# Appendix

## Known substitution Group Problems and Workarounds in .NET

When using XSD.exe, there is an issue where substitution groups having members in a namespace different from the one the substitution group head is declared will not always serialize. This issue will manifest itself during testing with unexpected exceptions occurring due to null values in the rendered classes.

Note that the circumstances under which a substitution group member will or will not serialize are somewhat dependent on the overall complexity of the IEPD. However, they can be broken down into the following scenarios:

1. A substitution group with a head element & members in different namespaces.
2. A substitution group without head and extension types in different namespaces.

Also note that it does not matter which namespaces are used. All NIEM examples are shown here but the same holds true for extensions. For this discussion, an IEPD using only NIEM components was fabricated that meets the cases as follows:

<xsd:complexType name="ExchangePackageType">

<xsd:sequence>

<xsd:element ext="PackageName"/>

<xsd:element ref="nc:Activity" minOccurs="0"/>

<xsd:element ref="nc:TangibleItem"/>

<xsd:element ref="nc:Conveyance"/>

</xsd:sequence>

</xsd:complexType>

<xsd:element name="ExchangePackage" type="sub:ExchangePackageType" nillable="true"/>

Each case described above is explained in the context of the above IEPD (and its NIEM subset) below.

### Case 1: A substitution group with a head element & members in different namespaces

In this case, the element nc:Activity/nc:ActivityDate/nc:DateRepresentation has the substitution members nc:Date, nc:DateTime, nc:Year, it:Month, and it:DayOfWeek specified in the NIEM subset schema. In this example, nc:DateRepresentation has a cardinality of 0 to unbounded.

Observe the following snippet of the rendered class DateType:

[System.Xml.Serialization.XmlRootAttribute("ActivityDate",

Namespace="http://niem.gov/niem/niem-core/2.0", IsNullable=true)]

public partial class DateType : ComplexObjectType {

private object[] itemsField;

/// <remarks/>

[System.Xml.Serialization.XmlElementAttribute("Date", typeof(date), IsNullable=true)]

[System.Xml.Serialization.XmlElementAttribute("DateTime", typeof(dateTime),

IsNullable=true)]

[System.Xml.Serialization.XmlElementAttribute("Year", typeof(gYear), IsNullable=true)]

public object[] Items {…

You can see that the XmlElementAttribute above the Items array only contains the substitution group members that were contained in the niem-core namespace. The 2 members from the intelligence namespace are not shown, which means that deserializing the following XML instance fragment would not occur on the intelligence time values.

To remedy this issue, go into the rendered class file and add an XmlElementAttribute attribute for each element name from another namespace, specifying the type and the namespace as shown below.

[System.Xml.Serialization.XmlRootAttribute("ActivityDate",

Namespace="http://niem.gov/niem/niem-core/2.0", IsNullable=true)]

public partial class DateType : ComplexObjectType {

private object[] itemsField;

/// <remarks/>

[System.Xml.Serialization.XmlElementAttribute("DayOfMonth", typeof(DayType),

Namespace = "<http://niem.gov/niem/domains/intelligence/2.1>", IsNullable = true)]

[System.Xml.Serialization.XmlElementAttribute("Month", typeof(gMonth),

Namespace = "<http://niem.gov/niem/domains/intelligence/2.1>", IsNullable = true)]

[System.Xml.Serialization.XmlElementAttribute("Date", typeof(date), IsNullable=true)]

[System.Xml.Serialization.XmlElementAttribute("DateTime", typeof(dateTime),

IsNullable=true)]

[System.Xml.Serialization.XmlElementAttribute("Year", typeof(gYear), IsNullable=true)]

public object[] Items {…

### Case 2: A substitution group without head and extension types in different namespaces

In this case, the element nc:Conveyance is being implicitly substituted with cbrn:Conveyance which is an extension that adds an augmentation to nc:Conveyance. Note that in this case the cardinality of nc:Conveyance within ExchangePackageType is one (required; no more than one). Looking at the rendered class for the “TangibleItem” element, you will see that there is no element name associated with the Conveyance XmlElementAttribute attribute.

[System.Xml.Serialization.XmlElementAttribute(

Namespace="http://niem.gov/niem/niem-core/2.0", IsNullable=true)]

public TangibleItemType TangibleItem {

get {… *[deletia]*

}

/// <remarks/>

[System.Xml.Serialization.XmlElementAttribute(

Namespace = "<http://niem.gov/niem/niem-core/2.0>", IsNullable = true)]

public ConveyanceType Conveyance {

get {…[*deletia]*

}

Consequently, if an XML instance contains a cbrn:conveyance element substituted in place of nc:conveyance, when deserialized, the Conveyance item in class Tangible will be a null value. To remedy this issue, the highlighted XmlElementAttribute attribute above must be replaced with the ones below, where both must explicitly state the element name and the associated namespace.

[System.Xml.Serialization.XmlElementAttribute(

Namespace="http://niem.gov/niem/niem-core/2.0", IsNullable=true)]

public TangibleItemType TangibleItem {

get {… *[deletia]*

}

/// <remarks/>

[System.Xml.Serialization.XmlElementAttribute("Conveyance", typeof(ConveyanceType),

Namespace = "<http://niem.gov/niem/niem-core/2.0>", IsNullable = true)]

[System.Xml.Serialization.XmlElementAttribute("Conveyance", typeof(ConveyanceType1),

Namespace = "<http://niem.gov/niem/domains/cbrn/2.1>", IsNullable = true)]

public ConveyanceType Conveyance {

get {…[*deletia]*

}

Finally, note that the key difference between these two different cases comes down to the cardinality specified for the element. Neither case picks up the substitution group members in the other namespaces. The only difference is that Case 1, having 0 to unbounded cardinality, illustrates how the problem looks with the array and the other illustrates the case of a single type.

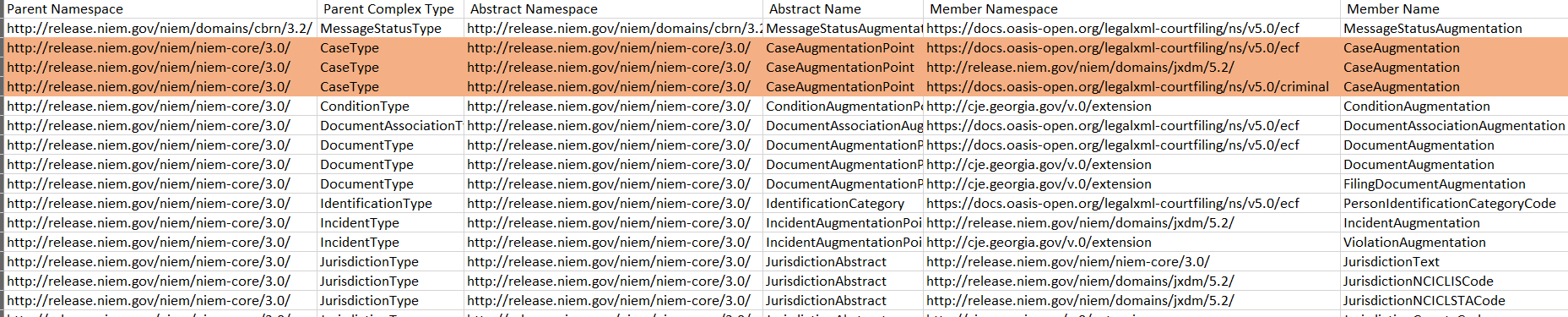
### Examples

To view all substitution groups that need to be manually corrected, we can use the GetSubstitutions tool that can be downloaded from http://bitbucket.org/urli/efilingreviewrefimplh, and is located in the source folder “tool-downloads\GetSubstitutions.zip”. After unzipping the project, use the command line to navigate to the Debug folder. Execute the GetSubtitutions.exe using the following structure:



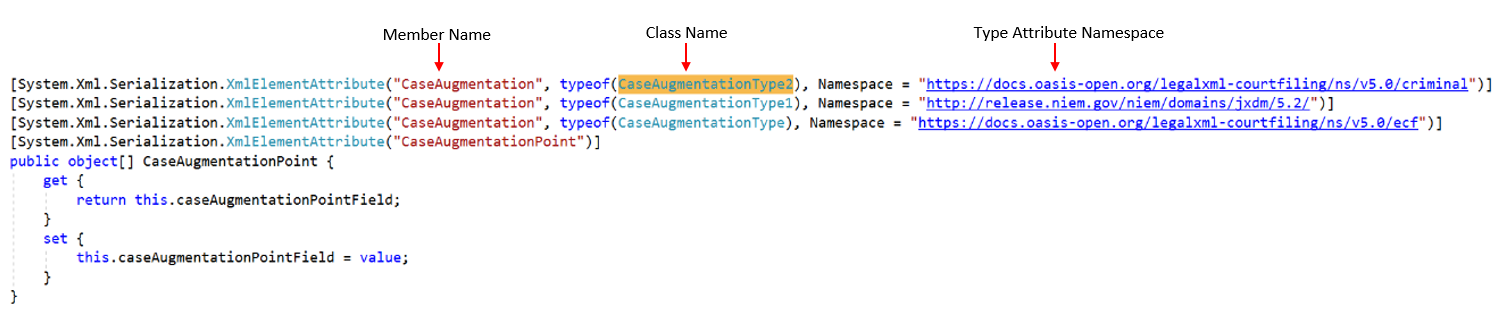
Once run, it will create an Excel spreadsheet in the same Debug folder. This spreadsheet specifies members of a substitution group that have at least one element which was not properly added onto the XsdBindingObject. To fix this, we can cross-reference this spreadsheet to the Xsd.

We will run through examples of two different scenarios of fixes as follows:

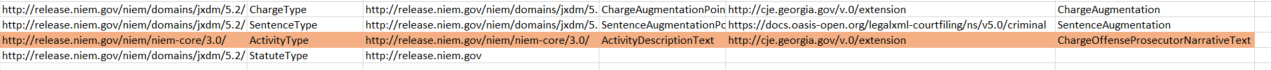
1. Substitution groups with text fields – Case Augmentation
2. Looking at the spreadsheet, we see three fields which have Parent Complex Type of *CaseType*, Abstract Name of *CaseAugmentationPoint* and Member Name of *CaseAugmentation*. What differentiates these three members is that they are derived from different namespaces.
3. If we search the XsdBindingObject for CaseAugmentationType, we will see that there are three cases: *CaseAugmentationType*, *CaseAugmentationType1*, and *CaseAugmentationType2*. Looking at any of the class definitions you will see an XML type attribute:



1. Now search for the Abstract Name, which is CaseAugmentationPoint in this example. Notice that the CaseAugmentationTypes have not been defined. We must add all three in as XMLElementAttributes. The structure is as follows:



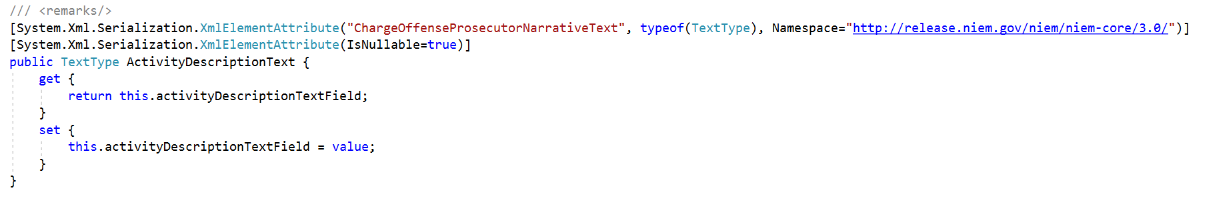
1. Substitution Groups with code lists – ChargeOffenseProsecutorNarrativeText
2. Similar to the previous example, the spreadsheet points out that member ChargeOffenseProsecutorNarrativeText is missing from the XsdBindingObject:



1. We can verify that the member is missing by searching for ActivityDescriptionText. Where this class is defined we can verify that the member attribute is missing.
2. Next, we will look for the namespace for this XMLElementAttribute. We can find this by searching for the class definition of ActivityType.
3. In the previous example, the member was existed as a XmlRootAttribute. However, the XmlRootAttribute is “Activity” in this case, not the name of the member. This makes identifying the type more difficult. We can find the type by looking at the schema. In this example, the schema is Georgia.xsd schema located in:

*CJEP\_EFiling\_SSP\schemas\extensions\georgia.xsd*

1. By doing a search for the member ChargeOffenseProsecutorNarrativeText, we can navigate to the element where it is defined. Here we can see that this element is of type nc:TextType. This will be the type to use in the XmlElementAttribute. Once constructed, the attribute will look something like this:



## XSD.exe and SVCUTIL.exe in .NET

XSD.exe and SVCUTIL.exe are both tools that will generate binding objects from your XML. The primary difference is that SVCUTIL generates the binding objects as well as the necessary classes and interfaces to implement a web service client. In fact, when you add a service reference into a .NET project, the resulting Reference.cs file was created using SVCUTIL. Besides the fact that that the SVCUTIL also generates the actual service classes, there is a subtle difference between the binding objects that are generated between the two; the XSD.exe generated classes specify the XmlRootAttribute, whereas the SVCUTIL.exe generated classes implement the INotifyPropertyChanged interface. The former-generated classes are useful when either release or test code will need to serialize fragments of the XML and not the full payload. The latter is useful if you are tying the binding objects to data-aware objects and want changes to the model to propagate to the data objects and vice-versa. From experience, the use of the binding objects generated from the XSD.exe comes into play more frequently.