Issues handling multicardinal XML

In this write-up I'll show how a collection of XML elements organized in a hierarchical data structure can be ingested using an out-of-the-box GeoEvent Server inbound connector configured to receive XML sent to a GeoEvent Server hosted REST endpoint using an HTTP/POST request. The event data will be processed to allow common attribution for an "incident" to be written to one feature service while data on responding units is written to a separate feature service. The records can then be related within the relational database backend using a one-to-many relationship.

First, let's take a look at the proposed XML structure. The data illustrated below represents an "incident" to which several responding vehicle units are dispatched. Information common to the incident is organized in an element named ***Common*** while a second data structure named ***Units*** at the same level as ***Common*** represents a collection of several vehicle units dispatched to respond to the to the incident.

```xml
<?xml version="1.0" encoding="utf-8"?>
<Incident>
  <Common>
    <Incident_Number>INC_58A3ED45</Incident_Number>
    <Location>
      <Latitude>32.375</Latitude>
      <Longitude>-115.125</Longitude>
    </Location>
  </Common>
  <Units>
    <Unit>
      <Unit_Id>VEH_B337C998</Unit_Id>
      <Dispatched>1581727047</Dispatched>
      <CodedValue>23</CodedValue>
    </Unit>
    <Unit>
      <Unit_Id>VEH_0C260805</Unit_Id>
      <Dispatched>1581727067</Dispatched>
      <CodedValue>36</CodedValue>
    </Unit>
    <Unit>
      <Unit_Id>VEH_A98D4422</Unit_Id>
      <Dispatched>1581727080</Dispatched>
      <CodedValue>41</CodedValue>
    </Unit>
    <Unit>
      <Unit_Id>VEH_3DA02DFA</Unit_Id>
      <Dispatched>1581727097</Dispatched>
      <CodedValue>57</CodedValue>
    </Unit>
    <Unit>
      <Unit_Id>VEH_78B0CD67</Unit_Id>
      <Dispatched>1581727113</Dispatched>
      <CodedValue>63</CodedValue>
    </Unit>
  </Units>
</Incident>
```

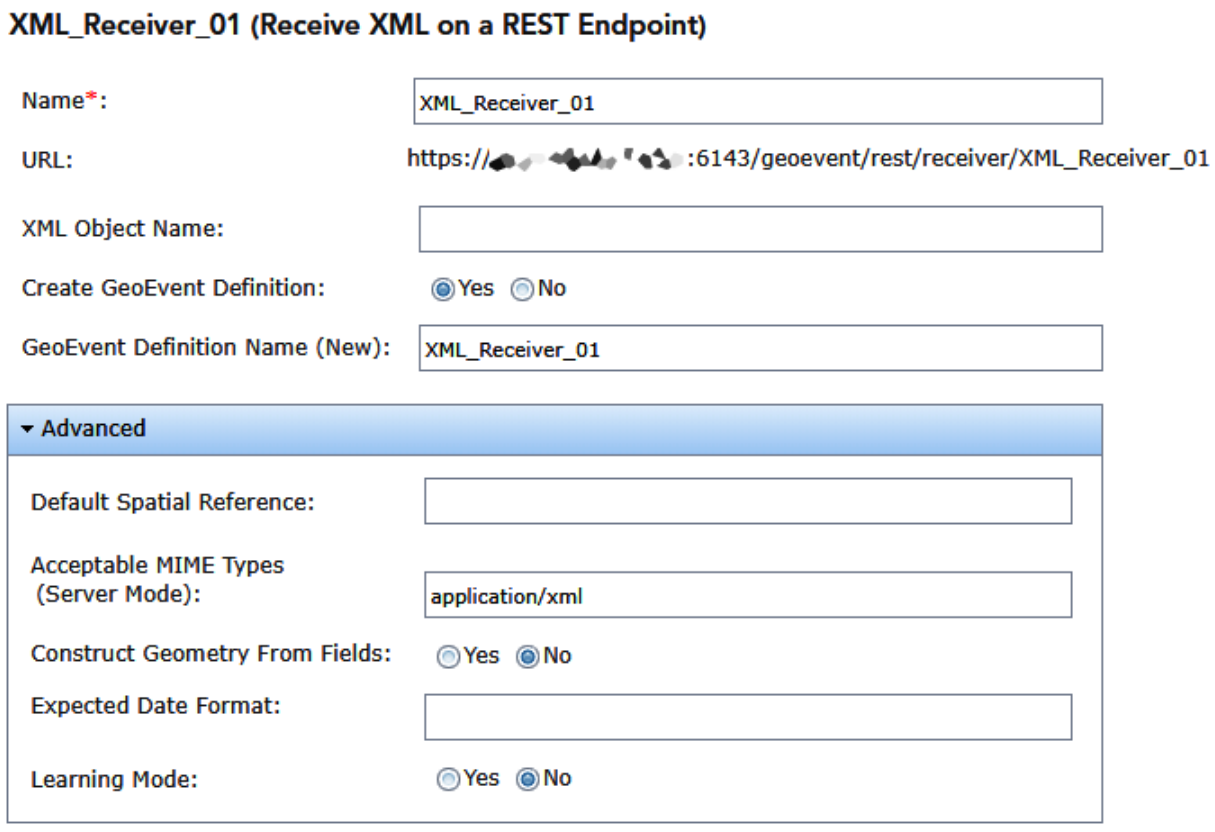
In the XML data structure illustrated above, notice that the cardinality of the ***Units*** element is intended to collect "many" individual ***Unit*** elements. Each ***Unit*** has its own vehicle identifier, a date/time the vehicle was dispatched (reported in epoch seconds) and a coded value which may or may not be unique across vehicles.

Configuring a *Receive XML on a REST Endpoint* inbound connector

In the blog XML Data Structures - Characteristics and Limitations I mentioned that GeoEvent Server uses third party libraries to translate received XML to JSON. While I am not sure when the change was introduced, it seems to me that translation was handled differently prior to the 10.7 product release. Specifically, something seems to have changed within the XML parser and how root-level nodes are recognized.

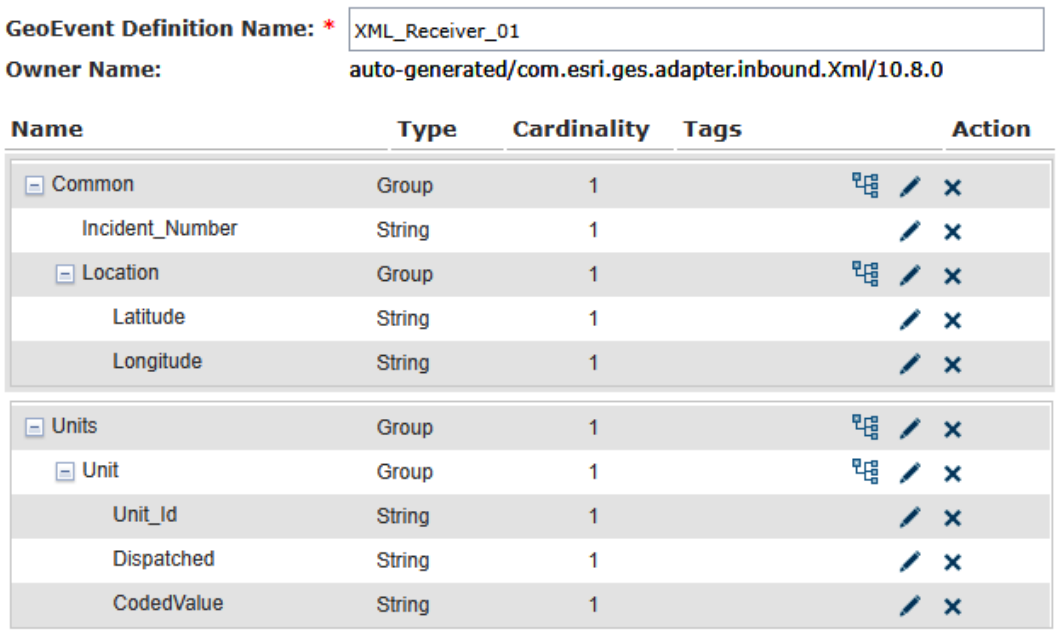
For example, given the XML we want to ingest for this exercise, I noticed that if we want to use ***Incident*** as the natural root element of the data structure the inbound adapter prefers that its *XML Object Name* property be left unspecified. My testing has suggested that I specify an element other than ***Incident*** only if I want the XML parser to delve into the data structure and parse a sub-structure within the XML. In this case I would like data from both ***Common*** and ***Units*** to be included in event records ingest for processing, so at least initially I will leave the *XML Object Name* property unspecified.

For this exercise I want to receive XML sent to a GeoEvent Server hosted REST endpoint using an HTTP/POST request, so my first step is to configure a new *Receive XML on a REST Endpoint*. The XML does not incorporate any declared namespace(s) in its attribute names, so it should be relatively straight forward to have the input create a GeoEvent Definition for me based on the first data record received.



Notice to simplify initial adaptation of the data I have chosen to ***not*** have the input try to construct a geometry for me using coordinate data found beneath the element ***Common***.

The initial adaptation of the XML data structure produced the GeoEvent Definition illustrated below.



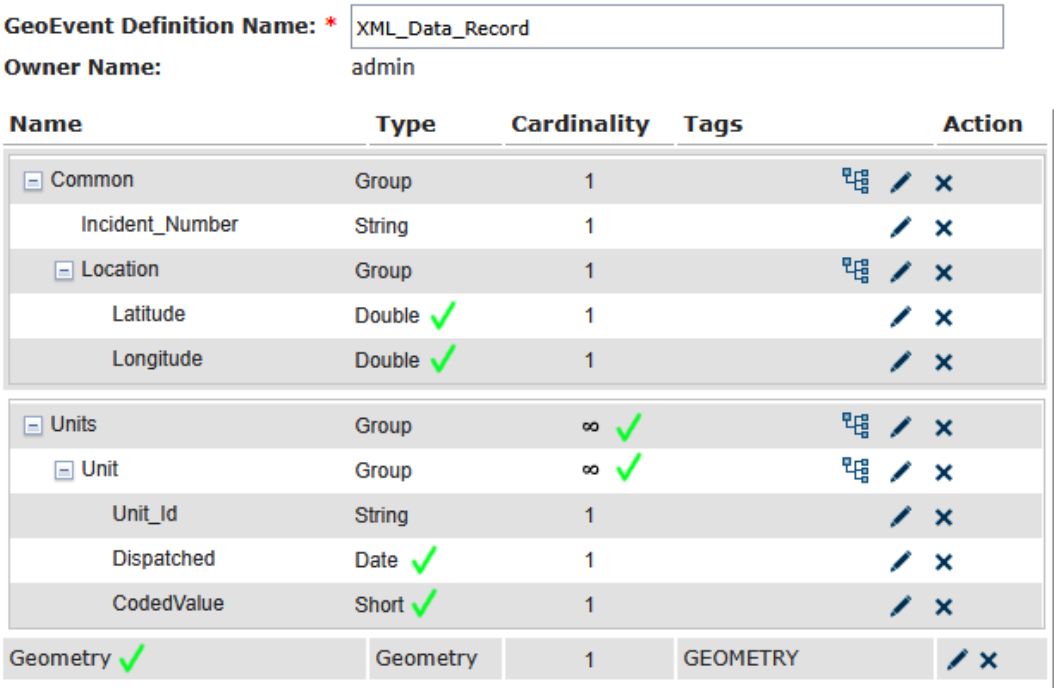
GeoEvent Definition Name: * XML_Receiver_01

Owner Name: auto-generated/com.esri.ges.adapter.inbound.Xml/10.8.0

Name	Type	Cardinality	Tags	Action
Common	Group	1		
Incident_Number	String	1		
Location	Group	1		
Latitude	String	1		
Longitude	String	1		
Units	Group	1		
Unit	Group	1		
Unit_Id	String	1		
Dispatched	String	1		
CodedValue	String	1		

Examining the above GeoEvent Definition we see that the hierarchy beneath ***Incident*** was parsed more or less correctly, but we know from the blog [XML Data Structures - Characteristics and Limitations] that the cardinality of ***Units*** at least was probably not correctly recognized. This is likely because dissimilar sub-structures, in this case ***Common*** and ***Units***, are grouped beneath a single root node ***Incident***.

It took some experimentation, but I found that I could copy the input's guess of the expected event record schema and edit my copy of the GeoEvent Definition to handle both ***Units*** and ***Unit*** as multicardinal elements. This was necessary in order to be able to receive and adapt subsequent blocks of XML whose structure mirrors the schema used in this exercise. I also took the opportunity to correct the data type for the coordinate values being received, the vehicle dispatch date/time, my vehicle's coded value, and add a Geometry field.



GeoEvent Definition Name: * XML_Data_Record

Owner Name: admin

Name	Type	Cardinality	Tags	Action
Common	Group	1		
Incident_Number	String	1		
Location	Group	1		
Latitude	Double ✓	1		
Longitude	Double ✓	1		
Units	Group	∞ ✓		
Unit	Group	∞ ✓		
Unit_Id	String	1		
Dispatched	Date ✓	1		
CodedValue	Short ✓	1		
Geometry ✓	Geometry	1	GEOMETRY	

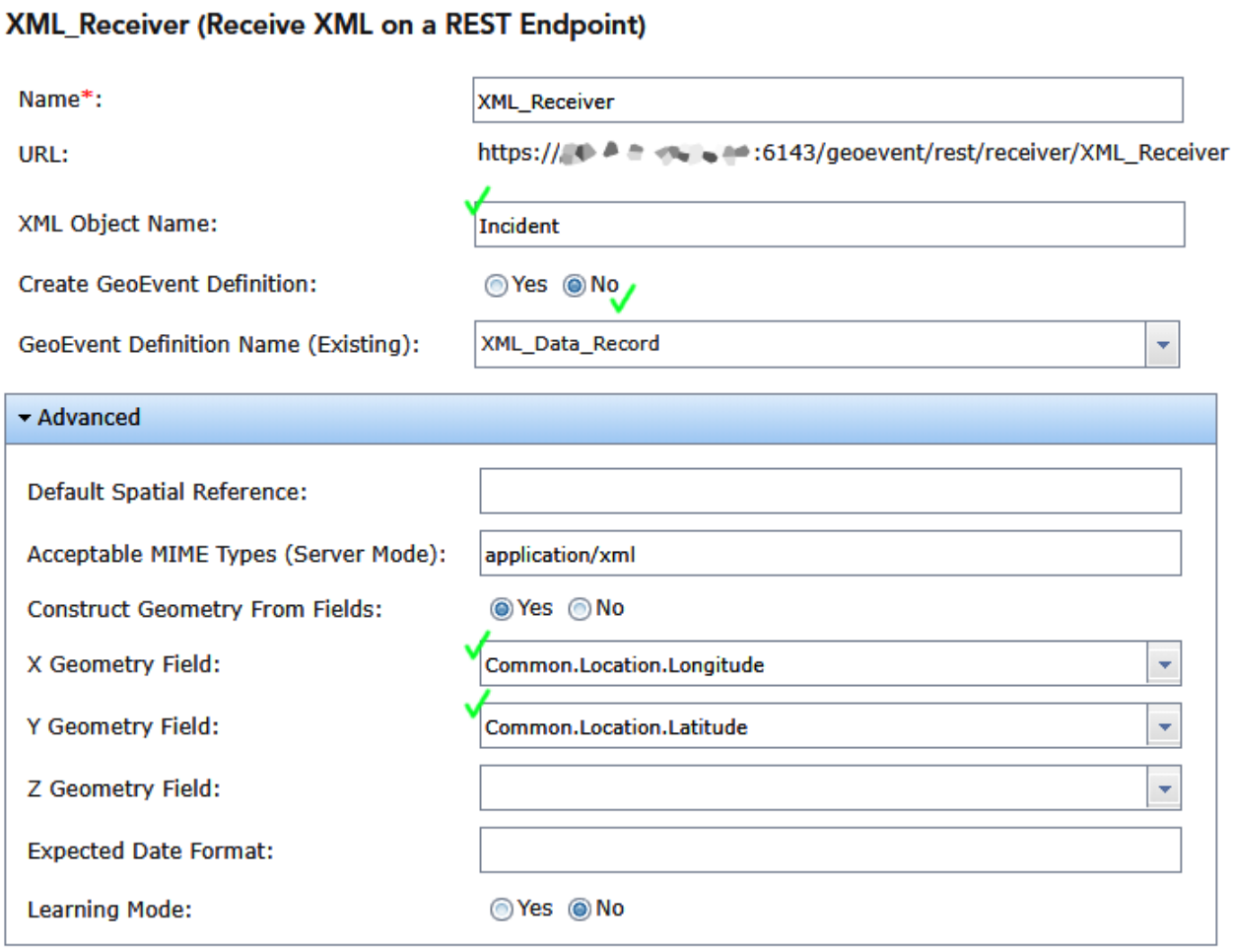
Reconfiguring the *Receive XML on a REST Endpoint input*

Once I have a GeoEvent Definition tailored specifically for data I expect to receive, I need to go back to the input and edit it specify to use my tailored GeoEvent Definition rather than attempting to create a GeoEvent Definition for me. There are actually a few edits I want to make to my input's configuration; I want to:

- Specify the event definition I want the input to use
- Have the input construct a Geometry for me from coordinate values received in the data
- Specify an *XML Object Name* the parser can use to identify the root of the data structure

I find this last change counterintuitive and confusing. For some reason when the parser is allowed to assume singular cardinality for all elements it does not want to be told that ***Incident*** should be treated as the root of the data structure. Changing the cardinality of the ***Units*** and ***Unit*** elements broke my input's ability to parse the XML structure, however, until I specified which node should be considered the root. This is behavior I am pretty sure changed when underlying libraries used to translate XML to JSON were updated.

The changes I made to my input are illustrated below.



Successfully ingesting the XML data is only half of the battle however. I am rather unhappy that both the ***Units*** and ***Unit*** elements have to be handled as multicardinal when, in my opinion, ***Units*** can be represented as a collection of simple objects. But trial and error demonstrated that this is what I had to do to get the input to repeatedly ingest and adapt the sample XML for this exercise.

Limitations of the *Multicardinal Field Splitter* processor

An important best practice I followed earlier when needing to edit a GeoEvent Definition created for me by an input was to make a copy of the event definition and then edit my copy. This prevents a component, the input in this case, from deleting or committing changes to an event definition it owns and inadvertently discarding or overwriting changes I deliberately made.

Unfortunately, I have to violate this best practice recommendation when using a pair of Multicardinal Field Splitter processors in series to flatten the ***Units*** and ***Unit*** element hierarchies. This will present us with some additional challenges we have to work around.

The current implementation of the Multicardinal Field Splitter processor does not allow two or more instances of the processor to be placed in series. When the first processor introduces a new attribute *childIndex* into the GeoEvent Definition of event records it produces the second instance of the processor is unable to create its version of the event definition because there is already an attribute with the name *childIndex*. We can work around this, but the work around is a bit tricky.

We have to allow some XML to be received and processed by the first Multicardinal Field Splitter, splitting the structure on the ***Units*** element, so that the processor can create its managed GeoEvent Definition. We can then edit the managed event definition, owned by the processor, to remove the *childIndex* attribute field, and allow some more XML to be received and processed so the second Multicardinal Field Splitter, splitting the structure on the ***Unit*** element, can create its managed GeoEvent Definition.

You have to remember to edit the managed GeoEvent Definition every time you make an edit to the GeoEvent Service and publish your changes because the Multicardinal Field Splitter will delete the GeoEvent Definition it owns which you have edited, reintroducing the *childIndex* attribute, preventing the second Multicardinal Field Splitter processor from creating its managed GeoEvent Definition.

Here are warning and error messages you can expect to be logged as part of this process:

```
com.esri.geoevent.processor.multicardinalfieldsplitter.MulticardinalFieldSplitter
   GeoEventDefinitionManager failed to add temporary GeoEventDefinition:
   MC_Field_Splitter_02 has duplicate fields
```

```
com.esri.ges.messaging.jms.GeoEventCreatorImpl
   WARNING: provided field values doesnt match in size with GeoEvent Definition ({0})
```

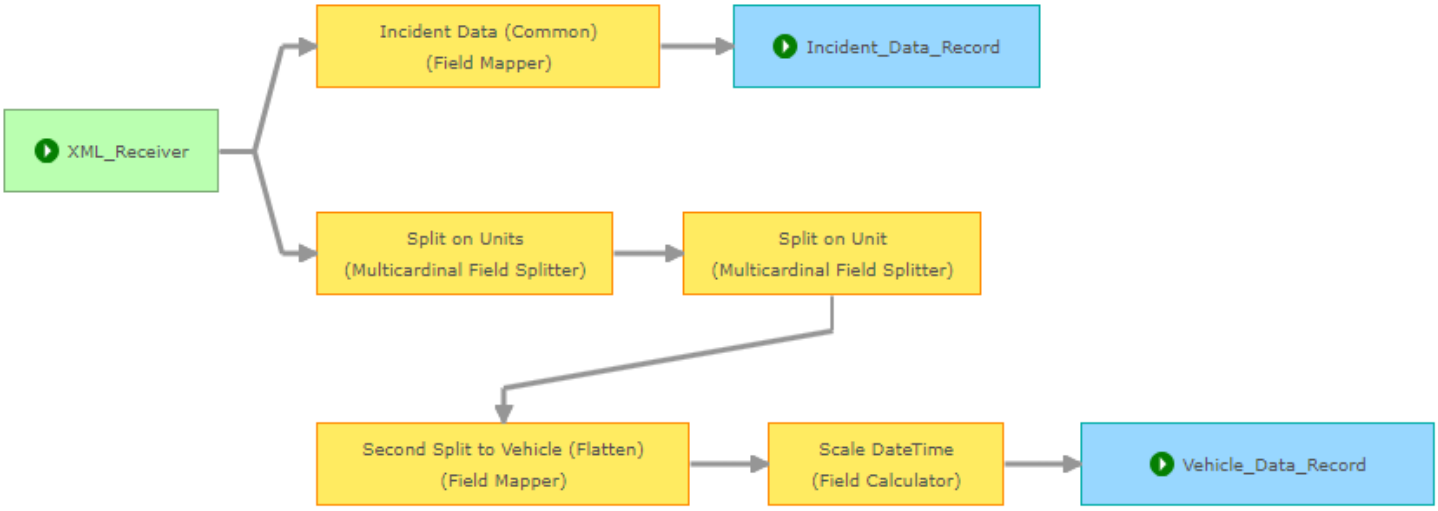
The first error message is logged when the second Multicardinal Field Splitter in the pair/series receives an event record which already has a field named *childIndex*. When you edit the GeoEvent Definition to delete the unwanted field and send the XML data through the GeoEvent Service again the first processor logs the latter error message warning that the GeoEvent Definition being used does not have the expected fields (because you deleted the attribute *childIndex* from its managed GeoEvent Definition).

You will, unfortunately, have to accept these error messages being logged or disable the logged messages as described in the blog GeoEvent Configuration: Turning Log Messages Off.

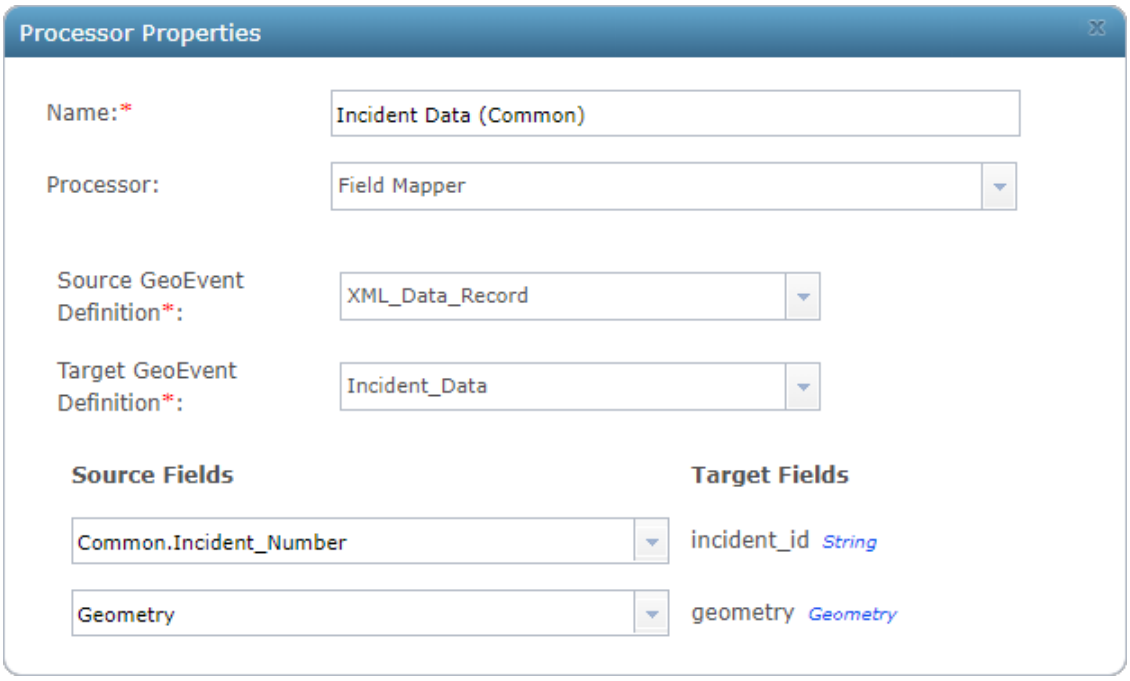
Let's take a look at the final GeoEvent Service configured to process the XML data and persist the event data as feature records in a traditional relational geodatabase's feature service.

Designing and publishing the GeoEvent Service

Once we have successfully ingest the XML and understand the limitations of using a pair of Multicardinal Field Splitter processors to flatten the event record hierarchy for us, we are left with the challenge of field mapping the processed event record(s) in order to write the common data for each incident into one feature service's feature record set while also persisting data describing responding units in a separate feature record set via a second feature service.



The upper branch maps values for the incident identifier and geometry from each event record and then adds/updates feature records using the incident identifier as the TRACK_ID. The field mapper, event data received by the processor, and event data mapped to the feature record's schema(s) are illustrated below.



Received XML (translated to JSON)

```
{
    "Common": {
        "Incident_Number": "INC_58A3ED45",
        "Location": {
            "Latitude": 32.375,
            "Longitude": -115.125
        }
    },
    "Units": [{
        "Unit": [{
            "Unit_Id": "VEH_B337C998",
            "Dispatched": 1581727047,
            "CodedValue": 23
        }, {
            "Unit_Id": "VEH_0C260805",
            "Dispatched": 1581727067,
            "CodedValue": 36
        }, {
            "Unit_Id": "VEH_A98D4422",
            "Dispatched": 1581727080,
            "CodedValue": 41
        }, {
            "Unit_Id": "VEH_3DA02DFA",
            "Dispatched": 1581727097,
            "CodedValue": 57
        }, {
            "Unit_Id": "VEH_78B0CD67",
            "Dispatched": 1581727113,
            "CodedValue": 63
        }]
    }],
    "Geometry": {
        "x": -115.125,
        "y": 32.375,
        "z": 0,
        "spatialReference": {
            "wkid": 4326
        }
    }
}
```

Incident_Data (feature schema)

```
{
  "incident_id" : "INC_58A3ED45",
  "geometry" : {
    "x" : -115.125,
    "y" : 32.375,
    "z" : 0,
    "spatialReference" : {
      "wkid" : 4326
    }
  }
}
```

Vehicle_Data (feature schema)

```
{
  "incident_id" : "INC_58A3ED45",
  "unit_id" : "VEH_B337C998",
  "dispatch_dt" : 1581727047000,
  "coded_value" : 23
},{
  "incident_id" : "INC_58A3ED45",
  "unit_id" : "VEH_3DA02DFA",
  "dispatch_dt" : 1581727097000,
  "coded_value" : 57
},{
  "incident_id" : "INC_58A3ED45",
  "unit_id" : "VEH_A98D4422",
  "dispatch_dt" : 1581727080000,
  "coded_value" : 41
},{
  "incident_id" : "INC_58A3ED45",
  "unit_id" : "VEH_0C260805",
  "dispatch_dt" : 1581727067000,
  "coded_value" : 36
},{
  "incident_id" : "INC_58A3ED45",
  "unit_id" : "VEH_78B0CD67",
  "dispatch_dt" : 1581727113000,
  "coded_value" : 63
}
```

The lower branch, with the Multicardinal Field Splitter processors, flattens the ***Units*** and ***Unit*** hierarchies, maps the event record schema to the feature service's schema, and scales out the epoch date/time from seconds to milliseconds. The processors participating in that workflow are illustrated below.

Processor Properties

Name:* Split on Units

Processor: Multicardinal Field Splitter

Field to Split: Units

Resulting GeoEvent Definition Name: MC_Field_Splitter_01

Processor Properties

Name:* Split on Unit

Processor: Multicardinal Field Splitter

Field to Split: Unit

Resulting GeoEvent Definition Name: MC_Field_Splitter_02

Processor Properties

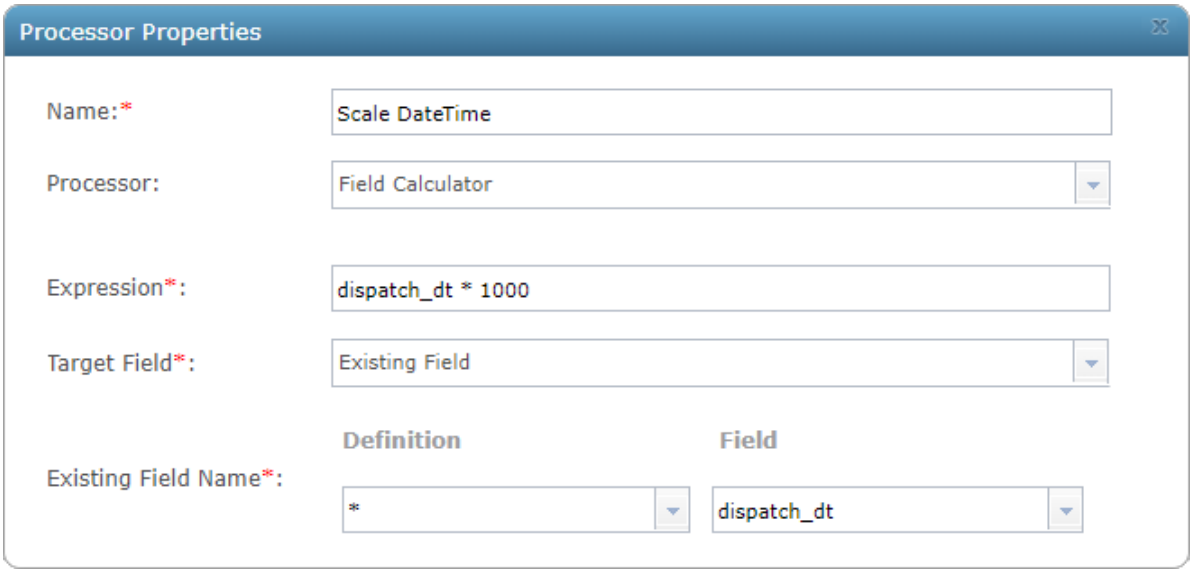
Name:* Second Split to Vehicle (Flatten)

Processor: Field Mapper

Source GeoEvent Definition*: MC_Field_Splitter_02

Target GeoEvent Definition*: Vehicle_Data

Source Fields	Target Fields
Common.Incident_Number	incident_id *String*
Unit_Id	unit_id *String*
Dispatched	dispatch_dt *Date*
CodedValue	coded_value *Short*



Remember, any changes you make to the GeoEvent Service, to alter a processor's configuration, add or change an input or output, will likely drive the Multicardinal Field Splitter processors to delete their managed GeoEvent Definitions when you publish the changes to the GeoEvent Service.

This will force you to send some data through the GeoEvent Service, edit the recreated event definition ***MC_Field_Splitter_01*** to delete the unwanted *childIndex* attribute, then send some more data through the GeoEvent Service so that the second Multicardinal Field Splitter processor can create its GeoEvent Definition.

If you double-click Field Mapper configured to map values from ***MC_Field_Splitter_02*** to the ***Vehicle_Data*** at a point in time that the GeoEvent Definition ***MC_Field_Splitter_02*** does not exist, you may have to send XML data through the GeoEvent Service to ensure the needed GeoEvent Definitions are created, then fix the Field Mapper and publish the GeoEvent Service. This may delete the managed event definitions forcing you to send data through to re-create them – but you should be able to leave the Field Mapper's configuration untouched and it will discover the GeoEvent Services in time to handle the mapping to the feature record schema used to store information on vehicles responding to an incident.