Adobe Systems Inc.
AMF 3 Specification
Category: ActionScript Serialization

Action Message Format -- AMF 3

Abstract
Action Message Format (AMF) is a compact binary format that is used to serialize ActionScript object graphs. Once serialized an AMF encoded object graph may be used to persist and retrieve the public state of an application across sessions or allow two endpoints to communicate through the exchange of strongly typed data.

AMF was introduced in Flash Player 6 in 2001 and remained unchanged with the introduction of ActionScript 2.0 in Flash Player 7 and with the release of Flash Player 8. This version of AMF is referred to as AMF 0 (See [AMF0]). In Flash Player 9, Action Script 3.0 was introduced along with a new ActionScript Virtual Machine (AVM+) - the new data types and language features made possible by these improvements prompted AMF to be updated. Given the opportunity to release a new version of AMF, several optimizations were also made to the encoding format to remove redundant information from serialized data. This specification defines this updated version of AMF, namely AMF 3.

Table of Contents

1 Introduction

1.1 Purpose

Action Message Format (AMF) is a compact binary format that is used to serialize ActionScript object graphs. Once serialized an AMF encoded object graph may be used to persist and retrieve the public state of an application across sessions or allow two endpoints to communicate through the exchange of strongly typed data. The first version of AMF, referred to as AMF 0, supports sending complex objects by reference which helps to avoid sending redundant instances in an object graph. It also allows endpoints to restore object relationships and support circular references while avoiding problems such as infinite recursion during serialization. A new version of AMF, referred to as AMF 3 to coincide with the release of ActionScript 3.0, improves on AMF 0 by sending object traits and strings by reference in addition to object instances. AMF 3 also supports some new data types introduced in ActionScript 3.0.

1.2 Notational Conventions

1.2.1 Augmented BNF

Type definitions in this specification use Augmented Backus-Naur Form (ABNF) syntax [RFC2234]. The reader should be familiar with this notation before reading this document.

1.3 Basic Rules

Throughout this document bytes are assumed to be octets, or 8-bits.

```
U8          =  An unsigned byte (8-bits, an octet)
U16         =  An unsigned 16-bit integer in big endian
               (network) byte order
U32         =  An unsigned 32-bit integer in big endian (network)
               byte order
DOUBLE      =  8 byte IEEE-754 double precision floating point
               value in network byte order (sign bit in low
               memory).
MB          =  A megabyte or 1048576 bytes.
```

More complicated data type rules require special treatment which is outlined below.

1.3.1 Variable Length Unsigned 29-bit Integer Encoding

AMF 3 makes use of a special compact format for writing integers to reduce the number of bytes required for encoding. As with a normal 32-bit integer, up to 4 bytes are required to hold the value however the high bit of the first 3 bytes are used as flags to determine whether the next byte is part of the integer. With up to 3 bits of the 32 bits being used as flags, only 29 significant bits remain for encoding an integer. This means the largest unsigned integer value that can be represented is $2^{29} - 1$.

```
         (hex)           :              (binary)
0x00000000 - 0x0000007F  :  0xxxxxxx
0x00000080 - 0x00003FFF  :  1xxxxxxx 0xxxxxxx
0x00004000 - 0x001FFFFF  :  1xxxxxxx 1xxxxxxx 0xxxxxxx
0x00200000 - 0x3FFFFFFF  :  1xxxxxxx 1xxxxxxx 1xxxxxxx xxxxxxxx
0x40000000 - 0xFFFFFFFF  :  throw range exception
```

In ABNF syntax, the variable length unsigned 29-bit integer type is described as follows:

```
U29         =  U29-1 | U29-2 | U29-3 | U29-4
U29-1       =  %x00-7F
U29-2       =  %x80-FF %x00-7F
U29-3       =  %x80-FF %x80-FF %x00-7F
U29-4       =  %x80-FF %x80-FF %x80-FF %x00-FF
```

1.3.2 Strings and UTF-8

AMF 0 and AMF 3 use (non-modified) UTF-8 to encode strings. UTF-8 is the abbreviation for 8-bit Unicode Transformation Format. UTF-8 strings are typically preceded with a byte-length header followed by a sequence of variable length (1 to 4 octets) encoded Unicode code-points. AMF 3 uses a slightly modified byte-length header; a detailed description is provided below and referred to throughout the document.

```
         (hex)           :              (binary)
0x00000000 - 0x0000007F  :  0xxxxxxx
0x00000080 - 0x000007FF  :  110xxxxx 10xxxxxx
0x00000800 - 0x0000FFFF  :  1110xxxx 10xxxxxx 10xxxxxx
0x00010000 - 0x0010FFFF  :  11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
```

In ABNF syntax, [RFC3629] describes UTF-8 as follows:

```
UTF8-char   =  UTF8-1 | UTF8-2 | UTF8-3 | UTF8-4
UTF8-1      =  %x00-7F
UTF8-2      =  %xC2-DF UTF8-tail
UTF8-3      =  %xE0 %xA0-BF UTF8-tail | %xE1-EC 2(UTF8-tail) |
               %xED %x80-9F UTF8-tail | %xEE-EF 2(UTF8-tail)
UTF8-4      =  %xF0 %x90-BF 2(UTF8-tail) | %xF1-F3 3(UTF8-tail) |
               %xF4 %x80-8F 2(UTF8-tail)
UTF8-tail   =  %x80-BF
```

For AMF 3 a string can be encoded as a string literal or a string reference. A variable length unsigned 29-bit integer is used for the header and the first bit is flag that specifies which type of string is encoded. If the flag is 1, a string literal is encoded and the remaining bits are used to encode the byte-length of the UTF-8 encoded String. If the flag is 0, then a string reference is encoded and the remaining bits are used to encode an index to the implicit string reference table.

```
U29S-ref     =  U29    ; The first (low) bit is a flag with
                       ; value 0. The remaining 1 to 28
                       ; significant bits are used to encode a
                       ; string reference table index (an
                       ; integer).

U29S-value   =  U29    ; The first (low) bit is a flag with
                       ; value 1. The remaining 1 to 28
                       ; significant bits are used to encode the
                       ; byte-length of the UTF-8 encoded
                       ; representation of the string

UTF-8-empty  =  0x01   ; The UTF-8-vr empty string which is
                       ; never sent by reference.

UTF-8-vr     =  U29S-ref | (U29S-value *(UTF8-char))
```

Note that this encoding imposes some theoretical limits on the use of Strings. The number of unique Strings that can be sent by reference is limited to $2^{28} - 1$, and the byte-length of each UTF-8 encoded String is limited to $2^{28} - 1$ bytes (approx 256 MB).

2. Technical Summary

2.1 Summary of improvements

The following is a table of the improvements and changes in AMF 3:

- Object traits can now be sent by reference
- Strings can now be sent by reference
- int/uint type support
- flash.utils.ByteArray type support, can also be sent by reference
- flash.utils.IExternalizable support
- Variable length encoding scheme for integers to reduce data size
- References are sent using variable length integer
- String UTF-8 length uses variable length integer
- Array count uses variable length integer
- A single Array type marker covers both strict and ECMA Arrays
- Dates no longer send timezone information
- Dates can now be sent by reference

- XMLDocument UTF-8 length uses variable length integer
- XMLDocument can now be sent by reference
- XML type support, can also be sent by reference
- XML UTF-8 length uses variable length integer
- ByteArray type length uses variable length integer
- Boolean true and false are now sent as one byte type markers
- Unsupported type marker has been removed
- Reserved RecordSet and Movieclip type markers have been removed

2.2 Reference Tables

In AMF 3, Strings, Complex Objects (which in AMF 3 are defined as anonymous Objects, typed Objects, Arrays, Dates, XMLDocument, XML, and ByteArrays) and an Object Type's Traits can now be sent by reference. This means that instead of sending redundant information, these components of AMF can simply refer to an earlier occurrence of a component. This reference is an integer forming a zero-based index that is encoded in the component information, typically in the first number that appears after the relevant type marker (see the type definitions for Object, Array, Date, XMLDocument, XML and ByteArray below for exact details). These indexes form a virtual "table" of references that a deserializer and serializer must maintain when reading and writing AMF 3 formatted data.

Note that 3 separate reference tables are used for Strings, Complex Objects and Object Traits respectively.

3 AMF 3 Data Types

3.1 Overview

There are 13 types in AMF 3. A type marker is one byte in length and describes the type of encoded data that follows.

```
marker              =  U8
```

The set of possible type markers are listed below (values are represented in hexadecimal format):

```
undefined-marker    =  0x00
null-marker         =  0x01
false-marker        =  0x02
true-marker         =  0x03
integer-marker      =  0x04
double-marker       =  0x05
string-marker       =  0x06
xml-doc-marker      =  0x07
date-marker         =  0x08
array-marker        =  0x09
object-marker       =  0x0A
xml-marker          =  0x0B
```

```
byte-array-marker   =  0x0C
```

Type markers may be followed by the actual encoded type data, or if the marker represents a single possible value (such as null) then no further information needs to be encoded.

```
value-type  =  undefined-marker | null-marker | false-marker |
               true-marker | integer-type | double-type |
               string-type | xml-doc-type | date-type |
               array-type | object-type | xml-type |
               byte-array-type
```

AMF 3 makes use of three reference tables for strings, objects and traits (the characteristics of objects that define a strong type such as the class name and public member names). These tables are considered implicit as they are not encoded as a unique entity in the format. Each type that can be sent by reference may instead be encoded using an index to the appropriate reference table. Strings can be sent by reference using an index to the string table. Object, Array, XML, XMLDocument, ByteArray, Date and instances of user defined Classes can be sent by reference using an index to the object table. Objects and instances of user defined Classes have trait information which can also be sent by reference using an index to the traits table.

3.2 undefined Type

The undefined type is represented by the undefined type marker. No further information is encoded for this value.

```
undefined-type      =  undefined-marker
```

Note that endpoints other than the AVM may not have the concept of undefined and may choose to represent undefined as null.

3.3 null Type

The null type is represented by the null type marker. No further information is encoded for this value.

```
null-type           =  null-marker
```

3.4 false Type

The false type is represented by the false type marker and is used to encode a Boolean value of false. Note that in ActionScript 3.0 the concept of a primitive and Object form of Boolean does not exist. No further information is encoded for this value.

```
false-type          =  false-marker
```

3.5 true type

The true type is represented by the true type marker and is used to encode a Boolean value of true. Note that in ActionScript 3.0 the concept of a primitive and Object form of Boolean does not exist. No further information is encoded for this value.

```
true-type          =  true-marker
```

3.6 integer type

In AMF 3 integers are serialized using a variable length unsigned 29-bit integer. The ActionScript 3.0 integer types - a signed 'int' type and an unsigned 'uint' type - are also represented using 29-bits in AVM+. If the value of an unsigned integer (uint) is greater or equal to 2^{29} or if the value of a signed integer (int) is greater than or equal to 2^{28} then it will be represented by AVM+ as a double and thus serialized in using the AMF 3 double type.

```
integer-type       =  integer-marker U29
```

3.7 double type

The AMF 3 double type is encoded in the same manner as the AMF 0 Number type. This type is used to encode an ActionScript Number or an ActionScript int of value greater than or equal to 2^{28} or an ActionScript uint of value greater than or equal to 2^{29}. The encoded value is always an 8 byte IEEE-754 double precision floating point value in network byte order (sign bit in low memory).

```
double-type        =  double-marker DOUBLE
```

3.8 String type

ActionScript String values are represented using a single string type in AMF 3 - the concept of string and long string types from AMF 0 is not used.

Strings can be sent as a reference to a previously occurring String by using an index to the implicit string reference table.

Strings are encoding using UTF-8 - however the header may either describe a string literal or a string reference.

The empty String is never sent by reference.

```
string-type        =  string-marker UTF-8-vr
```

3.9 XMLDocument type

ActionScript 3.0 introduced a new XML type (see 3.13) however the legacy XMLDocument type is retained in the language as flash.xml.XMLDocument. Similar to AMF 0, the structure of an XMLDocument needs to be flattened into a string representation for serialization. As with other strings in AMF, the content is encoded in UTF-8.

XMLDocuments can be sent as a reference to a previously occurring XMLDocument instance by using an index to the implicit object reference table.

```
U29X-value    =  U29   ; The first (low) bit is a flag with
                       ; value 1. The remaining 1 to 28
                       ; significant bits are used to encode the
                       ; byte-length of the UTF-8 encoded
                       ; representation of the XML or
                       ; XMLDocument.

xml-doc-type  =  xml-doc-marker (U29O-ref | (U29X-value
                 *(UTF8-char)))
```

Note that this encoding imposes some theoretical limits on the use of XMLDocument. The byte-length of each UTF-8 encoded XMLDocument instance is limited to 2^{28} - 1 bytes (approx 256 MB).

3.10 Date type

In AMF 3 an ActionScript Date is serialized simply as the number of milliseconds elapsed since the epoch of midnight, 1st Jan 1970 in the UTC time zone. Local time zone information is not sent.

Dates can be sent as a reference to a previously occurring Date instance by using an index to the implicit object reference table.

```
U29D-value   =  U29        ; The first (low) bit is a flag with
                           ; value 1. The remaining bits are not
                           ; used.

date-time    =  DOUBLE     ; A 64-bit integer value transported
                           ; as a double.

date-type    =  date-marker (U29O-ref | (U29D-value date-time))
```

3.11 Array type

ActionScript Arrays are described based on the nature of their indices, i.e. their type and how they are positioned in the Array. The following table outlines the terms and their meaning:

strict	contains only ordinal (numeric) indices
dense	ordinal indices start at 0 and do not contain gaps between successive indices (that is, every index is defined from 0 for the length of the array)
sparse	contains at least one gap between two indices
associative	contains at least one non-ordinal (string) index (sometimes referred to as an ECMA Array)

AMF considers Arrays in two parts, the dense portion and the associative portion. The binary representation of the associative portion consists of name/value pairs (potentially none) terminated by an empty string. The binary representation of the dense portion is the size of the dense portion (potentially zero) followed by an ordered list of values (potentially none). The order these are written in AMF is first the size of the dense portion, an empty string terminated list of name/value pairs, followed by size values.

Arrays can be sent as a reference to a previously occurring Array by using an index to the implicit object reference table.

```
U29A-value   =  U29    ; The first (low) bit is a flag with
                       ; value 1. The remaining 1 to 28
                       ; significant bits are used to encode the
                       ; count of the dense portion of the
                       ; Array.

assoc-value  =  UTF-8-vr value-type

array-type   =  array-marker (U29O-ref | (U29A-value
                (UTF-8-empty | *(assoc-value) UTF-8-empty)
                *(value-type)))
```

3.12 Object type

A single AMF 3 type handles ActionScript Objects and custom user classes. The term 'traits' is used to describe the defining characteristics of a class. In addition to 'anonymous' objects and 'typed' objects, ActionScript 3.0 introduces two further traits to describe how objects are serialized, namely 'dynamic' and 'externalizable'. The following table outlines the terms and their meanings:

Anonymous	an instance of the actual ActionScript Object type or an instance of a Class without a registered alias (that will be treated like an Object on deserialization)
Typed	an instance of a Class with a registered alias
Dynamic	an instance of a Class definition with the dynamic trait declared; public variable members can be added and removed from instances dynamically at runtime
Externalizable	an instance of a Class that implements flash.utils.IExternalizable and completely controls the serialization of its members (no property names are included in the trait information).

In addition to these characteristics, an object's traits information may also include a set of public variable and public read-writeable property names defined on a Class (i.e. public members that are not Functions). The order of the member names is important as the member values that follow the traits information will be in the exact same order. These members are considered sealed members as they are explicitly defined by the type.

If the type is dynamic, a further section may be included after the sealed members that lists dynamic members as name / value pairs. One continues to read in dynamic members until a name that is the empty string is encountered.

Objects can be sent as a reference to a previously occurring Object by using an index to the implicit object reference table. Further more, trait information can also be sent as a reference to a previously occurring set of traits by using an index to the implicit traits reference table.

```
U29O-ref         =  U29   ; The first (low) bit is a flag
                          ; (representing whether an instance
                          ; follows) with value 0 to imply that
                          ; this is not an instance but a
                          ; reference. The remaining 1 to 28
                          ; significant bits are used to encode an
                          ; object reference index (an integer).

U29O-traits-ref  =  U29   ; The first (low) bit is a flag with
                          ; value 1. The second bit is a flag
                          ; (representing whether a trait
                          ; reference follows) with value 0 to
                          ; imply that this objects traits are
                          ; being sent by reference. The remaining
                          ; 1 to 27 significant bits are used to
                          ; encode a trait reference index (an
                          ; integer).

U29O-traits-ext  =  U29   ; The first (low) bit is a flag with
                          ; value 1. The second bit is a flag with
                          ; value 1. The third bit is a flag with
                          ; value 1. The remaining 1 to 26
                          ; significant bits are not significant
                          ; (the traits member count would always
                          ; be 0).

U29O-traits      =  U29   ; The first (low) bit is a flag with
                          ; value 1. The second bit is a flag with
                          ; value 1. The third bit is a flag with
                          ; value 0.  The fourth bit is a flag
                          ; specifying whether the type is
                          ; dynamic. A value of 0 implies not
                          ; dynamic, a value of 1 implies dynamic.
                          ; Dynamic types may have a set of name
                          ; value pairs for dynamic members after
                          ; the sealed member   section. The
                          ; remaining 1 to 25 significant bits are
                          ; used to encode the number of sealed
                          ; traits member names that follow after
                          ; the class name (an integer).
```

```
class-name       =  UTF-8-vr    ; Note: use the empty string for
                                ; anonymous classes.

dynamic-member   =  UTF-8-vr    ; Another dynamic member follows
                    value-type  ; until the string-type is the
                                ; empty string.

object-type      =  object-marker (U29O-ref | (U29O-traits-ext
                    class-name *(U8)) | U29O-traits-ref | (U29O-
                    traits class-name *(UTF-8-vr))) *(value-type)
                    *(dynamic-member)))
```

Note that for `U29O-traits-ext`, after the class-name follows an indeterminable number of bytes as `*(U8)`. This represents the completely custom serialization of "externalizable" types. The client and server have an agreement as to how to read in this information.

3.13 XML type

ActionScript 3.0 introduces a new XML type that supports E4X syntax. For serialization purposes the XML type needs to be flattened into a string representation. As with other strings in AMF, the content is encoded using UTF-8.

XML instances can be sent as a reference to a previously occurring XML instance by using an index to the implicit object reference table.

```
xml-type      =  xml-marker (U29O-ref |
                 (U29X-value *(UTF8-char)))
```

Note that this encoding imposes some theoretical limits on the use of XML. The byte-length of each UTF-8 encoded XML instance is limited to 2^{28} - 1 bytes (approx 256 MB).

3.14 ByteArray type

ActionScript 3.0 introduces a new type to hold an Array of bytes, namely ByteArray. AMF 3 serializes this type using a variable length encoding 29-bit integer for the byte-length prefix followed by the raw bytes of the ByteArray.

ByteArray instances can be sent as a reference to a previously occurring ByteArray instance by using an index to the implicit object reference table.

```
U29B-value    =  U29    ; The first (low) bit is a flag with
                        ; value 1. The remaining 1 to 28
                        ; significant bits are used to encode the
                        ; byte-length of the ByteArray.

bytearray-type  =  bytearray-marker (U29O-ref | U29B-value *(U8))
```

Note that this encoding imposes some theoretical limits on the use of ByteArray. The maximum byte-length of each ByteArray instance is limited to 2^{28} - 1 bytes (approx 256 MB).

4. Usages of AMF 3

4.1 NetConnection and AMF 3

In addition to serializing ActionScript types, AMF can be used in the asynchronous invocations of remote services. A simple messaging structure is used to send a batch of requests to a remote endpoint. The format of this messaging structure is AMF 0 (See [AMF0]. A context header value or message body can switch to AMF 3 encoding using the special avmplus-object-marker type.

Similar to AMF 0, AMF 3 object reference tables, object trait reference tables and string reference tables must be reset each time a new context header or message is processed.

4.1.1 NetConnection in ActionScript 3.0

The qualified class name for NetConnection in ActionScript 3.0 is flash.net.NetConnection. This class continues to use a responder to handle result and status responses from a remote endpoint, however, a strongly typed Responder class is now required. The fully qualified class name is flash.net.Responder. For events other than normal result and status responses NetConnection dispatches events for which the developer can add listeners. These events are outlined below:

asyncError	Dispatched when an exception is thrown asynchronously - i.e. from native asynchronous code.
ioError	Dispatched when an input or output error occurs that causes a network operation to fail.
netStatus	Dispatched when a NetConnection object is reporting its status or error condition.
securityError	Dispatched if a call to NetConnection.call() attempts to connect to a server outside the caller's security sandbox.

To handle an AMF context header a suitable method needs to be available, matching the header name. NetConnection is now a sealed type so either it must be subclassed or an object with a suitable implementation needs to be set for the NetConnection client property.

4.2 ByteArray, IDataInput and IDataOutput

ActionScript 3.0 introduced a new type to support the manipulation of raw data in the form of an Array of bytes, namely flash.utils.ByteArray. To assist with ActionScript Object serialization and copying, ByteArray implements flash.utils.IDataInput and flash.utils.IDataOutput. These interfaces specify utility methods that help write common types to byte streams. Two methods of interest are IDataOutput.writeObject and IDataInput.readObject. These methods encode objects using AMF. The version of AMF used to encode object data is controlled by the ByteArray.objectEncoding method, which can be set to either AMF 3 or AMF 0. An enumeration type, flash.net.ObjectEncoding, holds the constants for the versions of AMF - ObjectEncoding.AMF0 and ObjectEncoding.AMF3 respectively.

Note that ByteArray.writeObject uses one version of AMF to encode the entire object. Unlike NetConnection, ByteArray does not start out in AMF 0 and switch to AMF 3 (with the objectEncoding property set to AMF 3). Also note that ByteArray uses a new set of implicit reference tables for objects, object traits and strings for each readObject and writeObject call.

5. Normative References

[AMF0] Adobe Systems Inc. "Action Message Format - AMF 0", June 2006.

[RFC2234] D. Crocker., et. al. "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 3629, November 2003.