Article
Julian Matthews · Jul 21, 2021 4m read

Splitting ORU Messages using ObjectScript and DTL

Over the years, I have found myself needing to create multiple HL7 messages based on a single inbound message. Usually these take the form of an order or result from a lab. Each time I have approached it, I have tried to start from scratch under the belief that the previous attempt could have been done better.

Recently, the need arose again and I was able to create a solution that I wasn't ashamed of. My main concern was that I would always either find myself getting buried in a BPL, or use ObjectScript and attempt to edit messages using the SetValueAt Method for the HL7 Message Class.

Problem

When System A processes multiple orders for a single patient, the result will come in a single message with repeating ORCgrp with the OBR and OBX segments contained within this. System B can only receive a single OBR per message.

Approach

Develop an ObjectScript process that will split a single message into multiple based on the number of ORCgrp repetitions, and do so while only manipulating the HL7 message with a DTL.

Execution

The Code (part 1)

To start, we need a class that extends Ens.BusinessProcess, and expects a HL7 message on request:

```
Class Demo.Processes.MessageSplitter Extends Ens.BusinessProcess{
 Method OnRequest(pRequest As EnsLib.HL7.Message) As %Status{
    Quit $$$OK
    }
}
```

The next step is to loop through the message based on the number of ORCgrp repetitions. For this we will need 2 things:

1. The number of repetitions
2. A For loop

To get the number of repetitions, we can grab the count from the message using the following code:

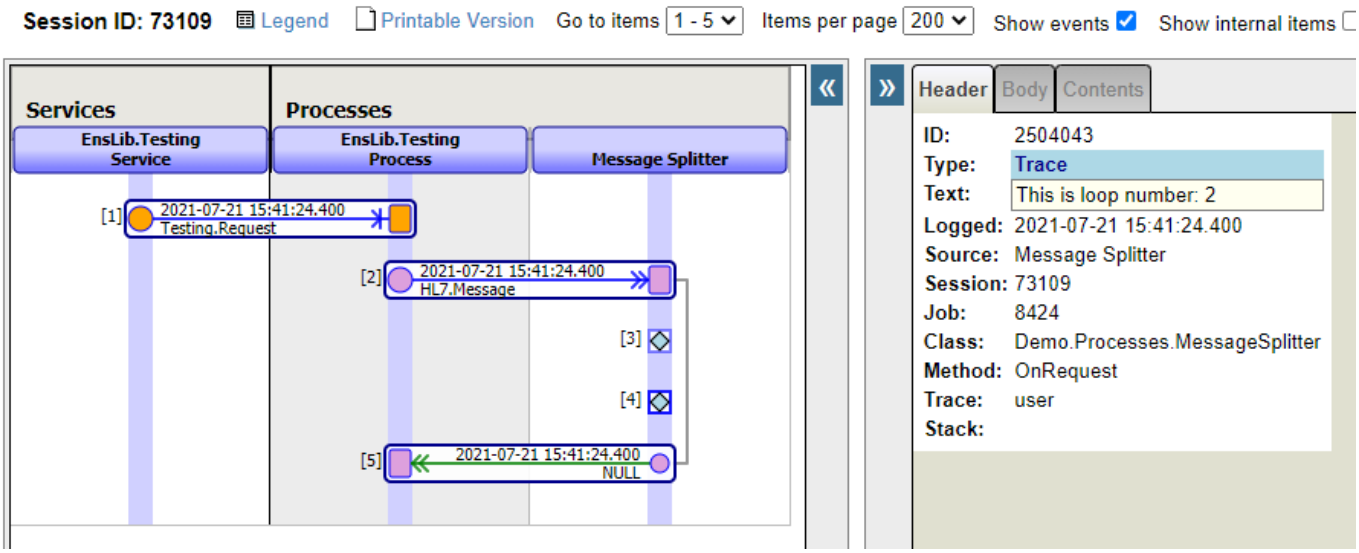
```
Set ORCCount = pRequest.GetValueAt("PIDgrpgrp(1).ORCgrp(*)")
```

This will set the variable "ORCCount" to the number of repetitions.

Putting this together with a For loop and a trace to see some output looks like this:

```
Method OnRequest(pRequest As EnsLib.HL7.Message) As %Status{
    Set ORCCount = pRequest.GetValueAt("PIDgrpgrp(1).ORCgrp(*)")
    For i=1:1:ORCCount {
        $$$TRACE("This is loop number: "_i)
    }
    Quit $$$OK
}
```

Passing a message with two ORCgrp repetitions through this process from within a production gives us:



As you can see, we get two traces.

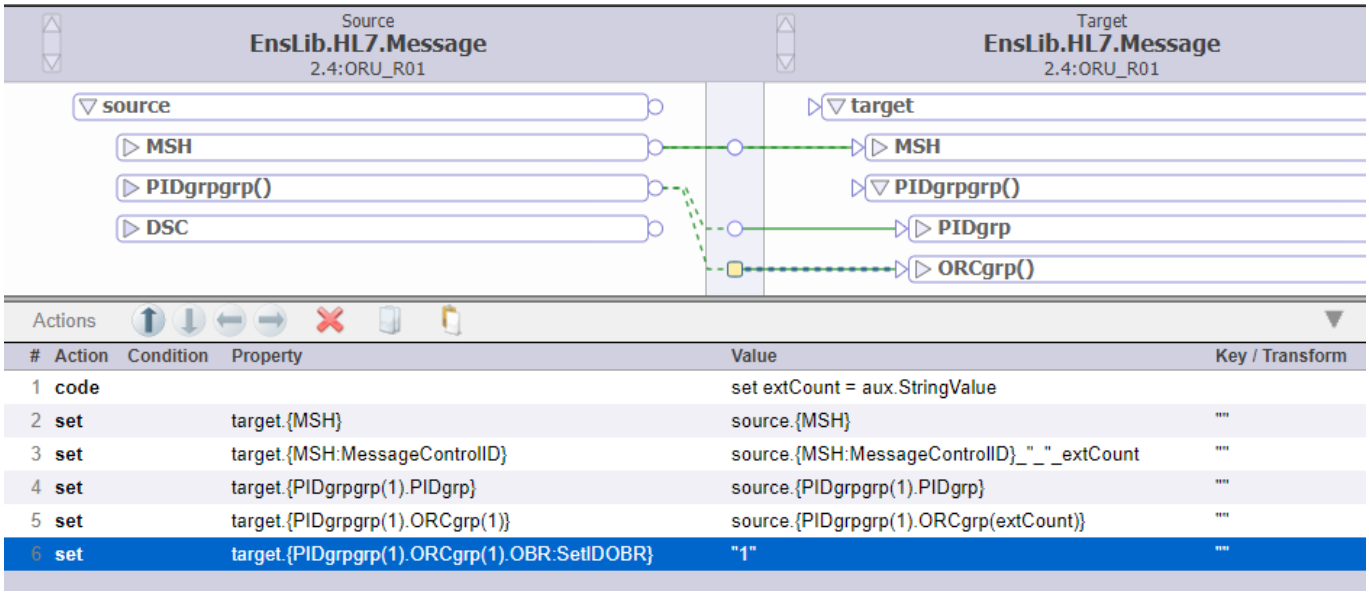
Now from here, we need to be able to call a transform, and also be able to tell that transform which iteration of the ORCgrp it should be returning. For this, we will use the sometimes overlooked "aux" parameter for transform classes.

The Transform

The transform itself can be very simple. All we want for this is to:

1. Copy the MSH Header while making the MessageControlID unique to your split message
2. Set the first ORCgrp of the target message to the iteration we are on from the source
3. Set the Target SetIDOBR to "1", as it will always be the first in the target message.

For this, our transform should look a little like this:



But hang on - where is aux.StringValue getting data?

To find that out, we need to go back to the code...

The Code (part 2)

We can pass a class to the transform via the aux parameter, and for this scenario I am just going to use a string container. We are also going to send the output of the transform to a target so we can see the results:

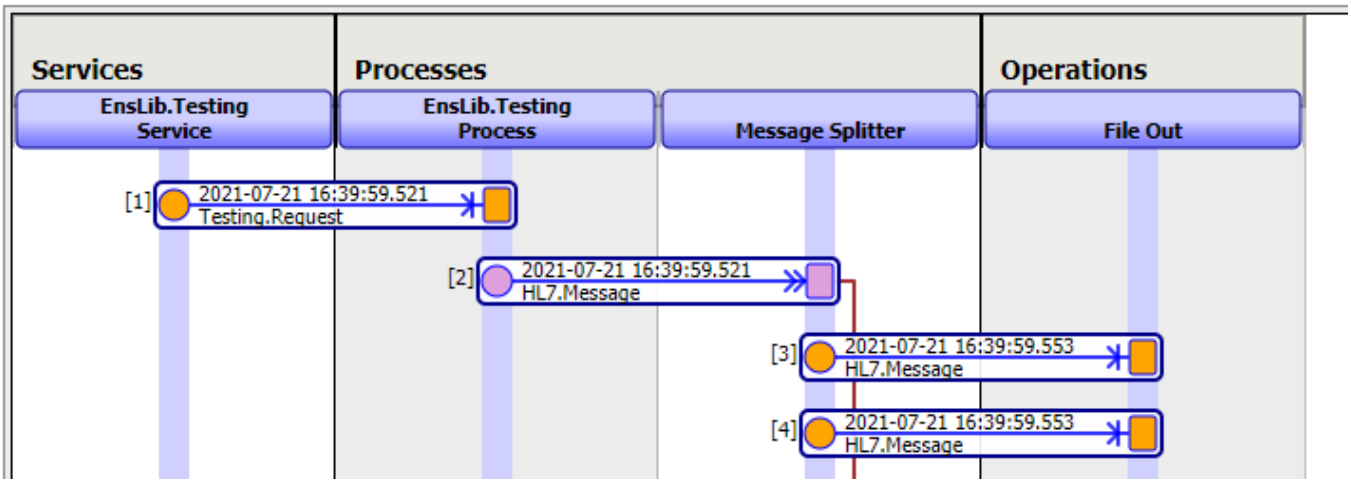
```
Class Demo.Processes.MessageSplitter Extends Ens.BusinessProcess{

Property TargetConfigName As Ens.DataType.ConfigName;
Parameter SETTINGS = "TargetConfigName";

Method OnRequest(pRequest As EnsLib.HL7.Message) As %Status{

    Set ORCCount = pRequest.GetValueAt("PIDgrpgrp(1).ORCgrp(*)")
    For i=1:1:ORCCount {
        Set contextString = ##class(Ens.StringContainer).%New()
        Set contextString.StringValue = i
        $$$QuitOnError(##Class(Demo.Transformations.R01Split).Transform(pRequest,.Spl
itR01,.contextString))
        $$$QuitOnError(..SendRequestAsync(..TargetConfigName,SplitR01,0))
    }
    Quit $$$OK
  }
}
```

Then in the production we set a destination using the new settings option we created with the property and parameter at the top of the class, and we should see something like this:



Conclusion

As I said at the start of this, I always seem to develop a solution to this type of issue and then look back and think it could be done better. This is no exception, but it's certainly better than previous iterations.

To improve this in the short term, I will be adding descriptive comments to the ObjectScript. Longer term, I would like to be able to add a setting for the transformation class so that it can be controlled from outside of the ObjectScript.

Overall, I can see this as an approach I will be taking for future developments, and not completely starting from scratch.

(ps - I am happy sharing the code for this, however I can only attach a pdf to this article. This feels a little light to be something for Open Exchange, but if there is any interest in me uploading it there or somewhere else, please let me know)

#DTL #HL7 #ObjectScript #Caché #Ensemble

Source URL: https://community.intersystems.com/post/splitting-oru-messages-using-objectscript-and-dtl