DATA MARKUP LANGUAGE - TRANSLATION

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

Specification Version: V3.1 Draft

CONTENTS

OVERVIEW

Parsing of DML content requires lookup of identifiers in a table called the translation. This translation can be defined in the header of the DML document, implicitly (i.e. hard-coded), and/or by referencing Translation documents that can themselves be either DML or XML documents. The encoding and language used to describe translations is identical in the DML header and a DML Translation document body, and is logically the same in an XML translation document. DML can also avoid the use of a translation lookup by encoding the necessary information in the DML stream using inline identification, which is useful in many experimental applications but is not compact.

The <DML:Header> of a content document can include translation documents by providing a URI. Translation documents can also reference additional translation documents much in the way that a header file in the C language can continue to include additional header files. This allows the construction of complex file formats. For example, the translation definitions required to describe the encoding of video data can be quite complex and large. The translation definitions required to describe a slide show file format can also be quite complex and large if the slideshow provides a rich feature set. Such a translation can be simplified by defining the slideshow translation in one translation document and using a <DML:Include-Translation DML:URI=…/> directive to embed the video rules inside of a <Video> container. This improves readability in the slideshow translation, supports cacheability, and allows separation of responsibility for the two complex documents.

The translation definition provides "associations" between each DML node and properties of the node. The properties described in an association include the name of the node and the type of the node. In order to locate the properties for a given node, both the DML ID and the current context must be considered. The context refers to the set of containers that the node is contained inside, or more simply as the location on the tree.

DML Translations should be written in XML and generally represented in XML. Some implementations may only understand DML, and a DML version of the translation document will be required; however conversion utilities from XML to DML are available. Whenever practical, DML readers and writers *should* support XML translation documents.

TRANSLATION REFERENCE FROM CONTENT DML

The translation language provides the <DML:Include-Translation> directive. The DML:URI attribute specifies the URI at which the translation document can be accessed. This directive allows the DML Header to reference a translation document. Since both the DML Header and translation documents share the same translation language, the translation document can then reference additional translation documents if necessary. In the case of a DML fragment (which contains no header), reference to a translation document is usually implied and may still be accessed using a priori or hardcoded knowledge.

For developmental or small DML projects, the simplest mode of operation for DML is to use inline identification for all content. This mode is not as efficient as DML identifiers. When maximum efficiency is required, then the project should move toward a translation document instead. The creation of an XML-based translation document can be helpful in laying out the structure of a file format as it strongly resembles a schema.

For proprietary formats that are not intended for interchange (or only interchange to other instances of the same software), the use of a URN scheme is still recommended for the URI. For example, a <DML:Include-Translation/> statement might have the attribute URI="urn:uuid:6e8bc430-9c3a-11d9-9669-0800200c9a66". The proprietary

software must simply be programmed to recognize the URN and have an internal copy of the resource (or a URL) when it encounters it. When a generic parser encounters a URN that it does not recognize, it can prompt the user to provide a file for the translation document.

Two built-in DML Translations must be supported by any fully generic DML parser: the dml3 and the tsl2 translations. The dml3 translation contains a reference to the tsl2 translation within the DML:Header context. The tsl2 translation describes the translation language described in this document. The dml3 translation describes the remainder of basic DML structure and contains the tsl2 translation within its DML:Header container. These built-in translations are described in XML format at the end of the "Data Markup Language" and this specification. All DML readers should automatically assume the built-in dml3 translation is available, but it can be explicitly referenced as:

<DML:Include-Translation DML:URI="urn:dml:dml3"/>

A DML translation document has its own header that should identify the translation language as:

<DML:Include-Translation DML:URI="urn:dml:tsl2" />

DML Content can also be provided using only inline identification, creating a self-contained document. Alternatively, a DML:Header can fully describe the translation of the document's body to achieve a self-contained document, but this is not a recommended practice for two reasons: 1) translation documents are better suited to XML for readability and maintainability and 2) the use of a separate translation document provides cacheability and compactness.

CAPABILITIES ANALYSIS FROM TRANSLATION

The capabilities required by a DML reader in order to parse a given DML document are also contained in the DML header and its referenced translation documents. The list of primitives required for a particular file format can be determined by examining `<DML:Include-Primitives DML:Set="…" />` directives. See the Primitive Sets section and the "Data Markup Language – Primitives" specification for more information.

CONTEXT-SENSITIVE IDENTIFIERS

Each DML container *may* include a "local translation". Locating the properties for a node (the "identification scan") involves first scanning the local translation (if present) of the current container. If no local translation is found, then the parent container is checked for a local translation, and so on until a container is found with a translation (or the "global" translation is reached). Once a local translation is found, a search begins for the DML ID. If no DML ID is matched in the local translation, the parent translation is scanned. This excludes any related lower-level translations from the parent, or in other words, the scan proceeds only "up" the tree, never back down through siblings.

The identification scan proceeds first up the DML *content* tree to find the first local translation. Once a local translation is found, the scan proceeds up the DML *translation* tree to locate a DML ID match. This algorithm provides a fast search while also permitting flexibility.

By way of an example, consider the following DML tree:

```
<Top-Level>
```

```
      <Alpha Code="A" />
      <Alpha Code="B" />
      <Beta><Code>3</Code></Beta>
      <Alpha Code="D"><Beta><Code>4</Code></Beta></Alpha>
</Top-Level>
```

In the DML Translation, Code might be the name for:

- Two different top-level global DML IDs each with name "Code" but with different types (one being a string and the other an integer.)
- DML IDs located in two different contexts – one inside of the Alpha local translation and one inside of the Beta local translation.

The translation language provides an optional "usage" attribute to describe primitives that allows them to be reserved for attribute or element use. A DML Parser *may* enforce this attribute, and higher-level software *may* use an XML schema for the document to enforce more complex grammatical rules. A DML Parser *may* enforce requirements for unique node names within a context, such as disallowing "Code" in the above example from referencing two different primitives in the same context. This would still make re-use of the same name permissible in different contexts, such as the 2nd case where the definition of "Code" is constrained to the local Alpha and local Beta translations.

The scan sequencing/algorithm described above clarifies complexities that may be contained in the translation definitions. Consider another example:

```
<DML:Header><DML:Include-Translation DML:URI="urn:dml:tsl2" /></DML:Header>
<DMLTranslation
      DML:Version="2"
      >

      <Container id="1" name="Slideshow">
            <Container id="1" name="New-Slide" />
      </Container>
      <Container id="2" name="Video">
            <Node id="1" name="Bitrate" type="string" />
            <DML:Include-Translation
                  DML:URI="http://www.optics.arizona.edu/asl/DML/Video.dml"
                  />
      </Container>
      <Container id="3" name="Audio">
            <Node id="1" name="Bitrate" type="uint" />
      </Container>
</DMLTranslation>
```

And related DML content as follows:

```
<Slideshow>
      <Audio Bitrate="65536" />
      <New-Slide>
            <Video Bitrate="variable"><Audio Bitrate="16384" /></Video>
      </New-Slide>
</Slideshow>
```

The first time Bitrate is encountered, the scan algorithm finds a local translation associated with the immediate container, Audio. In the Audio local translation, Birate is a uint attribute with ID 1. The next time Bitrate is

encountered, Video is the local translation and Bitrate is a string attribute with ID 1. The last encounter has Audio again as the local translation and Bitrate is recognized as a uint with ID 1.

In this example, the Bitrate definition is being overridden, but an important point is that the Bitrate definition of the Video translation is not available inside of an Audio translation, and vice-versa. This is because both Bitrate associations were made inside of a local translation, and the scan algorithm begins by locating the current DML translation through content examination and then proceeds up the translation tree. As soon as the Audio or Video container is identified, the algorithm proceeds only up the translation tree, and neither Audio nor Video is contained inside each other in the translation tree (even if they were contained within one another in content). In this example, both Audio and Video are defined at the global level, but if Audio were defined inside of Video in the translation then it would be impossible for Audio to exist outside of Video in the content as well. When it is desirable for an identification to be accessible to multiple contexts, it is achieved simply by providing the necessary definitions at a higher level. Further constraints on what nodes may be contained within each other can be by schema definition and enforcement.

Also note that built-in DML IDs associated with DML structure cannot be redefined and it is an error to attempt to do so. This includes the End-Attribute marker node, End-Container marker node, comment node, and padding nodes. The tsl2 nodes associated with the DML:Header can be redefined since they are being used inside another container (with the exception of DML:Header itself) and have local translation context only.

In addition to the optional uniqueness rules specified above, there are two constraints on the use of DML associations that a DML writer *must* follow:

1. Each DML ID must be unique within a single translation level.
2. Two different DML IDs cannot have identical properties (name and type) within a single translation level. An identical set of properties might exist at a higher level – this permits “redefining” from the name side just as redefining is permitted on the DML ID.

By the bottom-up search mechanism defined, associations can also be made solely at the top-level. Associations made at the top-level can appear at *any* level of the DML tree, unless the DML ID is redefined inside a local translation.

An example of local translation context being employed can be found in the base dml3 translation itself. The DML:Header container is defined to contain a local translation, which is the tsl2 translation included by URN. A translation document also uses the tsl2 translation as its top-level translation.

DML TRANSLATION DOCUMENT FORMAT

A DML-Based translation document contains its own DML Header and a body of <DML:Translation> type. While DML is specified as version 3, the translation language is in version 2 with this specification, thus a DML:Version=”2” attribute should be present on the body container. The header *must* contain a <DML:Include-Translation DML:URI=”urn:dml:tsl2” /> directive.

The presence of a DML:Header in both the content and translation document can be confusing. The DML:Header always provides information necessary to decode the body of *that* document. The DML:Header of the content document describes how to decode the content body while the DML:Header of the translation document describes how to decode the body of the translation document. Since the translation document relies entirely on built-in primitives, a translation document’s header should be identical in all instances, described in XML as:

```
<DML:Header
      DML:Version=”3” DML:ReadVersion=”3” DML:DocType=”DML:Translation”
      >
      <DML:Include-Translation DML:URI=”urn:dml:tsl2” />
</DML:Header>
```

The specification of Version 3 in the header refers to the version of DML required to decode the translation document's body. The translation document body would specify a version of 2, because that is the version of translation content.

XML TRANSLATION DOCUMENT FORMAT

To use an XML translation document, simply use a root-node of <DML:Translation>. An XML preprocessing directive is allowed to precede the root-node, and standard XML format applies. When using an XML translation document, no DML:Header is necessary.

TRANSLATION BODY

In both DML and XML Translation documents, the translation document body consists of a <DML:Translation> container as the top-level. A DML:TranslationURN attribute *may* be attached to the <DML:Translation> container. When present, then the DML Reader *should* verify the URN against the DML:URN provided in the DML:Include-Translation directive of any documents that reference it. This can be used to verify that the correct translation document was located. For example, a translation document is provided at a certain URI and various software depends on the translation. If the translation document at the URI is modified and content documents provide a URN in their DML:Include-Translation directives, then changing the URN provides an error detection mechanism for older documents. Note that recommended practice in a production environment is to also change the URI for each new translation version.

Between the <DML:Translation> opening and </DML:Translation> tags, the translation language is employed. This is the same language used in the <DML:Header>.

TRANSLATION LANGUAGE

The translation document provides a permanent association between a textual, human-readable identifier and a DML Identifier, the type of each node, and local translation contexts. A single translation file can be generated in order to describe a new “file format” and can be re-used. Any number of DML documents or streams (content) can then be exchanged and understood by a recipient with only a single translation document.

Associations in the translation document are specified by the following mechanisms:

- Node definition. Describes a single association between a DML identifier and a node’s properties, including its name.
- Container definition. Describes an association between a DML identifier and the container’s properties, including its name. Can also define a nested local translation that applies to nodes located inside the container.
- Include element. Merges the contents of another DML Translation document with this one at the present level. See further discussion below.

The associations described in the “built-in” DML translation structure *must not* be reused in either name or DML Identifier. Since the built-in DML translation utilizes the DML: prefix on every item, this is easy to avoid. This permits a simplified parser to recognize the DML identifiers in a “hard-coded” way.

ALLOWED TYPES

The allowed values for the type attribute with the base set are: “uint”, “string”, and “data”. The type attribute is not permitted on containers. Additional allowed types may be enabled by the <DML:Include-Primitives DML:Set=”…”/> directive. The “common” primitive set adds types “int”, “boolean”, “single”, “double”, and “datetime”. See the Primitive Sets section and the “Data Markup Language – Primitives” specification for more information.

INCLUDE-TRANSLATION DIRECTIVES

Another translation can be merged by a statement similar to the following, placed in the <DML:Header> or <DML:Translation> body:

```
<DML:Include-Translation DML:URI=”http://www.dmlexamples.org/example.dml” />
```

A translation document may be included inside of a container’s local translation context as well, and this is utilized in the base dml3 translation for the definition of the <DML:Header> container’s local translation. Logically, the referenced translation document body (contents of the <DML:Translation> container) can replace the <DML:Include-Translation> statement in-place.

TRANSLATION INCLUSIONS AND RENUMBERING

The <DML:Include-Translation DML:URI=”…” /> element in the translation document body provides a mechanism by which another DML translation can be incorporated into the existing one. This is useful for complicated translation definitions such as Video, which might need to be incorporated into another document type, such as a Slideshow. It is better to reference the large Video translation than to copy-and-paste it into the Slideshow translation document.

The <DML:Include-Translation DML:URI=”…” /> element may be placed below the global level. The included definitions then become part of the local translation, replacing the include directive in-place.

It is possible that included translations may conflict with each other. For example, perhaps separate Video and Image translation both re-use the DML ID 5 for their top-level containers. Both can still be included by renumbering one of them using a <Renumber> directive placed immediately following the <DML:Include-Translation> directive. Often only a single renumbering is necessary, since conflicting definitions are only likely at the highest conflicting level.

Primitive sets that are utilized by the translation being included are also merged into the outer translation document, but their codec and configuration can be overridden. The codec and configuration can be further overridden by the content document’s DML header, which has the highest precedence on codec selection as it can be processed last. Primitive sets are discussed next.

INCLUDE-PRIMITIVES DIRECTIVES

Primitive set inclusion directives within a translation body must appear before the type is used to define any nodes. A primitive-set can be referenced by a statement similar to the following:

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

This is sufficient to enable the use of the primitive set in node definitions, but many primitive-sets require specification of a codec before use in DML Content. This can be specified at the same time as the set, or it can be given by a later directive. The syntax is:

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

Furthermore, some codecs require configuration. This can also be given during the initial <DML:Include-Primitives /> directive, or it can be provided anytime later in the translation language. Codec configuration is discussed further below.

The DML:Set attribute provides a string that identifies predefined sets of primitives. The DML:Codec attribute can select a specific encoding/decoding mechanisms for the primitive set. The currently defined primitive sets are discussed in the "Data Markup Language – Primitives" specification. In addition to the extended primitive sets, a primitive base set is built in to any DML writer and reader. The base set is also described in "Data Markup Language – Primitives", but no <DML:Include-Primitives> statement is required to use the base set.

Each primitive-set included permits additional types to be utilized in the associations of nodes. Specifically, the type attribute supports an increased number of legal values. By means of an example, consider a sample XML-based translation document:

```
<DML:Translation DML:Version="2">
        <Container id="1" name="Bank">
            <Container id="1" name="Account">
                  <Node id="1" name="Account-Number" type="uint"/>
                  <Node id="2" name="Currency" type="string"/>
                  <Node id="3" name="Balance" type="uint"/>
            </Container>
        </Container>

</DML:Translation>
```

The above example defines a DML document type called "Bank". The top-level container (Bank) can contain any number of <Account> elements. Each account can have an account-number, currency, and balance. This example utilizes only the base set of primitives built-in to all DML parsers.

There is a significant problem with this translation configuration: the Balance is given as an unsigned integer. The bank will probably want to keep track of negative balances and change - the fractional part. A single-precision float would be one option, but suppose that the bank document needs to contain more precision – the rounding error associated with representing decimal numbers with base-2 floating-point may not be acceptable. A DML primitive set is defined for base-10 floating-point numbers. The primitive set is called the "decimal-float" set. A new example can be constructed with more precision.

```
<DML:Translation DML:Version=”2”>
        <DML:Include-Primitives DML:Set=”decimal-float” />

        <Container id=”1” name=”Bank”>
            <Container id=”1” name=”Account”>
                  <Node id=”1” name=”Account-Number” type=”uint”/>
                  <Node id=”2” name=”Currency” type=”string”/>
                  <Node id=”3” name=”Balance” type=”decimal-128”/>
            </Container>
        </Container>

</DML:Translation>
```

Two changes are highlighted in the above example – the include primitives directive and the change of type on the Balance attribute. A DML parser that does not support the “decimal-float” type can discover the “decimal-float” primitive-set is required and fail gracefully.

In the above example, the DML:Codec attribute was not attached to the DML:Include-Primitives directive. A decimal-float value can be encoded either in little or big-endian, and we have not specified which to use in this translation document. This is sufficient for a reader to know that a certain *type* is necessary to parse the document, but not sufficient for it to know *how* to decode it. An additional DML:Include-Primitives directive with DML:Primitives=”decimal-float” and specifying a DML:Codec must be present before the node occurs.

A content document can reference the above example with the following header:

```
<DML:Header>
      <DML:Include-Translation
            DML:URI=”http://dmlexamples.org/bank_example.xml” />
      <DML:Include-Primitives DML:Set=”decimal-float” DML:Codec=”be” />
</DML:Header>
```

As illustrated, the content document has final declaration on the choice of codec. Alternatively, the translation document *can* specify the codec, and the content document can always override this selection if needed. The recommended practice for most cases is that the translation document specify a codec, since this can provide a default and sometimes result in a more compact header in the content document.

Any number of <DML:Include-Primitives DML:Set=”…” /> statements may be included in the translation language of a document. Primitive-set inclusions are processed in the order they are encountered, with the content document’s DML:Header being processed last and having the highest precedence. Recommended usage in a translation document is to place primitive set include directives at the top-level, but a DML Parser *must* ignore the translation context when processing DML:Include-Primitives directives and process them in the order they are encountered. Types (besides the built-in base types) must be defined by a primitive-set inclusion before they can be used to describe nodes in the translation description, although the codec and configuration can be specified later.

The DML:Include-Primitives statement can also include a DML:CodecURI in order to guide the software to a location to potentially retrieve the codec from. A DML reader *must* fail gracefully when a request for an unsupported primitive-set/codec is encountered and the DML reader cannot or will not install the required codec.

Primitive configuration is accomplished by providing additional DML elements inside the DML:Include-Primitives container. The DML extension software must accept either DML or XML configuration information. If the configuration information is provided in a DML Header, then use of inline identification will be required as the header has not yet been processed and the DML IDs utilized would not be defined. If the configuration information is provided in a DML Translation document, the DML Translation document's header may define DML IDs to be employed. As an example:

```
<DML:Include-Primitives DML:Set="video" DML:Codec="example">
      <Video-Configuration
            CompressionScheme="imaginative"
            MagicNumber="1234" />
</DML:Include-Primitives>
```

When the above example is encountered, the DML reader should identify and activate the "example" codec software if it is installed and sufficient security or privileges are available. The DML reader must then provide access to the configuration information to the codec. If the above example were encoded directly in a DML:Header, then there would be way that the <Video-Configuration> container could have a DML ID defined in advance. Therefore, the <Video-Configuration> container, CompressionScheme attribute, and MagicNumber attribute would need to be specified using inline identification.

See the "Data Markup Language – Primitives" specification for more information.

TRANSLATION LANGUAGE DEFINITIONS

In addition to the structural DML nodes provided as part of DML v3, the following definitions are made for translation documents and DML headers. All DML IDs are specified in hexadecimal.

Name	DML ID	Type	Description
DML Translation Documents			
DML:Translation	[4][74]	Container	Top-level (body) container for a DML Translation document.
DML:URN	TSL2:[20]	String attr	Optional URN providing verification of DML Translation Document identity. Can be placed on the DML:Translation and on the DML:Include-Translation directive referencing the translation document. If present in both locations, the URN must match for the translation to be utilized.
DML Header and Translation Documents: Include-Translation Directives			
DML:Include-Translation	TSL2:[2]	Container	See "Data Markup Language – Primitives" and "Data Markup Language – Translation Document" specifications.
DML:URI	TSL2:[15]	String attr	URI providing location of the DML Translation Document to include.
DML Header and Translation Documents: Include-Primitives Directives			
DML:Include-Primitives	TSL2:[3]		
DML:Set	TSL2:[1F]	String attr.	Specifies an additional primitive set required for this document. See also "Data Markup Language – Primitives" specification.
DML:Codec	TSL2:[20]	String attr.	When used in combination with DML:Primitives attribute of DML:Include, specifies a codec for representing the additional primitives. See also "Data Markup Language – Primitives" specification.
DML:CodecURI	TSL2:[21]	String attr.	Optional. Can be used in combination with DML:Include of primitive codecs to identify a source for the specific codec software. The DML Reader *must not* acquire software from this URI without first gaining proper confirmation and authorization from the user. A DML Reader *may* choose to completely disregard this attribute.
DML Header and Translation Documents – Association Declarations			
Container	TSL2:[28]	Container	Identifies an association between a DML ID given by "id" attribute and a container type with name given by "name" attribute.

Name	DML ID	Type	Description
Node	TSL2:[29]	Container	Identifies an association between a DML ID given by “id” attribute and a primitive type with name given by “name” attribute and type given by “type” attribute. Can include an optional “usage” attribute.
id	TSL2:[2A]	UInt attr.	Specifies the DML ID in a Container or Node declaration.
name	TSL2:[2B]	String attr.	Specifies the name in a Container or Node declaration. Must be compliant with XML naming conventions.
type	TSL2:[2C]	String attr.	Specifies the primitive type in a Node declaration. Must be a value supported by the current set of primitives available at the time of the declaration.
usage	TSL2:[2D]	String attr.	Optional. Specifies the intended usage of the Node as either an “attribute”, “element”, or “any” (default).
Renumber	TSL2:[2E]	Container	Replaces a previous identification with a new one. Must contain an ‘id’ attribute and a ‘new-id’ attribute.
new-id	TSL2:[2F]	UInt attr.	DML Identifier to replace previous ‘id’ value with in a Renumber directive.
XMLRoot	TSL2:[32]	Container	Optional. Provides attributes that should be attached to the top-level body container during any XML conversion. See “Data Markup Language” specification for more information.

BUILT-IN TRANSLATION DATA

A built-in translation set is defined that applies whenever the TranslationURI matches the translation language URN “urn:dml:tsl2”. This built-in translation language set is summarized in XML as:

```
<DML:Translation DML:URN=”urn:dml:tsl2”>

       <Container id=”1140” name=”DML:Translation” />

       <Node id=”20” name=”DML:URN” type=”string” usage=”attribute”/>
       <Container id=”2” name=”DML:Include-Translation” />
       <Node id=”21” name=”DML:URI” type=”string” usage=”attribute”/>

       <Container id=”3” name=”DML:Include-Primitives” />
       <Node id=”31” name=”DML:Set” type=”string” usage=”attribute” />
       <Node id=”32” name=”DML:Codec” type=”string” usage=”attribute” />
       <Node id=”33” name=”DML:CodecURI” type=”string” usage=”attribute” />

       <Container id=”40” name=”Container” />
       <Container id=”41” name=”Node” />
       <Node id=”42” name=”name” type=”string” usage=”attribute” />
       <Node id=”43” name=”id” type=”uint” usage=”attribute” />
       <Node id=”44” name=”type” type=”string” usage=”attribute” />
       <Node id=”45” name=”usage” type=”string” usage=”attribute” />
       <Container id=”46” name=”Renumber” />
       <Node id=”47” name=”new-id” type=”uint” usage=”attribute” />

       <Container id=”50” name=”XMLRoot” />

</DML:Translation>
```

When defining new translations, it is recommended to avoid use of the DML ID range from 120 to 127 and from 1088 through 1200 in order to avoid conflicts with existing DML definitions. It is mandatory to avoid use of DML IDs found in the DML3 translation.