DATA MARKUP LANGUAGE – PRIMITIVE SETS

Contact: Wiley Black (WBlack@optics.arizona.edu)

Specification Version: V3.1

CONTENTS

OVERVIEW

The Data Markup Language (DML) defines the encoding of container structure within a document or stream. Primitive types are included in the DML which provide the building blocks of data represented by the document. Rather than define a strict set of primitives from which all implementations must adhere, DML provides a small base set of primitives and a mechanism to select additional sets of primitives.

This system separates the DML encoding from the primitive encoding, and allows variation of primitive encodings without requiring changes to the DML structural specification.

The base set of primitives supported by DML is called the “base primitives” and includes:

- Unsigned Integer
- String (UTF8)
- Byte Arrays

The following extensions are presently defined:

- Common types
    - Signed Integer
    - Boolean
    - Floating-Point (Single and Double Precision, base-2)
    - Date and Time
    - Available in Little-Endian or Big-Endian
- Extended Precision Floating Point (Base-2)
- Base-10 Floating Point
- Arrays & Matrices
    - Little-Endian
    - Big-Endian
- Compressed Opaque Data (lossless)
- Encrypted Opaque Data

Including a primitive set into a translation requires specifying the primitives to be utilized. Using a primitive in DML content also requires that the appropriate codec be specified. For example, including arrays involves specifying that you require array primitives and that they should use either a big or little endian codec.

It is possible to request primitives without specifying their codec up-front. For example, a video format might specify that a video primitive will occur, but the codec is not selected in the translation document. The individual content document includes an addition request for the same primitives but specifies a codec.

It is possible for primitive encodings to conflict – for example a single translation cannot include both big and little-endian arrays and matrices. The DML parsing rule is that the last primitive directive overrides any previous statements. Since the DML content document header is parsed last, it receives final say on codecs; although a DML reader may throw an error message if a requested codec is not supported and cannot be installed.

Use of extended primitives in a file format or DML stream requires adding an inclusion statement to the DML header or translation document body. The inclusion statement has a form:

```
<DML:Include-Primitives DML:Set="…"/>
```

Or,

```
<DML:Include-Primitives DML:Set="…" DML:Codec="…"/>
```

The DML:Set attribute is a string identifying the particular primitive set to be included. The DML:Codec attribute specifies the encoding applied to the primitives. Optionally, the <DML:Include-Primitive> directive can be a container with arbitrary additional elements that are passed to the codec for configuration.

The rules for include primitive statements are:

- If a translation document will be used, the <DML:Include-Primitives DML:Set="…" /> statement must be placed in the translation document before the primitive type is referenced.
- If the extended primitive will be referenced from inline identification, then a <DML:Include-Primitives DML:Set="…" /> directive must appear in the document's header or included translation documents.
- Multiple inclusion of the same primitive set has no effect, but a codec specification can override a previous codec specification. Any configuration specification also overrides previous configuration specification, which allows a translation to specify a default codec and configuration without mandating that it be the final choice used in content.
- <DML:Include-Primitives /> directives are processed in the order they are encountered after all <DML:Include-Translation /> directives have been logically completed.
- <DML:Include-Primitives /> directives *may* be placed inside of a <container> definition in translation language, but the context is ignored by the DML Reader.

Once the primitive set is included, additional type values are permitted within the translation document and inline identification.

A DML Reader *may* choose not to fully implement all primitive sets, although a DML reader *must* support the base primitives set. Rules for reader support are:

- A DML Reader *may* attempt to add or install support for a new codec or primitive set that it encounters, but the DML Reader *must* take proper security precautions and acquire proper authorization before installing and enabling unknown executable software.
- A DML Reader *must* fail gracefully when it encounters a primitive node with a codec that it does not support.
- A DML Reader *may* fail gracefully when it encounters an <DML:Include-Primitives> statement with either a primitive set or codec that it does not support.

All primitives are prefaced by a node head as discussed in the "Data Markup Language" specification. Many primitives make use of Compact-64 values to represent a data size, also described in the "Data Markup Language" specification.

The existence of a primitive in one of the sets defined here does not preclude another definition to accomplish the same purpose. For example, the definition of Booleans provided by the common set is simple, easy to implement, and covers most use cases; however a more compact definition is possible and might be preferred in some

situations. Types may not conflict between multiple primitive sets that are included simultaneously. For example, a DML Reader *should* generate an error if two separate primitive sets are included that both define "int", and *must* generate an error if a primitive node is encountered for which multiple primitive sets (and thus codecs) apply.

SUMMARY OF PRIMITIVE SETS

The currently defined primitive sets include the built-in base set, the common primitives, extended precision floating-point, base-10 floating-point, arrays & matrix primitives, compression, and encryption primitives. Multiple codecs exist for all the primitive sets except the base set.

Most primitive sets include two codecs – "be" and "le". These refer to big-endian and little-endian respectively.

Primitive Set Name	**Available Codecs**	**Available Primitive Types**
base	N/A	uint string array-U8
common	be le	int boolean single double datetime
ext-precision	be le	double-ext quad
decimal-float	be le	decimal-128
arrays	be le	array-U16 array-U24 array-U32 array-U64 array-I8 array-I16 array-I24 array-I32 array-I64 array-SF array-DF array-DT array-S matrix-U8 matrix-U16 matrix-U24 matrix-U32 matrix-U64 matrix-I8 matrix-I16 matrix-I24 matrix-I32 matrix-I64 matrix-SF matrix-DF
decimal-array	be	array-10F

	le	matrix-10F
dml-ec1	v1	encrypted-dml compressed-dml

BASE PRIMITIVE ENCODING

A primitive type may be either an attribute or an element as discussed in the DML specification, but the distinction affects only grammar rules and encoding is the same in either case. A single "built-in" codec is used for the base set.

UNSIGNED INTEGER TYPES

When the type marker indicates an unsigned integer (type="uint"), the node consists of a single Compact-64 value. Values must be zero-padded in memory after reading. For example, if the value 5 is read from a single-byte into a 64-bit word in memory, it must be zero padded to 0x0000000000000005.

UTF-8 STRING TYPES

When the type marker indicates a string type (type="string"), a data size value is encoded as Compact-64 and is followed by the string in UTF-8 encoding. Data size provides the length of the string in bytes, not characters, but does not count itself or the node head. The string is not terminated since its data size is specified.

BYTE ARRAY ENCODING ("OPAQUE DATA")

When the type marker indicates a byte array data type (type="array-U8"), the node consists of a Compact-64 data size indicator followed by data. The data is a collection of bytes, of length data size, which the parser can read but not interpret.

This encoding follows the same template as the array primitive set, except that for byte-sized data there are no endian considerations and unsigned data has no sign considerations.

COMMON PRIMITIVES

Additional commonly used types are provided by the common primitive set for elementary signed integers, Booleans, floating-point, and date and time values. These types can be encoded in either big or little-endian, and so there are two codecs available for representation. To use the set, one of the following directives must be present in the translation document and/or DML Header:

```
<DML:Include-Primitives DML:Set="common" DML:Codec="be" />
```

The "be" codec provides the big-endian form of these types. Alternatively,

```
<DML:Include-Primitives DML:Set="common" DML:Codec="le" />
```

The "le" codec provides the little-endian form of these types.

SIGNED INTEGER TYPES

When the type marker indicates a signed integer (type="int"), the primitive contains a signed integer up to 64-bits in length. A 2's complement value similar to the Compact-64 encoding is used, but the value ranges change for a signed number. Values must be sign-extended after reading. For example, -2 will be read as 0xFE and can be sign-extended to a 64-bit word as 0xFFFFFFFFFFFFFFFE.

The Compact-S64 (signed) encoding is given as:

Binary Encoding	**Minimum Value**	**Maximum Value**
1xxx xxxx	-2^{6}	$2^{6}-1$
01xx xxxx xxxx xxxx	-2^{13}	$2^{13}-1$
001x xxxx xxxx xxxx xxxx xxxx	-2^{20}	$2^{20}-1$
0001 xxxx xxxx xxxx … xxxx xxxx	-2^{27}	$2^{27}-1$
0000 1xxx xxxx xxxx … xxxx xxxx	-2^{34}	$2^{34}-1$
0000 01xx xxxx xxxx … xxxx xxxx	-2^{41}	$2^{41}-1$
0000 001x xxxx xxxx … xxxx xxxx	-2^{48}	$2^{48}-1$
0000 0001 xxxx xxxx … xxxx xxxx	-2^{55}	$2^{55}-1$
0000 0000 xxxx xxxx … xxxx xxxx	-2^{63}	$2^{63}-1$

Where the shortest binary encoding for which the value is contained between the minimum and maximum values is used. As with all the Compact encodings, big-endian sequencing is utilized when multiple bytes are present.

Caution: the use of big-endian in the signed integer type is counterintuitive given that a little-endian codec can be specified. The "be" and "le" codecs of the common set only differ for the fixed-length primitives: floating-point and date/time.

BOOLEAN TYPES

When the type marker indicates a Boolean (type="boolean"), the primitive consists of a single byte of data. False values are represented by zero and true values are represented by any non-zero value. For a sufficiently low DML ID, a Boolean primitive is encoded in 2 bytes.

FLOATING-POINT TYPES

When the type marker indicates a floating-point value (type="single" or type="double"), the data content provides a base-2 floating-point value following the common IEEE 754 representation. When the type marker is "single", the data content is 32-bits. When the type marker is "double", the data content is 64-bits. The bytes are encoded in either little or big-endian depending on the codec.

DATE AND TIME TYPES

When the type marker indicates a date value (type="datetime"), the data content is a signed 8-byte integer representing a date and time as the number of nanoseconds elapsed since the beginning of the millennium (at 2001-01-01T00:00:00.000000000 UTC.) Negative values can represent dates before the millennium. The value is always specified in the UTC time zone. The bytes are encoded in either little or big-endian depending on the codec. This format supports nanosecond precision for the date range of 1708-09-22T00:12:44.000000000Z through 2293-04-11T23:47:16.000000000Z (approx. +/- 293 years from Epoch).

EXTENDED PRECISION FLOATING-POINT PRIMITIVES

When extended precision floating-point is required, the following directive must be present in the translation document and/or DML Header:

```
<DML:Include-Primitives DML:Set="ext-precision" DML:Codec="be" />
```

Or,

```
<DML:Include-Primitives DML:Set="ext-precision" DML:Codec="le" />
```

Two types are added by this set, both representing base-2 floating-point values in big-endian IEEE-754-2008 representation:

- Double-extended (10 bytes) when type="double-ext".
- Quadruple precision if data size is 16 bytes when type="quad".

The encoding follows the same pattern as the common floating-point types except for their additional size and precision.

BASE-10 FLOATING-POINT PRIMITIVES

When base-10 floating-point primitives are required, the following directive must be present in the translation document and/or DML Header:

```
<DML:Include-Primitives DML:Set="decimal-float" DML:Codec="be" />
```

Or,

```
<DML:Include-Primitives DML:Set="decimal-float" DML:Codec="le" />
```

Once included, one additional primitive type encoding (decimal-128 type) is defined by the translation and permitted in content.

DECIMAL FLOATING-POINT TYPE

The decimal floating-point type (type="decimal-128") provides encoding of a floating-point number utilizing a base-10 radix. The decimal type is described and encoded by the IEEE 754 specification. The decimal-128 type is 16 bytes. Unlike ordinary floating-point types, it permits exact representation of quantities defined in base-10, and is particularly useful for financial calculations.

ARRAY & MATRIX PRIMITIVES

There are innumerable ways to encode and represent arrays and matrices. We define two encodings that will support the majority of cases encountered.

The big-endian array and matrix primitives are included by the directive:

```
<DML:Include-Primitives DML:Set="arrays" DML:Codec="be" />
```

The little-endian array and matrix primitives are included by the directive:

```
<DML:Include-Primitives DML:Set="arrays" DML:Codec="le" />
```

Several array types are endian-neutral and are defined identically in either codec. With the exception of strings, both arrays and matrices follow a consistent pattern. Therefore, they add available types "array-xxx" and "matrix-xxx" to the available type selections where the "xxx" suffix is described in table 1.

Name Suffix	Description
-U8	Unsigned 8-bit integer *, **
-U16	Unsigned 16-bit integer
-U24	Unsigned 24-bit integer
-U32	Unsigned 32-bit integer
-U64	Unsigned 64-bit integer
-I8	Signed 8-bit integer **
-I16	Signed 16-bit integer
-I24	Signed 24-bit integer
-I32	Signed 32-bit integer
-I64	Signed 64-bit integer
-SF	Single-precision (32-bit) floating-point
-DF	Double-precision (64-bit) floating-point
-DT	Date/time (8-byte) ***
-S	String **, ***

* The array-U8 type comes from the base set but matches the encoding template of all arrays.
** Types which are endian-neutral and have an identical representation in either codec.
*** Types which are allowed for arrays but not for matrices.

Table 1. Suffixes and matching type

ARRAY ENCODING

With the exception of string arrays, a node with array type contains a simple encoding scheme consisting of a count (Compact-64) followed by some number of unit primitives using a fixed unit size (i.e. 4 bytes per 32-bit signed integer). The count gives the number of unit primitives in the array. The unit primitive utilized (and its unit size) are identified by the suffix in the type specification for the node. Signed unit primitives use 2's complement format.

The count is stored in a Compact-64 value and is not affected by the codec selection. Following the Compact-64 count, each unit primitive is written in either big or little-endian format, with the first unit primitive representing the lowest index of the array. The number of bytes in the array is equal to the count times the size of the unit primitive.

For example, if the count is 3 and the DML ID indicates that the type is "Array-I32", then the encoding consists of the DML Identifier, followed by count (encoded as 0x83 in this case), and three 32-bit integers in sequence. The node occupies a total of 14 bytes.

STRING ARRAY ENCODING

Strings are variable-length, so their encoding varies from other arrays. First, a Compact-64 gives the count of the number of unit strings that follow. Then, each unit string is prefixed with a data size indicator as a Compact-64. The data size indicator gives the number of bytes (not characters) in each string.

Empty unit strings are permitted by providing a data size indicator of zero. An empty string is immediately followed by the data size indicator for the next string.

2-D MATRIX

All matrices begin with two Compact-64 values. The first value gives the number of columns and the second value gives the number of rows. A linear array follows in a row-major representation. The array contains fixed units (i.e. 4 bytes per each 32-bit signed integer). The Compact-64 values are not affected by codec selection, but each fixed-length unit primitive is stored in little or big endian depending on codec.

String and date-time matrices are not defined in this set.

BASE-10 FLOATING-POINT ARRAYS AND MATRICES

Primitive arrays of base-10 floating-point values, with 16 byte units, can be included with the statement:

```
<DML:Include-Primitives DML:Set="decimal-array" DML:Codec="be" />
```

Or.

```
<DML:Include-Primitives DML:Set="decimal-array" DML:Codec="le" />
```

Encoding for these arrays follows the same pattern as the integer and floating-point arrays described above. Two new primitive types become available: "array-10F" and "matrix-10F".

DML ENCRYPTION AND COMPRESSION

Two additional primitives are defined in concert with the DML Encryption and Compression layer (DML-EC). These primitives and the DML-EC layer are outlined in the separate specification, "DML: Encryption and Compression". The primitives are not intended for use outside of the structure defined in the DML-EC layer.

DEFINING ADDITIONAL PRIMITIVES

The Data Markup Language is designed to expand. Extensibility is provided by the DML structure as well as its ability to support flexible encodings including a variety of compressed and encrypted data structures.

It is impossible to anticipate all needs, however it should be pointed out that use of the predefined primitive sets permits a high degree of compatibility with available DML software. The definition of custom primitive sets is therefore recommended only when consideration of the existing primitives have been exhausted.

Should a custom primitive set be required, it is recommended that the manufacturer/origination name be made part of the primitives attribute. For example, "ArbitraryCompany-fixed-point" might be a highly-unique name for a primitive set of special fixed-point integers.