Article
Julian Matthews · Mar 17, 2023 7m read

Tutorial - Creating a HL7 TCP Operation for Granular Error Handling

Introduction

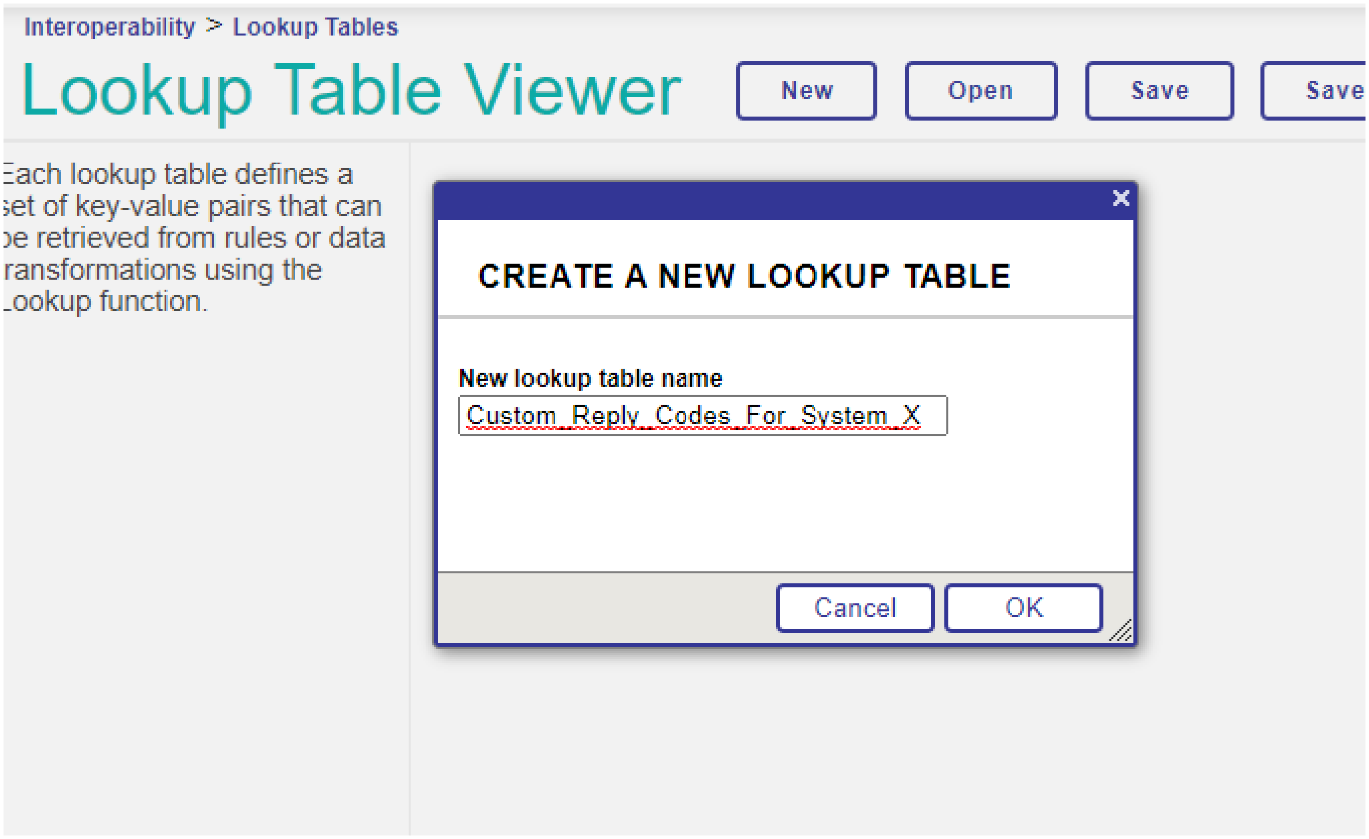
Say you have a receiving system that accepts HL7 and provides error messages in field ERR:3.9 in the ACK it returns. You require a different reply code action depending on the error message, however the Reply Code Actions settings for the operation do not provide this level of granularity. One option could be to create a process that takes the ACK and then completes the action you were expecting, however things can get a bit messy if the action is to retry the message, especially when trying to view a message trace.

My approach went a different direction, and makes use of an extension of EnsLib.HL7.Operation.TCPOperation and a lookup table to contain a portion of the error messages alongside the required Reply Code Action for each error.

Step 1: Creating the Lookup Table

For the list of Errors and the Reply Codes you require, we need to use the built in Lookup Table functionality from the Management Portal. This allows us to keep the configuration accessible to users without the need for running SQL queries to update a table or editing the Operation Class.

Add a new lookup with a name that makes sense for the system this is being used for:

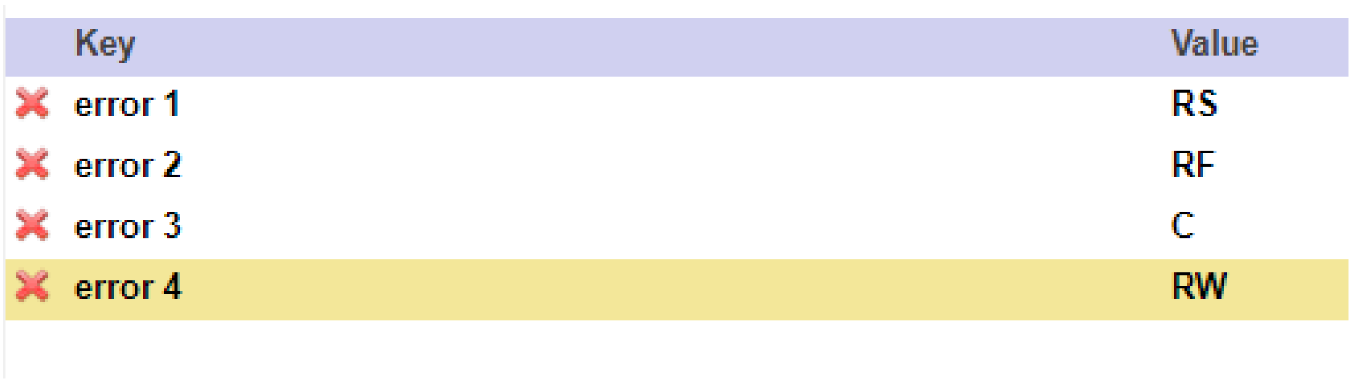


Each lookup table defines a set of key-value pairs that can be retrieved from rules or data transformations using the Lookup function.

CREATE A NEW LOOKUP TABLE

New lookup table name
Custom_Reply_Codes_For_System_X

Add the errors you're interested in as the Key, and the required Reply Code as the Value and then save your changes:



	Key	Value
✖	error 1	RS
✖	error 2	RF
✖	error 3	C
✖	error 4	RW

Step 2: Creating the Custom Operation Class

The class we are creating will need to extend EnsLib.HL7.Operation.TCPOperation, so first set this up:

```
Class
 Demo.Operations.SystemX.CustomHL7TCPO
peration Extends EnsLib.HL7.Operation.TCPOperation
{

}
```

The method we're interest in replacing is OnGetReplyAction which exists in EnsLib.HL7.Operation.ReplyStandard, and this class itself extended by EnsLib.HL7.Operation.TCPOperation:

```
Class
 Demo.Operations.SystemX.CustomHL7TCPO
peration Extends EnsLib.HL7.Operation.TCPOperation
{
    Method OnGetReplyAction(pRequest As
 EnsLib.HL7.Message, Output pResponse As
 EnsLib.HL7.Message, ByRef pSC As %Status) As %String
    {
        //Do Stuff Here!
    }
}
```

Now, we need a way of getting the error message from the ACK returned in the segment ERR:3.9. For this, we can loop through the segments of the ACK and then, if the segment is "ERR", set a variable to the value of segment 3.9 to be evaluated later. In addition, we should probably make sure we're only evaluating the response if it's an object, so lets wrap the code in an IF statement that checks it's an object:

```
Class
 Demo.Operations.SystemX.CustomHL7TCPO
```

```
peration Extends EnsLib.HL7.Operation.TCPOperation
{
    Method OnGetReplyAction(pRequest As
 EnsLib.HL7.Message, Output pResponse As
 EnsLib.HL7.Message, ByRef pSC As %Status) As %String
    {
        If $ISOBJECT(pResponse){
            Set (errorValue,isErr) = ""
            //loop over HL7
            Set SegCount = pResponse.SegCount
            For i=1:1:SegCount
            {
                Set segment = pResponse.GetSegmentAt(i)
                If (segment.Name="ERR")
                {
                    Set isErr = 1
                    Set errorValue=segment.GetValueAt("3.9")
                }
            }
        }
    }
}
```

We now have the value of ERR:3.9 in the variable "errorValue", and now we get to use Embedded SQL!

Embedded SQL? But didn't we put the error codes into a lookup table?

Behind the scenes, the Lookup Table available from the management portal is stored within a SQL Table called "Ens_Util.LookupTable". Using Embedded SQL to then query this as opposed to using a function that checks for an exact match means we have the power of SQL Operators to be more clever with our query. In our case, I want results where the error from the HL7 is *like* the value in the lookup table:

```
Class
 Demo.Operations.SystemX.CustomHL7TCPO
peration Extends EnsLib.HL7.Operation.TCPOperation
{
    Method OnGetReplyAction(pRequest As
 EnsLib.HL7.Message, Output pResponse As
 EnsLib.HL7.Message, ByRef pSC As %Status) As %String
    {
        If $ISOBJECT(pResponse){
            Set (errorValue,isErr) = ""
            //loop over HL7
            Set SegCount = pResponse.SegCount
            For i=1:1:SegCount
            {
                Set segment = pResponse.GetSegmentAt(i)
                If (segment.Name="ERR")
                {
                    Set isErr = 1
                    Set errorValue=segment.GetValueAt("3.9")
                }
            }
            If isErr{
                &SQL(
                    Select DataValue into :lookupResult
                    FROM Ens_Util.LookupTable
                    WHERE TableName = 'Custom_Reply_Codes_For_System_X' and
                    :errorValue like CONCAT('%',KeyName,'%')
                    )

//If the above query returned a result, quit with the value from the lookup table
                If (SQLCODE=0){
                    Quit lookupResult
                    }
            }
        }
    }
}
```

Quick Note on SQLCODE

The variable SQLCODE is born into existence by using Embedded SQL. If the query returns results, it has a value of 0, and no results returns a value of 100 (don't ask me why). A negative value indicates that the SQL query encountered an error. To get the error text from within your class, you could add in a trace with the following:

$$$TRACE("SQLCODE="$SYSTEM.SQL.Functions.SQLCODE(SQLCODE))

Alternatively, you could review the documentation for SQL Error Messages here.

Step 3: Draw the rest of the owl

I have to admit, at this point I cheated. I still wanted the functionality from the original method if we didn't get a match in our lookup table (such as the use of the reply code actions configured in the settings of the Operation when there's no match against the lookup table), so I added the content of the original method after the code I've written. This means that the operation continues to function as before if there's no match, or the message was successfully sent and there's no error:

```
Class
 Demo.Operations.SystemX.CustomHL7TCPO
peration Extends EnsLib.HL7.Operation.TCPOperation
{
    Method OnGetReplyAction(pRequest As
 EnsLib.HL7.Message, Output pResponse As
 EnsLib.HL7.Message, ByRef pSC As %Status) As %String
    {
        If $ISOBJECT(pResponse){
            Set (errorValue,isErr) = ""
            //loop over HL7
            Set SegCount = pResponse.SegCount
            For i=1:1:SegCount
            {
                Set segment = pResponse.GetSegmentAt(i)
                If (segment.Name="ERR")
                {
                    Set isErr = 1
                    Set errorValue=segment.GetValueAt("3.9")
                }
            }
            If isErr{
                &SQL(
                    Select DataValue into :lookupResult
                    FROM Ens_Util.LookupTable
                    WHERE TableName = 'Custom_Reply_Codes_For_System_X' and
                    :errorValue like CONCAT('%',KeyName,'%')
                    )

//If the above query returned a result, quit with the value from the lookup table
                If (SQLCODE=0){
                    Quit lookupResult
                    }
            }
        }

        Set (tCode,tAckCode,tAckCodeU,tFullAction,tText)=""
        Set nActions=$SELECT(""=$ZSTRIP(..ReplyCodeActions,"<>W"):0, 1:$LENGTH(
..ReplyCodeActions,","))
        /*Truncated for sanity*/
    }
}
```

There is a risk with doing this in that Intersystems could find that there's a glaring issue with the code in the original method, and rewrite it in the next release. Meanwhile, your operation is still using the old code blissfully unaware.

I would love to see a built in way of doing what I have attempted to achieve here, or a method that exists in EnsLib.HL7.Operation.ReplyStandard to be overwritten to allow for the actions I have taken in steps 1 and 2 but allowing the developer to then return a specific value that defers the processing to the original OnGetReplyAction. I have found myself in the past having a conversation similar to:

> Them: Hey, can you make it so that you only retry a message if the code is AE?
>
> Me: Sure, I just need to change a simple setting, and will take 30 seconds.
>
> Them: Awesome! Can you also make it so that you retry an AR, except when ERR:3.9 contains "Error with component xyz, please try again later"
>
> Me: That's not going to take 30 seconds...

Next Steps

Beyond my desired feature mentioned at the end of step 3, to improve the above you could make it so that the operation is agnostic to the lookup table name, and instead the details for the lookup table form part of the configuration of the Operation:

```
Class
 Demo.Operations.Generic.CustomTCPOperation Extends EnsLib.HL7.Operation.TCPOperation
{
    Property LookupTableName As %String;
    Parameter SETTINGS = "LookupTableName:Basic";
    Method OnGetReplyAction(pRequest As
 EnsLib.HL7.Message, Output pResponse As
 EnsLib.HL7.Message, ByRef pSC As %Status) As %String
```

```
    {
        /*Truncation*/
            If isErr{
                Set pTableName = ..LookupTableName
                &SQL(
                    Select DataValue into :lookupResult
                    FROM Ens_Util.LookupTable
                    WHERE TableName = :pTableName and
                    :errorValue like CONCAT('%',KeyName,'%')

                    )
        /*Truncation*/
```

This would give you a reusable Class instead of a custom one specific to a single receiving system.

#Business Operation #HL7 #SQL #Tutorial #Caché #Ensemble #InterSystems IRIS for Health