Article
Lorenzo Scalese · Déc 16, 2022 4m de lecture

Répartition des messages ORU à l'aide d'ObjectScript et DTL

Au fil des ans, je me suis souvent retrouvé dans la nécessité de créer plusieurs messages HL7 à partir d'un seul message entrant. Il s'agit généralement d'une commande ou d'un résultat provenant d'un laboratoire. Chaque fois que j'ai abordé ce problème, j'ai essayé de repartir de zéro en pensant que la tentative précédente aurait pu être mieux faite.

Récemment, le besoin est réapparu et j'ai pu créer une solution dont je n'avais pas honte. Mon principal souci était que je me retrouvais toujours à m'enterrer dans une BPL, ou à utiliser ObjectScript et à tenter de modifier des messages en utilisant la méthode SetValueAt pour la classe de messages HL7.

Problème

Lorsque le Système A traite plusieurs commandes pour un seul patient, le résultat est transmis dans un seul message avec un ORCgrp répété, contenant les segments OBR et OBX. Le Système B ne peut recevoir qu'un seul OBR par message.

Approche

Développez un processus ObjectScript qui divisera un message unique en plusieurs en fonction du nombre de répétitions ORCgrp, et ce, en manipulant uniquement le message HL7 avec un DTL.

Exécution

Le code (partie 1)

Pour commencer, nous avons besoin d'une classe qui étend Ens.BusinessProcess, et qui attend un message HL7 sur demande :

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

L'étape suivante consiste à boucler le message en fonction du nombre de répétitions ORCgrp. Pour cela, 2 choses sont nécessaires :

1. Le nombre de répétitions
2. Une boucle de type For

Pour obtenir le nombre de répétitions, nous pouvons récupérer le compte du message en utilisant le code suivant :

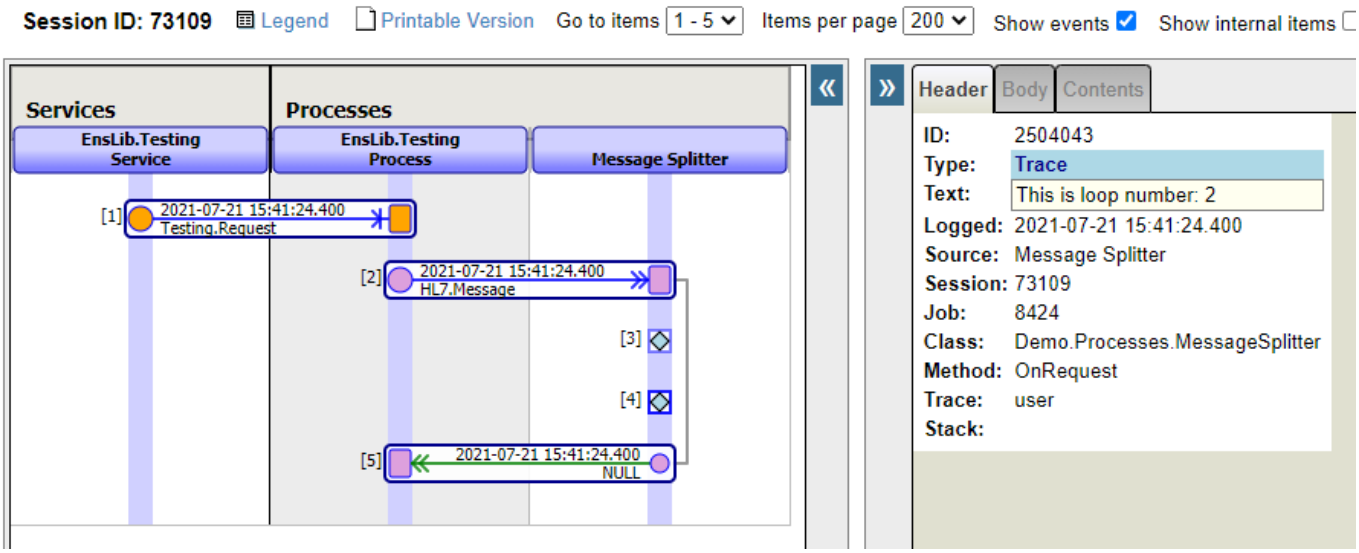
```
Set ORCCount = pRequest.GetValueAt("PIDgrpgrp(1).ORCgrp(*)")
```

Cela va définir la variable "ORCCount" en fonction du nombre de répétitions.

En combinant cela avec une boucle type For et une trace pour voir la sortie, cela ressemble à ceci :

```
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
}
```

En faisant passer un message avec deux répétitions ORCgrp par ce processus à partir d'une production, nous obtenons :



Comme vous pouvez le voir, nous obtenons deux traces.

A partir de là, nous devons être en mesure d'appeler une transformée, et aussi d'indiquer à cette transformée quelle itération de l'ORCgrp elle doit retourner. Pour cela, nous allons utiliser le paramètre "aux", parfois négligé, pour les classes de transformée.

La transformée

La transformée elle-même peut être très simple. Tout ce que nous voulons pour cela est le suivant :

1. Copier l'en-tête MSH tout en rendant le MessageControlID unique à votre message splitté
2. Définir le premier ORCgrp du message cible à l'itération sur laquelle nous sommes depuis la source.
3. Définir la valeur Target SetIDOBR à " 1 ", car elle sera toujours la première dans le message cible.

Pour cela, notre transformée devrait ressembler à ceci :

Mais attendez - où aux. StringValue obtient-il des données ?

Pour le savoir, il faut revenir au code...

Le code (partie 2)

Nous pouvons transmettre une classe à la transformée via le paramètre aux, et pour ce scénario, je vais juste utiliser un conteneur string. Nous allons également envoyer le résultat de la transformation à une cible afin de voir les résultats :

```
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
       $$$QuitOnError(##Class(Demo.Transformations.R01Split).Transform(pRequest,.Spli
tR01,.contextString))
        $$$QuitOnError(..SendRequestAsync(..TargetConfigName,SplitR01,0))
    }
   Quit $$$OK
 }
}
```

Ensuite, dans la production, nous définissons une destination en utilisant la nouvelle option de paramètres que nous avons créée avec la propriété et le paramètre en tête de la classe, et nous verrons quelque chose comme ceci :

Conclusion

Comme je l'ai dit au début de cet article, j'ai toujours l'impression de développer une solution à ce type de problème, et ensuite je regarde en arrière et je pense que cela pourrait être mieux fait. Cette solution ne fait pas exception, mais elle est certainement meilleure que les itérations précédentes.

Pour améliorer cela à court terme, je vais ajouter des commentaires descriptifs à l'ObjectScript. A plus long terme, j'aimerais pouvoir ajouter un paramètre pour la classe de transformation afin qu'elle puisse être contrôlée depuis l'extérieur de l'ObjectScript.

De façon générale, je considère qu'il s'agit d'une approche que j'adopterai pour les prochains développements et que je ne partirai pas complètement de zéro.

(PS - Je suis heureux de partager le code pour cela, mais je ne peux que joindre un pdf à cet article. Cela semble un peu léger pour être quelque chose pour Open Exchange, mais s'il y a un intérêt pour moi de le télécharger là ou ailleurs, s'il vous plaît faites le moi savoir)

#DTL #HL7 #ObjectScript #Caché #Ensemble

dobjectscript-et-dtl