

Adobe Systems Inc.**AMF 3 規格仕様**カテゴリ: **ActionScript** シリアライゼーション

Action Message Format -- AMF 3

著作権表記

Copyright (c) Adobe Systems Inc. (2002-2006). All Rights Reserved.

概要

Action Message Format (AMF) は、ActionScript オブジェクトグラフのシリアライズに用いられるコンパクトなバイナリ形式です。シリアライズされた AMF エンコードドオブジェクトグラフは、セッション間でアプリケーションのパブリックステートを永続・抽出するために使用したり、厳格に型付けされたデータ交換を介した、2 つのエンドポイント間での通信を可能にします。

AMF は 2001 年に Flash Player 6 とともに登場し、Flash Player 7 の ActionScript 2.0 リリース時、ならびに Flash Player 8 のリリース時には、変更が施されませんでした。この当初のバージョンの AMF は AMF 0 ([AMF0] を参照) と呼ばれています。Flash Player 9 の登場にあわせて、最新の ActionScript Virtual Machine (AVM+) と Action Script 3.0 が登場しました。これらの改善・更新によって実現された、数々の新たなデータ型や言語機能に対応するには、AMF 自体にも更新が必要とされました。今回、新しいバージョンの AMF をリリースする機会を受けて、シリアライズされたデータ内に重複する情報が含まれることを防止できるよう、エンコーディング形式に対しても数々の最適化が施されています。本スペックは、更新版の AMF である AMF 3 の規格仕様を定義するものです。

目次

- 1 はじめに
 - 1.1 目的
 - 1.2 記法基準
 - 1.2.1 ABNF (Augmented BNF)
 - 1.3 基本規則
 - 1.3.1 可変長符号なしの 29 ビット整数エンコーディング
 - 1.3.2 スtring と UTF-8 について
- 2 技術概要
 - 2.1 改善点のまとめ
 - 2.2 参照テーブル
- 3 AMF 3 のデータ型
 - 3.1 概要
 - 3.2 undefined データ型
 - 3.3 null データ型
 - 3.4 false データ型
 - 3.5 true データ型
 - 3.6 integer データ型
 - 3.7 double データ型
 - 3.8 String データ型
 - 3.9 XMLDocument データ型
 - 3.10 Date データ型

- 3.11 Array データ型
- 3.12 Object データ型
- 3.13 XML データ型
- 3.14 ByteArray データ型
- 4 AMF 3 の使用
 - 4.1 NetConnection と AMF 3
 - 4.1.1 ActionScript 3.0 での NetConnection
 - 4.2 ByteArray、IDataInput および IDataOutput
- 5 引用規格

1 はじめに

1.1 目的

Action Message Format (AMF) は、ActionScript オブジェクトグラフのシリアライズに用いられるコンパクトなバイナリ形式です。シリアライズされた AMF エンコードドオブジェクトグラフは、セッション間でアプリケーションのパブリックステートを永続・抽出するために使用したり、厳格に型付けされたデータ交換を介した、2つのエンドポイント間での通信を可能にします。AMF の最初のバージョンである AMF 0 は複雑なオブジェクトの参照での送信をサポートし、オブジェクトグラフ内の冗長的なインスタンスが不必要に繰り返し送信されることを防止することができます。また、AMF 0 はエンドポイントにおけるオブジェクトの各種関係の復元、循環的な参照のサポートを提供するとともに、シリアライズ処理中に無限ループなどの問題が発生することを防止します。ActionScript 3.0 の登場にあわせてリリースされた最新バージョンの AMF、AMF 3 はオブジェクトインスタンスだけではなく、オブジェクトの特性やストリングも参照として送信できるよう、AMF 0 に数々の改善を施しています。また、AMF 3 には ActionScript 3.0 にて追加された、新しいデータ型のサポートも含まれています。

1.2 記法基準

1.2.1 ABNF (Augmented BNF)

本スペックの型定義には ABNF (Augmented Backus-Naur Form) 記法 [RFC2234] が用いられています。本文書を読み進むには、この記法に精通している必要があります。

1.3 基本規則

本文書では、バイトがオクテット (8 ビット) であることを前提とします。

U8	=	符号なしのバイト (8 ビット、オクテット)
U16	=	ビッグエンディアン (network) バイトオーダーで記された符号なしの 16 ビット整数
U32	=	ビッグエンディアン (network) バイトオーダーで記された符号なしの 32 ビット整数
DOUBLE	=	ネットワークバイトオーダーで記された 8 バイト IEEE-754 倍精度浮動小数点値 (符号ビットをローメモリで表現)
MB	=	メガバイト (1048576 バイト)

より複雑なデータ型の規則では特殊な扱い方が必要となります。これらは以下に解説されています。

1.3.1 可変長符号なしの 29 ビット整数エンコーディング

AMF 3 は、整数を記述する際にコンパクトな特殊フォーマットを利用することで、エンコーディング時に必要なバイト数を削減します。通常の 32 ビット整数同様、値を保持するためには最大 4 バイトが必要となりますが、最初の 3 バイトの上位ビットは、次のバイトが整数の一部であることを示すフラグとして用いられます。32 ビット中最大 3 ビットがフラグとして使用されるので、残りの 29 ビットのみが整数をエンコーディングするために使用できます。したがって、表現可能な最大の符号なし整数は $2^{29} - 1$ になります。

(hex)	:	(binary)
0x00000000 - 0x0000007F	:	0xxxxxxxx
0x00000080 - 0x00003FFF	:	1xxxxxxxx 0xxxxxxxx
0x00004000 - 0x001FFFFF	:	1xxxxxxxx 1xxxxxxxx 0xxxxxxxx
0x00200000 - 0x3FFFFFFF	:	1xxxxxxxx 1xxxxxxxx 1xxxxxxxx xxxxxxxx
0x40000000 - 0xFFFFFFFF	:	レンジ例外が発生

ABNF 記法では、可変長符号なし 29 ビット整数型を以下のように表します。

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 スtringと UTF-8 について

AMF 0 および AMF 3 は (非修正の) UTF-8 で String をエンコードします。UTF-8 は、8 ビット Unicode Transformation Format の略にあたります。通常、UTF-8 String はバイト長ヘッダで始まり、その後、一連の可変長 (1~4 オクテット) のエンコード済み Unicode コードポイントが続きます。AMF 3 は一部変更されたバイト長ヘッダを使用します。これらの解説は以下に提供されているとともに、本文書内で明確に表示されています。

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

ABNF 記法 [RFC3629] では、UTF-8 を以下の体裁で表します。

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

AMF 3 では、文字列を文字列リテラルまたは文字列参照としてエンコードできます。ヘッダには可変長符号なし 29 ビット整数が用いられ、最初のビットは、どの種類の文字列がエンコードされているかを示すフラグです。フラグが 0 の場合は文字列リテラルがエンコードされ、以後のビットは UTF-8 エンコードされた文字列のバイト長をエンコードするために使用されます。フラグが 1 の場合は文字列参照がエンコードされ、以後のビットは暗示的な文字列参照テーブルへのインデックスをエンコードするために使用されます。

U29S-ref	=	U29	; 最初の (ロー) ビットは値が 0 のフラグ。 ; 残りの 1~28 ビットは文字列参照テーブル ; インデックス (整数) をエンコードするために使用。
U29S-value	=	U29	; 最初の (ロー) ビットは値が 1 のフラグ。 ; 残りの 1~28 ビットは UTF-8 でエンコードされ、 ; 表現された文字列のバイト長を ; エンコードするために使用。
UTF-8-empty	=	0x01	; UTF-8-vr の空文字列。 ; 参照で送信されることは ; 一切ありません。
UTF-8-vr	=	U29S-ref (U29S-value *(UTF8-char))	

このエンコーディング方法には、文字列を使用する上で理論制限値が発生する点に注意する必要があります。参照で送信できる一意的な文字列の数は $2^{28} - 1$ に制限されます。また、UTF-8 でエンコードされた各文字列のバイト長は $2^{28} - 1$ バイト (約 256 MB) に制限されます。

2 技術概要

2.1 改善点のまとめ

AMF 3 における改善点および変更点を以下にまとめます。

- オブジェクトの `trait` を参照で送信できるようになりました
- スtring を参照で送信できるようになりました
- `int/uint` データ型のサポート
- `flash.utils.ByteArray` データ型のサポート、参照で送信することも可能
- `flash.utils.IExternalizable` のサポート
- 可変長エンコーディングスキームで整数をエンコードすることによるデータサイズの抑制
- 可変長整数を用いた参照の送信
- String UTF-8 長で可変長整数を使用
- 配列カウンタで可変長整数を使用
- 単一の Array タイプマーカで `strict` と `ECMA` の両方の配列に対応
- 日付とともにタイムゾーン情報が送信されなくなりました
- 日付を参照で送信できるようになりました
- `XMLDocument` UTF-8 長で可変長整数を使用
- `XMLDocument` を参照で送信できるようになりました
- `XML` データ型のサポート、参照で送信することも可能
- `XML` UTF-8 長で可変長整数を使用
- `ByteArray` タイプ長で可変長整数を使用
- ブール値の `true` および `false` が 1 バイトのタイプマーカとして送信されるようになりました
- `Unsupported` タイプマーカが削除されています
- `Reserved RecordSet` および `Movieclip` タイプマーカが削除されています

2.2 参照テーブル

AMF 3 では、String、複雑なオブジェクト (AMF 3 では匿名オブジェクト、型指定のなされたオブジェクト、Array、Date、XMLDocument、XML または ByteArray として定義) およびオブジェクトタイプの Trait が参照で送信できるようになりました。つまり、これらの AMF コンポーネントに関しては、同じ情報を繰り返し送信するのではなく、単に当該コンポーネントの以前の発生を参照するだけで済みます。この参照はゼロベースのインデックスを構成する整数で行われ、コンポーネント情報内にエンコードされます。通常は、所定のタイプマーカの直後に現れる最初の数字としてエンコードされます (詳細については、以下に示す Object、Array、Date、XMLDocument、XML および ByteArray のタイプ定義欄を参照のこと)。これらのインデックスによって、AMF 3 形式データの読み書き時にシリアライズおよびデシリアライズが必要とする、仮想的な参照テーブルが構成されます。

String、複雑なオブジェクト、オブジェクト Trait に対して、それぞれ個別の参照テーブルが用意される点に注意してください。

3 AMF 3 のデータ型

3.1 概要

AMF 3 には 13 種類のデータ型が用意されています。タイプマーカの長さは 1 バイトであり、タイプマーカに続くエンコード済みデータの種類を示します。

```
marker = U8
```

使用可能なタイプマーカの一覧を以下に示します(値の形式は 16 進法)。

```
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
```

タイプマーカの後は、エンコードされた実際の型データを続けることができます。ただし、マーカが 1 つの値のみしか示し得ないような場合 (null など) は、他の情報をエンコードする必要はありません。

```
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 は、ストリング、オブジェクトおよび **trait** (クラス名やパブリックメンバ名といった厳格なタイプを定義するオブジェクト特性) 用の 3 つの参照テーブルを利用します。これらのテーブルはフォーマット内で独自のエンティティとしてエンコードされないため、暗示的に扱われます。参照で送信可能な各データ型は、代替策として所定の参照テーブルへのインデックスを用いてエンコードできます。ストリングの場合は、ストリングテーブルへのインデックスを利用して参照で送信できます。Object、Array、XML、XMLDocument、ByteArray、Date およびユーザ定義クラスのインスタンスの場合は、オブジェクトテーブルへのインデックスを利用して参照で送信できます。Trait 情報を有するユーザ定義クラスのオブジェクトおよびインスタンスの場合は、trait テーブルへのインデックスを利用して参照で送信できます。

3.2 undefined データ型

undefined データ型は undefined タイプマーカによって表されます。値に他の情報をエンコードする必要はありません。

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

AVM 以外のエンドポイントでは undefined のコンセプトが非サポートであることがあります。この場合、undefined が null として扱われるケースがある点に注意してください。

3.3 null データ型

null データ型は null タイプマーカによって表されます。値に他の情報をエンコードする必要はありません。

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

3.4 false データ型

false データ型は false タイプマーカによって表され、ブール値の false をエンコードするために使用されます。ActionScript 3.0 では、プリミティブ型およびオブジェクト型のブール値という概念が存在しない点に注意してください。値に他の情報をエンコードする必要はありません。

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

3.5 true データ型

true データ型は true タイプマーカによって表され、ブール値の true をエンコードするために使用されます。ActionScript 3.0 では、プリミティブ型およびオブジェクト型のブール値という概念が存在しない点に注意してください。値に他の情報をエンコードする必要はありません。

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

3.6 integer データ型

AMF 3 では、可変長符号なし 29 ビット整数を用いて整数がシリアライズされます。AVM+では、ActionScript 3.0 の整数データ型 (符号付きの int データ型と符号なしの uint データ型) も 29 ビットを用いて表現されます。符号なし整数 (uint) の値が 2^{29} 以上の場合、または符号付き整数 (int) の値が 2^{28} 以上の場合、これらは AVM+によって double として表され、AMF 3 double データ型を用いてシリアライズされます。

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

3.7 double データ型

AMF 3 の double データ型は、AMF 0 の Number データ型と同じようにエンコードされます。このデータ型は ActionScript の Number や、 2^{28} 以上の値の ActionScript int、および 2^{29} 以上の値の ActionScript uint をエンコードするために用いられます。エンコード済みの値は、常に、ネットワークバイトオーダーで記された 8 バイト IEEE-754 倍精度浮動小数点値 (符号ビットをローメモリで表現) です。

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

3.8 String データ型

AMF 3 では、ActionScript String の値が単一の string データ型で表現されます。AMF 0 で用いられていた string データ型と long string データ型の概念は使用されません。

String は、暗示的な string 参照テーブルへのインデックスを用いることで、以前に発生した String への参照としても送信できます。

文字列は UTF-8 でエンコーディングされます。ただし、ヘッダは文字列リテラル、文字列参照のいずれかを表現します。

空の String が参照で送信されることは一切ありません。

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

3.9 XMLDocument データ型

ActionScript 3.0 からは新しい XML データ型 (3.13 を参照) が登場しましたが、当該言語には、従来の XMLDocument データ型が flash.xml.XMLDocument として維持されています。シリアライゼーションにあたっては XMLDocument の構造を、AMF 0 でのシリアライゼーション同様に、文字列での表現に展開する必要があります。AMF の他の文字列同様に、当該コンテンツは UTF-8 でエンコードされます。

XMLDocument は、暗示的なオブジェクト参照テーブルへのインデックスを用いることで、以前に発生した XMLDocument インスタンスへの参照としても送信できます。

```
U29X-value          =    U29          ; 最初の (ロー) ビットは値が 1 のフラグ。
                        ; 残りの 1~28 ビットは XML または XMLDocument の
                        ; UTF-8 エンコード済み表現のバイト長を
                        ; エンコードするために使用。
```

```
xml-doc-type        =    xml-doc-marker (U290-ref | (U29X-value
                        *(UTF8-char)))
```

このエンコーディング方法では、XMLDocument の使用に関して、理論的な制限値が発生する点に注意が必要です。UTF-8 でエンコードされた XMLDocument インスタンスのバイト長は、各自 $2^{28} - 1$ バイト (約 256 MB) に制限されます。

3.10 Date データ型

AMF 3 の場合、ActionScript の Date は、単に UTC (協定世界時) タイムゾーンの 1970 年 1 月 1 日 0 時を原点とした経過時間を、ミリ秒で換算してシリアライズされます。ローカルタイムゾーン情報は送信されません。

Date は、暗示的なオブジェクト参照テーブルへのインデックスを用いることで、以前に発生した Date インスタンスへの参照としても送信できます。

```
U29D-value          =    U29          ; 最初の (ロー) ビットは値が 1 のフラグ。
                        ; 残りのビットは使用されません。
```

```
date-time           =    DOUBLE      ; 64 ビット整数値を double で通信。
```

```
date-type           =    date-marker (U290-ref | (U29D-value date-time))
```

3.11 Array データ型

ActionScript の Array は、インデックスの種類や配列内での位置関係といった、インデックスの特性に基づいて記述されます。条件とその解説を以下の表に示します。

strict	数順式(数値型)のインデックスのみを含む
dense	0 で始まる数順式のインデックスであり、連続するインデックス間に空きがない(つまり、0 から配列の全体長に至るまでのすべてのインデックスが順序よく使用されている)
sparse	2 つのインデックス間に最低 1 つの空きが含まれている
associative	数順式以外(ストリング)のインデックスが少なくとも 1 つ含まれている(ECMA Array と呼ばれることがあります)

AMF は、Array を緻密な部分と連想部分の 2 つのパーツとして考慮します。連想部分のバイナリ表現は、空のストリングで区切られた名前と値のペア(なしの場合あり)から構成されます。緻密部分のバイナリ表現は、当該部分のサイズ(ゼロの場合あり)と、順序立てられた値のリスト(なしの場合あり)から構成されます。AMF でこれらを記述する際は、緻密部分のサイズ、空のストリングで区切られた一連の名前と値のペア、そしてそのサイズの値になります。

Array は、暗示的なオブジェクト参照テーブルへのインデックスを用いることで、以前に発生した Array への参照としても送信できます。

U29A-value	=	U29	;	最初の (ロー) ビットは値が 1 のフラグ。 ;	残りの 1~28 ビットは配列の緻密部分のカウントを ;	エンコードするために使用。
assoc-value	=	UTF-8-vr value-type				
array-type	=	array-marker (U290-ref (U29A-value (UTF-8-empty *(assoc-value) UTF-8-empty) *(value-type)))				

3.12 Object データ型

AMF 3 では単一のデータ型が ActionScript Object とカスタムユーザクラスを処理します。「trait」という表現は、あるクラスの定義をなす、特性を記述するために使用されます。ActionScript 3.0 では、匿名オブジェクトと型指定のなされたオブジェクトに加えて、オブジェクトがどのようにシリアライズされるかを記述する方法として、「dynamic」と「externalizable」という 2 つの trait が加わります。条件とその解説を以下の表に示します。

Anonymous	実際の ActionScript Object データ型のインスタンス、または登録されたエイリアスを有さない Class のインスタンス(デシリアライズ時に Object 同様に扱われるもの)
Typed	登録されたエイリアスを有する Class のインスタンス
Dynamic	ダイナミック trait が宣言された Class 定義のインスタンス。実行時に、パブリック変数メンバを動的に追加またはインスタンスから削除することが可能
Externalizable	flash.utils.IExternalizable を実装し、当該メンバのシリアライゼーションを完全に制御する Class のインスタンス(trait 情報には一切のプロパティ名が含まれない)

これらの特性に加えて、オブジェクトの **trait** 情報には一連のパブリック変数と、**Class** に定義されたパブリック読み書き可能なプロパティ名 (関数ではないパブリックメンバなど) が含まれることがあります。この際、**trait** 情報に続くメンバ値の順序と同じになるように、メンバ名の順序に注意する必要があります。これらのメンバは、データ型によって明示的に定義されているので、シール済みのメンバとして考慮されます。

データ型が **dynamic** である場合は、ダイナミックメンバを名前と値のペアで記したシール済みメンバの後に、追加セクションを含めることができます。空ストリングの名前に遭遇するまで、ダイナミックメンバの読み取りが継続的行われます。

Object は、暗示的なオブジェクト参照テーブルへのインデックスを用いることで、以前に発生した **Object** への参照としても送信できます。また、暗示的な **trait** 参照テーブルへのインデックスを用いることで、**trait** 情報を、以前に発生した一連の **trait** への参照としても送信できます。

U290-ref	=	U29	<ul style="list-style-type: none"> ; 最初の (ロー) ビットは値が 0 のフラグ。 ; (インスタンスが後に続くかの指定)。 ; 値が 0 の場合は、インスタンスではなく、 ; 参照であることを示す。残りの 1~28 ビットは ; オブジェクト参照インデックス (整数) を ; エンコードするために使用。
U290-traits-ref	=	U29	<ul style="list-style-type: none"> ; 最初の (ロー) ビットは値が 1 のフラグ。 ; 第 2 ビットは値が 0 のフラグ ; (trait 参照が後に続くかの指定) であり、 ; 当該オブジェクト trait が参照として ; 送信されることを示す。残りの 1~27 ビットは ; trait 参照インデックス (整数) を ; エンコードするために使用。
U290-traits-ext	=	U29	<ul style="list-style-type: none"> ; 最初の (ロー) ビットは値が 1 のフラグ。 ; 第 2 ビットは値が 1 のフラグ。 ; 第 3 ビットは値が 1 のフラグ。 ; 残りの 1~26 ビットは重要性なし ; (trait メンバカウントは常に 0) 。
U290-traits	=	U29	<ul style="list-style-type: none"> ; 最初の (ロー) ビットは値が 1 のフラグ。 ; 第 2 ビットは値が 1 のフラグ。 ; 第 3 ビットは値が 0 のフラグ。 ; 第 4 ビットは、当該データ型が ; ダイナミックであることを指定するためのフラグ。 ; 値が 0 の場合は、ダイナミックではなく、 ; 値が 1 の場合はダイナミックであることを示す。 ; Dynamic データ型には、シール済みメンバ ; セクションの後にダイナミックメンバ用の ; 一連の名前と値のペアが含まれることがあります。 ; 残りの 1~25 ビットはクラス名 (整数) の後に続く、 ; 一連のシール済み trait メンバ名を ; エンコードするために使用。

<code>class-name</code>	=	<code>UTF-8-vr</code>	; メモ: 匿名クラスの場合は、 ; 空のストリングを使用。
<code>dynamic-member</code>	=	<code>UTF-8-vr</code> <code>value-type</code>	; <code>string-type</code> が ; 空ストリングになるまで、 ; 別のダイナミックメンバが ; 後に続きます。
<code>object-type</code>	=	<code>object-marker (U290-ref (U290-traits-ext class-name *(U8)) U290-traits-ref (U290-traits class-name *(UTF-8-vr))) *(value-type) *(dynamic-member))</code>	

`U290-traits-ext` では、`class-name` の後に、確定不可能な数のバイトとして `*(U8)` が続く点に注意してください。これにより、「externalizable」データ型の完全なカスタムシリアライゼーションが表現されます。クライアントとサーバの両方において、この情報をどのように解釈するか同意が行われます。

3.13 XML データ型

ActionScript 3.0 からは、E4X 構文をサポートする新しい XML データ型が登場しました。シリアライゼーションを行うためには、XML データ型をストリング表現に展開する必要があります。AMF の他のストリング同様、当該コンテンツは UTF-8 でエンコードされます。

XML インスタンスは、暗示的なオブジェクト参照テーブルへのインデックスを用いることで、以前に発生した XML インスタンスへの参照としても送信できます。

<code>xml-type</code>	=	<code>xml-marker (U290-ref (U29X-value *(UTF8-char)))</code>
-----------------------	---	--

このエンコーディング方法では、XML の使用に関して、理論的な制限値が発生する点に注意が必要です。UTF-8 でエンコードされた XML インスタンスのバイト長は、各自 $2^{28} - 1$ バイト(約 256 MB)に制限されます。

3.14 ByteArray データ型

ActionScript 3.0 からは、バイトの配列を保持するための新しいデータ型として、`ByteArray` が登場しました。AMF 3 は、バイト長接頭辞の部分に可変長エンコーディング 29 ビット整数を利用し、その後に `ByteArray` の生のバイトを記すことで、このデータ型をシリアライズします。

`ByteArray` インスタンスは、暗示的なオブジェクト参照テーブルへのインデックスを用いることで、以前に発生した `ByteArray` インスタンスへの参照としても送信できます。

<code>U29B-value</code>	=	<code>U29</code>	; 最初の (ロー) ビットは値が 1 のフラグ。 ; 残りの 1~28 ビットは <code>ByteArray</code> の ; バイト長をエンコードするために使用。
-------------------------	---	------------------	---

<code>bytearray-type</code>	=	<code>bytearray-marker (U290-ref U29B-value *(U8))</code>
-----------------------------	---	---

このエンコーディング方法では、`ByteArray` の使用に関して、理論的な制限値が発生する点に注意が必要です。`ByteArray` インスタンスの許容最大バイト長は、各自 $2^{28} - 1$ バイト(約 256 MB)に制限されます。

4 AMF 3 の使用

4.1 NetConnection と AMF 3

ActionScript データ型のシリアライズに加えて、AMF はリモートサービスの非同期的な呼び出しにも使用することができます。この際、リモートエンドポイントへの一連のリクエスト送信には、簡潔なメッセージング構造が用いられます。このメッセージング構造の書式は AMF 0 です ([AMF0] を参照)。コンテキストヘッダ値およびメッセージ本文の変換には、特殊な `avmplus-object-marker` タイプを利用して、AMF 3 エンコーディングに切り替えることができます。

AMF 0 同様に、新たなコンテキストヘッダおよびメッセージが処理される際には、その都度 AMF 3 のオブジェクト参照テーブル、オブジェクト `trait` 参照テーブルおよびストリング参照テーブルをリセットする必要があります。

4.1.1 ActionScript 3.0 での NetConnection

ActionScript 3.0 における `NetConnection` の有効なクラス名は `flash.net.NetConnection` です。このクラスは、リモートエンドポイントからの結果とステータスレスポンスの処理に、引き続きレスポндаを使用しますが、今後は厳格に型付けされた `Responder` クラスが必要となります。完全に有効なクラス名は `flash.net.Responder` です。通常の結果およびステータスレスポンス以外のイベントに関しては、デベロッパーがリスナを追加することができるイベントを `NetConnection` がディスパッチします。これらのイベントを以下にまとめます。

<code>asyncError</code>	非同期的に例外が発生した際にディスパッチ (ネイティブ非同期コードからなど)
<code>ioError</code>	ネットワーク処理の失敗の原因となるような I/O エラーが発生した際にディスパッチ
<code>netStatus</code>	<code>NetConnection</code> オブジェクトがステータスやエラー状況をレポートする際にディスパッチ
<code>securityError</code>	<code>NetConnection.call()</code> へのコールが、コーラーのセキュリティサンドボックス外にあるサーバへの接続を試みた際にディスパッチ

AMF コンテキストヘッダを処理するには、ヘッダ名に対応する適切なメソッドが用意されている必要があります。`NetConnection` は今後シール済みのデータ型であるため、サブクラスされているか、`NetConnection` クライアントプロパティに適切な実装が行われたオブジェクトである必要があります。

4.2 ByteArray、IDataInput および IDataOutput

ActionScript 3.0 からは、バイト配列の形式で生のデータが扱える、新しいデータ型の `flash.utils.ByteArray` がサポートされています。ActionScript Object のシリアライゼーションと複製を支援するために、`ByteArray` には `flash.utils.IDataInput` と `flash.utils.IDataOutput` が実装されています。これらのインタフェースは、汎用的なデータ型のバイトストリームへの書き出しを支援するユーティリティメソッドを指定します。注目に値するのは、`IDataOutput.writeObject` と `IDataInput.readObject` の 2 つのメソッドです。これらのメソッドは、AMF を利用してオブジェクトをエンコードします。オブジェクトデータのエンコードにどのバージョンの AMF を使用するかは `ByteArray.objectEncoding` メソッドによって制御され、AMF 3 または AMF 0 のいずれにも設定することができます。AMF バージョンごとの定数は、一覧データ型の `flash.net.ObjectEncoding` (`ObjectEncoding.AMF0` および `ObjectEncoding.AMF3`) によって、それぞれ保持されています。

`ByteArray.writeObject` はすべてのオブジェクトのエンコードに、単一バージョンの AMF を使用する点に注意してください。 `NetConnection` とは異なり、 `ByteArray` は、当初 AMF 0 を使用し、途中で AMF 3 に切り替わるようなことはありません (`objectEncoding` プロパティが AMF 3 に設定されている場合)。また、 `ByteArray` は `readObject` コールおよび `writeObject` コールごとに、オブジェクト、オブジェクト trait およびストリング用の新たな一連の暗示参照テーブルを使用する点にも注意が必要です。

5 引用規格

- [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.