length of the try block</td>
</tr>
<tr>
<td>CatchSize</td>
<td>UI16</td>
<td>Length of the catch block</td>
</tr>
<tr>
<td>FinallySize</td>
<td>UI16</td>
<td>Length of the finally block</td>
</tr>
<tr>
<td>CatchName</td>
<td>STRING</td>
<td>Name of the catch variable</td>
</tr>
<tr>
<td>CatchRegister</td>
<td>UI8</td>
<td>If CatchInRegisterFlag = 1, register to catch into</td>
</tr>
<tr>
<td>TryBody</td>
<td>UI8[TrySize]</td>
<td>Body of the try block</td>
</tr>
<tr>
<td>CatchBody</td>
<td>UI8[CatchSize]</td>
<td>Body of the catch block, if any</td>
</tr>
<tr>
<td>FinallyBody</td>
<td>UI8[FinallySize]</td>
<td>Body of the finally block, if any</td>
</tr>
</tbody>
</table>

NOTE: The CatchSize and FinallySize fields always exist, whether or not the CatchBlockFlag or FinallyBlockFlag settings are 1.

NOTE: The try, catch, and finally blocks do not use end tags to mark the end of their respective blocks. Instead, the length of a block is set by the TrySize, CatchSize, and FinallySize values.
ActionThrow implements the ActionScript `throw` keyword. ActionThrow is used to signal, or throw, an exceptional condition, which is handled by the exception handlers declared with `ActionTry`. If any code within the `try` block throws an object, control passes to the `catch` block, if one exists, then to the `finally` block, if one exists. The `finally` block always executes, regardless of whether an error was thrown.

If an exceptional condition occurs within a function and the function does not include a `catch` handler, the function and any caller functions are exited until a `catch` block is found (executing all `finally` handlers at all levels).

Any ActionScript data type can be thrown, though typically usage is to throw objects. ActionThrow pops the value to be thrown off the stack.
CHAPTER 8
Shapes

The Macromedia Flash (SWF) shape architecture is designed to be compact, flexible and rendered very quickly to the screen. It is similar to most vector formats in that shapes are defined by a list of edges called a path. A path may be closed, where the start and end of the path meet to close the figure, or open, where the path forms an open-ended stroke. A path may contain a mixture of straight edges, curved edges, and 'pen up and move' commands. This allows multiple disconnected figures to be described by a single shape structure.

A fill style defines the appearance of an area enclosed by a path. Fill styles supported by the SWF file format include a color, a gradient, or a bitmap image.

A line style defines the appearance of the outline of a path. The line style may be a stroke of any thickness and color.

Most vector formats allow only one fill and line style per path. The SWF file format extends this concept by allowing each edge to have its own line and fill style. This can have unpredictable results when fill styles change in the middle of a path.

Flash also supports two fill styles per edge, one for each side of the edge: FillStyle0 and FillStyle1. FillStyle0 should always be used first and then FillStyle1 if the shape is filled on both sides of the edge.

Shape overview

A shape is composed of the following elements:

- **CharacterId**: A 16-bit value that uniquely identifies this shape as a 'character' in the dictionary. The CharacterId can be referred to in control tags such as PlaceObject. Characters can be reused and combined with other characters to make more complex shapes.

- **Bounding box**: The rectangle that completely encloses the shape.

- **Fill style array**: A list of all the fill styles used in a shape.

- **Line style array**: A list of all the line styles used in a shape.
136 Shapes

Shape record array

A list of shape records. Shape records can define straight or curved edges, style changes, or move the drawing position.

Shape example

The following example appears to be a collection of shapes, but it can be described with a single DefineShape tag.

The red circle, red square and green rounded-rectangle are closed paths. The curved line is an open path. The red square consists of all straight edges, the red circle consists of all curved edges, while the rounded rectangle has curved edges interspersed with straight edges.

There are two fill styles, solid green and solid red, and two line styles, 1-pixel black, and 2-pixel black. The red circle and red square share the same fill and line styles. The rounded rectangle and curved line share the same line style.

Here's how to describe this example with the SWF file format.

Define the fill styles:

1. First, the fill styles are defined with a FILLSTYLEARRAY. The two unique fill styles are solid red and solid green.
2. This is followed by a LINESTYLEARRAY that includes the two unique line styles: 1-pixel black, and 2-pixel black.
3. This is followed by an array of shape records (see Shape records).

NOTE

Line and fill styles are defined only once and may be used (and reused) by any of the edges in the shape.
All shape records share a similar structure but can have varied meaning. A shape record can define straight or curved edge, a style change, or it can move the current drawing position.

Define the curved line:
1. The first shape record selects the 2-pixel-wide line style, and moves the drawing position to the top of the curved line by setting the StateMoveTo flag.
2. The next shape record is a curved edge, which ends to the bottom of the line. The path is not closed.

Define the red square:
1. The next shape record selects the 1-pixel line style and the red fill style. It also moves the drawing position to the upper-left corner of the red rectangle.
2. The following four shape records are straight edges. The last edge must end at the upper-left corner. Flash requires that closed figures be joined explicitly. That is, the first and last points must be coincident.

Define the red circle:
1. The next shape record does not change any style settings, but moves the drawing position to the top of the red circle.
2. The following eight shape records are curved edges that define the circle. Again, the path must finish where it started.

Define the green rounded-rectangle:
1. The next shape record selects the 2-pixel-wide line style, and the green fill. It also moves the drawing position to the upper left of the rounded-rectangle.
2. The following twelve shape records are a mixture of straight shape records (the sides) interspersed with curved shape records (the rounded corners). The path finishes where it began.
A gradient fill can be either a linear or a radial gradient. For an in-depth description of how gradients are defined, see Gradients.

Bitmap fills refer to a bitmap characterID. There are two styles: clipped and tiled. A clipped bitmap fill repeats the color on the edge of a bitmap if the fill extends beyond the edge of the bitmap. A tiled fill repeats the bitmap if the fill extends beyond the edge of the bitmap.

A fill style array enumerates a number of fill styles. The format of a fill style array is described in the following table:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FillStyleCount</td>
<td>UI8</td>
<td>Count of fill styles.</td>
</tr>
<tr>
<td>FillStyleCountExtended</td>
<td>UI16</td>
<td>Supported only for Shape2 and Shape3.</td>
</tr>
<tr>
<td>FillStyles</td>
<td>FILLSTYLE[FillStyleCount]</td>
<td>Array of fill styles.</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The format of a fill style value within the file is described in the following table:

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FillStyleType</td>
<td>Type of fill style:</td>
</tr>
<tr>
<td></td>
<td>0x00 = solid fill</td>
</tr>
<tr>
<td></td>
<td>0x10 = linear gradient fill</td>
</tr>
<tr>
<td></td>
<td>0x12 = radial gradient fill</td>
</tr>
<tr>
<td></td>
<td>0x13 = focal radial gradient fill</td>
</tr>
<tr>
<td></td>
<td>(SWF 8 file format and later only)</td>
</tr>
<tr>
<td></td>
<td>0x40 = repeating bitmap fill</td>
</tr>
<tr>
<td></td>
<td>0x41 = clipped bitmap fill</td>
</tr>
<tr>
<td></td>
<td>0x42 = non-smoothed repeating bitmap</td>
</tr>
<tr>
<td></td>
<td>0x43 = non-smoothed clipped bitmap</td>
</tr>
<tr>
<td>Color</td>
<td>Solid fill color with transparency</td>
</tr>
<tr>
<td>GradientMatrix</td>
<td>Matrix for gradient fill</td>
</tr>
<tr>
<td>Gradient</td>
<td>Gradient fill</td>
</tr>
<tr>
<td>BitmapId</td>
<td>ID of bitmap character for fill</td>
</tr>
<tr>
<td>BitmapMatrix</td>
<td>Matrix for bitmap fill</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
A line style array enumerates a number of line styles.

**LINESTYLEARRAY**

The format of a line style array is described in the following table:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LineStyleCount</td>
<td>UI8</td>
<td>Count of line styles.</td>
</tr>
<tr>
<td>LineStyleCountExtended</td>
<td>UI16</td>
<td>Extended count of line styles.</td>
</tr>
<tr>
<td>LineStyles</td>
<td>LINESTYLE[count] (if Shape1, Shape2, or Shape3) LINESTYLE2[count] (if Shape4)</td>
<td>Array of line styles.</td>
</tr>
</tbody>
</table>

**LINESTYLE**

Field | Type | Comment |
---|------|---------|
| Width | UI16 | Width of line in twips. |
| Color | RGB (Shape1 or Shape2) RGBA (Shape3) | Color value including alpha channel information for Shape3. |

**NOTE**

Before the introduction of LINESTYLE2 in SWF 8, all lines in the SWF file format have rounded joins and round caps. Different join styles and end styles can be simulated with a very narrow shape that looks identical to the desired stroke.

**NOTE**

The SWF file format has no native support for dashed or dotted line styles. A dashed line can be simulated by breaking up the path into a series of short lines.
LINESTYLE2 builds upon the capabilities of the LINESTYLE record by allowing the use of new types of joins and caps as well as scaling options and the ability to fill a stroke. In order to use LINESTYLE2, the shape must be defined with DefineShape4—not DefineShape2, DefineShape3.

While the LINESTYLE record permits only rounded joins and round caps, LINESTYLE2 also supports miter and bevel joins, and square caps and no caps. The following diagram illustrates the complete array of joins and caps:

When using LINESTYLE2 for a miter join, a MiterLimitFactor must be specified and is used to calculate the maximum miter length:

Maximum miter length = LINESTYLE2 MiterLimitFactor * LINESTYLE2 Width

If the miter join exceeds the maximum miter length, Flash Player will cut off the miter. Note that MiterLimitFactor is an 8.8 fixed-point value.

LINESTYLE2 also includes the option for pixel hinting to correct blurry vertical or horizontal lines.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The SHAPE structure defines a shape without fill style or line style information.

SHAPE is used by the DefineFont tag, to define character glyphs.

SHAPEWITHSTYLE extends the SHAPE structure by including fill style and line style information. SHAPEWITHSTYLE is used by the DefineShape tag.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumLineBits</td>
<td>UB[4]</td>
<td>Number of line index bits.</td>
</tr>
<tr>
<td>ShapeRecords</td>
<td>SHAPERECORD</td>
<td>Shape records (see following).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FillStyles</td>
<td>FILLSTYLEARRAY</td>
<td>Array of fill styles.</td>
</tr>
<tr>
<td>LineStyles</td>
<td>LINESTYLEARRAY</td>
<td>Array of line styles.</td>
</tr>
<tr>
<td>NumLineBits</td>
<td>UB[4]</td>
<td>Number of line index bits.</td>
</tr>
<tr>
<td>ShapeRecords</td>
<td>SHAPERECORD</td>
<td>Shape records (see following).</td>
</tr>
</tbody>
</table>

NOTE: The LINESTYLEARRAY and FILLSTYLEARRAY begin at index 1, not index 0.
The following diagram illustrates the SHAPEWITHSTYLE structure. First, the Fill styles and Line styles are defined. These are defined only once and are referred to by array index. The blue area represents the array of Shape records. The first shape record selects a fill from the fill style array, and moves the drawing position to the start of the shape. This is followed by a series of edge records that define the shape. The next record changes the fill style, and the subsequent edge records are filled using this new style.

This tag is a completely autonomous object. The style change records only refer to fill and line styles that have been defined in this tag.

Shape records

There are four types of shape records:

- End shape record
- Style change record
- Straight edge record
- Curved edge record

All shape records are byte aligned and begin with a TypeFlag. If the TypeFlag is zero, the shape record is a non-edge record, and a further five bits of flag information follow.
Shape structures

EndShapeRecord

The end shape record simply indicates the end of the shape record array. It is a non-edge record with all five flags equal to zero.

StyleChangeRecord

The style change record is also a non-edge record. It can be used to do the following:

1. Select a fill or line style for drawing.
2. Move the current drawing position (without drawing).
3. Replace the current fill and line style arrays with a new set of styles.

Because fill and line styles often change at the start of a new path, it is useful to perform more than one action in a single record. For example, say a DefineShape tag defines a red circle and a blue square. After the circle is closed, it is necessary to move the drawing position, and replace the red fill with the blue fill. The style change record can achieve this with a single shape record.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TypeFlag</td>
<td>UB[1]</td>
<td>Non-edge record flag. Always 0.</td>
</tr>
</tbody>
</table>

STYLECHANGERECORD

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TypeFlag</td>
<td>UB[1]</td>
<td>Non-edge record flag. Always 0.</td>
</tr>
<tr>
<td>StateNewStyles</td>
<td>UB[1]</td>
<td>New styles flag. Used by DefineShape2 and DefineShape3 only.</td>
</tr>
<tr>
<td>StateLineStyle</td>
<td>UB[1]</td>
<td>Line style change flag.</td>
</tr>
<tr>
<td>StateFillStyle1</td>
<td>UB[1]</td>
<td>Fill style 1 change flag.</td>
</tr>
<tr>
<td>StateFillStyle0</td>
<td>UB[1]</td>
<td>Fill style 0 change flag.</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
In the first shape record, MoveDeltaX and MoveDeltaY are relative to the shape origin. In subsequent shape records, MoveDeltaX and MoveDeltaY are relative to the current drawing position.

The style arrays begin at index 1, not index 0. FillStyle = 1 refers to the first style in the fill style array, FillStyle = 2 refers to the second style in the fill style array, and so on. A fill style index of zero means the path is not filled, and a line style index of zero means the path has no stroke. Initially the fill and line style indices are set to zero—no fill or stroke.

FillStyle0 and FillStyle1 Flash supports two fill styles per edge, one for each side of the edge: FillStyle0 and FillStyle1. For shapes that don’t self-intersect or overlap, FillStyle0 should be used. For overlapping shapes the situation is more complex.
For example, if a shape consists of two overlapping squares, and only FillStyle0 is defined, Flash Player renders a ‘hole’ where the paths overlap. This area can be filled using FillStyle1. In this situation, the rule is that for any directed vector, FillStyle0 is the color to the left of the vector, and FillStyle1 is the color to the right of the vector (as shown in the following diagram).

NOTE

FillStyle0 and FillStyle1 should not be confused with FILLSTYLEARRAY indices. FillStyle0 and FillStyle1 are variables that contain indices into the FILLSTYLEARRAY.
Edge records have a TypeFlag of 1. There are two types of edge records: straight and curved. The StraightFlag determines the type.

**StraightEdgeRecord**

The StraightEdgeRecord stores the edge as an $X$-$Y$ delta. The delta is added to the current drawing position, and this becomes the new drawing position. The edge is rendered between the old and new drawing positions.

Straight edge records support three types of lines:

1. General lines.
2. Horizontal lines.
3. Vertical lines.

General lines store both $X$ and $Y$ deltas, the horizontal and vertical lines store only the $X$ delta and $Y$ delta respectively.

---

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

---

**Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8**
The SWF file format differs from most vector file formats by using Quadratic Bezier curves rather than Cubic Bezier curves. PostScript uses Cubic Bezier curves, as do most drawing applications. The SWF file format uses Quadratic Bezier curves because they can be stored more compactly, and can be rendered more efficiently.

The following diagram shows a Quadratic Bezier curve and a Cubic Bezier curve. A Quadratic Bezier curve has 3 points: 2 on-curve anchor points, and 1 off-curve control point. A Cubic Bezier curve has 4 points: 2 on-curve anchor points, and 2 off-curve control points.

The curved-edge record stores the edge as two X-Y deltas. The three points that define the Quadratic Bezier are calculated like this:

1. The first anchor point is the current drawing position.
2. The control point is the current drawing position + ControlDelta.
3. The last anchor point is the current drawing position + ControlDelta + AnchorDelta.

The last anchor point becomes the current drawing position.

**CURVEDEDGERECORD**

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TypeFlag</td>
<td>This is an edge record. Always 1.</td>
</tr>
<tr>
<td>StraightFlag</td>
<td>Curved edge. Always 0.</td>
</tr>
<tr>
<td>NumBits</td>
<td>Number of bits per value (2 less than the actual number).</td>
</tr>
<tr>
<td>ControlDeltaX</td>
<td>X control point change.</td>
</tr>
<tr>
<td>ControlDeltaY</td>
<td>Y control point change.</td>
</tr>
<tr>
<td>AnchorDeltaX</td>
<td>X anchor point change.</td>
</tr>
<tr>
<td>AnchorDeltaY</td>
<td>Y anchor point change.</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Converting between quadratic and cubic Bezier curves

Replace the single off-curve control point of the quadratic Bezier curve with two new off-curve control points for the cubic Bezier curve. Place each new off-curve control point along the line between one of the on-curve anchor points and the original off-curve control point. The new off-curve control points should be 2/3 of the way from the on-curve anchor point to the original off-curve control point. The diagram of quadratic and cubic Bezier curves above illustrates this substitution.

A cubic Bezier curve can be approximated only with a quadratic Bezier curve, because you are going from a third-order curve to a second-order curve. This involves recursive subdivision of the curve, until the cubic curve and the quadratic equivalent are matched within some arbitrary tolerance.

For a discussion of how to approximate cubic Bezier curves with quadratic Bezier curves see the following:

- Converting Bezier Curves to Quadratic Splines at http://stevehollasch.com/cgindex/curves/cbez-quadspline.html

Shape tags

DefineShape

The DefineShape tag defines a shape for later use by control tags such as PlaceObject. The ShapeId uniquely identifies this shape as 'character' in the Dictionary. The ShapeBounds field is the rectangle that completely encloses the shape. The SHAPEWITHSTYLE structure includes all the paths, fill styles and line styles that make up the shape.

The minimum file format version is SWF 1.
DefineShape2 extends the capabilities of DefineShape with the ability to support more than 255 styles in the style list and multiple style lists in a single shape. The minimum file format version is SWF 2.

DefineShape3 extends the capabilities of DefineShape2 by extending all of the RGB color fields to support RGBA with alpha transparency. The minimum file format version is SWF 3.

DefineShape4 extends the capabilities of DefineShape3 by using a new line style record in the shape. LINESTYLE2 allows new types of joins and caps as well as scaling options and the ability to fill a stroke.

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
152 Shapes

DefineShape4 specifies not only the shape bounds but also the edge bounds of the shape. While the shape bounds are calculated along the outside of the strokes, the edge bounds are taken from the outside of the edges, as shown in the following diagram. The EdgeBounds field assists Flash Player in accurately determining certain layouts.

In addition, DefineShape4 includes new hinting flags UsesNonScalingStrokes and UsesScalingStrokes. These flags assist Flash Player in creating the best possible area for invalidation.

The minimum file format version is SWF 8.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORDHEADER</td>
</tr>
<tr>
<td></td>
<td>Tag type = 83.</td>
</tr>
<tr>
<td>ShapeId</td>
<td>UI16</td>
</tr>
<tr>
<td></td>
<td>ID for this character.</td>
</tr>
<tr>
<td>ShapeBounds</td>
<td>RECT</td>
</tr>
<tr>
<td></td>
<td>Bounds of the shape.</td>
</tr>
<tr>
<td>EdgeBounds</td>
<td>RECT</td>
</tr>
<tr>
<td></td>
<td>Bounds of the shape, excluding strokes.</td>
</tr>
<tr>
<td>Reserved</td>
<td>UB[6]</td>
</tr>
<tr>
<td></td>
<td>Must be 0.</td>
</tr>
<tr>
<td>UsesNonScalingStrokes</td>
<td>UB[1]</td>
</tr>
<tr>
<td></td>
<td>If 1, the shape contains at least one non-scaling stroke.</td>
</tr>
<tr>
<td>UsesScalingStrokes</td>
<td>UB[1]</td>
</tr>
<tr>
<td></td>
<td>If 1, the shape contains at least one scaling stroke.</td>
</tr>
<tr>
<td>Shapes</td>
<td>SHAPEWITHSTYLE</td>
</tr>
<tr>
<td></td>
<td>Shape information.</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Gradients are a special type of shape fill for Macromedia Flash (SWF) shapes. They create ramps of colors that interpolate between two or more fixed colors.

Here is an overview of the Macromedia Flash (SWF) gradient model:

- There are two styles of gradient: Linear and Radial. In addition, with the SWF 8 file format, a new radial gradient type is added to allow the focal point to be set.
- Each gradient has its own transformation matrix, and can be transformed independently of its parent shape.
- A gradient can have up to eight control points in SWF 7 file format and previous versions, or up to fifteen control points in SWF 8 and later. Colors are interpolated between the control points to create the color ramp.
- Each control point is defined by a ratio and an RGBA color. The ratio determines the position of the control point in the gradient; the RGBA value determines its color.

Following are some examples of Flash gradients (from left to right):

- A simple white-to-black linear gradient.
- A simple white-to-black radial gradient.
- A “rainbow” gradient consisting of seven control points; red, yellow, green, cyan, blue, purple, and red.
- A three-point gradient, where the end points are opaque and the center point is transparent. The result is a gradient in the alpha-channel that allows the diamond shape in the background to show through.
Gradients

Gradient transformations

All gradients are defined in a standard space called the gradient square. The gradient square is centered at (0,0), and extends from (-16384,-16384) to (16384,16384).

Each gradient is mapped from the gradient square to the display surface using a standard transformation matrix. This matrix is stored in the FILLSTYLE structure.

Example:

In the following diagram a linear gradient is mapped onto a circle 4096 units in diameter, and centered at (2048,2048).

The 2x3 MATRIX required for this mapping is:

\[
\begin{bmatrix}
0.125 & 0.000 \\
0.000 & 0.125 \\
2048.000 & 2048.000
\end{bmatrix}
\]

The gradient is scaled to one-eighth of its original size (32768 / 4096 = 8), and translated to (2048, 2048).

Gradient control points

The position of a control point in the gradient is determined by a ratio value between 0 and 255. For a linear gradient, a ratio of zero maps to the left side of the gradient square, and a ratio of 255 maps to the right side. For a radial gradient, a ratio of zero maps to the center point of the gradient square, and a ratio of 255 maps to the largest circle that fits inside the gradient square.

The color of a control point is determined by an RGBA value. An alpha value of zero means the gradient is completely transparent at this point. An alpha value of 255 means the gradient is completely opaque at this point.

Control points are sorted by ratio, with the smallest ratio first.
The gradient structures are part of the FILLSTYLE structure.

SWF 8 and later supports up to 15 gradient control points, spread modes and a new interpolation type.

Note that for the DefineShape, DefineShape2 or DefineShape3 tags, the SpreadMode and InterpolationMode fields must be 0, and the NumGradients field can not exceed 8.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpreadMode</td>
<td>UB[2]</td>
<td>0 = Pad mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Reflect mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = Repeat mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 = Reserved</td>
</tr>
<tr>
<td>InterpolationMode</td>
<td>UB[2]</td>
<td>0 = Normal RGB mode interpolation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Linear RGB mode interpolation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 and 3 = Reserved</td>
</tr>
<tr>
<td>NumGradients</td>
<td>UB[4]</td>
<td>1 to 15</td>
</tr>
</tbody>
</table>

GradientRecords GRADRECORD[nGrads]
Gradient records (see following)
FOCALGRADIENT must be declared in DefineShape4—not DefineShape, DefineShape2 or DefineShape3.

The value range is from -1.0 to 1.0, where -1.0 means the focal point is close to the left border of the radial gradient circle, 0.0 means that the focal point is in the center of the radial gradient circle, and 1.0 means that the focal point is close to the right border of the radial gradient circle.

GRADRECORD defines a gradient control point:

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
</table>
| SpreadMode | 0 = Pad mode  
1 = Reflect mode  
2 = Repeat mode  
3 = Reserved |
| InterpolationMode | 0 = Normal RGB mode interpolation  
1 = Linear RGB mode interpolation  
2 and 3 = Reserved |
| NumGradients | 1 to 15 |
| GradientRecords | GRADRECORD[nGrads] |
| FocalPoint | FIXED8 |

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratio</td>
<td>UI8</td>
</tr>
<tr>
<td>Color</td>
<td>RGB (Shape1 or Shape2) RGBA (Shape3)</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
CHAPTER 10

Bitmaps

The Macromedia Flash SWF 8 file format specification supports a variety of bitmap formats. All bitmaps are compressed to reduce file size. Lossy compression, best for imprecise images such as photographs, is provided by JPEG bitmaps; lossless compression, best for precise images such as diagrams, icons, or screen captures, is provided by ZLIB bitmaps. Both types of bitmaps can optionally contain alpha channel (transparency) information.

The JPEG format, officially defined as ITU T .81 or ISO/IEC 10918-1, is an open standard developed by the Independent Joint Photographic Experts Group. The JPEG format is not described in this document. For general information on the JPEG format, see JPEG at www.jpeg.org/. For a specification of the JPEG format, see the International Telecommunication Union at www.itu.int/ and search for recommendation T.81. The JPEG data in SWF files is encoded using the JPEG Interchange Format specified in Annex B. Flash Player also understands the popular JFIF format, an extension of the JPEG Interchange Format.

In all cases where arrays of non-JPEG pixel data are stored in bitmap tags, the pixels appear in row-major order, reading like English text, proceeding left to right within rows and top to bottom overall.

DefineBits

This tag defines a bitmap character with JPEG compression. It contains only the JPEG compressed image data (from the Frame Header onward). A separate JPEGTables tag contains the JPEG encoding data used to encode this image (the Tables/Misc segment).

The data in this tag begins with the JPEG SOI marker \texttt{0xFF, 0xD8} and ends with the EOI marker \texttt{0xFF, 0xD9}. Before the SWF 8 file format, SWF files could contain an erroneous header of \texttt{0xFF, 0xD9, 0xFF, 0xD8} before the JPEG SOI marker.

\textbf{NOTE}

Only one JPEGTables tag is allowed in a SWF file, and thus all bitmaps defined with DefineBits must share common encoding tables.
The minimum file format version is SWF 1.

JPEGTables

This tag defines the JPEG encoding table (the Tables/Misc segment) for all JPEG images defined using the DefineBits tag. There may only be one JPEGTables tag in a SWF file.

The data in this tag begins with the JPEG SOI marker \(0xFF, 0xD8\) and ends with the EOI marker \(0xFF, 0xD9\). Before SWF 8, SWF files could contain an erroneous header of \(0xFF, 0xD9, 0xFF, 0xD8\) before the JPEG SOI marker.

DefineBitsJPEG2

This tag defines a bitmap character with JPEG compression. It differs from DefineBits in that it contains both the JPEG encoding table and the JPEG image data. This tag allows multiple JPEG images with differing encoding tables to be defined within a single SWF file.

The data in this tag begins with the JPEG SOI marker \(0xFF, 0xD8\) and ends with the EOI marker \(0xFF, 0xD9\). Before SWF 8, SWF files could contain an erroneous header of \(0xFF, 0xD9, 0xFF, 0xD8\) before the JPEG SOI marker.
The minimum file format version is SWF 2.

DefineBitsJPEG3

Defines a bitmap character with JPEG compression. This tag extends DefineBitsJPEG2, adding alpha channel (transparency) data. Transparency is not a standard feature in JPEG images, so the alpha channel information is encoded separately from the JPEG data, and compressed using the ZLIB standard for compression. The data format used by the ZLIB library is described by Request for Comments (RFCs) documents 1950 to 1952.

The data in this tag begins with the JPEG SOI marker 0xFF, 0xD8 and ends with the EOI marker 0xFF, 0xD9. Before SWF 8, SWF files could contain an erroneous header of 0xFF, 0xD9, 0xFF, 0xD8 before the JPEG SOI marker.

The minimum file format version is SWF 3.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.
DefineBitsLossless

Defines a lossless bitmap character that contains RGB bitmap data compressed with ZLIB. The data format used by the ZLIB library is described by Request for Comments (RFCs) documents 1950 to 1952.

Two kinds of bitmaps are supported. Colormapped images define a colormap of up to 256 colors, each represented by a 24-bit RGB value, and then use 8-bit pixel values to index into the colormap. Direct images store actual pixel color values using 15 bits (32,768 colors) or 24 bits (about 17 million colors).

The minimum file format version is SWF 2.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORDHEADER (long)</td>
<td>Tag type = 20</td>
</tr>
<tr>
<td>CharacterID</td>
<td>UI16</td>
<td>ID for this character</td>
</tr>
<tr>
<td>BitmapFormat</td>
<td>UI8</td>
<td>Format of compressed data</td>
</tr>
<tr>
<td>BitmapWidth</td>
<td>UI16</td>
<td>Width of bitmap image</td>
</tr>
<tr>
<td>BitmapHeight</td>
<td>UI16</td>
<td>Height of bitmap image</td>
</tr>
<tr>
<td>BitmapColorTableSize</td>
<td>UI8</td>
<td>This value is one less than the actual number of colors in the color table, allowing for up to 256 colors.</td>
</tr>
<tr>
<td>ZlibBitmapData</td>
<td>If BitmapFormat = 3 COLORMAPDATA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If BitmapFormat = 4 or 5 BITMAPDATA</td>
<td>ZLIB compressed bitmap data</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader®or Adobe®Acrobat® version 6.0 or above.
The COLORMAPDATA and BITMAPDATA structures contain image data. These structures are each compressed as a single block of data. Their layouts before compression follow.

**NOTE**
Row widths in the pixel data fields of these structures must be rounded up to the next 32-bit word boundary. For example, an 8-bit image that is 253 pixels wide must be padded out to 256 bytes per line. To determine the appropriate padding, make sure to take into account the actual size of the individual pixel structures; 15-bit pixels occupy 2 bytes and 24-bit pixels occupy 4 bytes (see PIX15 and PIX24).

### COLORMAPDATA

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColorTableRGB</td>
<td>UI8[...]</td>
<td>Defines the mapping from color indices to RGB values. Number of RGB values is BitmapColorTableSize + 1.</td>
</tr>
<tr>
<td>ColormapPixelData</td>
<td>UI8[...]</td>
<td>Array of color indices. Number of entries is BitmapWidth * BitmapHeight, subject to padding (see Note preceding this table).</td>
</tr>
</tbody>
</table>

### BITMAPDATA

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitmapPixelData</td>
<td>UI8[...]</td>
<td>Array of pixel colors. Number of entries is BitmapWidth * BitmapHeight, subject to padding (see Note above).</td>
</tr>
</tbody>
</table>

### PIX15

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pix15Reserved</td>
<td>UB[1]</td>
<td>Always 0</td>
</tr>
<tr>
<td>Pix15Blue</td>
<td>UB[5]</td>
<td>Blue value</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
DefineBitsLossless2 extends DefineBitsLossless with support for transparency (alpha values).

The colormap colors in colormapped images are defined using RGBA values, and direct images store 32-bit ARGB colors for each pixel. The intermediate 15-bit color depth is not available in DefineBitsLossless2.

The minimum file format version is SWF 3.

---

**PIX24**

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pix24Reserved</td>
<td>UI8, always 0</td>
</tr>
<tr>
<td>Pix24Red</td>
<td>UI8, red value</td>
</tr>
<tr>
<td>Pix24Green</td>
<td>UI8, green value</td>
</tr>
<tr>
<td>Pix24Blue</td>
<td>UI8, blue value</td>
</tr>
</tbody>
</table>

**DefineBitsLossless2**

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORDHEADER (long), tag type = 36</td>
</tr>
<tr>
<td>CharacterID</td>
<td>UI16, ID for this character</td>
</tr>
<tr>
<td>BitmapFormat</td>
<td>UI8, format of compressed data</td>
</tr>
<tr>
<td>BitmapWidth</td>
<td>UI16, width of bitmap image</td>
</tr>
<tr>
<td>BitmapHeight</td>
<td>UI16, height of bitmap image</td>
</tr>
<tr>
<td>BitmapColorTableSize</td>
<td>UI8, if BitmapFormat = 3, this value is one less than the actual number of colors in the color table, allowing for up to 256 colors.</td>
</tr>
<tr>
<td>ZlibBitmapData</td>
<td>If BitmapFormat = 3, ZLIB compressed bitmap data, otherwise absent.</td>
</tr>
<tr>
<td></td>
<td>If BitmapFormat = 4 or 5, ALPHACOLORMAPDATA</td>
</tr>
<tr>
<td></td>
<td>If BitmapFormat = 5, ALPHABITMAPDATA</td>
</tr>
</tbody>
</table>

---

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.
The COLORMAPDATA and BITMAPDATA structures contain image data. These structures are each compressed as a single block of data. Their layouts before compression follow.

**NOTE**
Row widths in the pixel data field of ALPHACOLORMAPDATA must be rounded up to the next 32-bit word boundary. For example, an 8-bit image that is 253 pixels wide must be padded out to 256 bytes per line. Row widths in ALPHABITMAPDATA are always 32-bit aligned because the ARGB structure is 4 bytes.

<table>
<thead>
<tr>
<th>ALPHACOLORMAPDATA</th>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColorTableRGB</td>
<td></td>
<td>RGBA[ sizeof(ColorTableRGB) ]</td>
<td>Defines the mapping from color indices to RGBA values. Number of RGBA values is BitmapColorTableSize + 1.</td>
</tr>
<tr>
<td>ColormapPixelData</td>
<td></td>
<td>UI8[ sizeof(ColormapPixelData) ]</td>
<td>Array of color indices. Number of entries is BitmapWidth * BitmapHeight, subject to padding (see Note preceding this table).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ALPHABITMAPDATA</th>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>BitmapPixelData</td>
<td></td>
<td>ARGB[ sizeof(BitmapPixelData) ]</td>
<td>Array of pixel colors. Number of entries is BitmapWidth * BitmapHeight. The RGB data must already be multiplied by the alpha channel value.</td>
</tr>
</tbody>
</table>
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
CHAPTER 11

Shape Morphing

Shape morphing is the metamorphosis of one shape into another over time. The Macromedia Flash SWF 8 file format specification supports a flexible morphing model, which allows a number of shape attributes to vary during the morph. The SWF file format defines only the start and end states of the morph. Macromedia Flash Player is responsible for interpolating between the endpoints and generating the 'in-between' states.

The following shape attributes can be varied during the morph:

- The position of each edge in the shape.
- The color and thickness of the outline.
- The fill color of the shape (if filled with a color).
- The bitmap transform (if filled with a bitmap).
- The gradient transform (if filled with a gradient).
- The color and position of each point in the gradient (if filled with a gradient).

The following restrictions apply to morphing:

- The start and end shapes must have the same number of edges.
- The start and end shapes must have the same type of fill (that is, solid, gradient or bitmap).
- The style change records must be the same for the start and end shapes.
- If filled with a bitmap, both shapes must be filled with the same bitmap.
- If filled with a gradient, both gradients must have the same number of color points.
166 Shape Morphing

The following illustration shows a morph from a blue rectangle to a red quadrilateral over five frames. The green outlines represent the 'in-between' shapes of the morph sequence. Both shapes have the same number of points, and the same type of fill, namely a solid fill. Besides changing shape, the shape also gradually changes color from blue to red.

There are two tags involved in defining and playing a morph sequence:

- DefineMorphShape
- PlaceObject 2

DefineMorphShape defines the start and end states of the morph. A morph object does not use previously defined shapes; it is considered a special type of shape with only one character ID. DefineMorphShape contains a list of edges for both the start and end shapes. It also defines the fill and line styles, as they are at the start and end of the morph sequence.

The PlaceObject 2 tag displays the morph object at some point in time during the morph sequence. The ratio field controls how far the morph has progressed. A ratio of zero produces a shape identical to the start condition. A ratio of 65535 produces a shape identical to the end condition.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The DefineMorphShape tag defines the start and end states of a morph sequence. A morph object should be displayed with the PlaceObject2 tag, where the ratio field specifies how far the morph has progressed.

The minimum file format version is SWF 3.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORDHEADER</td>
<td>Tag type = 46</td>
</tr>
<tr>
<td>CharacterId</td>
<td>UI16</td>
<td>ID for this character</td>
</tr>
<tr>
<td>StartBounds</td>
<td>RECT</td>
<td>Bounds of the start shape</td>
</tr>
<tr>
<td>EndBounds</td>
<td>RECT</td>
<td>Bounds of the end shape</td>
</tr>
<tr>
<td>Offset</td>
<td>UI32</td>
<td>Indicates offset to EndEdges</td>
</tr>
<tr>
<td>MorphFillStyles</td>
<td>Morph fill styles</td>
<td>Fill style information is stored in the same manner as for a standard shape; however, each fill consists of interleaved information based on a single style type to accommodate morphing.</td>
</tr>
<tr>
<td>MorphLineStyles</td>
<td>MORPHLINESTYLES</td>
<td>Line style information is stored in the same manner as for a standard shape; however, each line consists of interleaved information based on a single style type to accommodate morphing.</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.
168 Shape Morphing

StartBounds

This defines the bounding-box of the shape at the start of the morph.

EndBounds

This defines the bounding-box at the end of the morph.

MorphFillStyles

This contains an array of interleaved fill styles for the start and end shapes. The fill style for the start shape is followed by the corresponding fill style for the end shape.

MorphLineStyles

This contains an array of interleaved line styles.

StartEdges

This array specifies the edges for the start shape, and the style change records for both shapes. Because the StyleChangeRecords must be the same for the start and end shapes, they are defined only in the StartEdges array.

EndEdges

This array specifies the edges for the end shape, and contains no style change records. The number of edges specified in StartEdges must equal the number of edges in EndEdges.

It is possible for an edge to change type over the course of a morph sequence. A straight edge can become a curved edge and vice versa. In this case, think of both edges as curved. A straight edge can be easily represented as a curve, by placing the off-curve (control) point at the midpoint of the straight edge, and the on-curve (anchor) point at the end of the straight edge. The calculation is as follows:

CurveControlDelta.x = StraightDelta.x / 2;
CurveControlDelta.y = StraightDelta.y / 2;
CurveAnchorDelta.x  = StraightDelta.x / 2;
CurveAnchorDelta.y  = StraightDelta.y / 2;

StartEdges

Contains the set of edges and the style bits that indicate style changes (for example, MoveTo, FillStyle, and LineStyle).

Number of edges must equal the number of edges in EndEdges.

EndEdges

Contains only the set of edges, with no style information.

Number of edges must equal the number of edges in StartEdges.

NOTE

Strictly speaking, MoveTo records fall into the category of StyleChangeRecords; however, they should be included in both the StartEdges and EndEdges arrays.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
DefineMorphShape2 extends the capabilities of DefineMorphShape by using a new morph line style record in the morph shape. MORPHLINESTYLE2 allows the use of new types of joins and caps as well as scaling options and the ability to fill the strokes of the morph shape.

DefineMorphShape2 specifies not only the shape bounds but also the edge bounds of the shape. While the shape bounds are calculated along the outside of the strokes, the edge bounds are taken from the outside of the edges. For an example of shape bounds versus edge bounds, see the diagram in DefineShape4. The new StartEdgeBounds and EndEdgeBounds fields assist Flash Player in accurately determining certain layouts.

In addition, DefineMorphShape2 includes new hinting information, UsesNonScalingStrokes and UsesScalingStrokes. These flags assist Flash Player in creating the best possible area for invalidation.

The minimum file format version is SWF 8.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td></td>
</tr>
<tr>
<td>CharacterId</td>
<td>UI16</td>
</tr>
<tr>
<td>StartBounds</td>
<td>RECT</td>
</tr>
<tr>
<td>EndBounds</td>
<td>RECT</td>
</tr>
<tr>
<td>StartEdgeBounds</td>
<td>RECT</td>
</tr>
<tr>
<td>EndEdgeBounds</td>
<td>RECT</td>
</tr>
<tr>
<td>Reserved</td>
<td>UB[6]</td>
</tr>
<tr>
<td>UsesNonScalingStrokes</td>
<td>UB[1]</td>
</tr>
<tr>
<td>UsesScalingStrokes</td>
<td>UB[1]</td>
</tr>
<tr>
<td>Offset</td>
<td>UI32</td>
</tr>
</tbody>
</table>
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
A morph fill style array enumerates a number of fill styles.

A fill style represents how a closed shape is filled in.

Field Type Comment
FillStyleCount Count = UI8 Count of fill styles.
FillStyleCountExtended If Count = 0xFF UI16 Extended count of fill styles.

MORPHFILLSTYLE

Field Type Comment
FillStyleType UI8 Type of fill style
0x00 = solid fill
0x10 = linear gradient fill
0x12 = radial gradient fill
0x40 = repeating bitmap
0x41 = clipped bitmap fill
0x42 = non-smoothed repeating bitmap
0x43 = non-smoothed clipped bitmap

StartColor If type = 0x00 RGBA Solid fill color with transparency information for start shape.
EndColor If type = 0x00 RGBA Solid fill color with transparency information for end shape.

StartGradientMatrix If type = 0x10 or 0x12 MATRIX Matrix for gradient fill for start shape.
EndGradientMatrix If type = 0x10 or 0x12 MATRIX Matrix for gradient fill for end shape.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Morph line styles

A morph line style array enumerates a number of fill styles.

The format of a line style array is described in the following table.

A line style represents a width and color of a line.

The format of a line style value within the file is described in the following table.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LineStyleCount</td>
<td>UI8</td>
<td>Count of line styles.</td>
</tr>
<tr>
<td>LineStyleCountExtended</td>
<td>UI16</td>
<td>Extended count of line styles.</td>
</tr>
<tr>
<td>LineStyles</td>
<td>MORPHLINESTYLE[count]</td>
<td>Array of line styles. (if MorphShape1) MORPHLINESTYLE2[count] (if MorphShape2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartWidth</td>
<td>UI16</td>
<td>Width of line in start shape in twips.</td>
</tr>
<tr>
<td>EndWidth</td>
<td>UI16</td>
<td>Width of line in end shape in twips.</td>
</tr>
<tr>
<td>StartColor</td>
<td>RGBA</td>
<td>Color value including alpha channel information for start shape.</td>
</tr>
<tr>
<td>EndColor</td>
<td>RGBA</td>
<td>Color value including alpha channel information for end shape.</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
MORPHLINESTYLE2 builds upon the capabilities of the MORPHLINESTYLE record by allowing the use of new types of joins and caps as well as scaling options and the ability to fill morph strokes. In order to use MORPHLINESTYLE2, the shape must be defined with DefineMorphShape2—not DefineMorphShape.

While the MORPHLINESTYLE record permits only rounded joins and round caps, MORPHLINESTYLE2 also supports miter and bevel joins, and square caps and no caps. For an illustration of the available joins and caps, see the diagram in the LINESTYLE2 description.

When using MORPHLINESTYLE for a miter join, a MiterLimitFactor must be specified and is used along with StartWidth or EndWidth to calculate the maximum miter length:

Max miter length = MORPHLINESTYLE2 MiterLimitFactor * MORPHLINESTYLE2 Width

If the miter join exceeds the maximum miter length, Flash Player will cut off the miter. Note that MiterLimitFactor is an 8.8 fixed-point value.

MORPHLINESTYLE2 also includes the option for pixel hinting in order to correct blurry vertical or horizontal lines.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Field</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>StartWidth</td>
<td>UI16</td>
<td>Width of line in start shape in twips.</td>
</tr>
<tr>
<td>EndWidth</td>
<td>UI16</td>
<td>Width of line in end shape in twips.</td>
</tr>
<tr>
<td>StartCapStyle</td>
<td>UB[2]</td>
<td>Start-cap style: 0 = Round cap 1 = No cap 2 = Square cap</td>
</tr>
<tr>
<td>JoinStyle</td>
<td>UB[2]</td>
<td>Join style: 0 = Round join 1 = Bevel join 2 = Miter join</td>
</tr>
<tr>
<td>HasFillFlag</td>
<td>UB[1]</td>
<td>If 1, fill is defined in FillType. If 0, uses StartColor and EndColor fields.</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoHScaleFlag</td>
<td>UB[1] If 1, stroke thickness will not scale if the object is scaled horizontally.</td>
</tr>
<tr>
<td>NoVScaleFlag</td>
<td>UB[1] If 1, stroke thickness will not scale if the object is scaled vertically.</td>
</tr>
<tr>
<td>PixelHintingFlag</td>
<td>UB[1] If 1, all anchors will be aligned to full pixels.</td>
</tr>
<tr>
<td>Reserved</td>
<td>UB[5] Must be 0.</td>
</tr>
<tr>
<td>NoClose</td>
<td>UB[1] If 1, stroke will not be closed if the stroke's last point matches its first point. Flash Player will apply caps instead of a join.</td>
</tr>
<tr>
<td>EndCapStyle</td>
<td>UB[2] End-cap style: 0 = Round cap, 1 = No cap, 2 = Square cap</td>
</tr>
<tr>
<td>MiterLimitFactor</td>
<td>If JoinStyle = 2, UI16 Miter limit factor as an 8.8 fixed-point value.</td>
</tr>
<tr>
<td>StartColor</td>
<td>If HasFillFlag = 0, RGBA Color value including alpha channel information for start shape.</td>
</tr>
<tr>
<td>EndColor</td>
<td>If HasFillFlag = 0, RGBA Color value including alpha channel information for end shape.</td>
</tr>
<tr>
<td>FillType</td>
<td>If HasFillFlag = 1, MORPHFILLSTYLE Fill style.</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The Macromedia Flash SWF 8 file format specification supports a variety of text-drawing primitives. In SWF 6 or later files, all text is represented using Unicode encodings, eliminating dependencies on playback locale for text and strings. Note that although Flash Player is fully Unicode-enabled, it can only draw left-to-right simple scripts and does not yet support Hebrew, Arabic, Thai, or other complex scripts.

Glyph text and device text

The SWF file format supports two kinds of text: glyph text and device text. Glyph text works by embedding character shapes in the SWF file, while device text uses the text rendering capabilities of the playback platform. Glyph text looks identical on all playback platforms. It can be drawn with either the standard anti-aliasing used by all shapes on a Flash Player Stage, or, in SWF 8 file format and later, rendered with FlashType font rendering technology. The usage of glyph text creates larger SWF files than for device text, especially in files that use many different characters from a large character set.

Device text is anti-aliased by the operating system that hosts Flash Player, and its appearance varies depending on the playback platform. Fonts for device text can be specified in two ways: directly, as a font name that will be sought verbatim on the playback platform; or indirectly, using one of a small number of special font names that are mapped to highly available fonts that differ in name from platform to platform, but are chosen to be as similar in appearance as possible across platforms.
Glyph text characters are defined using the DefineFont, DefineFont2, or DefineFont3 tag. Device text fonts are defined using the DefineFont and DefineFontInfo tags together, or the DefineFont2 tag. DefineFont2 tags for device text fonts do not need to include any character glyphs if they will only be used for dynamic text (see next section), although it is a good idea to include them if there is any doubt about the specified font being available at playback time on any platform. It is possible to use a given DefineFont or DefineFont2 tag as a glyph font for certain text blocks, and as a device font for others, as long as both glyphs and character codes are provided.

Static text and dynamic text

Text can be defined as **static text** or, in SWF 4 file format or later, **dynamic text**. Dynamic text can be changed programmatically at runtime, and, optionally, can be made editable for Macromedia Flash Player users as well. Dynamic text can emulate almost all features of static text; exact positioning of individual characters is the only advantage of static text, aside from implementation effort and version compatibility. Dynamic text also has many formatting capabilities that static text does not have. These rich formatting capabilities are expressed as a subset of HTML text-markup tags.

Static text is defined using the DefineText tag. Dynamic text is defined using the DefineEditText tag. Both of these tags make reference to DefineFont or DefineFont2 tags to obtain their character sources. DefineEditText tags require DefineFont2 tags rather than DefineFont tags; DefineText tags can use either DefineFont or DefineFont2 tags.

The DefineEditText tag provides a flag that indicates whether to use glyph text or device text. However, the DefineText tag does not. This means that, for static text, SWF file format provides no means to indicate whether to use glyph text or device text. This situation is resolved by runtime flags. Normally, all static text is rendered as glyph text. When a Flash Player plug-in is embedded in an HTML page, an HTML tag option called *devicefont* will cause Flash Player to render all static text as device text, if possible; as usual, glyph text is used as a fallback. The ability of the DefineEditText tag to specify glyph text or device text is another reason to consider dynamic text superior to static text.
Glyphs are defined once in a standard coordinate space called the **EM square**. The same set of glyphs are used for every point size of a given font. To render a glyph at different point sizes, Flash Player scales the glyph from EM coordinates to point-size coordinates.

Glyph fonts—without using FlashType—do not include any hinting information for improving the quality of small font sizes. However, anti-aliasing dramatically improves the legibility of scaled-down text. Glyph text remains legible down to about 12 points (viewed at 100%). At 12 points and lower, FlashType is recommended for readable glyph text. FlashType gives superior text quality at small point sizes and includes extra font meta-information for improved rendering.

TrueType fonts can be readily converted to SWF glyphs. A simple algorithm can replace the Quatradic B-splines (used by TrueType fonts) with Quadratic Bezier curves (used by SWF glyphs).

Example: To the left is the glyph for the TrueType letter 'b' of Monotype Arial. It is made up of curved and straight edges. Squares indicate on-curve points, and crosses indicate off-curve points. The black circle is the reference point for the glyph. The blue outline indicates the bounding box of the glyph.
The EM square is an imaginary square that is used to size and align glyphs. The EM square is generally large enough to completely contain all glyphs, including accented glyphs. It includes the font’s ascent, descent, and some extra spacing to prevent lines of text from colliding.

SWF glyphs are always defined on an EM square of 1024 by 1024 units. Glyphs from other sources (such as TrueType fonts) may be defined on a different EM square. To use these glyphs in SWF file format, they should be scaled to fit an EM square of 1024.

Converting TrueType fonts to SWF glyphs

TrueType glyphs are defined using Quadratic B-splines, which can be easily converted to the Quadratic Bezier curves used by SWF glyphs.

A TrueType B-spline is composed of one on-curve point, followed by many off-curve points, followed by another on-curve point. The midpoint between any two off-curve points is guaranteed to be on the curve. A SWF Bezier curve is composed of one on-curve point, followed by one off-curve point, followed by another on-curve point. The conversion from TrueType to SWF curves involves inserting a new on-curve point at the midpoint of two successive off-curve points.

Example: Following is a four point B-spline. P0 and P3 are on-curve points. P1 and P2 are successive off-curve points.
This curve can be represented as two Quadratic Bezier curves by inserting a new point M, at the midpoint of P1,P2. The result is two Quadratic Bezier curves; P0,P1,M and M,P2,P3.

The complete procedure for converting TrueType glyphs to SWF glyphs is as follows:

1. Negate the y-coordinate. (In TrueType glyphs, the y-axis points up; in SWF glyphs, the y-axis points down.)
2. Scale the x and y co-ordinates from the EM square of the TrueType font, to the EM square of the SWF glyph (always 1024).
3. Insert an on-curve (anchor) point at the midpoint of each pair of off-curve points.

Kerning and advance values

Kerning defines the horizontal distance between two glyphs. Some font systems store kerning information along with each font definition. SWF file format, in contrast, stores kerning information with every glyph instance (every character in a glyph text block). This is referred to as an advance value.

In the preceding example, the A glyph overlaps the V glyph. In this case, the advance is narrower than the width of the A glyph.

FlashType

Glyph text can be rendered using the normal Flash Player renderer or, in SWF 8 and later, with the FlashType system.

FlashType is a high-quality text renderer supported inside the Flash Player renderer. FlashType has the following advantages over using the normal renderer for text:

■ Readable, even at small point sizes.
■ Maintains the aesthetic look and feel of a font, even at small point sizes.
■ Supports pixel snapping for ultra-clear text (when left-aligned dynamic text is used).
■ Improved performance over glyph text, typically.
■ LCD sub-pixel rendering when Flash Player detects an LCD screen.
A limitation of FlashType, however, is that it does not animate well as compared to glyph text. FlashType uses Continuous Stroke Modulation (CSM) parameters to tune its performance. CSM is the continuous modulation of both stroke weight and edge sharpness. CSM uses two rendering parameters: inside and outside cutoff. Optimal values for these parameters are highly subjective and can depend on user preferences, lighting conditions, display properties, typeface, foreground and background colors, and point size. However, under most circumstances, high-quality type can be achieved with a small set of interpolated values.

The function that creates FlashType edges has an outside cutoff (below which the edge isn’t drawn) and an inside cutoff (above which the edge is opaque). Between these two cutoff values is a linear function ranging from zero at the outside cutoff to the maximum value at the inside cutoff.

Adjusting the outside and inside cutoff values affects stroke weight and edge sharpness. The spacing between these two parameters is comparable to twice the filter radius of classic anti-aliasing methods: a narrow spacing provides a sharper edge while a wider spacing provides a softer, more filtered edge. When the spacing is zero, the resulting density image is a bi-level bitmap. When the spacing is very wide, the resulting density image has a watercolor-like edge. Typically, users prefer sharp, high-contrast edges at small point sizes and softer edges for animated text and larger point sizes.

The outside cutoff typically has a negative value, the inside cutoff typically has a positive value, and their midpoint typically lies near zero. Adjusting these parameters to shift the midpoint towards negative infinity will increase the stroke weight; shifting the midpoint towards positive infinity will decrease the stroke weight. Note that the outside cutoff should always be less than or equal to the inside cutoff.

Flash Player creates a table of CSM parameters as a function of text size and text color for each FlashType font in use. This default table typically provides a suitable set of CSM settings across a wide range of point sizes. However, you can specify a user-defined table to replace the default table by using the ActionScript function `setAdvancedAntialiasingTable()`.

Of the four text types supported in SWF file format (static glyph, static device, dynamic glyph, and dynamic device), the most complex is static glyph text. The other types use simpler variations on the rules used for defining static glyph text.

NOTE
The CSM parameters are intended to make fonts more readable and not to create effects. Extreme values of CSM result in rendering artifacts. To apply effects to text, it is much better to use reasonable CSM values and then apply filters or blend effects.
Static glyph text is defined using two tags:

- The DefineFont tag defines a set of glyphs.
- The DefineText tag defines the text string that is displayed in the font.

The DefineFont tag defines all the glyphs used by subsequent DefineText tags. DefineFont includes an array of SHAPERECORDs, which describe the outlines of the glyphs. These shape records are the same records used by DefineShape to define non-text shapes. To keep file size to a minimum, only the glyphs actually used are included in the DefineFont tag.

The DefineText tag stores the actual text string(s) to be displayed, represented as a series of glyph indices. It also includes the bounding box of the text object, a transformation matrix, and style attributes such as color and size.

DefineText contains an array of TEXTRECORDs. A TEXTRECORD selects the current font, color, and point size, as well as the x and y position of the next character in the text. These styles apply to all characters that follow, until another TEXTRECORD changes the styles. A TEXTRECORD also contains an array of indices into the glyph table of the current font. Characters are not referred to by their character codes, as in traditional programming, but rather by an index into the glyph table. The glyph data also includes the advance value for each character in the text string.

Static glyph text example

Consider the example of displaying the static glyph text "bob" in the Arial font, with a point size of 24.

First, define the glyphs with a DefineFont tag. The glyph table, of type SHAPE, has two SHAPERECORDs. At index 0 is the shape of a lowercase b (see diagram). At index 1 is the shape of a lowercase o. (The second b in bob is a duplicate, and is not required). DefineFont also includes a unique ID so it can be selected by the DefineText tag.

Next, define the text itself with a DefineText tag. The TEXTRECORD sets the position of the first character, selects the Arial font (using the font’s character ID), and sets the point size to 24, so the font is scaled to the correct size. (Remember that glyphs are defined in EM coordinates—the actual point size is part of the DefineText tag). It also contains an array of GLYPHENTRYs. Each glyph entry contains an index into the font’s shape array. In this example, the first glyph entry has index 0 (which corresponds to the b shape), the second entry has index 1 (the o), and the third entry has index 0 (b again). Each GLYPHENTRY also includes an advance value for accurately positioning the glyph.

NOTE
A DefineFont tag must always come before any DefineText tags that refer to it.
The following diagram illustrates how the DefineText tag interacts with the DefineFont tag:

Font tags
DefineFont

The DefineFont tag defines the shape outlines of each glyph used in a particular font. Only the glyphs that are used by subsequent DefineText tags are actually defined. DefineFont tags cannot be used for dynamic text. Dynamic text requires the DefineFont2 tag.

The minimum file format version is SWF 1.

The font ID uniquely identifies the font. It can be used by subsequent DefineText tags to select the font. Like all SWF character IDs, font IDs must be unique among all character IDs in a SWF file.

If you provide a DefineFontInfo tag to go along with a DefineFont tag, be aware that the order of the glyphs in the DefineFont tag must match the order of the character codes in the DefineFontInfo tag, which must be sorted by code point order.
The OffsetTable and GlyphShapeTable are used together. These tables have the same number of entries, and there is a one-to-one ordering match between the order of the offsets and the order of the shapes. The OffsetTable points to locations in the GlyphShapeTable. Each offset entry stores the difference (in bytes) between the start of the offset table and the location of the corresponding shape. Because the GlyphShapeTable immediately follows the OffsetTable, the number of entries in each table (the number of glyphs in the font) can be inferred by dividing the first entry in the OffsetTable by two.

The first STYLECHANGERECORD of each SHAPE in the GlyphShapeTable does not use the LineStyle and LineStyles fields. In addition, the first STYLECHANGERECORD of each shape must have both fields StateFillStyle0 and FillStyle0 set to 1.

DefineFontInfo

The DefineFontInfo tag defines a mapping from a glyph font (defined with DefineFont) to a device font. It provides a font name and style to pass to the playback platform’s text engine, and a table of character codes that identifies the character represented by each glyph in the corresponding DefineFont tag, allowing the glyph indices of a DefineText tag to be converted to character strings.

The presence of a DefineFontInfo tag does not force a glyph font to become a device font; it merely makes the option available. The actual choice between glyph and device usage is made according to the value of devicefont (see the introduction) or the value of UseOutlines in a DefineEditText tag. If a device font is unavailable on a playback platform, Flash Player will fall back to glyph text.
The entries in the CodeTable must be sorted in ascending order by code point, by the value they provide. The order of the entries in the CodeTable must also match the order of the glyphs in the DefineFont tag to which this DefineFontInfo tag applies. This places a requirement on the ordering of glyphs in the corresponding DefineFont tag.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORD</td>
<td>Tag type = 13.</td>
</tr>
<tr>
<td>FontID</td>
<td>UI16</td>
<td>Font ID this information is for.</td>
</tr>
<tr>
<td>FontNameLen</td>
<td>UI8</td>
<td>Length of font name.</td>
</tr>
<tr>
<td>FontName</td>
<td>UI8[FontNameLen]</td>
<td>Name of the font (see following).</td>
</tr>
<tr>
<td>FontFlagsSmallText</td>
<td>UB[1]</td>
<td>SWF 7 file format or later: Font is small. Character glyphs are aligned on pixel boundaries for dynamic and input text.</td>
</tr>
<tr>
<td>FontFlagsShiftJIS</td>
<td>UB[1]</td>
<td>ShiftJIS character codes.</td>
</tr>
<tr>
<td>FontFlagsItalic</td>
<td>UB[1]</td>
<td>Font is italic.</td>
</tr>
<tr>
<td>FontFlagsBold</td>
<td>UB[1]</td>
<td>Font is bold.</td>
</tr>
<tr>
<td>FontFlagsWideCodes</td>
<td>UB[1]</td>
<td>If 1, CodeTable is UI16 array; otherwise, CodeTable is UI8 array.</td>
</tr>
<tr>
<td>CodeTable</td>
<td>If FontFlagsWideCodes</td>
<td>Glyph to code table, sorted in ascending order.</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
SWF 6 or later files require Unicode text encoding. One aspect of this requirement is that all character code tables are specified using UCS-2 (UCS-2 is generally the first 64k code points of UTF-16). This encoding uses a fixed 2 bytes for each character. This means that when a DefineFontInfo tag appears in a SWF 6 or later file, FontFlagsWideCodes must be set, FontFlagsShiftJIS and FontFlagsANSI must be cleared, and CodeTable must consist of UI16 entries (as always, in little-endian byte order) encoded in UCS-2.

Another Unicode requirement that applies to SWF 6 or later files is that font names must always be encoded using UTF-8. In SWF 5 or earlier files, font names are encoded in a platform-specific way, in the codepage of the system on which they were authored. The playback platform will interpret them using its current codepage, with potentially inconsistent results. If the playback platform is an ANSI system, font names will be interpreted as ANSI strings. If the playback platform is a Japanese shift-JIS system, font names will be interpreted as shift-JIS strings. Many other values for the playback platform’s codepage, which governs this decision, are possible. This playback locale dependency is undesirable, which is why SWF 6 file format moved toward a standard encoding for font names. Note that font name strings in the DefineFontInfo tag are not null-terminated; instead their length is specified by the FontNameLen field. FontNameLen specifies the number of bytes in FontName, which is not necessarily equal to the number of characters, since some encodings may use more than one byte per character.

Font names are normally used verbatim, passed directly to the playback platform’s font system in order to locate a font. However, there are several special indirect font names that are mapped to different actual font names depending on the playback platform. These indirect mappings are hard-coded into each platform-specific port of Flash Player, and the fonts for each platform are chosen from among system default fonts or other fonts that are very likely to be available. As a secondary consideration, the indirect mappings are chosen so as to maximize the similarity of indirect fonts across platforms.
The following tables describe the indirect font names that are supported. Western indirect fonts

### Japanese indirect fonts

#### DefineFontInfo2

When generating SWF 6 or later, it is recommended that you use the new DefineFontInfo2 tag rather than DefineFontInfo. DefineFontInfo2 is identical to DefineFontInfo, except that it adds a field for a language code. If you use the older DefineFontInfo, the language code will be assumed to be zero, which results in behavior that is dependent on the locale in which Flash Player is running.

<table>
<thead>
<tr>
<th>Font name</th>
<th>English Name</th>
<th>UTF-8 Byte String (hex)</th>
<th>Example appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>_sans</td>
<td>Hello world</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_serif</td>
<td>Hello world</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_typewriter</td>
<td>Hello world</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Japanese indirect fonts

<table>
<thead>
<tr>
<th>Font name</th>
<th>English Name</th>
<th>UTF-8 Byte String (hex)</th>
<th>Example appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gothic</td>
<td>English Name:</td>
<td>5F E3 82 B4 E3 82 B7</td>
<td>Example appearance:</td>
</tr>
<tr>
<td>Tohaba (Gothic Mono)</td>
<td></td>
<td>5F E7 AD 89 E5 B9 85</td>
<td></td>
</tr>
<tr>
<td>Mincho</td>
<td>English Name:</td>
<td>5F E6 98 8E E6 9C 9D</td>
<td>Example appearance:</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The DefineFont2 tag extends the functionality of DefineFont. Enhancements include:

- 32-bit entries in the OffsetTable, for fonts with more than 64K glyphs.
- Mapping to device fonts, by incorporating all the functionality of DefineFontInfo.
- Font metrics for improved layout of dynamic glyph text.

DefineFont2 tags are the only font definitions that can be used for dynamic text.

**DefineFontInfo2**

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORDHEADER</td>
<td>Tag type = 62.</td>
</tr>
<tr>
<td>FontID</td>
<td>UI16</td>
<td>Font ID this information is for.</td>
</tr>
<tr>
<td>FontNameLen</td>
<td>UI8</td>
<td>Length of font name.</td>
</tr>
<tr>
<td>FontName</td>
<td>UI8[FontNameLen]</td>
<td>Name of the font.</td>
</tr>
<tr>
<td>FontFlagsSmallText</td>
<td>UB[1]</td>
<td>SWF 7 or later: Font is small. Character glyphs are aligned on pixel boundaries for dynamic and input text.</td>
</tr>
<tr>
<td>FontFlagsShiftJIS</td>
<td>UB[1]</td>
<td>Always 0.</td>
</tr>
<tr>
<td>FontFlagsANSI</td>
<td>UB[1]</td>
<td>Always 0.</td>
</tr>
<tr>
<td>FontFlagsItalic</td>
<td>UB[1]</td>
<td>Font is italic.</td>
</tr>
<tr>
<td>FontFlagsBold</td>
<td>UB[1]</td>
<td>Font is bold.</td>
</tr>
<tr>
<td>LanguageCode</td>
<td>LANGCODE</td>
<td>Language ID.</td>
</tr>
<tr>
<td>CodeTable</td>
<td>UI16[nGlyphs]</td>
<td>Glyph to code table in UCS-2, sorted in ascending order.</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
In SWF 6 or later files, DefineFont2 has the same Unicode requirements as DefineFontInfo. Similarly to the DefineFontInfo tag, the CodeTable (and thus also the OffsetTable, GlyphShapeTable, and FontAdvanceTable) must be sorted in code point order. If a DefineFont2 tag will be used only for dynamic device text, and no glyph-rendering fallback is desired, set NumGlyphs to zero, and omit all tables having NumGlyphs entries. This will substantially reduce the size of the DefineFont2 tag. DefineFont2 tags without glyphs cannot support static text, which uses glyph indices to select characters, and also cannot support glyph text, which requires glyph shape definitions.

**DefineFont2 Field Type Comment**

- **CodeTable**
  - If FontFlagsWideCodes
    - UI16[NumGlyphs]
  - Otherwise UI8[NumGlyphs]
  - Sorted in ascending order.
  - Always UCS-2 in SWF 6 or later.

- **FontAscent**
  - If FontFlagsHasLayout
    - SI16
    - Font ascender height.

- **FontDescent**
  - If FontFlagsHasLayout
    - SI16
    - Font descender height.

- **FontLeading**
  - If FontFlagsHasLayout
    - SI16
    - Font leading height (see following).

- **FontAdvanceTable**
  - If FontFlagsHasLayout
    - SI16[NumGlyphs]
    - Advance value to be used for each glyph in dynamic glyph text.

- **FontBoundsTable**
  - If FontFlagsHasLayout
    - RECT[NumGlyphs]
    - Not used in Flash Player through version 7 (but must be present).

- **KerningCount**
  - If FontFlagsHasLayout
    - UI16
    - Not used in Flash Player through version 7 (always set to 0 to save space).

- **FontKerningTable**
  - If FontFlagsHasLayout
    - KERNINGRECORD
      - [KerningCount]
    - Not used in Flash Player through version 7 (omit with KerningCount of 0).
Layout information (ascent, descent, leading, advance table, bounds table, kerning table) is useful only for dynamic glyph text. This information takes the place of the per-character placement information that is used in static glyph text. The layout information in the DefineFont2 tag is fairly standard font-metrics information that can typically be extracted directly from a standard font definition, such as a TrueType font.

As with DefineFont, in DefineFont2 the first STYLECHANGERECORD of each SHAPE in the GlyphShapeTable does not use the LineStyle and LineStyles fields. In addition, the first STYLECHANGERECORD of each shape must have both fields StateFillStyle0 and FillStyle0 set to 1.

The DefineFont2 tag reserves space for a font bounds table and kerning table. This information is not used in Flash Player through version 7. However, this information must be present in order to constitute a well-formed DefineFont2 tag. Supply minimal (low-bit) RECTs for FontBoundsTable, and always set KerningCount to zero, which allows FontKerningTable to be omitted.

DefineFont3

The DefineFont3 tag is introduced along with the DefineFontAlignZones tag in SWF 8. The DefineFontAlignZones tag is optional but recommended for SWF files using FlashType, and it modifies the DefineFont3 tag.

The DefineFont3 tag extends the functionality of DefineFont2 by expressing the SHAPE coordinates in the GlyphShapeTable at 20 times the resolution. All the EMSquare coordinates are multiplied by 20 at export, allowing fractional resolution to 1/20 of a unit. This allows for more precisely defined glyphs and results in better visual quality.

The minimum file format version is SWF 8.

NOTE

Leading is a vertical line-spacing metric. It is the distance (in EM-square coordinates) between the bottom of the descender of one line and the top of the ascender of the next line.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
DefineFontAlignZones

The DefineFont3 tag can be modified by a DefineFontAlignZones tag. FlashType uses alignment zones to establish the borders of a glyph for pixel snapping. Alignment zones are critical for high-quality display of fonts.

The alignment zone defines a bounding box for strong vertical and horizontal components of a glyph. The box is described by a left coordinate, thickness, baseline coordinate, and height. Small thicknesses or heights are often set to 0.

For example, consider the letter I. The letter I has a strong horizontal at its baseline and the top of the letter. The letter I also has strong verticals that occur at the edges of the stem—not the short top bar or serif. These strong verticals and horizontals of the center block of the letter define the alignment zones.

FontAdvanceTable If FontFlagsHasLayout
SI16[NumGlyphs]
Advance value to be used for each glyph in dynamic glyph text.

FontBoundsTable If FontFlagsHasLayout
RECT[NumGlyphs]
Not used in Flash Player through version 7 (but must be present).

KerningCount If FontFlagsHasLayout UI16 Not used in Flash Player through version 7 (always set to 0 to save space).

FontKerningTable If FontFlagsHasLayout
KERNINGRECORD[KerningCount]
Not used in Flash Player through version 7 (omit with KerningCount of 0).
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The minimum file format version is SWF 1.

The `TextBounds` field is the rectangle that completely encloses all the characters in this text block.

The `GlyphBits` and `AdvanceBits` fields define the number of bits used for the `GlyphIndex` and `GlyphAdvance` fields, respectively, in each `GLYPHENTRY` record.

**Text records**

A `TEXTRECORD` sets text styles for subsequent characters. It can be used to select a font, change the text color, change the point size, insert a line break, or set the `x` and `y` position of the next character in the text. The new text styles apply until another `TEXTRECORD` changes the styles.

<table>
<thead>
<tr>
<th>DefineText Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECORDHEADER</td>
<td>Tag type = 11.</td>
</tr>
<tr>
<td>CharacterID</td>
<td>UI16 ID for this text character.</td>
</tr>
<tr>
<td>TextBounds</td>
<td>RECT Bounds of the text.</td>
</tr>
<tr>
<td>TextMatrix</td>
<td>MATRIX Transformation matrix for the text.</td>
</tr>
<tr>
<td>GlyphBits</td>
<td>UI8 Bits in each glyph index.</td>
</tr>
<tr>
<td>AdvanceBits</td>
<td>UI8 Bits in each advance value.</td>
</tr>
<tr>
<td>TextRecords</td>
<td>TEXTRECORD[zero or more] Text records.</td>
</tr>
<tr>
<td>EndOfRecordsFlag</td>
<td>UI8 Must be 0.</td>
</tr>
</tbody>
</table>
The TEXTRECORD also defines the actual characters in a text object. Characters are referred to by an index into the current font’s glyph table, not by a character code. Each TEXTRECORD contains a group of characters that all share the same text style, and are on the same line of text.

The **FontID** field is used to select a previously defined font. This ID uniquely identifies a DefineFont or DefineFont2 tag from earlier in the SWF file.

The **TextHeight** field defines the height of the font in twips. For example, a pixel height of 50 is equivalent to a TextHeight of 1000. (50 * 20 = 1000).

The **XOffset** field defines the offset from the left of the TextBounds rectangle to the reference point of the glyph (the point within the EM square from which the first curve segment departs). Typically, the reference point is on the baseline, near the left side of the glyph (see the example for Glyph text). The XOffset is generally used to create indented text or non-left-justified text. If there is no XOffset specified, the offset is assumed to be zero.

The **YOffset** field defines the offset from the top of the TextBounds rectangle to the reference point of the glyph. The TextYOffset is generally used to insert line breaks, moving the text position to the start of a new line.

<table>
<thead>
<tr>
<th>TEXTRECORD Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextRecordType</td>
<td>UB[1] Always 1.</td>
</tr>
<tr>
<td>StyleFlagsHasFont</td>
<td>UB[1] 1 if text font specified.</td>
</tr>
<tr>
<td>StyleFlagsHasColor</td>
<td>UB[1] 1 if text color specified.</td>
</tr>
<tr>
<td>StyleFlagsHasYOffset</td>
<td>UB[1] 1 if y offset specified.</td>
</tr>
<tr>
<td>StyleFlagsHasXOffset</td>
<td>UB[1] 1 if x offset specified.</td>
</tr>
<tr>
<td>FontID</td>
<td>If StyleFlagsHasFont UI16 Font ID for following text.</td>
</tr>
<tr>
<td>TextColor</td>
<td>If StyleFlagsHasColor RGB Font color for following text.</td>
</tr>
<tr>
<td></td>
<td>If this record is part of a DefineText2 tag then RGBA Font color for following text.</td>
</tr>
<tr>
<td>XOffset</td>
<td>If StyleFlagsHasXOffset SI16 x offset for following text.</td>
</tr>
<tr>
<td>YOffset</td>
<td>If StyleFlagsHasYOffset SI16 y offset for following text.</td>
</tr>
<tr>
<td>TextHeight</td>
<td>If hasFont UI16 Font height for following text.</td>
</tr>
<tr>
<td>GlyphCount</td>
<td>UI8 Number of glyphs in record.</td>
</tr>
<tr>
<td>GlyphEntries</td>
<td>GLYPHENTRY[GridCount] Glyph entry (see following).</td>
</tr>
</tbody>
</table>
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
The DefineEditText tag defines a dynamic text object, or text field. A text field is associated with an ActionScript variable name where the contents of the text field are stored. The SWF file can read and write the contents of the variable, which is always kept in sync with the text being displayed. If the ReadOnly flag is not set, users may change the value of a text field interactively.

Fonts used by DefineEditText must be defined using DefineFont2, not DefineFont.

The minimum file format version is SWF 4.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORDHEADER Tag type = 37.</td>
</tr>
<tr>
<td>CharacterID</td>
<td>UI16 ID for this dynamic text character.</td>
</tr>
<tr>
<td>Bounds</td>
<td>RECT Rectangle that completely encloses the text field.</td>
</tr>
<tr>
<td>HasText</td>
<td>UB[1] 0 = text field has no default text. 1 = text field initially displays the string specified by InitialText.</td>
</tr>
<tr>
<td>WordWrap</td>
<td>UB[1] 0 = text will not wrap and will scroll sideways. 1 = text will wrap automatically when the end of line is reached.</td>
</tr>
<tr>
<td>Multiline</td>
<td>UB[1] 0 = text field is one line only. 1 = text field is multi-line and scrollable.</td>
</tr>
<tr>
<td>Password</td>
<td>UB[1] 0 = characters are displayed as typed. 1 = all characters are displayed as an asterisk.</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>UB[1] 0 = text editing is enabled. 1 = text editing is disabled.</td>
</tr>
</tbody>
</table>
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
If the HTML flag is set, the contents of InitialText are interpreted as a limited subset of the HTML tag language, with a few additions not normally present in HTML. The following tags are supported:

- **Indent**
  - If HasLayout
  - UI16
  - Indent in twips.

- **Leading**
  - If HasLayout
  - SI16
  - Leading in twips (vertical distance between bottom of descender of one line and top of ascender of the next).

- **VariableName**
  - STRING
  - Name of the variable where the contents of the text field are stored. May be qualified with dot syntax or slash syntax for non-global variables.

- **InitialText**
  - If HasText
  - STRING
  - Text that is initially displayed.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;p&gt;</code></td>
<td>Defines a paragraph. The attribute align may be present, with value left, right, or center.</td>
</tr>
<tr>
<td><code>&lt;br&gt;</code></td>
<td>Inserts a line break.</td>
</tr>
<tr>
<td><code>&lt;a&gt;</code></td>
<td>Defines a hyperlink. The attribute href must be present. The attribute target is optional, and specifies a window name.</td>
</tr>
<tr>
<td><code>&lt;font&gt;</code></td>
<td>Defines a span of text that uses a given font. The following attributes are available:</td>
</tr>
<tr>
<td></td>
<td>• <strong>face</strong>, which specifies a font name that must match a font name supplied in a DefineFont2 tag</td>
</tr>
<tr>
<td></td>
<td>• <strong>size</strong>, which is specified in twips, and may include a leading '+' or '-' for relative sizes</td>
</tr>
<tr>
<td></td>
<td>• <strong>color</strong>, which is specified as a #RRGGBB hex triplet</td>
</tr>
<tr>
<td><code>&lt;b&gt;</code></td>
<td>Defines a span of bold text.</td>
</tr>
<tr>
<td><code>&lt;i&gt;</code></td>
<td>Defines a span of italic text.</td>
</tr>
<tr>
<td><code>&lt;u&gt;</code></td>
<td>Defines a span of underlined text.</td>
</tr>
<tr>
<td><code>&lt;li&gt;</code></td>
<td>Defines a bulleted paragraph. The <code>&lt;ul&gt;</code> tag is not necessary and is not supported. Numbered lists are not supported.</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
In addition to FlashType tags discussed earlier in this chapter, FlashType also supports a tag for modifying text fields. The CSMTextSettings tag modifies a previously streamed DefineText, DefineText2, or DefineEditText tag. The CSMTextSettings tag turns FlashType on or off for a text field, and can also be used to define FlashType quality and options. The minimum file format version is SWF 8.

<textformat> ... </textformat>

Defines a span of text with certain formatting options. The following attributes are available:

- leftmargin, which specifies the left margin in twips
- rightmargin, which specifies the right margin in twips
- indent, which specifies the left indent in twips
- blockindent, which specifies a block indent in twips
- leading, which specifies the leading in twips
- tabstops, which specifies a comma-separated list of tab stops, each specified in twips

<tab>

Inserts a tab character, which advances to the next tab stop as defined with the <textformat> tag.

---

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The Thickness and Sharpness fields are interpretations of the CSM parameters, applied to a particular text field. The thickness and sharpness setting will override the default CSM setting for that text field.

The CSM parameters, at render time, are computed from the thickness and sharpness:

outsideCutoff = \((0.5f \times \text{sharpness} - \text{thickness}) \times \text{fontSize}\);
insideCutoff = \((-0.5f \times \text{sharpness} - \text{thickness}) \times \text{fontSize}\);

Using the font size in the cutoff calculations results in linear scaling of CSM parameters, and linear scaling tends to be a poor approximation when significant scaling is applied. When a text field will scale, it is usually better to use the default table or provide your own anti-aliasing table.
CHAPTER 13

Sounds

The Macromedia Flash SWF 8 file format specification defines a small and efficient sound model. The SWF file format supports sample rates of 5.5, 11, 22, and 44 kHz in both stereo and mono. Macromedia Flash Player supports rate conversion and multichannel mixing of these sounds. The number of simultaneous channels supported depends on the CPU of specific platforms, but is typically three to eight channels.

There are two types of sounds in SWF file format:

■ Event sounds
■ Streaming sounds

Event sounds are played in response to some event such as a mouse click, or when Flash Player reaches a certain frame. Event sounds must be defined (downloaded) before they are used. They can be reused for multiple events if desired. Event sounds may also have a sound “style” that modifies how the basic sound is played.

Streaming sounds are downloaded and played in tight synchronization with the timeline. In this mode, sound packets are stored with each frame.

“5.5 kHz” = 5512 Hz
“11 kHz” = 11025 Hz
“22 kHz” = 22050 Hz
“44 kHz” = 44100 Hz

For information on including audio in the FLV File Format, please refer to the FLV Audio Tags section.

NOTE

The exact sample rates used are as follows. These are standard sample rates based on CD audio, which is sampled at 44100 Hz. The four sample rates are one-eighth, one-quarter, one-half, and exactly the 44100 Hz sampling rate.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
The SoundId field uniquely identifies the sound so it can be played by StartSound.

Format 0 (uncompressed) and Format 3 (uncompressed little-endian) are similar. Both encode uncompressed audio samples. For 8-bit samples, the two formats are identical. For 16-bit samples, the two formats differ in byte ordering. In Format 0, 16-bit samples are encoded and decoded according to the native byte ordering of the platform on which the encoder and Flash Player, respectively, are running. In Format 3, 16-bit samples are always encoded in little-endian order (least significant byte first), and are byte-swapped if necessary in Flash Player before playback. Format 0 is clearly disadvantageous because it introduces a playback platform dependency. For 16-bit samples, Format 3 is highly preferable to Format 0 for SWF 4 or later.

The contents of SoundData vary depending on the value of the SoundFormat field in the SoundStreamHead tag:

- If SoundFormat is 0 or 3, SoundData contains raw, uncompressed samples.
- If SoundFormat is 1, SoundData contains an ADPCM sound data record.
- If SoundFormat is 2, SoundData contains an MP3 sound data record.
- If SoundFormat is 6, SoundData contains Nellymoser data (see Nellymoser compression).

<table>
<thead>
<tr>
<th>Define Sound Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
StartSound

StartSound is a control tag that either starts (or stops) playing a sound defined by DefineSound. The SoundId field identifies which sound is to be played. The SoundInfo field defines how the sound is played. Stop a sound by setting the SyncStop flag in the SOUNDINFO record.

The minimum file format version is SWF 1.

### Sound styles

SOUNDINFO

The SOUNDINFO record modifies how an event sound is played. An event sound is defined with the DefineSound tag. Sound characteristics that can be modified include:

- Whether the sound loops (repeats) and how many times it loops.
- Where sound playback begins and ends.
- A sound **envelope** for time-based volume control.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORDHEADER</td>
<td>Tag type = 15.</td>
</tr>
<tr>
<td>SoundId</td>
<td>UI16</td>
<td>ID of sound character to play.</td>
</tr>
<tr>
<td>SoundInfo</td>
<td>SOUNDINFO</td>
<td>Sound style information.</td>
</tr>
<tr>
<td>SyncStop</td>
<td>UB[1]</td>
<td>Stop the sound now.</td>
</tr>
<tr>
<td>SyncNoMultiple</td>
<td>UB[1]</td>
<td>Don't start the sound if already playing.</td>
</tr>
<tr>
<td>HasLoops</td>
<td>UB[1]</td>
<td>Has loop information.</td>
</tr>
<tr>
<td>HasInPoint</td>
<td>UB[1]</td>
<td>Has in-point information.</td>
</tr>
<tr>
<td>InPoint</td>
<td>UI32</td>
<td>Number of samples to skip at beginning of sound.</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.
The SOUNDENVELOPE structure is defined as follows:

For mono sounds, set the LeftLevel and RightLevel fields to the same value. If the values differ, they will be averaged.

- **OutPoint**: If HasOutPoint UI32 Position in samples of last sample to play.
- **LoopCount**: If HasLoops UI16 Sound loop count.
- **EnvPoints**: If HasEnvelope UI8 Sound Envelope point count.
- **EnvelopeRecords**: If HasEnvelope SOUNDENVELOPE[EnvPoints] Sound Envelope records.

**SOUNDENVELOPE**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos44</td>
<td>UI32</td>
<td>Position of envelope point as a number of 44 kHz samples. Multiply accordingly if using a sampling rate less than 44 kHz.</td>
</tr>
<tr>
<td>LeftLevel</td>
<td>UI16</td>
<td>Volume level for left channel. Minimum is 0, maximum is 32768.</td>
</tr>
<tr>
<td>RightLevel</td>
<td>UI16</td>
<td>Volume level for right channel. Minimum is 0, maximum is 32768.</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
210 Sounds

Streaming sound

The SWF file format supports a streaming sound mode where sound data is played and downloaded in tight synchronization with the timeline. In this mode, sound packets are stored with each frame.

When streaming sound is present, and the playback CPU is too slow to maintain the desired SWF frame rate, Flash Player skips frames of animation in order to maintain sound synchronization and avoid dropping sound samples. (Actions from the skipped frames are still executed.)

The main timeline of a SWF file can only have a single streaming sound playing at a time, but each sprite can have its own streaming sound (see Sprites and Movie Clips).

SoundStreamHead

If a timeline contains streaming sound data, there must be a SoundStreamHead or SoundStreamHead2 tag before the first sound data block (see SoundStreamBlock). The SoundStreamHead tag defines the data format of the sound data, the recommended playback format, and the average number of samples per SoundStreamBlock.

The minimum file format version is SWF 1.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>0x0</td>
<td>RECORDHEADER, Tag type = 18.</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x0</td>
<td>UB[4], Always zero.</td>
</tr>
<tr>
<td>PlaybackSoundRate</td>
<td>0x0</td>
<td>UB[2], Playback sampling rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = 5.5 kHz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = 11 kHz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = 22 kHz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 = 44 kHz</td>
</tr>
<tr>
<td>PlaybackSoundSize</td>
<td>0x0</td>
<td>UB[1], Playback sample size.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Always 1 (16 bit).</td>
</tr>
<tr>
<td>PlaybackSoundType</td>
<td>0x0</td>
<td>UB[1], Number of playback channels:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mono or stereo.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = sndMono</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = sndStereo</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
The SoundStreamHead2 tag is identical to the SoundStreamHead tag, except it allows different values for StreamSoundCompression and StreamSoundSize (SWF 3 file format).

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>0x0</td>
<td>RECORDHEADER. Tag type = 45</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x0</td>
<td>Always zero.</td>
</tr>
<tr>
<td>PlaybackSoundRate</td>
<td>0x0</td>
<td>Playback sampling rate.</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>0 = 5.5 kHz</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>1 = 11 kHz</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>2 = 22 kHz</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>3 = 44 kHz</td>
</tr>
<tr>
<td>PlaybackSoundSize</td>
<td>0x0</td>
<td>Playback sample size.</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>0 = 8-bit</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>1 = 16-bit</td>
</tr>
<tr>
<td>PlaybackSoundType</td>
<td>0x0</td>
<td>Number of playback channels.</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>0 = sndMono</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>1 = sndStereo</td>
</tr>
<tr>
<td>StreamSoundCompression</td>
<td>0x0</td>
<td>Format of streaming sound data.</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>For explanation of 0 vs. 3, see DefineSound.</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>0 = uncompressed</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>1 = ADPCM</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>SWF 4 or later only:</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>2 = MP3</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>3 = uncompressed little-endian</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>SWF 6 or later only:</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>6 = Nellymoser</td>
</tr>
<tr>
<td>StreamSoundRate</td>
<td>0x0</td>
<td>The sampling rate of the streaming sound data.</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>5.5 kHz is not allowed for MP3.</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>0 = 5.5 kHz</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>1 = 11 kHz</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>2 = 22 kHz</td>
</tr>
<tr>
<td></td>
<td>0x0</td>
<td>3 = 44 kHz</td>
</tr>
</tbody>
</table>
The PlaybackSoundRate, PlaybackSoundSize, and PlaybackSoundType fields are advisory only; Flash Player may ignore them.

The SoundStreamBlock tag defines sound data that is interleaved with frame data so that sounds can be played as the SWF file is streamed over a network connection. The SoundStreamBlock tag must be preceded by a SoundStreamHead or SoundStreamHead2 tag. There may only be one SoundStreamBlock tag per SWF frame. The minimum file format version is SWF 1.

### StreamSoundSize

- **Type**: UB[1]
- **Size of each sample. Always 16 bit for compressed formats. May be 8 or 16 bit for uncompressed formats.**
  - 0 = 8-bit
  - 1 = 16-bit

### StreamSoundType

- **Type**: UB[1]
- **Number of channels in the streaming sound data.**
  - 0 = sndMono
  - 1 = sndStereo

### StreamSoundSampleCount

- **Type**: UI16
- **Average number of samples in each SoundStreamBlock. Not affected by mono/stereo setting; for stereo sounds this is the number of sample pairs.**

### LatencySeek

- **Type**: SI16
- **If StreamSoundCompression = 2**
- **Otherwise absent**
  - See MP3 sound data. The value here should match the SeekSamples field in the first SoundStreamBlock for this stream.

<table>
<thead>
<tr>
<th>SoundStreamBlock Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Tag type = 19.</td>
</tr>
<tr>
<td>StreamSoundData</td>
<td>Compressed sound data.</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
214 Sounds

The contents of StreamSoundData vary depending on the value of the StreamSoundCompression field in the SoundStreamHead tag:

- If StreamSoundCompression is 0 or 3, StreamSoundData contains raw, uncompressed samples.
- If StreamSoundCompression is 1, StreamSoundData contains an ADPCM sound data record.
- If StreamSoundCompression is 2, StreamSoundData contains an MP3 sound data record.
- If StreamSoundCompression is 6, StreamSoundData contains a NELLYMOSERDATA record.

Frame subdivision for streaming sound

The best streaming sound playback is obtained by providing a SoundStreamBlock tag in every SWF frame, and including the same number of sound samples in each SoundStreamBlock.

The ideal number of samples per SWF frame is easily determined: divide the sampling rate by the SWF frame rate. If this results in a non-integer number, write an occasional SoundStreamBlock with one more or one fewer samples, so that the average number of samples per frame remains as close as possible to the ideal number.

For uncompressed audio, it is possible to include an arbitrary number of samples in a SoundStreamBlock, so an ideal number of samples can be included in each SWF frame. For MP3 sound, the situation is different. MP3 data is itself organized into frames, and an MP3 frame contains a fixed number of samples (576 or 1152, depending on the sampling rate). SoundStreamBlocks containing MP3 data must contain whole MP3 frames rather than fragments, so a SoundStreamBlock with MP3 data always contains a number of samples that is a multiple of 576 or 1152.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
There are two requirements for keeping MP3 streaming sound in sync with SWF playback:

■ Distribute MP3 frames appropriately among SWF frames.
■ Provide appropriate SeekSamples values in SoundStreamBlock tags.

These techniques are described in the rest of this section.

For streaming ADPCM sound, the logic for distributing ADPCM packets among SWF frames is identical to distributing MP3 frames among SWF frames. However, for ADPCM sound, there is no concept of SeekSamples or latency. For this and other reasons, MP3 is a preferable format for SWF 4 or later files.

To determine the ideal number of MP3 frames for each SWF frame, divide the ideal number of samples per SWF frame by the number of samples per MP3 frame. This will usually result in a non-integer number. Achieve the ideal average by interleaving SoundStreamBlocks with different numbers of MP3 frames. For example, at a SWF frame rate of 12 and a sampling rate of 11 kHz, there are 576 samples per MP3 frame; the ideal number of MP3 frames per SWF frame is \((11025 / 12) / 576\), roughly 1.6; this can be achieved by writing SoundStreamBlocks with one or two MP3 frames. While writing SoundStreamBlocks, keep track of the difference between the ideal number of total samples and the total number of samples written so far. Put as many MP3 frames in each SoundStreamBlock as is possible without exceeding the ideal number. Then, in each SoundStreamBlock, use the difference between the ideal and actual numbers of samples as of the end of the prior SWF frame as the value of SeekSamples. This will enable Flash Player to begin sound playback at exactly the right point after a seek occurs.

Here is an illustration of this example:

The SoundStreamBlock in SWF Frame 1 contains one MP3 frame and has SeekSamples set to zero. Frame 2 contains two MP3 frames and has SeekSamples set to \(919 - 576 = 343\). Frame 3 contains one MP3 frame and has SeekSamples set to \(1838 - 1728 = 110\).

In continuous playback, Flash Player will string all of the MP3 frames together and play them at their natural sample rate, reading ahead in the SWF bitstream to build up a buffer of sound data (this is why it is acceptable to include less than the ideal number of samples in a SWF frame). After a seek to a particular frame, such as is prompted by an ActionGotoFrame, Flash Player will skip the number of samples indicated by SeekSamples. For example, after a seek to Frame 2, it will skip 343 samples of the SoundStreamBlock data from Frame 2, which will cause sound playback to begin at sample 919, the ideal value.
If the ideal number of MP3 frames per SWF frame is less than one, there will be SWF frames whose SoundStreamBlocks cannot accommodate any MP3 frames without exceeding the ideal number of samples. In this case, write a SoundStreamBlock with SampleCount = 0, SeekSamples = 0, and no MP3 data.

Some MP3 encoders have an initial latency, generating a number of silent or meaningless samples before the desired sound data begins. This can help the Flash Player MP3 decoder as well, providing some ramp-up data before the samples that are needed. In this situation, determine how many samples the initial latency occupies, and supply that number for SeekSamples in the first SoundStreamBlock. Flash Player will add this number to the SeekSamples for any other frame when performing a seek. Latency also affects the decision as to how many MP3 frames to put into a SoundStreamBlock. Here is a modification of the above example with a latency of 940 samples:

The SoundStreamBlock in SWF Frame 1 contains three MP3 frames, the maximum that can be accommodated without exceeding the ideal number of samples after adjusting for latency (represented by the leftward shift of the MP3 timeline above). The value of SeekSamples in Frame 1 is special; it represents the latency. Frame 2 contains one MP3 frame and has SeekSamples set to $919 - (1728 - 940) = 131$.

ADPCM compression

ADPCM (Adaptive Differential Pulse Code Modulation) is a family of audio compression and decompression algorithms. It is a simple but efficient compression scheme that avoids any licensing or patent issues that arise with more sophisticated sound compression schemes, and helps to keep player implementations small.

For SWF 4 or later files, MP3 compression is a preferable format. MP3 produces substantially better sound for a given bitrate.

ADPCM uses a modified Differential Pulse Code Modulation (DPCM) sampling technique where the encoding of a each sample is derived by calculating a “difference” (DPCM) value, and applying to this a complex formula which includes the previous quantization value. The result is a compressed code, which can recreate almost the same subjective audio quality.
A common implementation takes 16-bit linear PCM samples and converts them to 4-bit codes, yielding a compression rate of 4:1. Public domain C code written by Jack Jansen is available at ftp.cwi.nl/pub/audio/adpcm.zip.

The SWF file format extends Jansen’s implementation to support 2-, 3-, 4- and 5-bit ADPCM codes. When choosing a code size, there is the usual trade-off between file size and audio quality. The code tables used in SWF file format are as follows (note that each structure here provides only the unique lower half of the range, the upper half being an exact duplicate):

```c
int indexTable2[2] = {-1, 2};
int indexTable3[4] = {-1, -1, 2, 4};
int indexTable4[8] = {-1, -1, -1, -1, 2, 4, 6, 8};
int indexTable5[16] = {-1, -1, -1, -1, -1, -1, -1, -1, 1, 2, 4, 6, 8, 10, 13, 16};
```

ADPCM sound data

The ADPCMSOUNDATA record defines the size of the ADPCM codes used, and an array of ADPCMPACKETs which contain the ADPCM data.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdpcmCodeSize</td>
<td>UB[2]</td>
<td>0 = 2 bits/sample</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = 3 bits/sample</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = 4 bits/sample</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 = 5 bits/sample</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bits per ADPCM code less 2.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The actual size of each code is AdpcmCodeSize + 2.</td>
</tr>
<tr>
<td>AdpcmPackets</td>
<td>If SoundType = mono ADPCMMONOPACKET[one or more]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If SoundType = stereo ADPCMSTEREOPACKET[one or more]</td>
<td></td>
</tr>
</tbody>
</table>

Array of ADPCMPACKETs.
ADPCMPACKETs vary in structure depending on whether the sound is mono or stereo.

For an explanation of StepSizeTable, see the Jansen source code.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitialSample</td>
<td>First sample. Identical to first sample in uncompressed sound.</td>
</tr>
<tr>
<td>InitialIndex</td>
<td>Initial index into the ADPCM StepSizeTable.*</td>
</tr>
<tr>
<td>AdpcmCodeData</td>
<td>4096 ADPCM codes. Each sample is (AdpcmCodeSize + 2) bits.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitialSampleLeft</td>
<td>First sample for left channel. Identical to first sample in uncompressed sound.</td>
</tr>
<tr>
<td>InitialIndexLeft</td>
<td>Initial index into the ADPCM StepSizeTable* for left channel.</td>
</tr>
<tr>
<td>InitialSampleRight</td>
<td>First sample for right channel. Identical to first sample in uncompressed sound.</td>
</tr>
<tr>
<td>InitialIndexRight</td>
<td>Initial index into the ADPCM StepSizeTable* for right channel.</td>
</tr>
<tr>
<td>Channel data</td>
<td>8192 ADPCM codes per channel, total 8192. Each sample is (AdpcmCodeSize + 2) bits. Channel data is interleaved left, then right.</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
MP3 compression

MP3 is a sophisticated and complex audio compression algorithm supported in SWF 4 and later. It produces superior audio quality at better compression ratios than ADPCM. Generally speaking, MP3 refers to MPEG1 Layer 3; however, the SWF file format supports later versions of MPEG (V2 and 2.5) that were designed to support lower bitrates. Flash Player supports both CBR (constant bitrate) and VBR (variable bitrate) MP3 encoding. For more information on MP3, see www.mp3-tech.org/ and www.iis.fhg.de/amm/techinf/layer3/index.html. Writing an MP3 encoder is quite difficult, but public-domain MP3 encoding libraries may be available.

**NOTE**

Be aware that software and hardware MP3 encoders and decoders might have their own licensing requirements.

---

**NOTE**

For event sounds, SeekSamples is limited to specifying initial latency.

---

**Table: MP3 Sound Data**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeekSamples</td>
<td>SI16</td>
<td>Number of samples to skip.</td>
</tr>
<tr>
<td>Mp3Frames</td>
<td>MP3FRAME[zero or more]</td>
<td>Array of MP3 frames.</td>
</tr>
</tbody>
</table>

---

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader®or Adobe®Acrobat® version 6.0 or above.
The MP3FRAME record corresponds exactly to an MPEG audio frame that you would find in an MP3 music file. The first 32 bits of the frame contain header information, followed by an array of bytes, which are the encoded audio samples.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syncword</td>
<td>UB[11]</td>
</tr>
<tr>
<td>MpegVersion</td>
<td>UB[2]</td>
</tr>
<tr>
<td>Layer</td>
<td>UB[2]</td>
</tr>
<tr>
<td>ProtectionBit</td>
<td>UB[1]</td>
</tr>
</tbody>
</table>

- **Syncword**: UB[11]
- **MpegVersion**: 0 = MPEG Version 2.5, 1 = reserved, 2 = MPEG Version 2, 3 = MPEG Version 1
- **Layer**: 0 = reserved, 1 = Layer III, 2 = Layer II, 3 = Layer I
- **ProtectionBit**: 0 = Protected by CRC, 1 = Not protected

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
<table>
<thead>
<tr>
<th>Bitrate</th>
<th>MPEG1</th>
<th>MPEG2.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>free</td>
<td>free</td>
</tr>
<tr>
<td>1</td>
<td>32</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>48</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>56</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>64</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>80</td>
<td>48</td>
</tr>
<tr>
<td>7</td>
<td>96</td>
<td>56</td>
</tr>
<tr>
<td>8</td>
<td>112</td>
<td>64</td>
</tr>
<tr>
<td>9</td>
<td>128</td>
<td>80</td>
</tr>
<tr>
<td>10</td>
<td>160</td>
<td>96</td>
</tr>
<tr>
<td>11</td>
<td>192</td>
<td>112</td>
</tr>
<tr>
<td>12</td>
<td>224</td>
<td>128</td>
</tr>
<tr>
<td>13</td>
<td>256</td>
<td>144</td>
</tr>
<tr>
<td>14</td>
<td>320</td>
<td>160</td>
</tr>
<tr>
<td>15</td>
<td>bad</td>
<td>bad</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sampling Rate</th>
<th>MPEG1</th>
<th>MPEG2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>44100</td>
<td>22050</td>
</tr>
<tr>
<td>1</td>
<td>48000</td>
<td>24000</td>
</tr>
<tr>
<td>2</td>
<td>32000</td>
<td>16000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PaddingBit</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>frame is not padded</td>
</tr>
<tr>
<td>1</td>
<td>frame is padded with one extra slot</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reserved</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The size of the sample data is calculated like this (using integer arithmetic):

\[
\text{Size} = \left( \frac{(MpegVersion == MPEG1 ? 144 : 72) \times \text{Bitrate}}{\text{SamplingRate}} \right) + \text{PaddingBit} - 4
\]

For example: The size of the sample data for an MPEG1 frame with a Bitrate of 128000, a SamplingRate of 44100, and PaddingBit of 1 is:

\[
\text{Size} = \left( \frac{144 \times 128000}{44100} \right) + 1 - 4 = 414 \text{ bytes}
\]

Nellymoser compression

Starting with SWF 6, a compressed sound format called Nellymoser Asao is available. This is a single-channel (mono) format optimized for low-bitrate transmission of speech. The format was developed by Nellymoser Inc. at www.nellymoser.com.

A summary of the Nellymoser Asao encoding process is provided here. For full details of the Asao format, contact Nellymoser.

ChannelMode

- 0 = Stereo
- 1 = Joint stereo (Stereo)
- 2 = Dual channel
- 3 = Single channel (Mono)

ModeExtension

Copyright

- 0 = Audio is not copyrighted
- 1 = Audio is copyrighted

Original

- 0 = Copy of original media
- 1 = Original media

Emphasis

- 0 = none
- 1 = 50/15 ms
- 2 = reserved
- 3 = CCIT J.17

SampleData

The encoded audio samples.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Asao uses frequency-domain characteristics of sound for compression. Sound data is grouped
into frames of 256 samples. Each frame is converted into the frequency domain and the most
significant (highest-amplitude) frequencies are identified. A number of frequency bands are
selected for encoding; the rest are discarded. The bitstream for each frame then encodes which
frequency bands are in use and what their amplitudes are.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
CHAPTER 14

Buttons

Button characters in the Macromedia Flash SWF 8 file format serve as interactive elements. They can react programmatically to events that occur. The most common event to handle is a simple click from the mouse, but more complex events can be trapped as well.

Button states

A button object can be displayed in one of three states: up, over, or down. The up state is the default appearance of the button. The up state is displayed when the Flash file starts playing, and whenever the mouse is outside the button. The over state is displayed when the mouse is moved inside the button. This allows rollover or hover buttons in a Flash file. The down state is the clicked appearance of the button. It is displayed when the mouse is clicked inside the button.

A fourth state—the hit state—defines the active area of the button. This is an invisible state and is never displayed. It defines the area of the button that reacts to mouse clicks. This hit area is not necessarily rectangular and need not reflect the visible shape of the button.

Each state is made up of a collection of instances of characters from the dictionary. Each such instance is defined using a Button record, which, within a button definition, acts like a PlaceObject tag. For the up, over, and down states, these characters are placed on the display list when the button enters that state. For the hit-area state, these characters define the active area of the button.

The following is an example of a typical button and its four states. The button is initially blue. When the mouse is moved over the button, it changes to a purple color. When the mouse is pressed inside the button, the shading changes to simulate a depressed button. The fourth state—the hit area—is a simple rectangle. Anything outside this shape is outside the button, and anything inside this shape is inside the button.
The SWF file format has no native support for radio buttons or check boxes. There is no "checked" (selected) state, and buttons cannot "stick" down after the mouse is released. In addition, there is no way to group buttons into mutually exclusive choices. However, both of these behaviors can be simulated by using button actions.

**Button tracking**

Button tracking refers to how a button behaves as it tracks the movement of the mouse. A button object can track the mouse in one of two modes:

1. As a push button.
2. As a menu button.

If a push button is clicked, all mouse movement events are directed to the push button until the mouse button is released. This is called capturing the mouse. For example, if you click a push button and drag outside the button (without releasing), the button changes to the over state, and the pointer remains a pointing hand.

Menu buttons do not capture the mouse. If you click a menu button and drag outside, the button changes to the up state, and the pointer reverts to an arrow.

**Events, state transitions, and actions**

A button object can perform an action whenever a state transition occurs (that is, when the button changes from one state to another). A state transition occurs in response to some event, such as a mouse click, or mouse entering the button. In the SWF file format, events are described as state transitions. The following table shows possible state transitions and corresponding Flash events:

<table>
<thead>
<tr>
<th>State Transition</th>
<th>Flash Event</th>
<th>Description</th>
<th>Visual Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>IdleToOverUp</td>
<td>Roll Over</td>
<td>Mouse enters the hit area while the mouse button is up. Button changes from up to over state.</td>
<td></td>
</tr>
<tr>
<td>OverUpToIdle</td>
<td>Roll Out</td>
<td>Mouse leaves the hit area while the mouse button is up. Button changes from over to up state.</td>
<td></td>
</tr>
</tbody>
</table>
The following transitions only apply when tracking Push buttons:

<table>
<thead>
<tr>
<th>State Transition</th>
<th>Flash Event</th>
<th>Description</th>
<th>Visual Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>OverUpToOverDown</td>
<td>Press Mouse button is pressed while the mouse is inside the hit area.</td>
<td>Button changes from over to down state.</td>
<td></td>
</tr>
<tr>
<td>OverDownToOverUp</td>
<td>Release Mouse button is released while the mouse is inside the hit area.</td>
<td>Button changes from down to over state.</td>
<td></td>
</tr>
<tr>
<td>OutDownToIdle</td>
<td>Release Outside Mouse button is released outside the hit area while the mouse is captured.</td>
<td>Button changes from over to up state.</td>
<td></td>
</tr>
<tr>
<td>IdleToOverDown</td>
<td>Drag Over Mouse is dragged inside the hit area while the mouse button is down.</td>
<td>Button changes from up to down state.</td>
<td></td>
</tr>
<tr>
<td>OverDownToIdle</td>
<td>Drag Out Mouse is dragged outside the hit area while the mouse button is down.</td>
<td>Button changes from down to up state.</td>
<td></td>
</tr>
</tbody>
</table>

Often button actions are performed only on OverDownToOverUp (when the mouse button is released), but DefineButton2 allows actions to be triggered by any state transition.

A button object can perform any action supported by the SWF 3 actions.

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
A button record defines a character to be displayed in one or more button states. The ButtonState flags indicate which state (or states) the character belongs to. A one-to-one relationship does not exist between button records and button states. A single button record can apply to more than one button state (by setting multiple ButtonState flags), and multiple button records can be present for any button state.

Each button record also includes a transformation matrix and depth (stacking-order) information. These apply just as in a PlaceObject tag, except that both pieces of information are relative to the button character itself.

SWF 8 and later supports the new ButtonHasBlendMode and ButtonHasFilterList fields to support blend modes and bitmap filters on buttons. Flash Player 7 and earlier ignores these two fields.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
<th>Field Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ButtonReserved</td>
<td>Reserved bits; always 0</td>
<td>ButtonReserved</td>
<td>UB[2]</td>
</tr>
<tr>
<td>ButtonHasBlendMode</td>
<td>0 = No blend mode</td>
<td>ButtonHasBlendMode</td>
<td>UB[1]</td>
</tr>
<tr>
<td>ButtonHasFilterList</td>
<td>1 = Has filter list</td>
<td>ButtonHasFilterList</td>
<td>UB[1]</td>
</tr>
<tr>
<td>ButtonStateHitTest</td>
<td>Present in hit test state</td>
<td>ButtonStateHitTest</td>
<td>UB[1]</td>
</tr>
<tr>
<td>ButtonStateDown</td>
<td>Present in down state</td>
<td>ButtonStateDown</td>
<td>UB[1]</td>
</tr>
<tr>
<td>ButtonStateOver</td>
<td>Present in over state</td>
<td>ButtonStateOver</td>
<td>UB[1]</td>
</tr>
<tr>
<td>ButtonStateUp</td>
<td>Present in up state</td>
<td>ButtonStateUp</td>
<td>UB[1]</td>
</tr>
<tr>
<td>CharacterID</td>
<td>ID of character to place</td>
<td>CharacterID</td>
<td>UI16</td>
</tr>
<tr>
<td>PlaceDepth</td>
<td>Depth at which to place</td>
<td>PlaceDepth</td>
<td>UI16</td>
</tr>
<tr>
<td>PlaceMatrix</td>
<td>Transformation matrix for character</td>
<td>PlaceMatrix</td>
<td>MATRIX</td>
</tr>
<tr>
<td>ColorTransform</td>
<td>Character color transform</td>
<td>ColorTransform</td>
<td>CXFORMWITHALPHA</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
The DefineButton tag defines a button character for later use by control tags such as PlaceObject. DefineButton includes an array of Button records that represent the four button shapes: an up character, a mouse-over character, a down character, and a hit-area character. It is not necessary to define all four states, but at least one button record must be present. For example, if the same button record defines both the up and over states, only three button records are required to describe the button.

More than one button record per state is allowed. If two button records refer to the same state, both are displayed for that state.

DefineButton also includes an array of ACTIONRECORDs, which are performed when the button is clicked and released (see SWF 3 actions).

FILTERLIST

If within DefineButton2 and ButtonHasFilterList = 1,

List of filters on this button

BlendMode

If within DefineButton2 and ButtonHasBlendMode = 1,

UI8

0 or 1 = normal
2 = layer
3 = multiply
4 = screen
5 = lighten
6 = darken
7 = add
8 = subtract
9 = difference
10 = invert
11 = alpha
12 = erase
13 = overlay
14 = hardlight
Values 15 to 255 are reserved.
The minimum file format version is SWF 1.

DefineButton2 extends the capabilities of DefineButton by allowing any state transition to trigger actions.

The minimum file format version is SWF 3:

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORDHEADER, Tag type = 7</td>
</tr>
<tr>
<td>ButtonId</td>
<td>UI16, ID for this character</td>
</tr>
<tr>
<td>Characters</td>
<td>BUTTONRECORD[one or more]</td>
</tr>
<tr>
<td>CharacterEndFlag</td>
<td>UI8, Must be 0</td>
</tr>
<tr>
<td>Actions</td>
<td>ACTIONRECORD[zero or more]</td>
</tr>
<tr>
<td>ActionEndFlag</td>
<td>UI8, Must be 0</td>
</tr>
</tbody>
</table>

DefineButton2

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORDHEADER, Tag type = 34</td>
</tr>
<tr>
<td>ButtonId</td>
<td>UI16, ID for this character</td>
</tr>
<tr>
<td>ReservedFlags</td>
<td>UB[7], Always 0</td>
</tr>
<tr>
<td>TrackAsMenu</td>
<td>UB[1], 0 = track as normal button, 1 = track as menu button</td>
</tr>
<tr>
<td>ActionOffset</td>
<td>UI16, Offset in bytes from start of this field to the first BUTTONCONDACTION, or 0 if no actions occur</td>
</tr>
<tr>
<td>Characters</td>
<td>BUTTONRECORD[one or more]</td>
</tr>
<tr>
<td>CharacterEndFlag</td>
<td>UI8, Must be 0</td>
</tr>
<tr>
<td>Actions</td>
<td>BUTTONCONDACTION[zero or more]</td>
</tr>
<tr>
<td></td>
<td>Actions to execute at particular button events</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The actions associated with DefineButton2 are specified as follows:

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CondActionSize</td>
<td>Offset in bytes from start of this field to next BUTTONCONDACTION, or 0 if last action.</td>
</tr>
<tr>
<td>CondIdleToOverDown</td>
<td>UB[1] Idle to OverDown</td>
</tr>
<tr>
<td>CondOutDownToIdle</td>
<td>UB[1] OutDown to Idle</td>
</tr>
<tr>
<td>CondOutDownToOverDown</td>
<td>UB[1] OutDown to OverDown</td>
</tr>
<tr>
<td>CondOverDownToOutDown</td>
<td>UB[1] OverDown to OutDown</td>
</tr>
<tr>
<td>CondOverDownToOverUp</td>
<td>UB[1] OverDown to OverUp</td>
</tr>
<tr>
<td>CondOverUpToOverDown</td>
<td>UB[1] OverUp to OverDown</td>
</tr>
<tr>
<td>CondOverUpToIdle</td>
<td>UB[1] OverUp to Idle</td>
</tr>
<tr>
<td>CondIdleToOverUp</td>
<td>UB[1] Idle to OverUp</td>
</tr>
<tr>
<td>CondKeyPress</td>
<td>UB[7] SWF 4 or later: key code</td>
</tr>
<tr>
<td></td>
<td>Otherwise: always 0</td>
</tr>
</tbody>
</table>

Valid key codes:
- 1 = left arrow
- 2 = right arrow
- 3 = home
- 4 = end
- 5 = insert
- 6 = delete
- 8 = backspace
- 13 = enter
- 14 = up arrow
- 15 = down arrow
- 16 = page up
- 17 = page down
- 18 = tab
- 19 = escape
- 32 to 126: follows ASCII
For each event handler (each BUTTONCONDACTION), one or more of the Cond bit fields should be filled in. This specifies when the event handler should be executed. CondKeyPress specifies a particular key to trap. A CondKeyPress event handler is executed even if the button that it applies to does not have input focus. For the 32 to 126 ASCII key codes, the key event that is trapped is composite—it takes into account the effect of the Shift key. To trap raw key events, corresponding directly to keys on the keyboard (including the modifier keys themselves), use clip event handlers instead.

DefineButtonCxform

DefineButtonCxform defines the color transform for each shape and text character in a button. This is not used for DefineButton2, which includes its own CXFORM. The minimum file format version is SWF 2.

DefineButtonSound

DefineButtonSound tag defines which sounds (if any) are played on state transitions. The minimum file format version is SWF 2.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORDHEADER</td>
</tr>
<tr>
<td>ButtonId</td>
<td>UI16</td>
</tr>
<tr>
<td>ButtonColorTransform</td>
<td>CXFORM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORDHEADER</td>
</tr>
<tr>
<td>ButtonId</td>
<td>UI16</td>
</tr>
<tr>
<td>ButtonSoundChar0</td>
<td>UI16</td>
</tr>
<tr>
<td>ButtonSoundInfo0</td>
<td>SOUNDINFO (if ButtonSoundChar0 is nonzero)</td>
</tr>
<tr>
<td>ButtonSoundChar1</td>
<td>UI16</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
CHAPTER 15

Sprites and Movie Clips

A sprite corresponds to a movie clip in the Macromedia Flash authoring application. It is a Macromedia Flash SWF file contained within a SWF file, and supports many of the features of a regular Flash file, such as the following:

- Most of the control tags that can be used in the main file.
- A timeline that can stop, start, and play independently of the main file.
- A streaming soundtrack that is automatically mixed with the main soundtrack.

A sprite object is defined with a DefineSprite tag. It consists of a character ID, a frame count, and a series of control tags. Definition tags (such as DefineShape) are not allowed in the DefineSprite tag. All of the characters that control tags refer to in the sprite must be defined outside the sprite, and before the DefineSprite tag.

Once defined, a sprite is displayed with a PlaceObject2 tag in the main file. The transform (specified in PlaceObject) is concatenated with the transforms of objects placed inside the sprite. These objects behave like children of the sprite, so when the sprite is moved, the objects inside the sprite move too. Similarly, when the sprite is scaled or rotated, the child objects are also scaled or rotated. A sprite object stops playing automatically when it is removed from the display list.
Sprite names

When a sprite is placed on the display list, it can be given a name with the PlaceObject2 tag. This name is used to identify the sprite so that the main file (or other sprites) can perform actions inside the sprite. This is achieved with the SetTarget action (see ActionSetTarget).

For example, say a sprite object is placed in the main file with the name "spinner". The main file can send this sprite to the first frame in its timeline with the following action sequence:

1. SetTarget "spinner"
2. GotoFrame zero
3. SetTarget ""

Note: All actions following SetTarget apply to the spinner object until SetTarget "", which sets the action context back to the main file.

The SWF file format supports placing sprites within sprites, which can lead to complex hierarchies of objects. To handle this complexity, the SWF file format uses a naming convention similar to that used by file systems to identify sprites. For example, the following outline shows four sprites defined within the main file:

MainMovie.swf
- SpriteA (name: Jack)
  - SpriteA1 (name: Bert)
  - SpriteA2 (name: Ernie)
- SpriteB (name: Jill)

The following SetTarget paths identify the preceding sprites:

- /Jack targets SpriteA from the main file.
- ../ targets the main file from SpriteA.
- /Jack/Bert targets SpriteA1 from any other sprite or the main file.
- Bert targets SpriteA1 from SpriteA.
- ../Ernie targets SpriteA2 from SpriteA1.
- ../../Jill targets SpriteB from SpriteA1.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The DefineSprite tag defines a sprite character. It consists of a character ID and a frame count, followed by a series of control tags. The sprite is terminated with an End tag.

The length specified in the Header reflects the length of the entire DefineSprite tag, including the ControlTags field.

Definition tags (such as DefineShape) are not allowed in the DefineSprite tag. All of the characters that control tags refer to in the sprite must be defined in the main body of the file before the sprite is defined.

The minimum file format version is SWF 3.

The following tags are valid within a DefineSprite tag:

- ShowFrame
- StartSound
- PlaceObject
- FrameLabel
- PlaceObject2
- SoundStreamHead
- RemoveObject
- SoundStreamHead2
- RemoveObject2
- SoundStreamBlock
- All Actions (see Actions)
- End

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
CHAPTER 16

Video

The Macromedia Flash Player 6 and later supports video playback. Video can be provided to Flash Player in the following ways:

■ Embed video within a Macromedia Flash SWF file by using the SWF video tags.
■ Deliver a video stream over RTMP through the Macromedia Flash Communication Server, which, as one option, can obtain the video data from an FLV file format file.
■ Load an FLV file directly into Flash Player by using the NetStream.play ActionScript method. This method is only available in Flash Player 7 and later. The SWF and FLV file formats share a common video encoding format.

Please see the FLV File Format chapter for complete information about the FLV file format.

Sorenson H.263 Bitstream Format

As of SWF 6, a single video format, called Sorenson H.263, is available. This format is based on H.263, an open video encoding standard that is maintained by the ITU. Copies of the H.263 standard can be obtained at www.itu.int/.

All references to the H.263 standard in this document refer to the draft version of H.263, dated May 1996, sometimes referred to as H.263v1. This is distinct from the revised version of H.263, dated February 1998, sometimes referred to as H.263v2 or H.263+, and currently the in-force version of H.263 according to the ITU.

The Sorenson H.263 video format differs slightly from H.263. For the most part, it is a subset of H.263, with some advanced features removed and a few additions. These changes are described in this section.

The Sorenson H.263 video format was developed by Sorenson Media (www.sorenson.com). Existing products that can produce Flash video are the Macromedia Flash MX authoring application, and Sorenson Squeeze for Macromedia Flash MX, a professional video compression application. You can license the Sorenson Spark codec to perform Flash video encoding; contact Sorenson Media for details.
Summary of differences from H.263

The following H.263 features are removed from the Sorenson H.263 video format:

- GOB (group of blocks) layer
- Split-screen indicator
- Document camera indicator
- Picture freeze release
- Syntax-based arithmetic coding
- PB frames
- Continuous-presence multipoint
- Overlapped block-motion compensation

The following non-H.263 features are added to the Sorenson H.263 video format:

- Disposable frames (difference frames with no future dependencies)
- Arbitrary picture width and height up to 65535 pixels
- Unrestricted motion vector support is always on
- A deblocking flag is available to suggest the use of a deblocking filter

To support these differences, the Sorenson H.263 video format uses different headers than H.263 at both the picture layer and the Macroblock layer. The GOB layer is absent.

Two versions of the Sorenson H.263 video format are defined. In version 0, the block layer is identical to H.263. In version 1, escape codes in transform coefficients are encoded differently than in H.263. Version 0 and version 1 have no other differences.

Video packet

The video packet is the top-level structural element in a Sorenson H.263 video packet. It corresponds to the picture layer in H.263 section 5.1. This structure is included within the VideoFrame tag in the SWF file format, and also within the VIDEODATA structure in the FLV file format.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>PictureStartCode</td>
<td>Similar to H.263 5.1.1</td>
</tr>
<tr>
<td>Version</td>
<td>Video format version</td>
</tr>
<tr>
<td>TemporalReference</td>
<td>See H.263 5.1.2</td>
</tr>
</tbody>
</table>
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The macro block is the next layer down in the video structure. It corresponds to the macro block layer in H.263 section 5.3.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodedMacroblockFlag</td>
<td>UB[1]</td>
</tr>
<tr>
<td>MacroblockType</td>
<td>varies</td>
</tr>
<tr>
<td>BlockPattern</td>
<td>varies</td>
</tr>
<tr>
<td>QuantizerInformation</td>
<td>UB[2]</td>
</tr>
<tr>
<td>MotionVectorData</td>
<td>varies[2]</td>
</tr>
<tr>
<td>ExtraMotionVectorData</td>
<td>varies[6]</td>
</tr>
<tr>
<td>BlockData</td>
<td>BLOCKDATA[6]</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Block data is the lowest layer in the video structure. In version 0 of the Sorenson H.263 video format, this layer follows H.263 section 5.4 exactly.

In version 1 of the Sorenson H.263 video format, escape codes in transform coefficients (see H.263 section 5.4.2) are encoded differently. When the ESCAPE code 0000 011 appears, the next bit is a format bit that indicates the subsequent bit layout for LAST, RUN, and LEVEL. In both cases, one bit is used for LAST and six bits are used for RUN. If the format bit is 0, seven bits are used for LEVEL; if the format bit is 1, eleven bits are used for LEVEL. The 7-bit and 11-bit LEVEL tables, which replace table 14 in H.263, as the following table shows:

<table>
<thead>
<tr>
<th>Index</th>
<th>Level Code</th>
<th>Index</th>
<th>Level Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>-64</td>
<td>FORBIDDEN</td>
<td>-1024</td>
<td>FORBIDDEN</td>
</tr>
<tr>
<td>0</td>
<td>1000 001</td>
<td>0</td>
<td>FORBIDDEN</td>
</tr>
<tr>
<td>1</td>
<td>0000 001</td>
<td>1023</td>
<td>0000 0000</td>
</tr>
<tr>
<td>2</td>
<td>0000 010</td>
<td>1024</td>
<td>0000 0000</td>
</tr>
<tr>
<td>63</td>
<td>1023 0111</td>
<td>1023</td>
<td>0111 1111 111</td>
</tr>
<tr>
<td>64</td>
<td>1024 0110</td>
<td>1024</td>
<td>0111 1111 111</td>
</tr>
</tbody>
</table>

As of SWF 7, an additional video format, called screen video, is available. Screen video is a simple lossless sequential-bitmap format with blocked interframing. It is designed for sending captures of computer screens in action.

Pixel data in the screen video format is compressed by using the ZLIB open standard.
Each frame in a screen video sequence is formatted as a series of blocks. These blocks form a grid over the image. In a keyframe, every block is sent. In an interframe, one or more blocks contain no data, which indicates that the bitmap region represented by that block was not changed since the last keyframe.

Blocks have width and height that range from 16 to 256 in multiples of 16. Block height is not required to match block width. The block size must not change except at a keyframe. Blocks are ordered from the bottom left of the image to the top right, in rows. A fixed layout of blocks exists for any given combination of block size and image size. To determine the number of blocks in a row of the grid, divide the image width by the block width. If the result is not an integer, the end of each row has one partial block, which contains only the number of pixels necessary to fill the remaining width of the image. The same logic applies to the image height, block height, number of rows in the grid, and partial blocks in the final row. It is important to understand the partial-block algorithm to create correct blocks, since the pixels within a partial block are extracted with implicit knowledge of the width and height of the block.

The following is an example of blocking. The image in this example is 120 x 80 pixels, and the block size is 32 x 32.

Video packet

The video packet is the top-level structural element in a screen video packet. This structure is included within the VideoFrame tag in the SWF file format, and also within the VIDEODATA structure in the FLV file format.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
Screen Video Version 2 uses several techniques to reduce the amount of data for each screen block.

In the initial Screen Video version, each block of screen data is a complete buffer of compressed data that can be decompressed to a full 24-bit color image for that block. In the Screen Video V2 format, the screen data blocks can be incomplete updates of the image area, similar to the concept of keyframes and interframes.

In the Screen Video V2 format, block data comes in two types:

- **Keyblock**
  - contains complete information for the block. The contents can be decompressed to obtain the complete block image.

- **Interblock**
  - requires additional data, from a previous image or another block, to construct the full block image.

### Table: Image Block Field Types

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSize</td>
<td>Size of the compressed block data that follows. If this is an interframe, and this block is not changed since the last keyframe, DataSize is 0 and the Data field is absent.</td>
</tr>
<tr>
<td>Data</td>
<td>If DataSize &gt; 0, compressed pixel data using ZLIB. Pixels are ordered from bottom left to top right in rows. Each pixel is three bytes: B, G, R.</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Video Packet V2 is the top-level structural element in a screen video packet for Screen Video Version 2. This structure is included within the VideoFrame tag in SWF file format, and also within the VIDEODATA structure in FLV file format.

The data consists of information about the image sub-block dimensions and grid size, followed by the data for each block.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlockWidth</td>
<td>Pixel width of each block in the grid. This value is stored as (actualWidth / 16) - 1, so possible block sizes are a multiple of 16 and not more than 256.</td>
</tr>
<tr>
<td>ImageWidth</td>
<td>Pixel width of the full image.</td>
</tr>
<tr>
<td>BlockHeight</td>
<td>Pixel height of each block in the grid. This value is stored as (actualHeight / 16) - 1, so possible block sizes are a multiple of 16 and not more than 256.</td>
</tr>
<tr>
<td>ImageHeight</td>
<td>Pixel height of the full image.</td>
</tr>
<tr>
<td>Reserved</td>
<td>Must be 0</td>
</tr>
<tr>
<td>HasIFrameImage</td>
<td>If 1, has IFrameImage</td>
</tr>
<tr>
<td>HasPaletteInfo</td>
<td>If 1, has PaletteInfo</td>
</tr>
<tr>
<td>PaletteInfo</td>
<td>If HasPaletteInfo, IMAGEBLOCK</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The Image Block V2 structure represents one block in a frame. Blocks of image data. See Block format for details of how to calculate n.

Blocks are ordered from bottom left to top right in rows and can be a combination of keyblocks and interblocks.

If HasIFrameImage, Blocks of image data representing interblocks that must be combined with the previous keyblocks to produce the image. See Block format for details of how to calculate n. Blocks are ordered from bottom left to top right in rows.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSize</td>
<td>UB[16]</td>
</tr>
<tr>
<td>Format</td>
<td>IMAGEFORMAT</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The IMAGEFORMAT byte describes the color depth and compression of the IMAGEBLOCKV2 structure.

If Format’s HasDiffBlock = 1, IMAGEDIFFPOSITION

If Format’s ZlibPrimeCompressCurrent = 1, IMAGEPRIMEPOSITION

Describes the format and compression of Data

If DataSize > 0, UI8[DataSize]

Pixel data compressed using ZLIB. Pixels are ordered from bottom left to top right in rows. Each pixel is three bytes: B, G, R.

IMAGEFORMAT
Field Type Comment
Reserved UB[3] Must be 0
ColorDepth UB[2] 00: 24-bit RGB image
10: 15/7-bit hybrid color image
HasDiffBlocks UB[1] If 1, the data starts and stops on specific rows within the block and does not represent the entire block.
ZlibPrimeCompressCurrent UB[1] If 1, the current data block was produced with the ZLIB priming technique of compression.
ZlibPrimeCompressPrevious UB[1] If 1, the previous data block was produced with the ZLIB priming technique of compression.

IMAGEBLOCKV2
Field Type Comment
Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Like the Sorenson H.263 video format, the On2 Truemotion VP6 video format uses color information encoded in the YCbCr color space described in the ITU-R BT.601 standard. This color information is stored as YUV 4:2:2 using unsigned 8-bit values for each color component. The following algorithm can be used to convert forward from RGB color space pixel data to YUV color space:

FUNCTION SATURATE(A) = MIN(255, MAX(0, A))

Y = SATURATE(+ (0.257 * R) + (0.504 * G) + (0.098 * B) + 16)

U = SATURATE(- (0.148 * R) - (0.291 * G) + (0.439 * B) + 128)

V = SATURATE(+ (0.439 * R) - (0.368 * G) - (0.071 * B) + 128)

This algorithm can be used to convert YUV color space pixel data back to RGB color space:

B = SATURATE(1.164(Y - 16) + 2.018(U - 128))

G = SATURATE(1.164(Y - 16) - 0.813(V - 128) - 0.391(U - 128))

R = SATURATE(1.164(Y - 16) + 1.596(V - 128))

In addition to supporting standard On2 Truemotion VP6 video streams, Flash Player 8 adds support for an extra alpha channel that is used to simulate transparency. The alpha channel is encoded by using a second On2 Truemotion VP6 stream that contains the alpha channel information. To encode this type of video stream, the preceding RGB-to-YUV algorithm should be used on the RGB color channels of the premultiplied ARGB color space pixel data.

With the resulting YUV color space data, the video data can be encoded as the first video stream.

For the second video stream, the following algorithm can be used to obtain the YUV video data from the alpha channel of the premultiplied ARGB color space pixel data:

Y = A

U = 0

V = 0

To decode alpha channel video streams, assume that the first video stream returns YUV-encoded color channels for a screen pixel in the form of three channels named Y1, U1, and V1. The second video stream returns this data as Y2, U2, and V2. The resulting alpha premultiplied ARGB pixel values are obtained by using the following algorithm:

B = MIN(Y2, SATURATE(1.164(Y1 - 16) + 2.018(U1 - 128)))

G = MIN(Y2, SATURATE(1.164(Y1 - 16) - 0.813(V1 - 128) - 0.391(U1 - 128)))

R = MIN(Y2, SATURATE(1.164(Y1 - 16) + 1.596(V1 - 128)))

A = Y2

The U2 and V2 channels are not currently used at decode time.

NOTE

At encode time, the second video stream must contain at least as many key frames as the first video stream. Each key frame occurring in the first video stream must force a key frame in the second video stream at encode time so that the combined video stream stays seekable.
The VP6 FLV video packet represents a VP6 video frame within an FLV file.

The VP6 FLV Alpha video packet represents a VP6 video frame with alpha channel information within FLV files.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>HorizontalAdjustment</td>
<td>Number of pixels to subtract from the total width. The resulting width is used on the stage, and the rightmost pixels of the video is cropped.</td>
</tr>
<tr>
<td>VerticalAdjustment</td>
<td>Number of pixels to subtract from the total height. The resulting height is used on the stage, and the rightmost pixels of the video is cropped.</td>
</tr>
<tr>
<td>Data</td>
<td>Raw VP6 video stream data</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader®or Adobe®Acrobat® version 6.0 or above.

---

**VP6 SWF video packet**

The VP6 SWF video packet represents a VP6 video frame within SWF files.

**VP6 SWF Alpha video packet**

The VP6 SWF Alpha video packet represents a VP6 video frame with alpha channel information within SWF files.

---

**SWF video tags**

The following tags define embedded video data within a SWF file. These tags are permissible only in SWF 6 or later.

Video embedded in a SWF file is always streamed: video frames are located in the SWF frames with which they are temporally associated, and video playback can begin before an entire video stream is downloaded. This process is comparable to the way that streaming sounds are defined (see Streaming sound).

---

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>UI8[n]</td>
</tr>
<tr>
<td>AlphaData</td>
<td>UI8[n]</td>
</tr>
</tbody>
</table>

---

**Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8**
DefineVideoStream defines a video character that can later be placed on the display list (see The Display List).

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>RECORDHEADER</td>
<td>Tag type = 60</td>
</tr>
<tr>
<td>CharacterID</td>
<td>UI16</td>
<td>ID for this video character</td>
</tr>
<tr>
<td>NumFrames</td>
<td>UI16</td>
<td>Number of VideoFrame tags</td>
</tr>
<tr>
<td>Width</td>
<td>UI16</td>
<td>Width in pixels</td>
</tr>
<tr>
<td>Height</td>
<td>UI16</td>
<td>Height in pixels</td>
</tr>
<tr>
<td>VideoFlagsReserved</td>
<td>UB[4]</td>
<td>Must be 0</td>
</tr>
</tbody>
</table>
| VideoFlagsDeblocking | UB[3] | 000 = use VIDEOPACKET value  
                      |                   | 001 = off  
                      |                   | 010 = Level 1 (Fast deblocking filter)  
                      |                   | 011 = Level 2 (VP6 only, better deblocking filter)  
                      |                   | 100 = Level 3 (VP6 only, better deblocking plus fast deringing filter)  
                      |                   | 101 = Level 4 (VP6 only, better deblocking plus better deringing filter)  
                      |                   | 110 = Reserved  
                      |                   | 111 = Reserved       |
| VideoFlagsSmoothing | UB[1] | 0 = smoothing off (faster)  
                      |                   | 1 = smoothing on (higher quality)                                        |
| CodecID       | UI8   | 2 = Sorenson H.263  
                      |                   | 3 = Screen video (SWF 7 and later only)  
                      |                   | 4 = VP6 (SWF 8 and later only)  
                      |                   | 5 = VP6 video with alpha channel (SWF 8 and later only)         |

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.
SWF video tags 255

VideoFrame provides a single frame of video data for a video character that is already defined with DefineVideoStream.

In playback, the time sequencing of video frames depends on the SWF frame rate only. When SWF playback reaches a particular SWF frame, the video images from any VideoFrame tags in that SWF frame are rendered. Any timing mechanisms built into the video payload are ignored.

A VideoFrame tag is not needed for every video character in every frame number specified. A VideoFrame tag merely sets video data associated with a particular frame number; it does not automatically display a video frame. To display a video frame, specify the frame number as the Ratio field in PlaceObject2 or PlaceObject3.

### VideoFrame Field Type Comment

- **Header**
  - Tag type = 61

- **StreamID**
  - UI16
  - ID of video stream character of which this frame is a part

- **FrameNum**
  - UI16
  - Sequence number of this frame within its video stream

- **VideoData**
  - if CodecID = 2
    - H263VIDEOPACKET
  - if CodecID = 3
    - SCREENVIDEOPACKET
  - if CodecID = 4
    - VP6SWFVIDEOPACKET
  - if CodecID = 5
    - VP6SWFALPHAVIDEOPACKET
  - if CodecID = 6
    - SCREENV2VIDEOPACKET

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
CHAPTER 17

FLV File Format

Starting with SWF 6, Flash Player can exchange audio, video, and data over RTMP connections with the Macromedia Flash Communication Server. One way to feed data to Flash Communication Server (and thus on to Flash Player clients) is from files in the Flash Video (FLV) file format. Starting with SWF 7, Flash Player can also play FLV files directly with MIME type video/x-flv.

An FLV file encodes synchronized audio and video streams. The audio and video data within FLV files are encoded in the same way as audio and video within SWF files.

This document describes FLV version 1.

Each tag type in an FLV file constitutes a single stream. There can be, at most, one audio and one video stream, synchronized together, in an FLV file. You cannot define multiple independent streams of a single type.

NOTE

FLV files, unlike SWF files, store multibyte integers in big-endian byte order. This means, for example, the number 300 (0x12C) as a UI16 in SWF file format is represented by the byte sequence 0x2C 0x01, while as a UI16 in FLV file format, it is represented by the byte sequence 0x01 0x2C. Also note that FLV uses a 3-byte integer type, UI24, that is not used in SWF.
The FLV Header

All FLV files begin with the following header:

The DataOffset field usually has a value of 9 for FLV version 1. This field is present in order to accommodate larger headers in future versions.

The FLV File Body

After the FLV header, the remainder of an FLV file consists of alternating back-pointers and tags. They interleave as shown in the following table:

<table>
<thead>
<tr>
<th>FLV File Header</th>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature</td>
<td>UI8</td>
<td>Signature byte always 'F'</td>
</tr>
<tr>
<td>Signature</td>
<td>UI8</td>
<td>Signature byte always 'L'</td>
</tr>
<tr>
<td>Signature</td>
<td>UI8</td>
<td>Signature byte always 'V'</td>
</tr>
<tr>
<td>Version</td>
<td>UI8</td>
<td>File version (for example, 0x01 for FLV version 1)</td>
</tr>
<tr>
<td>TypeFlagsReserved</td>
<td>UB[5]</td>
<td>Must be 0</td>
</tr>
<tr>
<td>TypeFlagsAudio</td>
<td>UB[1]</td>
<td>Audio tags are present</td>
</tr>
<tr>
<td>TypeFlagsReserved</td>
<td>UB[1]</td>
<td>Must be 0</td>
</tr>
<tr>
<td>TypeFlagsVideo</td>
<td>UB[1]</td>
<td>Video tags are present</td>
</tr>
<tr>
<td>DataOffset</td>
<td>UI32</td>
<td>Offset in bytes from start of file to start of body (that is, size of header)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FLV File Body</th>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>PreviousTagSize0</td>
<td>UI32</td>
<td>Always 0</td>
</tr>
<tr>
<td>Tag1</td>
<td>FLVTAG</td>
<td>First tag</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
FLV Tags

FLV tags have the following format:

`PreviousTagSize1`  
`UI32`  
Size of previous tag, including its header.

For FLV version 1, this is the previous tag's DataSize plus 11.

`Tag2`  
`FLVTAG`  
Second tag

...  

`PreviousTagSizeN-1`  
`UI32`  
Size of second-to-last tag

`TagN`  
`FLVTAG`  
Last tag

`PreviousTagSizeN`  
`UI32`  
Size of last tag

FLV File Body

Field Type Comment

`TagType`  
`UI8`  
Type of this tag. Values are:

8: audio
9: video
18: script data
all others: reserved

`DataSize`  
`UI24`  
Length of the data in the Data field

`Timestamp`  
`UI24`  
Time in milliseconds at which the data in this tag applies. This is relative to the first tag in the FLV file, which always has a timestamp of 0.

`TimestampExtended`  
`UI8`  
Extension of the Timestamp field to form a UI32 value. This field represents the upper 8 bits, while the previous Timestamp field represents the lower 24 bits of the time in milliseconds.
In playback, the time sequencing of FLV tags depends on the FLV timestamps only. Any timing mechanisms built into the payload data format are ignored.

Audio Tags

Audio tags are similar to the DefineSound tag in the SWF file format, and their payload data is identical, except for the additional Nellymoser 8kHz format, which is not permitted in SWF.

StreamID

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>Always 0</td>
</tr>
</tbody>
</table>

Data

| If TagType = 8 | AUDIODATA |
| If TagType = 9 | VIDEODATA |
| If TagType = 18 | SCRIPTDATAOBJECT |

Body of the tag

AUDIODATA

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoundFormat</td>
<td>0 = uncompressed &lt;br&gt;1 = ADPCM &lt;br&gt;2 = MP3 &lt;br&gt;5 = Nellymoser 8kHz mono &lt;br&gt;6 = Nellymoser</td>
</tr>
</tbody>
</table>

Format of SoundData

<table>
<thead>
<tr>
<th>SoundRate</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 = 5.5 kHz &lt;br&gt;1 = 11 kHz &lt;br&gt;2 = 22 kHz &lt;br&gt;3 = 44 kHz</td>
<td></td>
</tr>
</tbody>
</table>

Sampling rate

<table>
<thead>
<tr>
<th>SoundSize</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 = snd8Bit &lt;br&gt;1 = snd16Bit</td>
<td></td>
</tr>
</tbody>
</table>

Size of each sample

SoundType

<table>
<thead>
<tr>
<th>SoundType</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 = sndMono &lt;br&gt;1 = sndStereo</td>
<td></td>
</tr>
</tbody>
</table>

Mono or stereo sound

For Nellymoser: always 0

SoundData

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoundData</td>
<td>UI8[size of sound data]</td>
</tr>
</tbody>
</table>

Sound data—varies by format

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Nellymoser 8kHz is a special case—the 8kHz sampling rate is not supported in other formats, and the SoundRate bits can't represent this value. When Nellymoser 8kHz mono is specified in SoundFormat, the SoundRate and SoundType fields are ignored. For other Nellymoser sampling rates, specify the normal Nellymoser SoundFormat and use the SoundRate and SoundType fields as usual.

Video Tags

Video tags are similar to the VideoFrame tag in SWF, and their payload data is identical.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FrameType</td>
<td>0x0: keyframe, 0x1: inter frame, 0x2: disposable inter frame (H.263 only)</td>
</tr>
<tr>
<td>CodecID</td>
<td>0x0: Sorenson H.263, 0x1: Screen video, 0x2: On2 VP6, 0x3: On2 VP6 with alpha channel, 0x4: Screen video version 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VideoData</th>
</tr>
</thead>
<tbody>
<tr>
<td>If CodecID = 2: H263VIDEOPACKET</td>
</tr>
<tr>
<td>If CodecID = 3: SCREENVIDEOPACKET</td>
</tr>
<tr>
<td>If CodecID = 4: VP6FLVVIDEOPACKET</td>
</tr>
<tr>
<td>If CodecID = 5: VP6FLVALPHAVIDEOPACKET</td>
</tr>
<tr>
<td>If CodecID = 6: SCREENV2VIDEOPACKET</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Data Tags

Data tags encapsulate single-method invocation, which is usually called on a NetStream object in the Flash Player. Data tags are formed from a method name and a set of arguments.

**SCRIPTDATAOBJECT** and **SCRIPTDATAOBJECTEND**

A **SCRIPTDATAOBJECT** record defines object data in ActionScript. Lists of **SCRIPTDATAOBJECT** records are terminated by using the **SCRIPTDATAOBJECTEND** tag.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectNameType</td>
<td>UI8</td>
</tr>
<tr>
<td>ObjectName</td>
<td>SCRIPTDATASTRING</td>
</tr>
<tr>
<td>ObjectData</td>
<td>SCRIPTDATAVALUE</td>
</tr>
<tr>
<td>ObjectEndMarker1</td>
<td>UI8</td>
</tr>
<tr>
<td>ObjectEndMarker2</td>
<td>UI24</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader® or Adobe®Acrobat® version 6.0 or above.
The SCRIPTDATASTRING and SCRIPTDATALONGSTRING records are used to define strings for data tags. The StringData field must be zero-terminated, and the StringLength field must include the zero terminator.

The SCRIPTDATALONGSTRING record can be used to specify strings larger than 65535 characters.

A SCRIPTDATAVALUE record represents an abstract definition of an ActionScript value or object. It can contain a list of values, objects, variables or arrays.

<table>
<thead>
<tr>
<th>Field Type Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>StringLength</strong></td>
</tr>
<tr>
<td>String length in bytes</td>
</tr>
<tr>
<td><strong>StringData</strong></td>
</tr>
<tr>
<td>String data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Type Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>StringLength</strong></td>
</tr>
<tr>
<td>String length in bytes</td>
</tr>
<tr>
<td><strong>StringData</strong></td>
</tr>
<tr>
<td>String data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Type Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>Type of the variable:</td>
</tr>
<tr>
<td>0 = Number type</td>
</tr>
<tr>
<td>1 = Boolean type</td>
</tr>
<tr>
<td>2 = String type</td>
</tr>
<tr>
<td>3 = Object type</td>
</tr>
<tr>
<td>4 = MovieClip type</td>
</tr>
<tr>
<td>5 = Null type</td>
</tr>
<tr>
<td>6 = Undefined type</td>
</tr>
<tr>
<td>7 = Reference type</td>
</tr>
<tr>
<td>8 = ECMA array type</td>
</tr>
<tr>
<td>10 = Strict array type</td>
</tr>
<tr>
<td>11 = Date type</td>
</tr>
<tr>
<td>12 = Long string type</td>
</tr>
<tr>
<td><strong>ECMAArrayLength</strong></td>
</tr>
<tr>
<td>Number of fields of ECMA array</td>
</tr>
</tbody>
</table>
A SCRIPTDATAVARIABLE record defines variable data in ActionScript. Lists of SCRIPTDATAVARIABLE records are terminated by using the SCRIPTDATAVARIABLEEND tag.

ScriptDataValue
If Type = 0, DOUBLE
If Type = 1, UI8
If Type = 2, SCRIPTDATASTRING
If Type = 3, SCRIPTDATAOBJECT[n]
If Type = 4, SCRIPTDATASTRING defining the MovieClip path
If Type = 7, UI16
If Type = 8, SCRIPTDATAVARIABLE[ECMAArrayLength]
If Type = 10, SCRIPTDATAVARIABLE[n]
If Type = 11, SCRIPTDATADATE
If Type = 12, SCRIPTDATALONGSTRING

Script data values

ScriptDataValueTerminator
If Type = 3, SCRIPTDATAOBJECTEND
If Type = 10, SCRIPTDATAVARIABLEEND

Field Type Comment
VariableName SCRIPTDATASTRING Name of the variable
VariableData SCRIPTDATAVALUE Data of the variable
SCRIPTDATAVARIABLEEND

Field Type Comment
VariableEndMarker1 UI24 Always 9

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
A SCRIPTDATADATE record defines a particular date and time.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateTime</td>
<td>UI64</td>
</tr>
<tr>
<td></td>
<td>Number of milliseconds since Jan 1, 1970 UTC.</td>
</tr>
<tr>
<td>LocalDateTimeOffset</td>
<td>SI16</td>
</tr>
<tr>
<td></td>
<td>Local time offset in minutes from UTC. For time zones located west of Greenwich, UK, this is a negative number. Time zones east of Greenwich, UK, are positive.</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
To write Macromedia Flash SWF files, you must be able to read and understand the raw bits and bytes. This appendix examines a simple, one-frame Flash file that contains only a rectangle.

Here is a hex dump of the SWF file:

```
000000  46 57 53 03 4F 00 00 00    78 00 05 5F 00 00 0F A0
000010  00 00 0C 01 00 43 02 FF    FF FF BF 00 23 00 00 00
000020  01 00 70 FB 49 97 0D 0C    7D 50 00 01 14 00 00 00
000030  00 01 25 C9 92 0D 21 ED    48 87 65 30 3B 6D E1 D8
000040  B4 00 00 86 06 06 01 00    01 00 00 40 00 00 00
```

A SWF file always begins with a header. It describes the file version, the length of the file in bytes, the frame size in twips (twentieths of a pixel), frame rate in frames per second, and the frame count.

**SWF File Header**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature</td>
<td>UI8</td>
<td>Signature byte:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;F&quot; indicated uncompressed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;C&quot; indicates compressed (SWF 6 or later only)</td>
</tr>
<tr>
<td>Signature</td>
<td>UI8</td>
<td>Signature byte always &quot;W&quot;</td>
</tr>
<tr>
<td>Signature</td>
<td>UI8</td>
<td>Signature byte always &quot;S&quot;</td>
</tr>
<tr>
<td>Version</td>
<td>UI8</td>
<td>Single byte file version (for example, 0x06 for SWF 6)</td>
</tr>
<tr>
<td>FileLength</td>
<td>UI32</td>
<td>Length of entire file in bytes</td>
</tr>
<tr>
<td>FrameSize</td>
<td>RECT</td>
<td>Frame size in twips</td>
</tr>
<tr>
<td>FrameRate</td>
<td>UI16</td>
<td>Frame delay in 8.8 fixed number of frames per second</td>
</tr>
<tr>
<td>FrameCount</td>
<td>UI16</td>
<td>Total number of frames in file</td>
</tr>
</tbody>
</table>

* The types are defined in Basic Data Types.
The first three bytes are the standard signature for all SWF files. They are the ASCII values of the characters 'F' (or 'C'), 'W', and 'S' in that order. The fourth byte indicates the version of the file.

The next four bytes represent an unsigned 32-bit integer indicating the file size. Here's where it starts getting tricky and machine architecture gets involved. The next four bytes are 0x4F000000 so that would imply that the file length is 1325400064 bytes, a very large number which doesn't make sense. What we failed to do is swap all the bytes.

In SWF files, bytes are swapped whenever reading words and dwords such that a 32-bit value B1B2B3B4 is written as B4B3B2B1, and a 16-bit value B1B2 is written as B2B1. Single bytes are written unchanged since there is no bit-swapping. The reason for this is the differences in storage and retrieval between the Mac and PC processors.

Reversing the bytes we can read the four bytes correctly and see that file is 79 bytes long.

The next nine bytes represent a data structure used in the SWF format called a Rectangle. Here is the file description of a rectangle:

To understand these bytes, we need to look at the individual bits.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nbits</td>
<td>Bits in each rect value field</td>
</tr>
<tr>
<td>Xmin</td>
<td>x minimum position for rect</td>
</tr>
<tr>
<td>Xmax</td>
<td>x maximum position for rect</td>
</tr>
<tr>
<td>Ymin</td>
<td>y minimum position for rect</td>
</tr>
<tr>
<td>Ymax</td>
<td>y maximum position for rect</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
Another subtle point about the SWF file representation is that reading and writing bits is different from reading and writing words and dwords. When reading and writing bits, no byte-swapping occurs. This is because when Flash is reading an $n$-bit field, it reads a byte at a time until it has read all $n$ bits. Therefore, the next five bits are read in order and evaluate to 15.

What if $N$-bit has a value of sixteen? This is exactly the size of a word so do we read the following fields as words and swap bytes? No. Fields described by bit size are always read a byte at a time. No swapping, just read the next $n$ bits in that order.

For the header, the rectangle is used to store the file dimensions with $X_{\text{max}}$ corresponding to the file width and $Y_{\text{max}}$ corresponding to the file height, both in twips. In SWF format, a twip is a twentieth of a pixel, so if we convert to pixels, we see that our file is 550 x 400.

Now we have looked at all of the fields of the rectangle and evaluated them, but what about those last seven bits which are all 0s. Well, they were just “filled.”

0000000 = filled bits

After the end of any structure, if the structure does not completely fill up its last byte, then that last byte is filled with 0s to keep the next item byte aligned. So if the next item is a word or dword, you can read it as such and not worry about being in the middle of a byte. In this case, only one bit in the last byte is used so the last seven bits are filled with 0s.

Next in the header is the frame rate. It is supposed to be stored as a 16-bit integer, but the first byte (or last depending on how you look at it) is completely ignored. So the frame rate is 12 fps.

Next is the frame count, which is also a 16-bit integer. So the frame count is 1.

0x000C

➜ 0x0C00

➜ 0x0C

12 = frame rate

0x0100

➜ 0x0001(byte swapping)

➜ 1 = frame count
Now we are finished with the header. After the header is a series of tagged data blocks. Here is a description of a tag (this is simplifying somewhat; byte swapping is necessary):

There are two types of tags: the short and long tag header. Regardless of which case you have, you begin by looking at the first word.

The first 10 bits of the tag are the unsigned tag type. The tag type indicates what type of data is to follow in the body of the data block to follow. In this case, the value of the tag type is 9, which corresponds to a SetBackgroundColor block. The last six unsigned bits of the tag header indicate the length of the data block to follow if it is 62 bytes or less. If the length of the data block is more than 62 bytes, then this field has all 1s and the length is indicated in the following dword. For the tag we are looking at, the field does not have all 1s, so it does indicate the actual length which is 3 bytes.

0000001001 = 9 = SetBackgroundColor
000011 = 3 = body length

Since we know that the length of the body is 3 bytes, let’s take a look at it. A SetBackgroundColor tag only contains the 3-byte RGB color description so we evaluate it as such. A color is its own 3-byte data type so there is no byte swapping.

0XFFFFFF = white
The next tag is a long tag and is a DefineShape tag. Here is the file description of DefineShape:

The body of a DefineShape is composed of an unsigned 16-bit character ID, a rectangle defining the bounds for the shape, and a SHAPEWITHSTYLE structure which contains shape information.

Now the Rect which defines the boundaries:

```
70 FB 49 97 0D 0C 7D 50
0111 0000  1111 1011  0100 1001  1001 0111  0000 1101   0000
1100  0111 1101  0101 0000
01110 = 14 = Nbits
```

```
0xBF00
0x00BF
```

```
0000 0000 1011 1111
```

```
0000000010 = 2 = DefineShape
```

```
111111 = 63 = body length (indicates that the body length value is in the next dword)
```

```
0x23000000
0x00000023
```

35 = body length

```
DefineShape
```

Field Type Comment

<table>
<thead>
<tr>
<th>Header</th>
<th>RECORDHEADER</th>
<th>Tag type = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShapeId</td>
<td>UI16</td>
<td>ID for this character</td>
</tr>
<tr>
<td>ShapeBounds</td>
<td>RECT</td>
<td>Bounds of the shape</td>
</tr>
<tr>
<td>Shapes</td>
<td>SHAPEWITHSTYLE</td>
<td>Shape information</td>
</tr>
</tbody>
</table>

```
0x0100 ➜ 0x0001
```

```
1 = shape ID
```

```
00011111011010 = 2010 = Xmin
```

```
01001100101110 = 4910 = Xmax
```

```
00011010000110 = 1670 = Ymin
```

```
00111110101010 = 4010 = Ymax
```

```
000 = fill bits
```

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The following table describes the SHAPEWITHSTYLE structure:

A fill style array itself has three fields. The first field is an 8-bit integer count which indicates how many fill styles are in the array. This count works similar to the tag's length field in that if it is all 1s, you have to look at the next 16 bits to get the actual length. Here is the file description:

In this case, the 8-bit count is equal to 0 so there is nothing to follow it.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FillStyleCount</td>
<td>UI8</td>
<td>Count of fill styles</td>
</tr>
<tr>
<td>FillStyleCountExtended</td>
<td>UI16</td>
<td>Extended count of fill styles. Supported only for Shape2 and Shape3.</td>
</tr>
<tr>
<td>FillStyles</td>
<td>FILLSTYLE[FillStyleCount]</td>
<td>Array of fill styles</td>
</tr>
</tbody>
</table>

A line style array is exactly the same as a fill style array except it stores line styles. Here is the file description:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LineStyleCount</td>
<td>UI8</td>
<td>Count of line styles</td>
</tr>
<tr>
<td>LineStyleCountExtended</td>
<td>UI16</td>
<td>Extended count of line styles</td>
</tr>
<tr>
<td>LineStyles</td>
<td>LINESTYLE[count]</td>
<td>Array of line styles</td>
</tr>
</tbody>
</table>
So there is one line style in the array. A line style has two parts, an unsigned 16-bit integer indicating the width of a line in twips, and a color. Here is the file description:
The color in this case is a 24-bit RGB, but if we were doing a DefineShape3, it would be a 32-bit RGBA where alpha is the transparency of the color.

<table>
<thead>
<tr>
<th>FIELD</th>
<th>TYPE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>UI16</td>
<td>Width of line in twips</td>
</tr>
<tr>
<td>Color</td>
<td>RGB (Shape1 or Shape2) RGBA (Shape3)</td>
<td>Color value including alpha channel information for Shape3</td>
</tr>
</tbody>
</table>

0x0 = 0 = NumFillBits
0x1 = 1 = NumLineBits
Now for the array of shape records. The following four tables describe the four types of shape records. Here are the file descriptions:

### ENDSHAPERECORD

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TypeFlag</td>
<td>UB[1]</td>
<td>Non-edge record flag</td>
</tr>
<tr>
<td>EndOfShape</td>
<td>UB[5]</td>
<td>End of shape flag</td>
</tr>
</tbody>
</table>

### STYLECHANGERECORD

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TypeFlag</td>
<td>UB[1]</td>
<td>Non-edge record flag</td>
</tr>
<tr>
<td>StateNewStyles</td>
<td>UB[1]</td>
<td>New styles flag. Used by DefineShape2 and DefineShape3 only.</td>
</tr>
<tr>
<td>StateLineStyle</td>
<td>UB[1]</td>
<td>Line style change flag</td>
</tr>
<tr>
<td>StateFillStyle1</td>
<td>UB[1]</td>
<td>Fill style 1 change flag</td>
</tr>
<tr>
<td>StateFillStyle0</td>
<td>UB[1]</td>
<td>Fill style 0 change flag</td>
</tr>
<tr>
<td>StateMoveTo</td>
<td>UB[1]</td>
<td>Move to flag</td>
</tr>
<tr>
<td>MoveDeltaX</td>
<td>SB[MoveBits]</td>
<td>Delta X value</td>
</tr>
<tr>
<td>MoveDeltaY</td>
<td>SB[MoveBits]</td>
<td>Delta Y value</td>
</tr>
<tr>
<td>FillStyle0</td>
<td>UB[FillBits]</td>
<td>Fill 0 Style</td>
</tr>
<tr>
<td>FillStyle1</td>
<td>UB[FillBits]</td>
<td>Fill 1 Style</td>
</tr>
<tr>
<td>LineStyle</td>
<td>UB[LineBits]</td>
<td>Line Style</td>
</tr>
<tr>
<td>FillStyles</td>
<td>FILLSTYLEARRAY</td>
<td>Array of new fill styles</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe®Reader®or Adobe®Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TypeFlag</td>
<td>UB[1]</td>
<td>This is an edge record. Always 1.</td>
</tr>
<tr>
<td>StraightFlag</td>
<td>UB[1]</td>
<td>Curved edge. Always 0.</td>
</tr>
<tr>
<td>NumBits</td>
<td>UB[4]</td>
<td>Number of bits per value. (two less than the actual number)</td>
</tr>
<tr>
<td>ControlDeltaX</td>
<td>SB[NumBits+2]</td>
<td>X control point change</td>
</tr>
<tr>
<td>ControlDeltaY</td>
<td>SB[NumBits+2]</td>
<td>Y control point change</td>
</tr>
<tr>
<td>AnchorDeltaX</td>
<td>SB[NumBits+2]</td>
<td>X anchor point change</td>
</tr>
<tr>
<td>AnchorDeltaY</td>
<td>SB[NumBits+2]</td>
<td>Y anchor point change</td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8
277

ENDSHAPERECORD defines the end of the shape record array.

STYLECHANGERECORD defines changes in line style, fill style, position, or a new set of styles. STRAIGHTEDGERECORD and CURVEDEDGERECORD define a straight or curved edge, respectively. The first bit in a shape record is a type flag. A 0 corresponds to a non-edge record, and a 1 corresponds to an edge record. Looking at the first bit of our first shape record, we see that it is not an edge record. Now we must look at the next five bits which are all flags that tell us what is to follow. If all of the five bits are 0, then that is a type 0 shape record and defines the end of the array of shape records.

`25 C9 92 0D 21`  
0010 0101  1100 1001  1001 0010  0000 1101  0010 0001
0 = 0 = non edge record

`01001`  
01001100100100  = 4900 (twips) = 245 pixels = MoveDeltaX
00011010010000  = 1680 = 84 pixels = MoveDeltaY

Since the line style flag is true, the next field is a NumLineBits = 1 bit field representing the line style. This field is equal to 1. This means that the line style for the line to follow is the first one in the line style array.

Now for the rest of the shape records:

`ED 48 87 65 30 3B 6D E1 D8 B4 00 00`  
1110 1101 0100 1000 1000 0111 0110 0101 0011 0000 0011 1011 0110 1101 1110 0001 1101 1000 1011 0100 0000 0000 0000 0000

The next shape record begins with a 1, so it is an edge record.
The next bit indicates if it is a straight or curved edge. It is a 1, which stands for a straight edge. The next four bits indicate the size of any delta fields which follow. The formula for the NumBits value is 2 + whatever the value of that 4-bit field. In this case, the value of NumBits is 13. Following the NumBits field is a 1-bit line flag. This indicates whether the line being described is a general line or horizontal/vertical line. The value of 0 corresponds to a hor/vert line, so the next bit is a VertLineFlag field and indicates whether the line is horizontal or vertical. The value of the bit is 1 which corresponds to a vertical line. The next field for a vertical line is the signed DeltaY field which is nbits = 13 bits long. The value corresponds to 116 pixels. That is the end of the shape record.

The next three records are very similar to the last one:

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
Finally, the last shape record begins with a 0 which means it is not an edge record. Furthermore, all of its flag bits are equal to 0, which means that it is the last shape record and we are through with our shape record array.

0 = 0 = non-edge record
000000 = flags (since they are all 0, this is the end of the shape record array)

Since we are done with our structure, we must now fill our last byte with 0s to keep byte aligned.

000000 = filled 0s

We are also done with our shape with style since the shape record array is the last element of the shape with style. Since we are already byte aligned, we can go on to our next tagged data block.

The Tag type of the block is equal to 26 which corresponds to a PlaceObject2. The length field has a value of 6 so the length of the data block is 6 bytes.

0000011010 = 26 = tag type = PlaceObject2
000110 = 6 = length

Here is the file description of the PlaceObject2 tag:

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Tag type = 26.</td>
</tr>
<tr>
<td>PlaceFlagHasClipActions</td>
<td>SWF 5 or later: has clip actions (sprite characters only). Otherwise: always 0.</td>
</tr>
<tr>
<td>PlaceFlagHasClipDepth</td>
<td>Has clip depth.</td>
</tr>
<tr>
<td>PlaceFlagHasName</td>
<td>Has name.</td>
</tr>
<tr>
<td>PlaceFlagHasRatio</td>
<td>Has ratio.</td>
</tr>
<tr>
<td>PlaceFlagHasColorTransform</td>
<td>Has color transform.</td>
</tr>
<tr>
<td>PlaceFlagHasMatrix</td>
<td>Has matrix.</td>
</tr>
</tbody>
</table>

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The first eight bits of the body are all flags indicating what is to follow. A 1 in the sixth bit indicates that the body has a transform matrix, and the 1 in the seventh bit indicates that the object to be placed has a character ID.

Following the flags is a 16-bit unsigned integer which indicates the depth of the character. In this case, the depth is 1, which makes sense since the rectangle is the only object in the file. Since the object has a character ID, the next field in the body is the unsigned 16-bit ID. Since the rectangle is the only object in the file, the ID of the rectangle is 1.

<table>
<thead>
<tr>
<th>PlaceFlagHasCharacter</th>
<th>PlaceFlagMove</th>
<th>Depth</th>
<th>CharacterId</th>
<th>Matrix</th>
<th>ColorTransform</th>
<th>Ratio</th>
<th>Name</th>
<th>ClipDepth</th>
<th>ClipActions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0100</td>
<td>0x0001</td>
<td>depth = 1</td>
<td>character ID = 1</td>
<td>0x0000</td>
<td></td>
<td>0x0001</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
The final field for this PlaceObject2 is the transform matrix. Here is the file description:

Since this shape has no transform information, the matrix is empty. All of its flag bits have values of zero. This is not super efficient but it is valid.

Since we are done with our PlaceObject2, let's take a look at our next tag.

**Tag type = 1 = ShowFrame**

**length = 0**

We see that the tag is an instruction to show the frame. A ShowFrame has no body. Its length is 0, so we move on to the next tag.

**Tag type = 0 = end**

**length = 0**

We have reached the end tag which signals the end of our SWF file.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasScale</td>
<td>UB[1]</td>
<td>Has scale values if equal to 1.</td>
</tr>
<tr>
<td>NScaleBits</td>
<td>UB[5]</td>
<td>Bits in each scale value field.</td>
</tr>
<tr>
<td>ScaleX</td>
<td>FB[NScaleBits]</td>
<td>x scale value.</td>
</tr>
<tr>
<td>ScaleY</td>
<td>FB[NScaleBits]</td>
<td>y scale value.</td>
</tr>
<tr>
<td>HasRotate</td>
<td>UB[1]</td>
<td>Has rotate and skew values if equal to 1.</td>
</tr>
<tr>
<td>NRotateBits</td>
<td>UB[5]</td>
<td>Bits in each rotate value field.</td>
</tr>
<tr>
<td>RotateSkew0</td>
<td>FB[NRotateBits]</td>
<td>First rotate and skew value.</td>
</tr>
<tr>
<td>RotateSkew1</td>
<td>FB[NRotateBits]</td>
<td>Second rotate and skew value.</td>
</tr>
<tr>
<td>NTranslateBits</td>
<td>UB[5]</td>
<td>Bits in each translate value field.</td>
</tr>
<tr>
<td>TranslateX</td>
<td>SB[NTranslateBits]</td>
<td>x translate value in twips.</td>
</tr>
<tr>
<td>TranslateY</td>
<td>SB[NTranslateBits]</td>
<td>y translate value in twips.</td>
</tr>
</tbody>
</table>

0x00 ➜ completely empty matrix with leftover bits filled
0x4000 ➜ 0x0040
0x0000 0000 0100 0000
0x0000 0000 0000 0000

To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.
To view the contents of this document, you must agree to the License Agreement. Requires Adobe® Reader® or Adobe® Acrobat® version 6.0 or above.

Macromedia Flash (SWF) and Flash Video (FLV) File Format Specification Version 8