Using REST Services and Operations in Productions

Version 2024.2
2024-09-05

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Using REST Services and Operations in Productions
PDF generated on 2024-09-05
InterSystems IRIS® Version 2024.2

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

1

Creating REST Services in Productions

This page describes briefly how to create a Business Service that is also a REST service. This REST service will be able to receive REST requests and pass them to a Business Process or Business Operation elsewhere in the production.

The approach varies by your needs. If you want to:

- Parse and process the request in the production—use a subclass of %CSP.REST and call the **Ens.Director.CreateBusinessService()** method to instantiate the class as a business service. This service uses the Web port. For details on implementing a subclass of %CSP.REST, see Creating REST Services.
- Pass through a REST URL to an external server with minimal changes—use the pass-through REST service, EnsLib.REST.GenericService. For details on using the pass-through REST service, see the sections on pass-through business services in Configuring ESB Services and Operations and Pass-through Service and Operation Walkthrough.

For details on implementing a subclass of %CSP.REST, see Creating REST Services.

Note: InterSystems IRIS provides a built-in business service, EnsLib.REST.Service, that can be used instead of subclassing `%CSP.REST`. Because this business service works with the HTTP/REST inbound adapter, the dispatch methods receive additional arguments containing the input and output streams that the HTTP Adapter relies on. Classes receiving forwarded `<Map>` requests should extend `%CSP.REST` and not `EnsLib.REST.Service`.

2

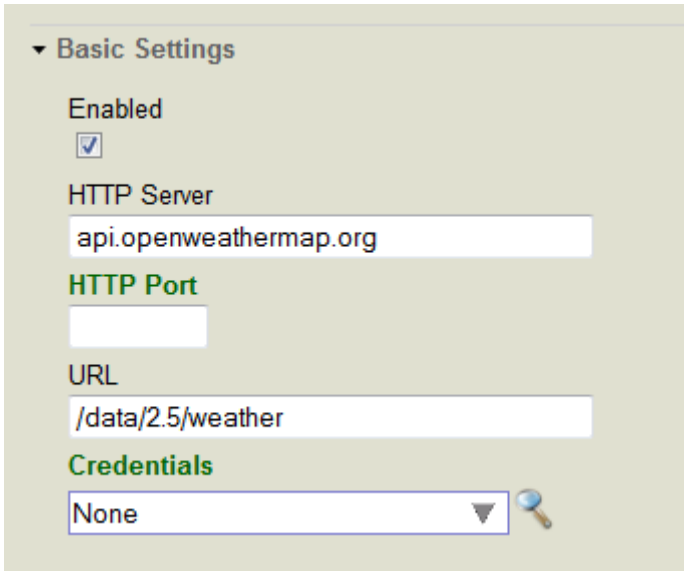
Creating REST Operations in Productions

This page describes briefly how to create a REST operation: a Business Operation that invokes an external REST service.

2.1 Basics

1. Define the REST operation class. Create a subclass of EnsLib.REST.Operation, which uses the InterSystems IRIS® outbound HTTP adapter, described in Using the HTTP Outbound Adapter.
2. Within your class, define the behavior of the business operation, as described generally in Defining Business Operations. Because you want this business operation to invoke an external REST service, invoke one or more of the methods of the HTTP adapter, depending on which HTTP operation you want to use:
   - **GetURL()**—uses the HTTP GET operation.
   - **PostURL()**—uses the HTTP POST operation.
   - **PutURL()**—uses the HTTP PUT operation.
   - **DeleteURL()**—uses the HTTP DELETE operation.
   - **SendFormDataArray()**—allows you to specify the HTTP operation as a parameter.

   Note that these operations all operate relative to the base URL that's specified by the production configuration.
3. Add the business operation to the production and configure it following the normal practice. Now you can specify the location of the external REST service. For example, to define a REST operation that calls a weather service, you could configure the operation as follows:



If you do not have a business process running, you can run and test this and other operations in **Interoperability** > **Configure** > **Production** page by selecting your operation and then selecting Test on the **Actions** tab.

2.2 Example

For example, the following extension of EnsLib.REST.Operation calls the weather REST service and provides a city name as a parameter:

Class Definition

```
Class Test.REST.WeatherOperation Extends EnsLib.REST.Operation
{

Parameter INVOCATION = "Queue";

Method getWeather(
   pRequest As Test.REST.WeatherRequest,
   Output pResponse As Test.REST.WeatherResponse) As %Status
{

   try {
      // Prepare and log the call
      // Append the city to the URL configured for adapter
      Set tURL=..Adapter.URL_"?q="_pRequest.City_"&units=imperial"

      // Execute the call
      Set tSC=..Adapter.GetURL(tURL,.tHttpResponse)

      // Return the response
     If $$$ISERR(tSC)&&$IsObject(tHttpResponse)&&$IsObject(tHttpResponse.Data)&&tHttpResponse.Data.Size
 {
         Set tSC=$$$ERROR($$$EnsErrGeneral,$$$StatusDisplayString(tSC)_":"_tHttpResponse.Data.Read())
      }
      Quit:$$$ISERR(tSC)
      If $IsObject(tHttpResponse) {
         // Instantiate the response object
         set pResponse = ##class(Test.REST.WeatherResponse).%New()
         // Convert JSON into a Proxy Object
         set tSC = ..JSONStreamToObject(tHttpResponse.Data, .tProxy)
         if (tSC){
            // Set response properties from the Proxy Object
            set pResponse.Temperature = tProxy.main.temp_"F"
            set pResponse.Humidity = tProxy.main.humidity_"%"
            set pResponse.MaxTemp = tProxy.main."temp_max"_"F"
            set pResponse.MinTemp = tProxy.main."temp_min"_"F"
            set pResponse.Pressure = tProxy.main.pressure_" mbar"
```

```
            set pResponse.WindSpeed = tProxy.wind.speed_" MPH"
            set pResponse.WindDirection = tProxy.wind.deg_" degrees"
            // Convert from POSIX time
            set pResponse.Sunrise = $ZT($PIECE($ZDTH(tProxy.sys.sunrise, -2),",",2),3)
            set pResponse.Sunset = $ZT($PIECE($ZDTH(tProxy.sys.sunset, -2),",",2),3)
          }
        }
   }catch{
       Set tSC=$$$SystemError
   }
   Quit tSC
}

XData MessageMap
{

<MapItems>
  <MapItem MessageType="Test.REST.WeatherRequest">
    <Method>getWeather</Method>
  </MapItem>
</MapItems>
}

}
```

The message sent to the operation specifies the city:

Class Definition

```
Class Test.REST.WeatherRequest Extends (%Persistent, Ens.Util.MessageBodyMethods)
{
   Property City As %String;
}
```

This operation calls the JSONStreamToObject() method and returns an InterSystems IRIS object that makes the elements of the JSON accessible. The message returned by this sample returns the following properties taken from the JSON stream:

Class Definition

```
Class Test.REST.WeatherResponse Extends (%Persistent, Ens.Util.MessageBodyMethods)
{
   Property Temperature As %String;
   Property MinTemp As %String;
   Property MaxTemp As %String;
   Property Pressure As %String;
   Property Humidity As %String;
   Property WindSpeed As %String;
   Property WindDirection As %String;
   Property Sunrise As %String;
   Property Sunset As %String;
 }
```

2.3 Variation: Posting JSON Data

If you need the REST operation to post JSON data, some adaptations are needed.

1. The HTTP adapter needs to specify the HTTP ContentType header appropriately. This means that you will need to create your own HTTP adapter class as follows:

Class Definition

```
Class My.REST.Client.HTTPOutboundAdapter Extends EnsLib.HTTP.OutboundAdapter
{
/// Send a POST to the configured Server, Port and URL, sending form data to the named form variables.
Method Post(Output pHttpResponse As %Net.HttpResponse, pFormVarNames As %String, pData...) As
%Status {
     quit ..SendFormDataArray(.pHttpResponse, "POST", ..GetRequest(), .pFormVarNames, .pData)
}

ClassMethod GetRequest() As %Net.HttpRequest
{
     set request = ##class(%Net.HttpRequest).%New()
     set request.ContentType = "application/json"
     quit request
}

}
```

2. Create a custom business operation class that uses your new adapter class.
3. Create a JSON-formatted string:
   a. Create an instance of %DynamicObject.
   b. Set properties of that object.
   c. Use the **%ToJSON()** method to serialize the object.

   For example:

ObjectScript

```
 //Use a %Library.DynamicObject to prepare the REST POST request
 Set tRequest = ##class(%DynamicObject).%New()
 Set tRequest.transactionid=pRequest.transactionid
 Set tRequest.participantid=pRequest.participantid
 Set tRequest.authstatus=pRequest.authstatus
 Set tRequest.reason=pRequest.reason
 set tPayload = tRequest.%ToJSON()
```

4. Post that string to the REST service by calling the **Post** method of the adapter. The following shows an example:

ObjectScript

```
 Set tSC=..Adapter.Post(.tHttpResponse, , tPayload)
```

Note that the second parameter is blank in this case.