**Equivalent message serializations in NIEM**

NIEM supports multiple *message serializations*; at present, XML and JSON. NIEM defines a mapping from each supported serialization to RDF, and so every NIEM message can be converted to a RDF graph. When two messages map to the same graph, they are equivalent, representing the same information.

Question: Does every NIEM message have an equivalent in all supported serializations? Or is it possible to have (for instance) a NIEM JSON message that cannot be represented in NIEM XML? We compare example messages to find the answer.

A NIEM XML message always has a message element, as follows. (In all examples, some components are omitted for clarity.)

  <Message>

    <Foo>Hello

    <Foo>World

    <Bar>Goodbye

You can’t have a NIEM XML message without a single root element containing the message. There’s no such thing as a bare collection of elements in XML.

A message specification defines one or more *message formats*. A message format defines a class of messages, specifying both the required and optional content of an instance message belonging to that class, and also the meaning of that content. In order to define the message format for the above example, the message format would provide a model expressed in XML Schema, as shown below. In addition, a message format also has to declare the name of the message element (in this case, Message).

  <xs:complexType name=”MessageType”>

<xs:element ref=”Foo” maxOccurs="unbounded"/>

<xs:element ref="Bar"/>

</xs:complexType>

<xs:element name="Foo" type="niem-xs:string"/>

<xs:element name="Bar" type="niem-xs:string"/>

You could also express the model as a model instance, described below, and then the message format would declare the name of the message *property.* (*Element* is an XML concept; *property* is the technology-neutral term.)

Model

Property name="Foo", datatype="xs:string"

Property name="Bar", datatype="xs:string"

Property name="Message", class="MessageType"

Class name="MessageType"

HasProperty name="Foo" minOccurs=1 maxOccurs="unbounded" sequence=1

HasProperty name="Bar" minOccurs=1 maxOccurs=1 sequence=2

Datatype name="xs:string"

Now, in NIEM JSON, we don't have elements; we have objects containing key/value pairs, arrays, and scalars. JSON doesn't require anything equivalent to the message element in XML. You could put one into your message anyway, like this:

  {

    "@context": *something*,

    "Message": {

"Foo": [ "Hello", "World" ],

"Bar": "Goodbye"

And that's exactly what I did in the first version of the NIEM XML/JSON translator. But that's gross. Every JSON developer will hate it. The natural representation omits that unnecessary "Message" pair, like this:

  {

    "@context": *something*,

"Foo": [ "Hello", "World" ],

"Bar": "Goodbye"

}

In theory, the message format for this JSON message could use the XML Schema above. In practice, it will use the model instance. It won't declare a message *property*; instead, it will declare a message *class* (in this case, MessageType).

The RDF graph for the "natural" NIEM JSON message looks like this:

\_:msg rdf:type ns:MessageType ;

ns:Foo "Hello" ;

ns:Foo "World" ;

ns:Bar "Goodbye" .

I originally thought the RDF graph for the NIEM XML message would be:

\_:msg ns:Message \_:n1 .

\_:n1 rdf:type ns:MessageType ;

ns:Foo "Hello" ;

ns:Foo "World" ;

ns:Bar "Goodbye" .

The difference is the first tuple representing the message element. I thought this was required by the NDR. In which case, a NIEM XML message and a natural NIEM JSON message can never have the same RDF. There can't be equivalent messages. Blech.

A closer examination of NDR 5.0 shows that there is *no mapping* for the message element. None of the rules in section 5.6.3 apply. The NDR doesn't say anything about it. So there is no tuple for the message element. The RDF graphs are the same.

**Answer:** You can have equivalent NIEM XML and NIEM JSON messages. Hurray! When going from NIEM XML to NIEM JSON, we'll omit the message element. Going the other way, we have to put it back in. In that case, the message format may give us the name of the message element, or we may infer it from the message class.

**Conclusion:** I think the following bullets are true and/or advisable:

1. It's always possible to have equivalent NIEM JSON and NIEM XML messages
2. It's always possible to convert between those serializations with no loss of information
3. That will be true for any other supported serialization
4. A model instance is a NIEM message, so a model instance can be in any serialization
5. A message format can provide its model in XML Schema or as a model instance
6. A message format is defined in terms of a model and a *class* (not a *property*) in that model
7. A message format has a name, which is also the name of the XML message element
8. We will be able to generate artifacts for any serialization from a message format

**Postscript – on self describing data**

I have been a proponent of NIEM as self-describing data. A NIEM message contains identifiers for the resources that define the meaning of that message. Sometimes you do not need this property, but sometimes you really do.

The message element in NIEM XML is a convenient way to make self-describing data. I have long wanted a convention or rule to say that the URI of the message element declaration is the URI of its message format. Suppose you receive a NIEM XML message beginning with:

<ns:Message xmlns:ns="http://example.com/MyNS/">

According to the NDR, the URI of the message element declaration corresponding to that element is http://example.com/MyNS/#Message. We may interpret as the URI of the message format. If we have a copy of that message format, then we know exactly what this message means and can process it with confidence. If we do not have a copy of the message format, we may still be able to obtain it by resolving the namespace URI, or by looking for that URI in a registry. This is about as close to self-describing data as you can get.

There is nothing quite so convenient in NIEM JSON messages. The only way to make self-describing NIEM JSON data is to include the @context pair in the message. The value of the @context pair would be the identifier of the message format, as follows

    "@context": "http://example.com/MyNS/#Message",

"Foo": [ "Hello", "World" ],

"Bar": "Goodbye"

When a message specification defines more than one message format, this convention would require a separate JSON-LD context for each. That may be acceptable when we are generating the context from the model instance.