Article
Nathan Ng · Sep 6, 2017 4m read

Polling an External REST API with Ensemble

Preface

Before we begin, I'd like to mention that I am by no means an Ensemble expert, so take this with a grain of salt and please feel free to offer any suggestions for improvement. That being said, I have enjoyed working with Ensemble and wanted to share the approach I took to poll an external REST API for patient data in the hopes that it might help others with a similar goal.

If you get bored easily and just want to see some code, feel free to jump straight to the example on github. It uses dummy data (for obvious reasons) but the core principles should be the same.

The Goal

The goal of the project was to read a global containing a list of critical results and query an external REST API to determine whether they had been viewed. If they were, we could perform some logic on our side and remove them from our pending list, otherwise we would check again on our next poll interval.

Design Decisions

I wanted to gain as much reuse out of my operations as possible and stick to the Ensemble way of doing things as best I could, so I decided on the following:

1. A service would loop through our Global and pass an Ens.Request message to a BPL
2. The BPL would be in charge of handling all the business logic (and storing application state as context variables)
3. All REST calls would be made via an operation, with the Server, URL, Action (e.g. GET, POST, etc), and Content Type being exposed as configuration settings.
4. The REST Operation would take a custom Ens.Request message with fields for Form Variables and/or a JSON Payload.
5. The REST Operation would return a custom Ens.Response message with a field for the Server's JSON response string.

In order to accomplish items 3- 5 I ended up extending the stock EnsLib.REST.Operation class.

The Operation

The full code for this operation can be found on github, but the gist of it is as follows:

OnMessage() Accepts a REST.Request and returns a REST.Response to the BPL

```
 Method OnMessage(pRequest As DLS.REST.Request, pResponse As
DLS.REST.Response) As %Status
  {

// This hook lets the developer override how form variables are set if necessary
```

```
       Set formVariables = ..PopulateFormVariables(pRequest)

// This hook lets the developer override how the JSON payload is set if necessary
       Set payload = ..PopulatePayload(pRequest)

// Check the action set in the settings configuration and make appropriate request
       If ..Action = "POST" {
          Set status = ..Adapter.Post(.tHttpResponse, formVariables, payload)
       }
       ELSEIF ..Action = "GET" {
          Set status = ..Adapter.Get(.tHttpResponse, formVariables, payload)
       }
       // Etc for POST and PUT

       // Save response string to variable
       Set responseString = tHttpResponse.Data.Read(100000)

// This hook allows the developer to override how the server's response is handled if
 necessary
       Set pResponse = ..HandleResponse(responseString)
  }
```

The BPL

As stated earlier, the BPL is in charge of the business logic, so it will need to both construct the REST.Request message and handle the REST.Response message.

BPL Context:

Before we do anything else, we'll want to define the following context variables:

1. context.PayloadA - this will store the JSON payload that will be passed to our REST Operation
2. context.ServerResponseA - this will store the JSON response from our REST Operation

Constructing the request:

- Use a CODE block to store a JSON string to the BPL context

```
Set payloadObj = {
  "myProperty": "myValue,
  "mySecondProperty": "mySecondValue"
}

Set context.PayloadA = payloadObj.$toJSON() // Note: use %ToJSON for 2016.2 and above
```

- Use a CALL block to call the operation. Make sure to set the Request Message Class to REST.Request and set the callrequest.Payload equal to your context.PayloadA (the value set in the above CODE block).

Handling the response:

- In the above CALL block, make sure the Response Message Class is set to REST.Response and set the

context.ServerResponseA equal to your callresponse.ServerResponse.
- Use a CODE block to handle the server response that is now stored to your BPL context

```
?Set responseObj = ##class(%Object).$fromJSON(context.ServerResponseA)

// Do something with this object
```

Note: If you are performing multiple operations that depend on each other, make sure your CALLs are synchronous and in the correct order.

Discussion

So that's the approach I took to polling an external REST API. Hopefully you found it useful, but please feel free to let me know if you have any questions or suggestions on how it could be improved.

If you'd like to play around with an example, you can find an extract of the classes/settings here.

Thanks for reading! - Nathan

#Business Operation #Business Process (BPL) #REST API #Ensemble

Source URL:https://community.intersystems.com/post/polling-external-rest-api-ensemble