

Peer Review of Agile Delta Recommendations

(to the U.S. Message Text Format 2017-1 Configuration Control Board)

6/16/2017

Version: 1.0

Irina Connelly
Research Scientist

Laura Burkhart
Senior Research Scientist

Webb Roberts
Senior Research Scientist

Frank W. Klucznik
Senior Research Scientist
Georgia Tech Research Institute

Government SME under Contract No: N00189-16-D-Z039; Task Order N0018917F0254
(as a subcontractor to Booz Allen Hamilton)

ABSTRACT: This paper summarizes a peer review conducted of the Agile Delta evaluation and recommendations to the U.S. Message Text Format 2017-1 Configuration Control Board concerning the development of eXtensible Markup Language formatted messages conforming to the National Information Exchange Model.

Change History

Version	Date	Change Description
1.0	16 Jun 17	Initial Release

Table of Contents

Contents

<i>Executive Summary</i>	<i>1</i>
<i>1. Purpose</i>	<i>2</i>
<i>2. Scope</i>	<i>2</i>
<i>3. Introduction</i>	<i>2</i>
<i>4. Background</i>	<i>3</i>
<i>5. Benefits of NIEM</i>	<i>4</i>
<i>6. Agile Delta Evaluation</i>	<i>5</i>
<i>7. NIEM Use of XML Concepts</i>	<i>6</i>
<i>8. Discussion of Observations</i>	<i>7</i>
<i>9. Agile Delta Recommendations</i>	<i>14</i>
<i>Appendix A: Agile Delta Review & Analysis</i>	<i>18</i>

Executive Summary

This paper summarizes a peer review conducted of the Agile Delta evaluation and recommendations to the U.S. Message Text Format (USMTF) 2017-1 Configuration Control Board (CCB) concerning the development of eXtensible Markup Language (XML) formatted messages (aka, XML-MTF) conforming to the National Information Exchange Model (NIEM).

USMTF's success in transforming their existing XML to NIEM-conformant XML depends on their ability to balance the needs of their community, programs implementing the USMTF standard and the broader DoD enterprise. Achieving balance requires compromise, and in this context, compromise requires that all parties make concessions to benefit the DoD enterprise. Compromise involves change, and change involves shifting paradigms and learning new ways to accomplish tasks. USMTF's success will also provide benefits to other organizations and entities not currently involved in this transformation process (e.g., NIEM user community, other military and non-military standards, NATO and coalition groups).

From a NIEM perspective, USMTF's success involves making decisions that strike balance between domain model management (e.g., developing broadly reusable schema components), information exchange specification development (e.g., generating Information Exchange Package Documentation (IEPD)), and IEPD implementation (e.g., system interface developers). In this paper, Georgia Tech Research Institute (GTRI) provided inputs to the first two areas, and the Agile Delta provided inputs on the latter area in hopes that the combined input from both teams would provide the USMTF CCB with sufficient data points to make the best decisions for moving forward with aligning their XML representation with NIEM.

The most relevant sections of this document are:

- Section 5 provides context by describing the benefits NIEM offers to USMTF and broader DoD
- Section 6 provides an overview of the Agile Delta evaluation and observations
- Section 7 introduces relevant NIEM concepts to help readers fully understand the discussion
- Section 8 summarizes and discusses observations from the evaluation
- Section 9 provides a discussion and comment on the Agile Delta recommendations
- Appendix A contains the material presented to the 2017-1 USMTF CCB

For most reviewers, this document will not contain any surprises. There are areas of agreement, areas of disagreement, and areas requiring further discussion. The areas of disagreement in this paper are not negative. Rather they represent opportunities to share and learn from each other's experience and differing points of view, and then share that combined knowledge with the USMTF community to the benefit of the community and DoD enterprise.

1. Purpose

This paper documents the results of a peer review analysis conducted on the Agile Delta recommendations presented to the U.S. Message Test Format (MTF) 2017-1 Configuration Control Board (CCB) in March 2017, concerning the development of eXtensible Markup Language (XML) formatted messages (aka, XML-MTF) conforming to the National Information Exchange Model (NIEM). The intent of this review was to reproduce and confirm observations reported by Agile Delta, and to evaluate the Agile Delta recommendations from a Department of Defense (DoD) enterprise perspective.

In the context of this paper, applying an “enterprise perspective” involves development of an implementation and execution strategy that uses a holistic approach designed to support business, information, process and technology changes required across the DoD to better support warfighter needs in the future.

2. Scope

Georgia Tech Research Institute’s (GTRI’s) peer review was limited to the scope of known software technologies used in the Agile Delta review, which included:

- Java Architecture for XML Binding (JAXB)¹
- Eclipse²
- Oxygen XML Editor³
- Apache Xerces⁴

All of the material in this paper related to the NIEM can be associated with one of three independent functional user groups:

- Domain Model Developers
- Information Exchange Package Documentation (IEPD) Developers
- IEPD Implementers

GTRI developed the discussion and recommendations in this document from an “enterprise perspective” as defined in the Purpose section above.

3. Introduction

USMTF’s success in transforming their existing XML to NIEM-conformant XML depends on their ability to balance the needs of their community with programs implementing the USMTF standard and broader DoD enterprise requirements. Achieving this balance requires compromise, and compromise requires that all parties make concessions to benefit the greater good (e.g., DoD enterprise). Compromise also involves change, and change involves shifting paradigms and learning new ways to accomplish tasks. USMTF’s success will also provide benefits to other organizations and entities not currently involved in

¹ JAXB. https://en.wikipedia.org/wiki/Java_Architecture_for_XML_Binding.

² Eclipse. <https://eclipse.org>.

³ Oxygen XML Editor. <https://www.oxygenxml.com>.

⁴ Apache Xerces Project. <http://xerces.apache.org>.

this transformation process (e.g., NIEM user community, other military and non-military standards, NATO and coalition groups).

From a NIEM perspective, USMTF's success involves making decisions that strike a balance between domain model management (e.g., developing broadly reusable schema components), information exchange specification (e.g., generating Information Exchange Package Documentation (IEPD)), and IEPD implementation (e.g., development of system interfaces). In this paper, GTRI provides inputs on the first two, while Agile Delta provides inputs on the latter in hopes that the combined input from both teams would provide the USMTF CCB with sufficient data points to make the best decisions for moving forward.

GTRI was able to replicate and validate Agile Delta's observations. GTRI's peer review involved four tasks:

- TASK I: GTRI configured an environment comprising the software and technologies listed in Section 2, and replicated the Agile Delta observations using independently developed NIEM XML components not associated with the Agile Delta evaluation.
- TASK II: Agile Delta presented GTRI with a demonstration of their evaluation using WebEx⁵ technology, and answered questions about their observations and recommendations.
- TASK III: GTRI used the 2016 USMTF COMSPOT IEPD and associated software provided by Agile Delta to replicate and validate their observations a second time.
- TASK IV: GTRI generated this report.

4. Background

After analysis and evaluation, the USMTF CCB approved the decision to migrate XML-MTF to a NIEM-conformant format in 2016. This decision took into account an enterprise perspective and supports DoDI 8320.07 policy to consider using NIEM as an enabler for ensuring data, information, and IT services are visible, accessible, understandable, trustworthy, and interoperable. The current USMTF plans include complete transformation of existing XML-MTF to NIEM-conformant XML by December 2017, and then submission of reusable schema components to the NIEM Military Operations (MilOps) Domain.

U.S. Army CECOM awarded Agile Delta a contract to support U.S. Military Standards including USMTF (aka, MIL-STD-6040) in 2017. One of the first tasks assigned to Agile Delta involved reviewing and evaluating the impact of the USMTF NIEM Interface Change Proposals (ICPs). The NIEM ICP review involved a technical review designed to provide feedback to the USMTF CCB regarding the impact of implementing NIEM XML across the Army. The intent of this feedback was to influence the format of USMTF's new XML to reduce implementation costs and potential delays in the future. Appendix A contains the Agile Delta brief initially presented at the 2017-1 CCB.

Georgia Tech Research Institute (GTRI) was tasked to conduct a peer review of the Agile Delta's analysis at the request of the Joint Staff J6 Deputy Director for Cyber and C4 Integration, Data and Services

⁵ WebEx. <https://www.webex.com>.

Division (JS J6 DD C5I, DSD), under a subcontract to Booz Allen Hamilton. GTRI is the lead technical and subject matter expert (SME) for NIEM on behalf of the NIEM Program Management Office (PMO) and the DoD NIEM Military Operations (MilOps) Domain on behalf of the JS J6 DD C5I, DSD.

5. Benefits of NIEM

This section briefly highlights benefits NIEM offers to the USMTF and other DoD communities, as well as describes how these benefits can be achieved. In order to realize the full benefit of NIEM, the USMTF community will need to use different components of the NIEM technical architecture. The adoption of NIEM will:

1. Benefit USMTF community by leveraging the existing NIEM technical framework and supporting standards, distributed governance, standard Information Exchange Package Documentation (IEPD) development process, and compliance to DoDI 8320.07 (e.g., Implementing the Sharing of Data, Information, and Information Technology Services in the DoD).
2. Benefit other DoD communities by making USMTF semantic content available for reuse by organizations across the Department, and external communities engaged in the NIEM process through NIEM distributed governance, technical framework, tool support and process automation.

The first benefit involves developing a NIEM-conformant representation of USMTF semantic content, and leveraging that content to generate Information Exchange Package Documentation (IEPD) specifications for the set of military text messages. NIEM-conformant IEPDs may include:

- a subset from a NIEM-conformant reference schema (e.g., NIEM Core, NIEM Domains),
- extension schema (e.g., extensions of the reference schema, new content defined by the community, and/or other external components),
- external schema supporting as appropriate, and
- supporting artifacts such as mappings, business rules, and etc.

Maximizing the second benefit requires the USMTF community develop a reference schema that is incorporated into the latest NIEM release, and published to the NIEM Schema Subset Generation Tool (SSGT)⁶, a publicly available web application. In addition to the public web application, the SSGT capability is also offered to DoD in a Virtual Machine (VM) package with both public and non-public DoD content for use on DoD and DoD contractor networks as appropriate. Adding the USMTF semantic content to the VM SSGT will allow DoD IEPD developers and data modelers to have access to tools available in the public domain (e.g., NIEM.gov), and provide them with the ability to reuse data components from NIEM Core, all NIEM Domains, as well as USMTF public and non-public content when creating new IEPDs and updating existing IEPDs.

⁶ SSGT. <https://tools.niem.gov/niemtools/ssgt/index.iepd>.

6. Agile Delta Evaluation

Agile Delta's overall objective involved assessing the impacts of NIEM-conformant XML-MTF on IEPD implementers and users. The Agile Delta team achieved this objective by evaluating the impact of updating the 2016 USMTF COMSPOT message to a NIEM v3.0-conformant XML format. Their evaluation included the following test cases:

- Evaluate impact on schema-driven binding models (e.g. JAXB)
- Evaluate impact on Integrated Development Environments (IDEs), such as Eclipse
- Evaluate impact on XML editors, such as Oxygen XML Editor
- Evaluate impact on schema-driven validators (e.g. Apache Xerces)

The PowerPoint presentation in Appendix A contains summary observations from the evaluation. The table below provides a selected list of the observations excerpted from the presentation for discussion. The table also includes the XML concept/construct associated with each individual comment.

Analysis of the observations in Table 1 associates the observations with a small number of common XML concepts (e.g., global types, global elements and substitution groups). Discussing the observations requires a fundamental understanding of the corresponding XML concepts employed in NIEM. Therefore, the authors of this paper provided a brief overview of these and other relevant NIEM concepts in Section 7, followed by a discussion of each observation in Section 8.

#	Observations	Slide #	XML Concept
1	NIEM Schemas result in many classes at code generation	14	Global Types
2	Updating to NIEM COMSPOT schema result in errors in Java code written for COMSPOT 2016 Schemas	18	N/A
3	NIEM COMSPOT schema require different code for un-marshalling ⁷ .	20	Global Elements
4	Substitution groups do not map well onto the type systems of popular programming languages, and require extra code to determine the associated concrete schema element.	22	Substitution Groups
5	Substitution groups negatively impact code completion functionality of Integrated Development Environments (IDEs)	23, 25	Substitution Groups
6	Garden of Eden schema result in more complex code (getters and XML element names no longer match) ⁸	24	Substitution Groups
7	Oxygen sample instance generation needs configuration to identify the root element	37	Global Elements

⁷ In the context of this paper, "Marshalling" refers to the process of converting the data or the objects into XML, and "unmarshalling" is the reverse process of converting XML back to its original data or object format.

⁸ Java objects contain private variables. Public "getters" and "setters" are software methods to retrieve the stored variable value or update the stored variable value. For example, the Java object representing a car might contain variables such as mileage, gas level, make, model, year etc. The "year" getter will return the year of the vehicle; the "year" setter will update the year of the vehicle stored in the Java object.

8	Error messages in Oxygen point to abstract elements vs. concrete substitutions	40	Substitution Groups
9	“Quick fix” functionality in Oxygen offers to insert “abstract” elements vs. corresponding concrete elements	41	Substitution Groups
10	Schema validator can no-longer be used to detect valid/invalid messages (indicates any global element as a valid message)	48	Global Elements
11	Error messaging by the schema validator Apache Xerces points to the abstract element as the missing content vs. identifying concrete substitutions.	48	Substitution Groups

Table 1: Agile Delta Observations

7. NIEM Use of XML Concepts

Global Elements and Types

The NIEM Naming and Design Rules outline a key way that NIEM aims to enable interoperability:

“The W3C XML Schema standard enables information interoperability and sharing by providing a common language for describing data precisely. The constructs it defines are basic metadata building blocks — baseline data types and structural components. Developers employ these building blocks to describe their own domain-oriented data semantics and structures, as well as structures for specific information exchanges and components for reuse across multiple information exchanges. Rules that profile allowable XML Schema constructs and describe how to use them help ensure that those components are consistent and reusable.”⁹

In order to facilitate as much reuse as possible; NIEM takes the following approach to local and global components:

“Every schema component defined by a NIEM-conformant schema that can have a name has a name. This means that there are no anonymous types, elements, or other components defined by NIEM. Once an application has determined the name (i.e., namespace and local name) of an attribute or element used in NIEM-conformant instances, it will also know the type of that attribute or element.

There are no local attributes or elements defined by NIEM, only global attributes and elements. This maximizes the ability of application developers to extend, restrict, or otherwise derive definitions of local components from NIEM-conformant components. Using named global components in schema maximizes the capacity for reuse.”¹⁰

Substitution Groups

Use of *Substitution Groups* is the preferred method in NIEM reference schema for obtaining flexible content models, that is, different representations or refinements of a semantic concept. For example, a

⁹ <https://reference.niem.gov/niem/specification/naming-and-design-rules/3.0/niem-ndr-3.0.html>

¹⁰ <https://reference.niem.gov/niem/specification/naming-and-design-rules/3.0/niem-ndr-3.0.html>

175 date may be represented in a multitude of formats, such as dd mmm yy, mmm dd, yyyy, and yy-dd-
176 mmm.

177 A primary reason for using *Substitution Groups* in reference schema is that *Substitution Groups* are
178 extensible, and enable IEPD developers and domain modelers to add new semantic content without
179 modifying the reference schema.

180 **Constraint Schemas**

181 In certain situations, it may be advantageous to use different schema constructs than what NIEM-
182 conformant schema allow. For example, in certain use cases an implementer may need to replace
183 named types with anonymous types, replace substitution groups with choice groups, or turn global
184 elements into local elements. These changes in format and form are sometimes necessary to
185 accommodate issues that may arise with various software development tools.

186 In order to accommodate these needs, NIEM employs the *Constraint Schema* construct. NIEM Naming
187 and Design Rules do not bound *Constraint Schemas*, which enables them to leverage any schema
188 construct necessary to achieve their objective. The only usage caveat is that all XML instances must be
189 valid against both the original IEPD schema and the *Constraint Schema*.

190 **NDR Naming Rules**

191 To support its goal of enabling broad reuse of content, NIEM strives to ensure that names are consistent
192 and well understood. To that end, NIEM establishes its pattern for naming of components based on the
193 ISO 11179-5 standard. The pattern indicates that a component name should consist of the following:

- 194 • Object-class qualifier terms (0 or more)
- 195 • An object class term (1)
- 196 • Property qualifier terms (0 or more)
- 197 • A property term (1)
- 198 • Representation qualifier terms (0 or more)
- 199 • A representation term (0 or 1)

200 For each of the portions of the component names identified above, the NDR contains rules along with
201 supporting rationale and examples.

202 **8. Discussion of Observations**

203 This section contains a summary description and discussion of the Agile Delta observations in Table 1.

204 **Comment 1: NIEM Schemas result in many classes at code generation**

205 DESCRIPTION: The Agile Delta evaluation demonstrated the NIEM COMSPOT schema produced 167 Java
206 classes compared to the 40 Java classes generated using the 2016 COMSPOT schema. The increase in
207 the number of Java classes generated is due to the increase in the number of global types present in
208 NIEM-conformant schema.

209 DISCUSSION: *Global Types* are a fundamental XML construct used in NIEM. Refer to *Global Types* in the
210 previous section for the rationale of this NIEM design decision.

211 If the number of Java classes is an issue, one mitigating solution that an implementer can apply is the
212 use of a *Constraint Schema*. *Constraint Schemas* may alleviate this problem by converting a select
213 number of types from global/named types to anonymous types, which will reduce the number of classes
214 generated by the Java binding tool.

215 Implementers can also consider generating Java classes into separate packages based on namespaces or
216 using a custom binding file designed to organize specific schema into predefined packages. This
217 approach organizes the Java classes into logical blocks and thus reduces the number of classes a
218 software developer is working with in any one package.

219 Another best practice to reduce the burden of implementing large NIEM IEPDs involves pre-generating
220 software code designed to perform the reading and writing functionality. This code can be shared and
221 distributed among implementers, and contributes to more consistent implementations at reduced costs
222 and with fewer errors. This practice enables an IEPD implementer to develop and manage Application
223 programs Interfaces (API) for messages, and then share them with other implementers, rather than
224 leaving this work for multiple implementers to complete independently.

225 Although there are methods for mitigating and managing the number of Java classes, keep in mind that
226 the exact level of effort will vary widely, and is dependent on the size and complexity of the schema as
227 well as the size and complexity of the implementing system.

228 **Comment 2: Updating to NIEM COMSPOT schema result in errors in Java code written for COMSPOT** 229 **2016 Schemas**

230 DESCRIPTION: The Agile Delta evaluation indicated that replacing the 2016 COMSPOT Schemas with
231 NIEM Schemas results in many errors in the software code generated.

232 DISCUSSION: After reviewing and replicating the Agile Delta findings, the authors of this paper
233 unanimously agree these errors are indicative of work that is normally required of software developers
234 to update application code when migrating from one message schema format to another. This is not a
235 NIEM-specific issue.

236 Change to a message structure implies that changes are required to the software code that reads and
237 writes the XML. Furthermore, the exact level of effort required to make changes will vary widely and is
238 dependent on the size and complexity of the schema, the size and complexity of the implementing
239 system, and the programming methodology (e.g., use of schema language binding tools such as JAXB).

240 **Comment 3: NIEM COMSPOT schema require different code for proper un-marshalling.**

241 DESCRIPTION: The Agile Delta analysis noted that NIEM COMSPOT schema require different
242 unmarshalling (i.e., going from XML to Java) code to read the NIEM-conformant message, because the
243 schema contained multiple global elements.

DISCUSSION: Any schema set containing more than one global element will require different unmarshalling code compared to schema with a single global element. This issue arises because the software application cannot deterministically predict the root element that unmarshalling will produce. From the application's perspective, the root element could be any of the global elements in the schema. Therefore in this use case, JAXB will return a generic element, and it is up to the software developer to set its proper type via casting (i.e., change an entity from one data type to another).

NIEM IEPD developers define root elements in the Model Package Documentation (MPD) catalog file (i.e., mpd-catalog.xml), Schematron rules, and/or human readable documentation in the IEPD. IEPD implementers may define root elements in a myriad of ways in their software code (e.g., element name defined in the software application, function call).

Unmarshalling is a one-time event per message instance and does not require a significant amount of software development work.

Comment 4: Substitution groups do not map well onto constructs of popular programming languages, and require extra code to determine which concrete element was used.

DESCRIPTION: Agile Delta noted in their analysis that *Substitution Groups* do not map well onto the type systems of popular programming languages, and therefore software code to work with them is particularly cumbersome for software developers.

DISCUSSION: The authors of this paper disagree with this assertion. *Substitution Groups* as an XML construct are in many ways similar to an abstract class in Java. Both *Substitution Groups* and Java abstract classes are generic; software applications will not know what the object is until it is sufficiently interrogated. The identity of an object instantiated as a *Substitution Group* can be identified via reflection¹¹ using the common Java operator *instanceof*. Refer to the Figure 1 example using *instanceof* to verify that *TextType* was used for eye color.

```
if (JAXBIntrospector.getValue(attendee.getPersonEyeColor()) instanceof TextType) {  
    TextType eyeColor = (TextType) JAXBIntrospector.getValue(attendee.getPersonEyeColor());  
    System.out.println(eyeColor.getValue());  
}
```

Figure 1: *instanceof* example

Conversations addressing the use of *Substitution Groups* in NIEM frequently introduce *xs:choice* as an alternative construct. Interestingly enough, it is *xs:choice* that does not have a close equivalent in modern programming languages. When *xs:choice* is used application software routinely creates objects to hold all possible elements, and the developer must test which one was actually instantiated by performing null checks of all possible options. Furthermore, since objects present in a choice group are converted by JAXB to software code that is identical to code for objects present in a sequence, a software developer may not realize the objects are part of a choice, leading them to assign values to

¹¹ In computer science, reflection is the ability of a computer program to examine, introspect, and modify its own structure and behavior at runtime.

276 more than one object. This conversion can cause problems when the objects are marshalled (i.e.,
277 copied from Java to XML).

278 From a software developer's point of view, *Substitution Groups* generate more relevant documentation
279 in JAXB than *xs:choice*. Take, for example, the USMTF Alternate Relocation Set, which contains an
280 "Alternate Relocation Position" that may be expressed in any one of seven ways:

- 281 • MgrsUtm100Meter
- 282 • MgrsUtm10Meter
- 283 • MgrsUtm1Meter
- 284 • LatLongDegrees
- 285 • LatLongMinutes
- 286 • LatLongSeconds
- 287 • NicsAlternateRelocationCode

288 Implementing this semantic concept in XML using *xs:choice* generates the comments and method in
289 Figure 2. Beyond the snippet of code, items under this element appear just as any other properties
290 might. There are no specific details presented that would indicate to a developer whether this
291 component is *xs:choice*, *xs:sequence*, or *xs:all*, which becomes problematic given there are different
292 business rules for handling each. This JAXB-generated code requires the programmer to review the XML
293 schema to determine what structure was used.

```
/**
 * Gets the value of the alternateRelocationPosition property.
 *
 * @return
 *     possible object is
 *     {@link AlternateRelocationType.AlternateRelocationPosition }
 *
 */
public AlternateRelocationType.AlternateRelocationPosition getAlternateRelocationPosition() {
    return alternateRelocationPosition;
}
```

Figure 2: *xs:choice* generated comments & method

296 In contrast, a representation of the same semantic component using *Substitution Group* generates the
297 comments and method in JAXB-generated code depicted in Figure 3. Based on the pattern of the
298 comments software developers can easily identify this as a *Substitution Group* and list of potential
299 elements provided in the comments section.

```

/**
 * The position for the alternate relocation.
 *
 * @return
 *     possible object is
 *     {@link JAXBElement }{@code <}{@link LocationLatLongDegreesType }{@code >}
 *     {@link JAXBElement }{@code <}{@link GeographicLocationLatLongMinutesType }{@code >}
 *     {@link JAXBElement }{@code <}{@link NICSLocationCodeType }{@code >}
 *     {@link JAXBElement }{@code <}{@link GeographicLocationLatLongSecondsType }{@code >}
 *     {@link JAXBElement }{@code <}{@link LocationMGRSUTM100MeterType }{@code >}
 *     {@link JAXBElement }{@code <}{@link LocationMGRSUTM1000MeterType }{@code >}
 *     {@link JAXBElement }{@code <}{@link LocationMGRSUTM10MeterType }{@code >}
 *     {@link JAXBElement }{@code <}{@link MGRSUTM1MeterType }{@code >}
 *     {@link JAXBElement }{@code <}{@link Object }{@code >}
 *
 */
public JAXBElement<?> getAlternateRelocationPositionAlternative() {
    return alternateRelocationPositionAlternative;
}

```

Figure 3: *Substitution Group* example comment & method

Comparing the JAXB-generated comments in Figure 2 and 3 shows the information provided for *Substitution Groups* is more informative than the information provided for *xs:choice*. Also, the lack of information provided for *xs:choice* requires IEPD implementers to review the XML schema to determine the actual XML structure used, whereas IEPD implementers will easily recognize the JAXB comments that are associated with a *Substitution Group* construct.

Furthermore, the software code for handling choice using JAXB is no more efficient than the code needed to handle substitution groups, because *xs:choice* employs a similar programming “if-else” structure that examines which element out of the choice group was instantiated, and which elements were not, by performing null checks.

Additionally, in situations where Substitution Groups cause issues with software tools, implementers can also consider employing Constraint Schemas to turn substitution groups into choice groups.

Comment 5: Substitution groups affect code completion functionality of IDEs

DESCRIPTION: The Agile Delta noted during their analysis that the Eclipse IDE is unable to provide code completion functionality when working with abstract elements and substitution groups.

DISCUSSION: It is true that when substitution groups are used, the corresponding code generated by JAXB employs a generic construct (JAXBElement) to hold the concrete element substituted. The getter and setter methods for the *Substitution Group Head*¹² take and return values using JAXBElement. In order to determine the exact element substituted, an implementer can use one of the following:

- Using reflection and *instanceof* as previously discussed, or
- By examining the element name

Additionally, JAXB generates getter and setter methods with comments that list the possible valid concrete objects. Refer to the discussion and example in Comment 4. Once developers perform instance

¹² A Substitution Group Head is the element for which other elements are substitutable. In the *Comment 4* example, the USMTF field element “Alternate Relocation Position” is the substitution group head for the following alternative field elements: MgrsUtm100Meter, MgrsUtm10Meter, MgrsUtm1Meter, etc.

or name checking, and the JAXBElement is cast to the proper expected object, code completion functionality works as expected.

The NIEM NDR states a clear preference for using *Substitution Groups* for obtaining flexible content models in reference schema. However the NDR allows the use of *xs:sequence* (REF, EXT)¹³ and *xs:choice* (EXT). The NDR does not allow use of *xs:all* in reference or extension schema. NIEM reference schema are intended to provide authoritative definitions of broadly reusable schema components made available to IEPD developers via the SSGT and NIEM release packages, while extension schema are intended for reuse within a more narrow scope than those defined by a reference schema (e.g., IEPD).¹⁴

Comment 6: Getters and XML element names no longer match

DESCRIPTION: Agile Delta noted during their analysis that getter and setter methods created for substitution group heads do not match the names of concrete components that will be instantiated in their place.

DISCUSSION: It is true the methods associated with the *Substitution Group Heads* do not reflect the type names for the possible concrete components that may appear. An implementer can refer to the schema to identify the possible substitution values, or to the comments included by JAXB annotating the possible type values. This is a normal amount of work expected of software developers regardless of interface format.

Comment 7: Oxygen sample instance generation needs configuration to identify the root element

DESCRIPTION: Agile Delta experimented with the functionality that can generate sample XML documents using Oxygen XML Editor. They contrasted the behavior of this functionality for the NIEM COMSPOT Schemas compared to the 2016 COMSPOT Schemas. Their analysis indicated that unexpected results occurred for the NIEM version of the schema (e.g., wrong message element picked).

DISCUSSION: We agree that sample XML document generation is a useful utility provided in the Oxygen editor. This utility comes with a configuration menu that allows the user to select from a range of available options. For any schema or set of schema that includes more than one global element, selection of the root element is necessary. This behavior is expected and is not unique to NIEM schema. Furthermore, applying the right configuration choices to achieve the desired behavior for NIEM schema is similar to that of any other XML format.

This issue can be mitigated by either adding documentation to the IEPD defining the root element or it can be fully resolved selecting the desired configuration options within Oxygen. Some familiarity with the USMTF message is required.

Comment 8: Error messages in Oxygen point to abstract elements vs. concrete substitutions

¹³ For the remainder of this document “REF” = NIEM reference schema and “EXT” = NIEM extension schema, and are used to identify applicability as prescribed by the NIEM NDR.

¹⁴ <https://reference.niem.gov/niem/specification/naming-and-design-rules/3.0/niem-ndr-3.0.html>

DESCRIPTION: Agile Delta experimented with schema validation functionality of XML editors such as Oxygen. Their analysis uncovered that while Oxygen correctly identifies the location of the missing element within a message, the error message indicates that the abstract element (head of substitution group) is missing vs. the concrete substitutable elements. However, code completion at the location of the missing element does suggest proper concrete elements.

DISCUSSION: This feature occurs because of Apache Xerces and is not a NIEM-specific issue. The remedy involves navigating to the location of the error in the document and triggering code completion functionality, which produces the correct concrete elements in the suggestions. The remedy requires knowledge of working with XML schema, Oxygen and Xerces. Reference Figures 4 and 5 below.

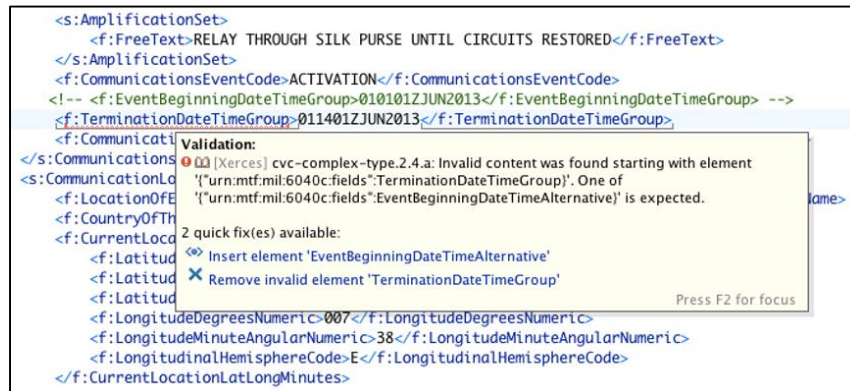


Figure 4: Xerces Code Completion

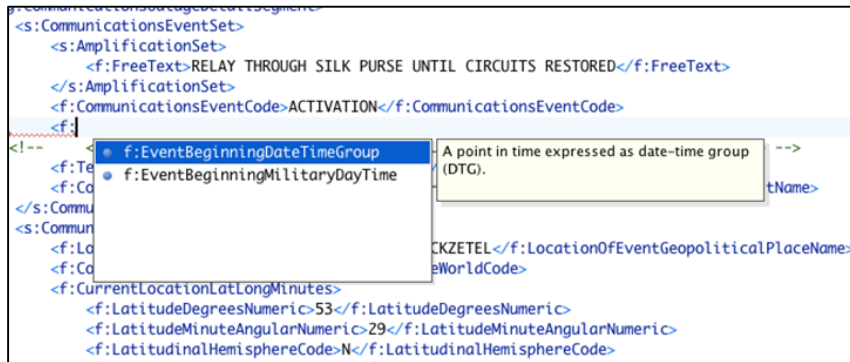


Figure 5: Oxygen Code Completion

Comment 9: “Quick fix” functionality in Oxygen offers to insert “abstract” elements vs. corresponding concrete elements

DESCRIPTION: Agile Delta experimented with the “Quick fix” functionality in the Oxygen XML Editor and its ability to repair errors due to missing substitutable elements. Their analysis identified that “Quick fix” is unable to repair the error since it suggests inserting the abstract substitution group head instead of the corresponding concrete elements. However, code completion at the location of the missing element does suggest proper concrete elements.

DISCUSSION: The remedy involves navigating to the location of the error in the document and triggering the code completion functionality, which produces a list of the correct concrete elements in the suggestion list. Refer to discussion in Comment 8 above for additional details.

Comment 10: Schema validator can no longer be used to detect valid/invalid messages (indicates any global element as a valid message)

DESCRIPTION: Agile Delta noted that when supplying an instance containing an element present in the NIEM COMSPOT schema, but not intended to be the root element of the message, Apache Xerces indicates that the instance is valid against the schema.

DISCUSSION: In the experience of the authors of this paper, the behavior described is expected for any schema that contains more than one global element. The NIEM NDR requires the global declaration of elements to support maximum reuse. A schema validator cannot differentiate the root element from any other globally-defined element. Consequently, knowledge of the IEPD is required, and as a best practice, the IEPD documentation identifies the allowable root element(s). This function is typically managed by other processes (e.g., application software, message pre-processor) and not in the schema or schema editors.

Comment 11: Error messaging by the schema validator (Apache Xerces) points to the abstract element as the missing content vs. identifying concrete substitutions.

DESCRIPTION: Agile Delta evaluated the error messages produced by the Apache Xerces schema validator. When the message is missing an element that was supposed to substitute for an abstract, the validator error message indicates that the abstract element is expected.

DISCUSSION: While it is correct to point out that inserting the abstract element in the location of the error will not resolve the error, an experienced software developer is able to determine a concrete substitution is needed at the location to resolve the error. Refer to previous discussions involving *Substitution Groups*.

Error messages generated by the Apache Xerces schema validator correctly identify the location of the error within the XML instance, and the reason for the error. However, Xerces does not go as far as suggesting an appropriate fix, which is a shortfall of the validator and not NIEM or XML structures.

9. Agile Delta Recommendations

In addition to observations documented during evaluation, Agile Delta provided a list of discrete recommendations. This section provides a discussion of those recommendations from a DoD enterprise perspective.

Recommendation 1: Minimize schema changes due to acronyms (slide 54)

INTERPRETATION: Slide 54 on page 31 recommends adopting ID and URI acronyms to minimize the amount of schema changes caused by spelling out of acronyms.

410 DISCUSSION: NIEM supports the use of local terminology, and provided the acronyms used by
411 USMTF do not conflict with any currently documented acronyms within the MilOps Domain
412 model, we concur with this recommendation.

413 **Recommendation 2: Reduce USMTF local terminology (slide 55)**

414 INTERPRETATION: Slide 55 on page 30 focuses on component naming rules and recommends
415 not using USMTF-specific terms when naming data components to ensure maximum reusability
416 across the DoD. For example, consider not terminating USMTF names with “Set”, “Alternative”,
417 “Segment”, etc.

418 DISCUSSION: We concur with this recommendation.

419 **Recommendation 3: Apply NIEM naming changes incrementally (slide 56)**

420 INTERPRETATION: Slide 56 on page 31 recommends renaming components to conform to the
421 NIEM NDR naming rules on an incremental message-by-message basis rather than making the
422 required changes at one time for all messages.

423 DISCUSSION: We strongly disagree with this recommendation because it does not allow the
424 DoD nor the USMTF community to realize the full benefits of NIEM. Refer to Section 3 of this
425 document. Incrementally updating USMTF on a message-by-message basis could extend the
426 work more than a decade, and incur additional costs of managing and maintaining two sets of
427 schema generation tools. In addition, the full scope of semantic content within the USMTF
428 message set would not be available for broader DoD and international use.

429 We recommend executing the current activity plan to convert all USMTF messages to NIEM-
430 conformant XML by the end of this calendar year. Once the conversion is complete, we
431 recommend developing and executing an adoption plan that allows programs to implement
432 messages over time through budgeted technology refreshes to minimize the financial impact.

433 **Recommendation 4: Minimize usage of substitution groups (slide 57-58)**

434 INTERPRETATION: Slides 57-58 on page 32 recommend minimizing use of *substitution groups*
435 within USMTF schema for the reasons noted in the observations section above.

436 DISCUSSION: We acknowledge that *substitution groups*, like all similar XML constructs (*e.g.*,
437 *xs:sequence*, *xs:choice*) have benefits and drawbacks. We recommend applying *substitution*
438 *groups* when the benefits outweigh the drawbacks. The design team for the USMTF schema will
439 need to weigh their options for each situation to determine whether a *substitution group* is the
440 appropriate solution or not. In situations where a *substitution group* is the appropriate modeling
441 option, but causes some undesirable behavior by the software development tools, we
442 recommend IEPD Implementers consider employing a *constraint schema* as an additional
443 mitigation technique.

444

Recommendation 5: Reduce reference schema (slide 59)

INTERPRETATION: Slide 59 on page 32 appears to recommend not using NIEM reference schema because of the issues encountered with the use of *substitution groups* during the Agile Delta evaluation. A series of follow on discussions and email threads exploring the use of NIEM extension schema reinforces this interpretation.

DISCUSSION: We disagree with this recommendation based on the stated argument that *substitution groups* are a problem. There are several reasons for our position on this recommendation, which include:

- Discussions and facts brought up in this paper regarding the preferred use of *Substitution Groups* over other options (e.g., *xs:choice*).
- The decision to use extension vs. reference schema is complex and multidimensional. It involves careful consideration of USMTF goals/objectives, benefits to the DoD enterprise, governance and configuration management, and funding of SSGT updates, among other considerations.

The USMTF CCB made the decision to use reference schema for all but message schema at the 2016-3 CCB.

Recommendation 6: Raise issues about the Garden of Eden pattern with NIEM (slide 60)

INTERPRETATION: Slide 60 on page 32 asserts the Garden of Eden pattern makes coding of web services more time-consuming, complex and error-prone, because of the increased amount of global components it includes. This also contributes to a reduced ability of schema validators to be used to determine message validity, again due to the fact that more than one global element is present in schema following this pattern. They recommend discussing this issue with the NIEM PMO.

DISCUSSION: Supporting the interoperability requirements of the DoD enterprise requires striking a balance between flexibility and agreement. Achieving this balance requires compromise from all stakeholders (e.g., domain modelers, IEPD developers and Implementers).

Globally defined elements is a cornerstone of NIEM's technical architecture that enables it to provide broadly reusable schema components to all who use NIEM. The DoD has evaluated, tested and demonstrated this technical architecture from high bandwidth headquarters level environments to denied, disconnected, intermittent, and low-bandwidth environments over the past 8 years, and found it more than acceptable. Furthermore, this paper discusses the management and resolution of issues associated with multiple global elements in schema.

As a best practice, we recommend the USMTF community include documentation in their message specifications that explicitly identify the root elements for each IEPD in MPD catalog, Schematron rules, and/or human-readable documentation.

481 **Recommendation 7: Engage with NIEM to work through identified issues (slide 61)**

482 INTERPRETATION: Slide 61 on page 33 proposes the USMTF community engage with the NIEM
483 PMO in order to:

- 484 • Identify unintended consequences and effects on program and user adoption
- 485 • Work with NIEM to make it more widely adopted and attractive to programs.
- 486 • Work on less disruptive transition strategies for large, existing programs

487 DISCUSSION: We concur. In response to our engagements with Agile Delta associated with this
488 review, GTRI submitted a change request proposal to relax the requirements for representation
489 terms in the NIEM v4.0 NDR to reduce the level of effort needed to implement USMTF IEPDs
490 across DoD. This change request was presented and discussed at the 24 Jun 2017 NIEM
491 Technical Architecture Committee (NTAC) and 25 Jun 2017 NIEM Business Architecture
492 Committee (NBAC) meetings, and was approved at the 07 Jun 2017, NTAC meeting for inclusion
493 in the NIEM v4.0 NDR.

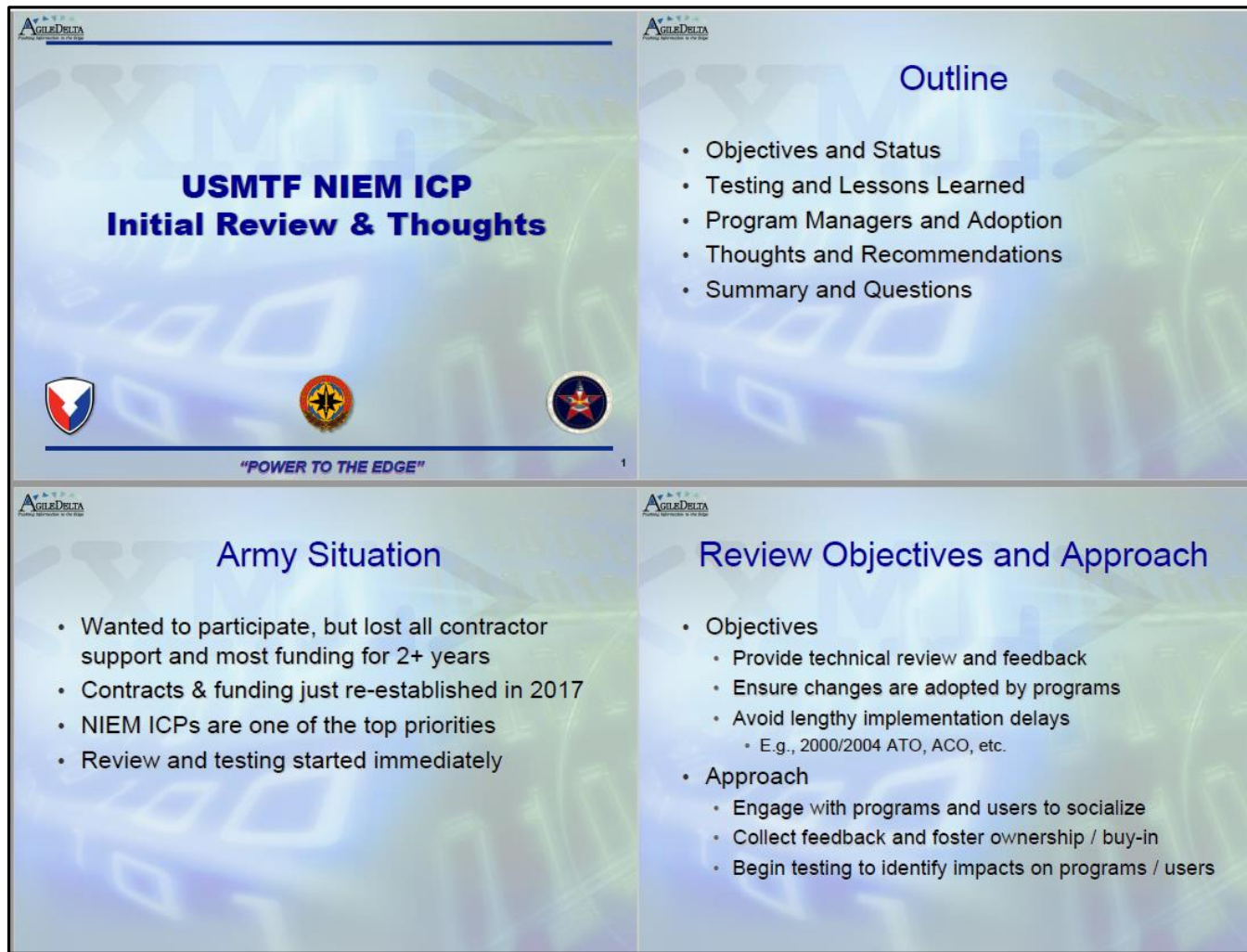
494 **Recommendation 8: Test everything (slide 62)**

495 INTERPRETATION: Slide 62 on page 33 recommends testing products before the standards are
496 finalized to reduce unanticipated consequences and provide a means to solicit feedback from
497 programs.

498 DISCUSSION: We agree that testing is important, and one of many methods to evaluate and
499 solicit feedback from programs. Having said that, we are also keenly aware that testing, like
500 programs, is restricted by time, budget and resourcing. It is impossible to “test everything” as
501 the slide infers. Testing requires a balanced approach that facilitates feedback, reduces risk, and
502 enables standards to advance, mature and keep pace with technological improvements to the
503 benefit of our ultimate customer: the warfighter.

504 **Appendix A: Agile Delta Review & Analysis**

505 This section of the document contains the complete 64 slide PowerPoint brief presented to the USMTF 2017-1 CCB 14-16 March 2017. This
506 material represents a summary of the Agile Delta's initial review and analysis of the NIEM ICPs. Although not all of the slides are numbered, they
507 are sequenced and should be read from left-to-right, top-to-bottom.



508

Current Status

- Completed initial ICP reviews
- Completed initial impact testing
- Started engaging with Army programs and users

ICP Review

- An amazing amount of work completed
 - Over 3000 changes impacting most messages & sets
 - Field changes are even more significant
- Addressing most NDR requirements
 - Garden of Eden schema design pattern
 - Replacing xs:choice with substitution groups
 - Acronym usage and documentation
 - NIEM representation terms (e.g., Code, Numeric, Text, Quantity, ID, etc.)

"Test everything; retain what is good."

Early Developer Guidance
1 Thessalonians 5:21

Testing and Lessons Learned

*"The intended consequences sometimes happen.
The unintended consequences always happen."*

Richard Rollman
VP Engineering, AgileDelta
(AKA Pessimist Prime)

"POWER TO THE EDGE"

7

W3C Approach

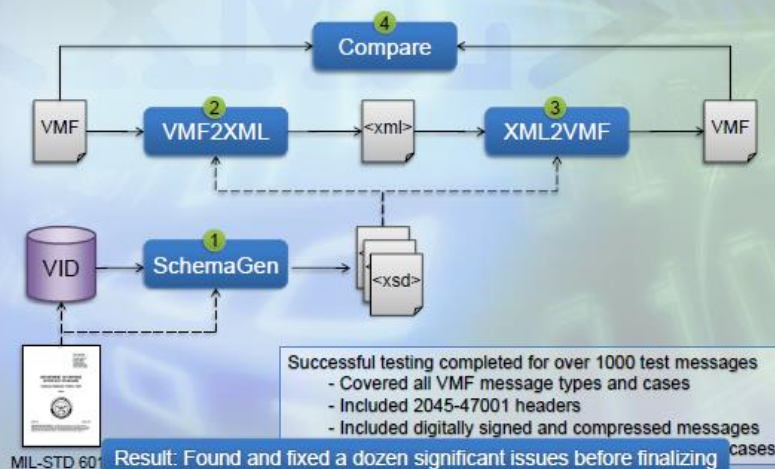


"Implementation experience is required to show that a specification is sufficiently clear, complete, and relevant to market needs, to ensure that independent interoperable implementations of each feature of the specification will be realized."

- For EX1, 3000+ test cases covering every aspect of spec
- 3 independent implementations
- Found several issues, even after review by world's top experts

I.e., all standards should be thoroughly tested before they are final

VML (XML-VMF) Approach



Schema-Driven Binding Models

- Binding Models
 - The primary method web-service platforms provide for sending, receiving and processing XML
 - Used by most net-centric systems (e.g., DCGS, AOC-WS, DDS, CANES ACS, GCCS, etc.)
- How they work
 - Developer provides schemas to a "binding compiler"
 - Binding compiler generates code for creating and processing all elements and types in the schema
 - Developer uses generated code to send / receive XML
- E.g., JAXB (Java API for XML Binding)
 - The Java standard for XML binding used by all Java application servers and web-service clients.

Initial USMTF NIEM Testing

- Primary Objective
 - Assess impacts on implementers and users
- Tests completed
 - Schema-driven binding models (i.e., JAXB)
 - Schema-driven editors and data mappers
 - Schema-driven validators

Binding Model Test

- Purpose: evaluate what a typical web-service developer must do to implement the NIEM ICPs
- Process
 - Create a simple application that uses JAXB to read and display data from a 2016 COMSPOT message
 - Update the app to use the NIEM COMSPOT message
 - Characterize level-of-effort and any notable issues

Binding Model Test

- Generate JAXB classes from 2016 COMSPOT schemas
- Review Eclipse IDE developer assistance (from schemas)
- Assess impact of typical change (i.e., add a field)
- Assess impact of updating to NIEM COMSPOT schemas

> Demo

JAXB Class Generation

2016 schemas generated 40 Java classes

NIEM schemas generated 167 Java classes due to increased global elements, augmentation points, substitution groups, etc.

JAXB Code using 2016 Schemas

```
// read the COMSPOT message into a comspot Java objects
File file = new File(args[0]);
JAXBContext jaxbContext = JAXBContext.newInstance(CommunicationsSpotReport.class);
Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
CommunicationsSpotReport comspot = (CommunicationsSpotReport) jaxbUnmarshaller.unmarshal(file);

System.out.println("Read the following COMSPOT message into Java objects.");
System.out.println();

// ExerciseIdentification
ExerciseIdentification exerciseId = comspot.getExerciseIdentification();
System.out.println("Exercise Identification:");
System.out.println("  Name: " + exerciseId.getExerciseName().getValue());
System.out.println("  Additional Name: " + exerciseId.getExerciseAdditionalIdentifier().getValue());
System.out.println();

// MessageIdentifier
MessageIdentifier msgId = comspot.getMessageIdentifier();
System.out.println("Message Identifier:");
System.out.println("  MTF Format: " + msgId.getMessageTextFormatIdentifier().getValue());
System.out.println("  MTF Standard: " + msgId.getStandard().getValue());
System.out.println("  MTF Version: " + msgId.getVersion().getValue());
System.out.println("  Originator: " + msgId.getOriginator().getValue());
System.out.println("  Serial #: " + msgId.getSerialNumber().getValue());
System.out.println("  Qualifier: " + msgId.getQualifier().getValue());
System.out.println("  Security Policy: " + msgId.getMessageSecurityPolicy().getValue());
System.out.println("  Classification: " + msgId.getMessageSecurityClassification().getValue());

System.out.println();
// show details from each output
System.out.println("Communication Output Details:");
List<CommunicationOutputDetails> outputDetails = comspot.getCommunicationOutputDetails();
for (int i = 0; i < outputDetails.size(); i++) {
    CommunicationOutputDetails output = outputDetails.get(i);
    System.out.println("  Output " + (i+1));
}
```

Note: Venetian Blind schemas result in very simple code where all types and getter functions match XML element names

IDE Help with 2016 Schemas

Typing partial name shows available matches

IDE provides code completion and help finding all desired items in message. Makes it really easy to write code without combing through schemas for names.

Impact of Typical Schema Change

```
// read the COMSPOT message into a comspot Java objects
File file = new File("src/test/resources/comspot.xml");
JAXBContext jaxbContext = JAXBContext.newInstance(CommunicationsSpotReportType.class);
Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
CommunicationsSpotReportType comspot = (CommunicationsSpotReportType) jaxbUnmarshaller.unmarshal(file);

System.out.println("Read the following COMSPOT message into Java objects.");
System.out.println();

ExerciseIdentification exerciseId = comspot.getExerciseIdentification();
System.out.println("Exercise Identification:");
System.out.println("  Nickname: " + exerciseId.getExerciseNickname().getValue());
System.out.println("  Additional Nickname: " + exerciseId.getExerciseAdditionalIdentifier().getExerciseAdditionalNickname().getValue());
System.out.println();

MessageIdentifier msgId = comspot.getMessageIdentifier();
System.out.println("Message Identification:");
System.out.println("  MTF Format: " + msgId.getMessageTextFormatIdentifier().getValue());
System.out.println("  MTF Standard: " + msgId.getMessageTextFormatIdentifier().getValue());
System.out.println("  MTF Version: " + msgId.getMessageTextFormatIdentifier().getValue());
System.out.println("  Originator: " + msgId.getOriginator().getValue());
System.out.println("  Serial #: " + msgId.getMessageSerialMarker().getValue());
System.out.println("  Qualifier: " + msgId.getQualifier().getValue());
System.out.println("  Security Policy: " + msgId.getMessageSecurityPolicy().getValue());
System.out.println("  Classification: " + msgId.getMessageSecurityClassification().getValue());

// show details from each outage
System.out.println("Communication Outage Details:");
List<CommunicationOutageDetails> outageDetails = comspot.getCommunicationOutageDetails();
for (int i = 0; i < outageDetails.size(); i++) {
    CommunicationOutageDetails outage = outageDetails.get(i);
    System.out.println("  Outage # " + (i+1));
}
```

Adding elements to schema generates no errors and requires no changes to code. Developer can update code to use newly added elements when and if needed.

Impact of NIEM Schema Change

```
// read the COMSPOT message into a comspot Java objects
File file = new File("src/test/resources/comspot.xml");
JAXBContext jaxbContext = JAXBContext.newInstance(CommunicationsSpotReportType.class);
Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
CommunicationsSpotReportType comspot = (CommunicationsSpotReportType) jaxbUnmarshaller.unmarshal(file);

System.out.println("Read the following COMSPOT message into Java objects.");
System.out.println();

ExerciseIdentification exerciseId = comspot.getExerciseIdentification();
System.out.println("Exercise Identification:");
System.out.println("  Nickname: " + exerciseId.getExerciseNickname().getValue());
System.out.println("  Additional Nickname: " + exerciseId.getExerciseAdditionalIdentifier().getExerciseAdditionalNickname().getValue());
System.out.println();

MessageIdentifier msgId = comspot.getMessageIdentifier();
System.out.println("Message Identification:");
System.out.println("  MTF Format: " + msgId.getMessageTextFormatIdentifier().getValue());
System.out.println("  MTF Standard: " + msgId.getMessageTextFormatIdentifier().getValue());
System.out.println("  MTF Version: " + msgId.getMessageTextFormatIdentifier().getValue());
System.out.println("  Originator: " + msgId.getOriginator().getValue());
System.out.println("  Serial #: " + msgId.getMessageSerialMarker().getValue());
System.out.println("  Qualifier: " + msgId.getQualifier().getValue());
System.out.println("  Security Policy: " + msgId.getMessageSecurityPolicy().getValue());
System.out.println("  Classification: " + msgId.getMessageSecurityClassification().getValue());

// show details from each outage
System.out.println("Communication Outage Details:");
List<CommunicationOutageDetails> outageDetails = comspot.getCommunicationOutageDetails();
for (int i = 0; i < outageDetails.size(); i++) {
    CommunicationOutageDetails outage = outageDetails.get(i);
    System.out.println("  Outage # " + (i+1));
}
```

Updating to NIEM schemas results in many errors. All the existing types and getters for messages, segments and sets no longer work due to name changes.

Impact of NIEM Schema Change

```
// read the COMSPOT message into a comspot Java objects
File file = new File("src/test/resources/comspot.xml");
JAXBContext jaxbContext = JAXBContext.newInstance(CommunicationsSpotReportType.class);
Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
CommunicationsSpotReportType comspot = (CommunicationsSpotReportType) jaxbUnmarshaller.unmarshal(file);

System.out.println("Read the following COMSPOT message into Java objects.");
System.out.println();

ExerciseIdentification exerciseId = comspot.getExerciseIdentification();
System.out.println("Exercise Identification:");
System.out.println("  Nickname: " + exerciseId.getExerciseNickname().getValue());
System.out.println("  Additional Nickname: " + exerciseId.getExerciseAdditionalIdentifier().getExerciseAdditionalNickname().getValue());
System.out.println();

MessageIdentifier msgId = comspot.getMessageIdentifier();
System.out.println("Message Identification:");
System.out.println("  MTF Format: " + msgId.getMessageTextFormatIdentifier().getValue());
System.out.println("  MTF Standard: " + msgId.getMessageTextFormatIdentifier().getValue());
System.out.println("  MTF Version: " + msgId.getMessageTextFormatIdentifier().getValue());
System.out.println("  Originator: " + msgId.getOriginator().getValue());
System.out.println("  Serial #: " + msgId.getMessageSerialMarker().getValue());
System.out.println("  Qualifier: " + msgId.getQualifier().getValue());
System.out.println("  Security Policy: " + msgId.getMessageSecurityPolicy().getValue());
System.out.println("  Classification: " + msgId.getMessageSecurityClassification().getValue());

// show details from each outage
System.out.println("Communication Outage Details:");
List<CommunicationOutageDetails> outageDetails = comspot.getCommunicationOutageDetails();
for (int i = 0; i < outageDetails.size(); i++) {
    CommunicationOutageDetails outage = outageDetails.get(i);
    System.out.println("  Outage # " + (i+1));
}
```

Fixing all these errors reveals 2 times more errors than we started with because the names and types for all fields, composites and alternates have also changed.

Impact of NIEM Schema Change

```
// read the COMSPOT message into a comspot Java objects
File file = new File("src/test/resources/comspot.xml");
JAXBContext jaxbContext = JAXBContext.newInstance(CommunicationsSpotReportType.class);
Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
CommunicationsSpotReportType comspot = (CommunicationsSpotReportType) jaxbUnmarshaller.unmarshal(file);

System.out.println("Read the following COMSPOT message into Java objects.");
System.out.println();

ExerciseIdentification exerciseId = comspot.getExerciseIdentification();
System.out.println("Exercise Identification:");
System.out.println("  Nickname: " + exerciseId.getExerciseNickname().getValue());
System.out.println("  Additional Nickname: " + exerciseId.getExerciseAdditionalIdentifier().getExerciseAdditionalNickname().getValue());
System.out.println();

MessageIdentifier msgId = comspot.getMessageIdentifier();
System.out.println("Message Identification:");
System.out.println("  MTF Format: " + msgId.getMessageTextFormatIdentifier().getValue());
System.out.println("  MTF Standard: " + msgId.getMessageTextFormatIdentifier().getValue());
System.out.println("  MTF Version: " + msgId.getMessageTextFormatIdentifier().getValue());
System.out.println("  Originator: " + msgId.getOriginator().getValue());
System.out.println("  Serial #: " + msgId.getMessageSerialMarker().getValue());
System.out.println("  Qualifier: " + msgId.getQualifier().getValue());
System.out.println("  Security Policy: " + msgId.getMessageSecurityPolicy().getValue());
System.out.println("  Classification: " + msgId.getMessageSecurityClassification().getValue());

// show details from each outage
System.out.println("Communication Outage Details:");
List<CommunicationOutageDetails> outageDetails = comspot.getCommunicationOutageDetails();
for (int i = 0; i < outageDetails.size(); i++) {
    CommunicationOutageDetails outage = outageDetails.get(i);
    System.out.println("  Outage # " + (i+1));
}
```

After fixing all the known errors, the program crashes due to the line above. Unlike Venetian Blind, the GoE schemas do not define a single top-level XML message. So, the code cannot assume JAXB will return a COMSPOT message.

Impact of NIEM Schema Change

```

ObjectFactory | *TestCOMPOT ja 12 | *TestCOMPOTNIEM | CommunicationsB | MessageBaseType
File file = new File(args[0]);
JAXBContext jaxbContext = JAXBContext.newInstance(CommunicationsSpotReportType.class);
Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
JAXBElement<?> rootElement = (JAXBElement<?>) jaxbUnmarshaller.unmarshal(file);
CommunicationsSpotReportType compot = null;
if (rootElement.getName().getLocalPart().equals("CommunicationsSpotReport")) {
    compot = (CommunicationsSpotReportType) rootElement.getValue();
}

System.out.println("Read the following COMPOT message into Java objects:");
System.out.println();

ExerciseIdentification exerciseId = compot.getExerciseIdentification();
System.out.println("Exercise Identification:");
System.out.println("  Nickname: " + exerciseId.getExerciseNickname().getValue());
System.out.println("  Additional Nickname: " + exerciseId.getExerciseAdditionalIdentifier().getExerciseAdditionalNickname().getValue());
System.out.println();

MsgDataSet msgId = compot.getMsgDataSet();
System.out.println("Message Identification:");
System.out.println("  MTF Format: " + msgId.getMessageTextFormat().getValue());
System.out.println("  MTF Standard: " + msgId.getMessageTextFormat().getStandard().getValue());
System.out.println("  MTF Version: " + msgId.getMessageTextFormat().getVersion().getValue());
System.out.println("  Originator: " + msgId.getOriginator().getValue());
System.out.println("  Serial #: " + msgId.getMessageSerialNumber().getValue());
System.out.println("  Qualifier: " + msgId.getQualifier().getValue());
System.out.println("  Security Policy: " + msgId.getMessageSecurityPolicy().getValue());
System.out.println("  Classification: " + msgId.getMessageSecurityClassification().getValue());
System.out.println();

// show details from each outgoing

```

Since the root element is unknown with GoE, JAXB returns a generic element that must be inspected before it can be converted to a useful COMPOT object. This requires 5 lines of code vs. the 1 line required with Venetian Blind schemas.

Impact of NIEM Schema Change

```

ObjectFactory | *TestCOMPOT ja 12 | *TestCOMPOTNIEM | CommunicationsB | MessageBaseType
someElement = event.getCommunicationsEventAlternative();
if (someElement.getName().getLocalPart().equals("CommunicationsEventCode")) {
    CommunicationsEventCodeType eventCode = (CommunicationsEventCodeType) someElement.getValue();
    System.out.println("  Event: " + eventCode.getValue());
}

someElement = event.getEventBeginningDateTimeAlternative();
if (someElement.getName().getLocalPart().equals("EventBeginningDateTimeGroup")) {
    DTGType dtg = (DTGType) someElement.getValue();
    System.out.println("    begin: " + dtg.getValue());
}

someElement = event.getTerminationDateTimeAlternative();
if (someElement.getName().getLocalPart().equals("TerminationDateTimeGroup")) {
    DTGType dtg = (DTGType) someElement.getValue();
    System.out.println("    end: " + dtg.getValue());
}

List<JAXBElement> elementsList = event.getCommunicationsDesignatorAlternative();
if (elementsList.get(0).getName().getLocalPart().equals("CommunicationsDesignatorCircuitName")) {
    CommunicationsCircuitNameType name = (CommunicationsCircuitNameType) elementsList.get(0).getValue();
    System.out.println("    Circuit: " + name.getValue());
}

System.out.println("  Note: " + event.getAmplificationSet().getAmplificationSet().getValue());
System.out.println();

System.out.println("  Location: " + location);
CommunicationsLocationType location = (CommunicationsLocationType) location;
someElement = location.getLocationEventBaseNameAlternative();
if (someElement.getName().getLocalPart().equals("LocationEventBaseNameAlternative")) {

```

The same thing occurs for every substitution group. Substitution groups do not map well onto the type systems of popular programming languages. As such, the programmer has to write more code to determine which element was returned.

Impact of NIEM Schema Change

```

ObjectFactory | *TestCOMPOT ja 12 | *TestCOMPOTNIEM | CommunicationsB | MessageBaseType
// read the COMPOT message into a compot Java object
File file = new File(args[0]);
JAXBContext jaxbContext = JAXBContext.newInstance(CommunicationsSpotReportType.class);
Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
JAXBElement<?> rootElement = (JAXBElement<?>) jaxbUnmarshaller.unmarshal(file);
CommunicationsSpotReportType compot = null;
if (rootElement.getName().getLocalPart().equals("CommunicationsSpotReport")) {
    compot = (CommunicationsSpotReportType) rootElement.getValue();
}

compot.getExerciseIdOperationAlternative().get()
System.out.println();

JAXBElement<?> exerciseId = compot.getExerciseIdOperationAlternative();
ExerciseIdType exerciseId = null;
if (exerciseId.getName().getLocalPart().equals("ExerciseIdSet")) {
    exerciseId = (ExerciseIdSetType) exerciseId.getValue();
}

System.out.println("Exercise Identification:");
System.out.println("  Nickname: " + exerciseId.getExerciseNickname().getValue());
System.out.println();

JAXBElement<?> exerciseIdID = compot.getExerciseIdOperationAlternative();
exerciseIdID.getExerciseIdAdditionalIdentifier();
if (exerciseIdID.getName().getLocalPart().equals("ExerciseAdditionalNicknameText")) {
    exerciseIdAdditionalNicknameType nickname = (ExerciseAdditionalNicknameType) exerciseIdID.getValue();
    System.out.println("  Additional Nickname: " + nickname.getValue());
}
System.out.println();

MsgDataSet msgId = compot.getMsgDataSet();
System.out.println("Message Identification:");
System.out.println("  MTF Format: " + msgId.getMessageTextFormat().getValue());

```

In addition, the IDE can no longer provide code completion or help finding desired items in the message as it could before. This makes the code much more time consuming to write and maintain and requires manually combing through schemas.

Impact of NIEM Schema Change

```

ObjectFactory | *TestCOMPOT ja 12 | *TestCOMPOTNIEM | CommunicationsB | MessageBaseType
someElement = event.getCommunicationsEventAlternative();
if (someElement.getName().getLocalPart().equals("CommunicationsEventCode")) {
    CommunicationsEventCodeType eventCode = (CommunicationsEventCodeType) someElement.getValue();
    System.out.println("  Event: " + eventCode.getValue());
}

someElement = event.getEventBeginningDateTimeAlternative();
if (someElement.getName().getLocalPart().equals("EventBeginningDateTimeGroup")) {
    DTGType dtg = (DTGType) someElement.getValue();
    System.out.println("    begin: " + dtg.getValue());
}

someElement = event.getTerminationDateTimeAlternative();
if (someElement.getName().getLocalPart().equals("TerminationDateTimeGroup")) {
    DTGType dtg = (DTGType) someElement.getValue();
    System.out.println("    end: " + dtg.getValue());
}

List<JAXBElement> elementsList = event.getCommunicationsDesignatorAlternative();
if (elementsList.get(0).getName().getLocalPart().equals("CommunicationsDesignatorCircuitName")) {
    CommunicationsCircuitNameType name = (CommunicationsCircuitNameType) elementsList.get(0).getValue();
    System.out.println("    Circuit: " + name.getValue());
}

System.out.println("  Note: " + event.getAmplificationSet().getAmplificationSet().getValue());
System.out.println();

System.out.println("  Location: " + location);
CommunicationsLocationType location = (CommunicationsLocationType) location;
someElement = location.getLocationEventBaseNameAlternative();
if (someElement.getName().getLocalPart().equals("LocationEventBaseNameAlternative")) {

```

GoE schemas result in more complex code where types, getters and XML element names no longer match. As such, the developer must comb through schemas to find type names.

Impact of NIEM Schema Change

```

ObjectFactory | TestCOMSPOT.java | TestCOMSPOT2.java | Communications8 | MessageBaseType
someElement = event.getCommunicationsEventAlternative();
if (someElement.getName().getLocalPart().equals("CommunicationsEventCode")) {
    CommunicationsEventCodeType eventCode = (CommunicationsEventCodeType)someElement.getValue();
    System.out.println("Event: " + eventCode.getValue());
}

someElement = event.getEventBeginningTimeAlternative();
if (someElement.getName().getLocalPart().equals("EventBeginningTimeGroup")) {
    DTType dtg = (DTType)someElement.getValue();
    System.out.println("Begin: " + dtg.getValue());
}

someElement = event.getTerminationDateAlternative();
if (someElement.getName().getLocalPart().equals("TerminationDateGroup")) {
    DTType dtg = (DTType)someElement.getValue();
    System.out.println("End: " + dtg.getValue());
}

List<JAXBElement> elementsList = event.getCommunicationsDesignatorAlternative();
if (elementsList.get(0).getName().getLocalPart().equals("CommunicationsDesignatorCircuitName")) {
    CommunicationsCircuitNameType name = (CommunicationsCircuitNameType)elementsList.get(0).getValue();
    System.out.println("Circuit: " + name.getValue());
}

System.out.println("Note: " + event.getAmplificationSet().getFreeText().getValue());

System.out.println("Location: " + location);
CommunicationsLocationType location = message.getCommunicationsLocationSet();
someElement = location.getLocationOfEventPressReleaseAlternative();
if (someElement.getName().getLocalPart().equals("LocationOfEventPressReleasePlaceName")) {

```

In addition, the IDE cannot provide code completion, detect errors or detect changes for element names inside quotes. Errors in these names will fail silently, causing incorrect behavior.

Lessons Learned: XML Names are Really Important

- Primary and best way for developers to identify specific information applications need from messages
- More stable than contextual or positional identifiers (e.g., 2nd field in 3rd set in 1st segment)
- Adding new names or moving old names is easy
 - Requires no code changes until application needs new item
- Changing names is not easy -- every name change requires code changes -- everywhere the data is used
- Note: This is a simple example with one small message
 - COMSPOT has 199 elements.
 - ATO has 1775. PURPLE has 1317. AIRSUPREQ has 824.

The best names are meaningful and stable

Names Changes not Limited to JAXB Impact all XML Interfaces

XSLT:

```
<xsl:template match="//MessageIdentifier/Originator" > ...
```

XQuery/XPath:

```
//CommunicationLocation[CountryOfTheWorld == 'AFG']"
```

SAX

```

public void startElement(String uri, String localName, ...
    if (localName.equals("CommunicationsEvent")) {
        ...
    } else if (localName.equals("TerminationDateTime")) {

```

DOM

```
ampn = document.getElementsByTagName("Amplification");
```

NIEM changes will break all these expressions

Lessons Learned Substitution Groups and GoE

- No corresponding programming language construct
 - Binding maps to generic "JAXBElement" with no schema-driven assistance (e.g., code completion & help-text) or error checking
- Significantly impacts application code
 - Increases code size & complexity (45% increase in our test)
 - Defeats IDE's ability to provide code completion
 - Requires developers to manually lookup names in schemas
 - Cannot use example messages -- names do not match
 - Defeats compiler's error detection
 - Increases runtime errors in the code
 - Runtime errors are harder to find and fix
 - Requires greater testing efforts

Schema-Driven Editor Test

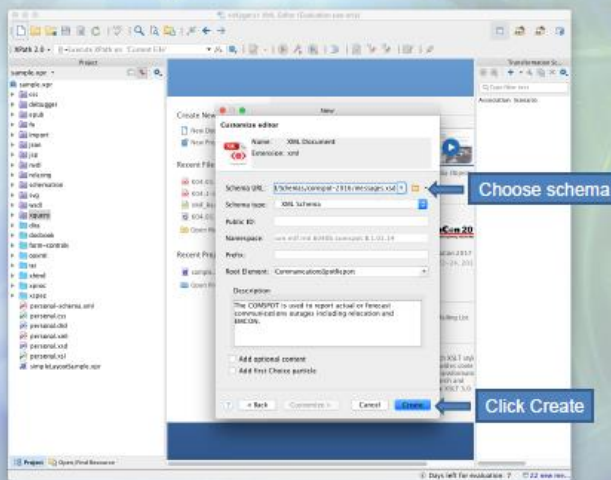
- Purpose: evaluate what a typical XML Editor user will experience switching to new schemas
- Process
 - Use 2016 COMSPOT schemas to create a message
 - Use NIEM COMSPOT schemas to create a message
 - Characterize level-of-effort and any notable issues

Schema-Driven Editor Test

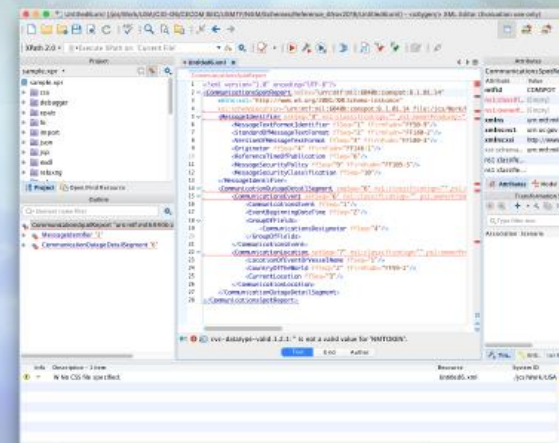
- Create new message using 2016 schemas
- Create new message using NIEM schemas
- Compare schema-driven help and validation

> Demo

Create New Message Using 2016 Schemas

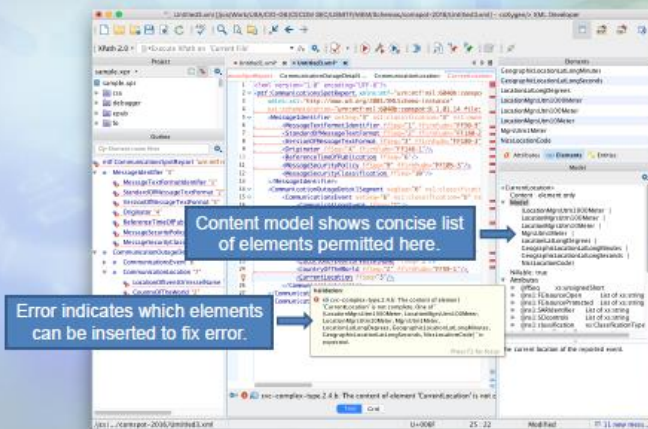


Create New Message Using 2016 Schemas

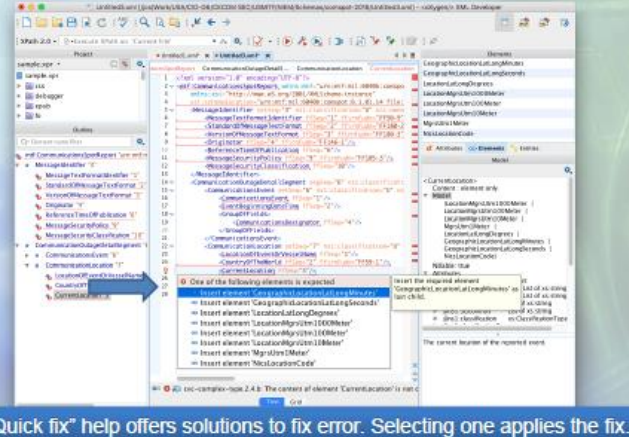


Editor creates new COMSPOT and provides empty template with mandatory items.

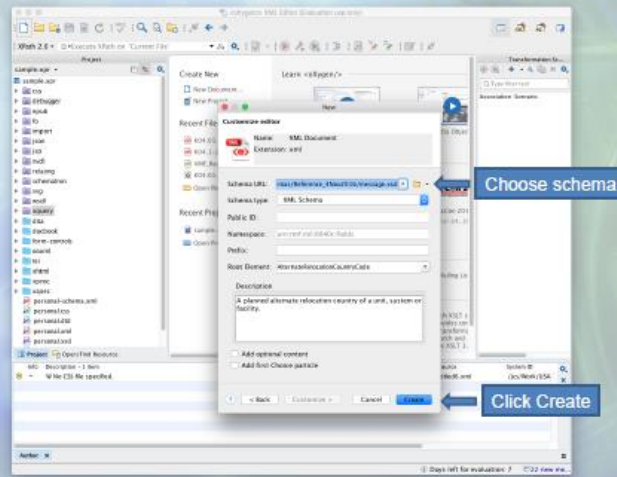
Schema-driven Help Using 2016 Schemas



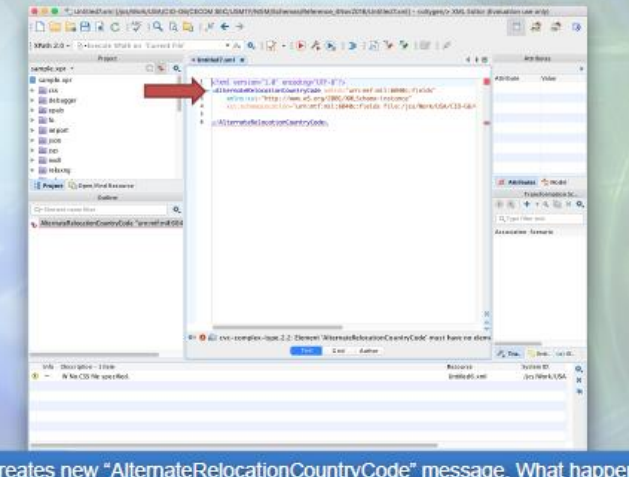
Schema-driven Help Using 2016 Schemas



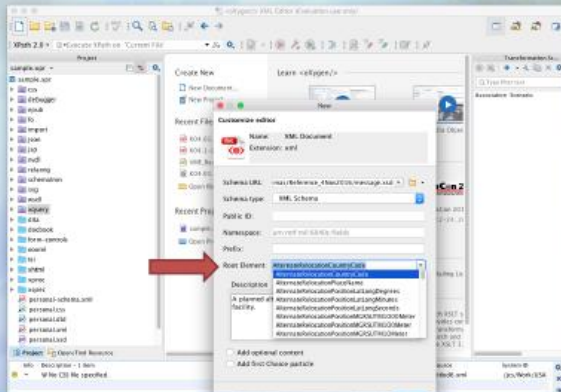
Create New Message Using NIEM Schemas



Create New Message Using NIEM Schemas

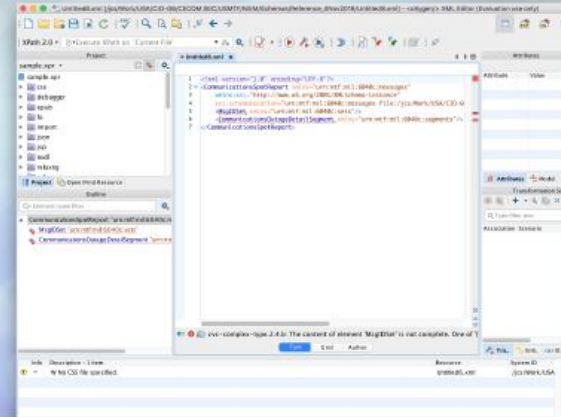


Create New Message Using NIEM Schemas



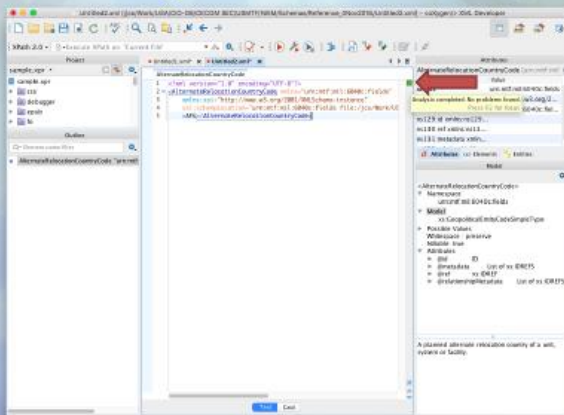
Returning to schema selection screen reveals this is one of about 170 possible messages defined by the COMSPOT schema. GoE makes all segs, sets, fields, etc. top-level elements, which means XML Schema considers them all valid messages.

Create New Message Using NIEM Schemas



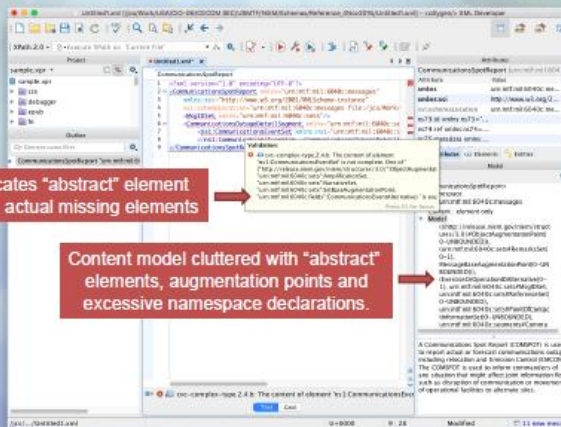
After scrolling through the list and selecting CommunicationsSpotReport, the user gets a blank template with far less completed (7 lines vs. 28 lines).

Create New Message Using NIEM Schemas



After inserting a country code into the original "AlternateRelocationCountryCode" message, the editor says it is a valid message according to the COMSPOT schema.

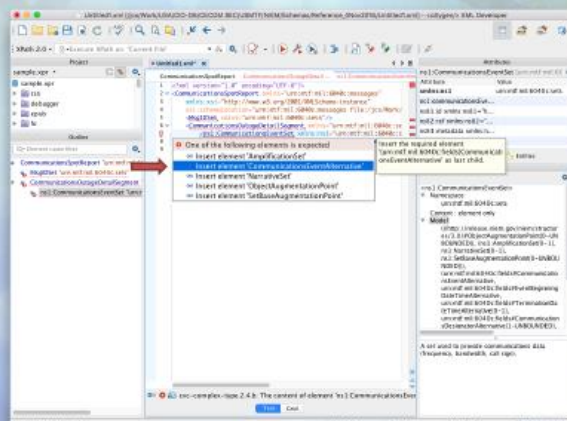
Schema-driven Help Using NIEM Schemas



Error indicates "abstract" element missing vs. actual missing elements

Content model cluttered with "abstract" elements, augmentation points and excessive namespace declarations.

Schema-driven Help Using NIEM Schemas

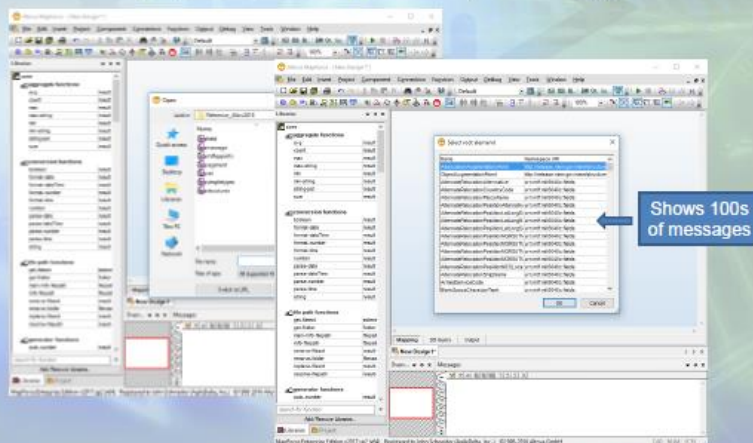


Quick fix help incorrectly offers to insert "abstract" element rather than the correct elements. Choosing this option creates another error rather than fixing the error.

Lessons Learned Schema-Driven Editor

- Garden of Eden
 - Each global element defines a valid message
 - NIEM schemas present users with hundreds / thousands of possible messages rather than one
 - Editor indicates any segment, set, or field is a valid message
 - Excessive namespace declarations clutter user interface
- NIEM schemas auto-complete far less of message (inserts 3 rather than 19 required elements)
- Substitution Groups
 - Reduce helpfulness of model help and error messages
 - Shows "abstract" elements rather than correct elements
 - Quick-fix inserts "abstract" elements rather than correct elements

Same Effects with Other Tools E.g., Altova MapForce, XMLSpy



Schema-Driven Validators

- Check structure and content of a message against all schema constraints
- Producers use as quality control before sending
- Consumers use to verify incoming message before "trusting" and processing them
- Guards use to ensure only known messages get through and as first check for malicious content

Lessons Learned Schema-Driven Validator

- Garden of Eden
 - Validator indicates any segment, set, or field is a valid message and can no longer be used to detect valid / invalid messages
 - May require separate schemas be developed for guards
- Substitution Groups
 - Reduces helpfulness of validator error messages
 - Reports missing "abstract" elements rather than correct elements
 - Augmentation points and excessive namespaces clutter displays and errors

Program Managers and Adoption

"POWER TO THE EDGE"

50

Program Manager Priorities

- Maximize requirements delivered
- With minimize cost, time and risk



The Challenge

- Never have the funds & time to do everything
 - Items that help meet funded requirements come first
 - Items with highest cost, effort and risk come last
 - Best bet is to minimize cost / benefit ratio



Risk: A high cost / benefit may delay implementation for many years

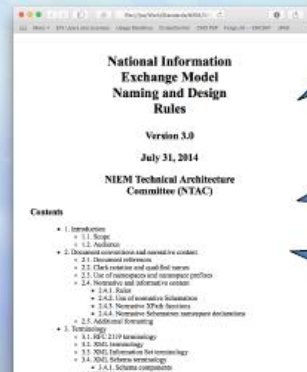
Thoughts and Recommendations

"POWER TO THE EDGE"

53

Minimize Name Changes Acronyms

Most acronym changes are not strictly required



"Acronyms and abbreviations have the ability to improve readability and comprehensibility of large, complex, or frequently used terms."

"They also obscure meaning and impair understanding when their definitions are not clear or when they are used injudiciously. They should be used with great care."

"[Rule 10-49]: The schema MUST use, in defined names, the abbreviations identified by Table 10." (i.e., ID and URI)

Recommendation: Minimize changes. Adopt ID & URI. Document others "as is."

Minimize Name Changes USMTF Local Terminology

- Reduce USMTF "local" terminology
 - E.g., elements representing USMTF sets are not required to end with "Set".
 - Minimize name changes to those strictly required to disambiguate names of sets, segments, fields, etc.
 - Use logical name changes that usefully identify the difference in purpose or meaning.

Note: Names that do not use USMTF terminology will appear less USMTF specific and are more likely to be reused by others.

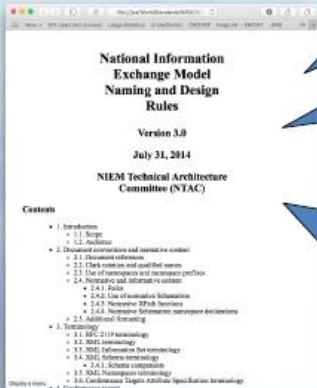
Minimize Name Changes General

- Consider applying other NIEM name changes incrementally (e.g., on new or modified items)
- Avoid hitting program managers with a large, untenable cost all at once.

I.e., Boil the frog

Minimize Substitution Groups

NIEM strives for reuse of common components



"The goal is to maximize interoperability and reuse."

"A real-world entity should be modeled in only one way. The definition of a type or element should appear once and only once."

"Multiple components of identical or closely similar semantics hinder interoperability because too many valid methods exist for representing the same data. For each data concept that must be represented, there should be only one component (and associated type) to represent it."

Minimize Substitution Groups

- However, NIEM uses XML Schema 1.0
 - Restricts each element to a single substitution group
 - Not possible to reuse a common representation of general purpose data concepts across substitution groups.
- Unintended consequence
 - Many general purpose elements that were widely reused must be converted to duplicate, special purpose representations

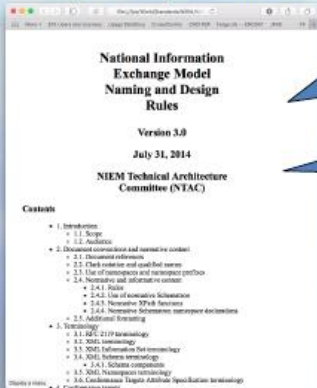
CircularAreaSet	StartPositionCircularAreaSet
HavenCircularAreaSet	WaitPositionCircularAreaSet
SubmarineOperatingAreaCircularAreaSet	MissileSeekerZoneCircularAreaSet
AEWEmploymentCircularAreaSet	TASManCircularAreaSet
EWAAircraftEmploymentCircularAreaSet	MPAMonCircularAreaSet
FEZCircularAreaSet	FixedWingASWMonCircularAreaSet
MEZCircularAreaSet	HelioMonCircularAreaSet
TGTAreaReportingCircularAreaSet	SubmarineMonCircularAreaSet

CircularArea → Becomes 26 total elements

Recommendation: Raise this issue with NIEM. Not likely the intended effect.

Reduce Reference Schemas

The primary source of substitution group issues



"A reference schema document is a schema document that is intended to provide the authoritative definitions of broadly reusable schema components."

"Reference schema documents are intended to be as regular and simple as possible."

- Intended for most general, most abstract core concepts
 - Does not apply to all MTF segs, sets, fields
 - Probably do not want 1000s from us
- May want to hold off until substitution group issues are resolved

Garden of Eden

- Impacts programs and end users
 - Makes coding web-services more time-consuming, complex, error-prone (similar to substitution groups)
 - Makes user experience more complex, time-consuming, error-prone
 - Removes ability to use schema validation for release control, error-detection, guards.
- I.e., there may be an apple in the GoE

Recommendation: Raise this issue with NIEM. Not likely the intended consequences.

Engage with NIEM

- USMTF is probably the largest program to which NIEM has been applied
 - Provides a fantastic way to get NIEM to programs
 - Provides an important source of lessons learned
- Identify unintended consequences and effects on program and user adoption
- Work with NIEM to make it more widely adopted and attractive to programs.
- Work on less disruptive transition strategies for large, existing programs

And Finally ...

"Test everything; retain what is good."

- Testing is important before finalizing standards
- No matter how many smart experts review a big change, they won't find everything
- Testing will identify complex interactions and unanticipated consequences
- And ... socialize changes with programs and solicit feedback

Recommendation: Before approval, implement more messages. Test them.

Summary

- **Objective**
 - Ensure changes are adopted by programs without lengthy delay
- **Testing results**
 - Name changes require significant code changes
 - Substitution groups and GoE make coding web-services more time-consuming, complex, error-prone
 - Substitution groups and GoE make user experience more complex, time-consuming, error-prone
- **Recommendations**
 - Minimize name changes (acronyms, USMTF local terms)
 - Minimize substitution groups, reference schemas and GoE
 - Engage with NIEM to share lessons learned
 - Do more testing before finalizing this change

Questions?

"POWER TO THE EDGE"

64