Creating REST Services and Clients with Ensemble

Version 2018.1
2018-12-14

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

Creating REST Services and Clients with Ensemble
Ensemble Version 2018.1 2018-12-14

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

About This Book

This book describes how an Ensemble programmer can create the following:

- REST services
- REST operations (clients to external REST services)

You should be familiar with the REST architectural style and with the HTTP protocol.

This book contains the following sections:

- REST Overview
- Developing a REST Service
- Developing a REST Operation
- Walkthrough the REST Sample in ENSDEMO
- Configuring Ensemble for REST Services

REST pass-through services and operations, are described in "Configuring Pass-through Business Services" and "Configuring Pass-through Business Operations" in *Using Ensemble as an ESB*. Ensemble's pass-through services and operations enable you to receive an HTTP, REST, or SOAP request and pass it through to an external service.

For a detailed outline, see the table of contents.

For more information, try the following sources:

- *Ensemble Best Practices* describes best practices for organizing and developing Ensemble productions.
- *Developing Ensemble Productions* explains how to perform the development tasks related to creating an Ensemble production.
- *Configuring Ensemble Productions* describes how to configure the settings for Ensemble productions, business hosts, and adapters. It provides details on settings not discussed in this book.
- *Creating REST Services in Caché* describes the specific standards followed by InterSystems product support for REST.
- *Projecting Objects to XML* describes how to project Caché objects to XML and how to control that projection. (If you want to use an object as an argument for a web method, the class must be XML-enabled.)

For general information, see the *InterSystems Documentation Guide*.

1

REST Overview

Ensemble provides the capabilities to implement a REST service that can be invoked from a REST call and to implement a REST operation that is a client that can call an external REST service.

REST, which is named from “Representational State Transfer,” has the following attributes:

- REST is an architectural style rather than a clearly defined format. Although REST is frequently implemented using HTTP for transporting messages and JSON for passing data, you can also pass data as XML or plain text. REST makes use of existing web standards such as HTTP, URL, XML, and JSON.
- REST is resource oriented. Typically a resource is identified by a URL and uses operations based explicitly on HTTP methods, such as GET, POST, PUT, and DELETE.
- REST typically has a small overhead. Although it can use XML to describe data, it typically uses JSON which is a light-weight data wrapper. JSON identifies data with tags but the tags are not specified in a formal schema definition and do not have explicit data types.

2

Developing a REST Service

This section describes how to provide a REST service from an Ensemble Business Service and to pass the request to a Business Process or Business Operation. It lists the best practice for different requirements.

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 Ensemble's CSP port.
- Pass through a REST URL to an external server with minimal changes—use the pass-through REST service, EnsLib.REST.GenericService.

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

For an example of using the **Ens.Director.CreateBusinessService()** method, use Studio to examine the class Demo.ZenService.Zen.WeatherReportForm.cls. Although this is a Zen service rather than a REST service, you would use the **Ens.Director.CreateBusinessService()** method in the same way.

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" in *Using Ensemble as an ESB*.

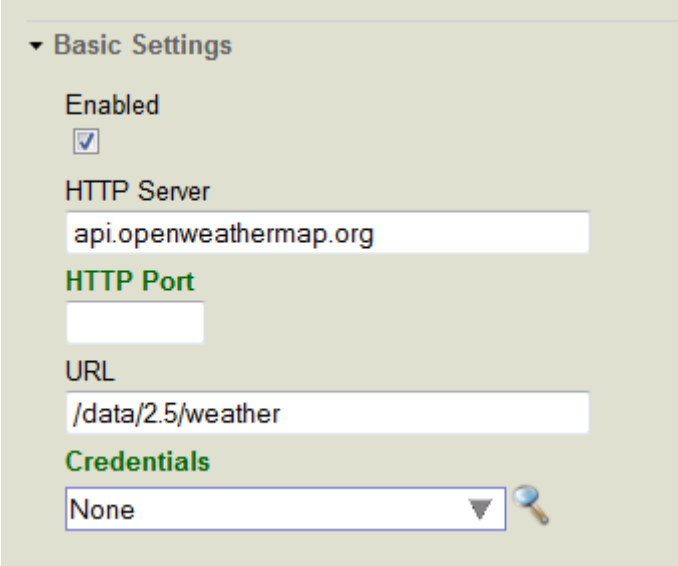
Note: Although Ensemble defines a class EnsLib.REST.Service, that is a subclass of %CSP.REST, we recommend that you not use this class because it provides an incomplete implementation of %CSP.REST. The only feature that EnsLib.REST.Service provides that is not available from %CSP.REST is the ability to use a special port, but we recommend against using a special port because it does not provide the robustness and security you get by using a commercial web server and the CSP port.

3

Developing a REST Operation

To develop a REST operation, you extend the class EnsLib.REST.Operation. The REST operation uses Ensemble's outbound HTTP adapter, which is described in Using the HTTP Outbound Adapter.

Using the HTTP Outbound Adapter, you specify the server, port, and address of the external web service in the HTTP adapter configuration. For example, to define a REST operation that calls a weather service, you could configure the operation as follows:



In your extension of EnsLib.REST.Operation, you specify the remaining parts of the URL by appending the value to the adapter URL property. Then you call one of the following adapter methods depending on what 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.

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

```
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:$$$ISERR(tSC)
      If $IsObject(tHttpResponse) {
         // Instantiate the response object
         set pResponse = ##class(Test.REST.WeatherResponse).%New()
         // Convert JSON into a Proxy Cache Object
         set tSC = ..JSONStreamToObject(tHttpResponse.Data, .tProxy)
         if (tSC){
            // Set response properties from the Proxy Object
            set pResponse.Temperature = tProxy.main.temp_"F"
            set pResponse.Humidity = tProxy.main.humidity_"%"
            set pResponse.MaxTemp = tProxy.main."temp_max"_"F"
            set pResponse.MinTemp = tProxy.main."temp_min"_"F"
            set pResponse.Pressure = tProxy.main.pressure_" mbar"
            set pResponse.WindSpeed = tProxy.wind.speed_" MPH"
            set pResponse.WindDirection = tProxy.wind.deg_" degrees"
            // Convert from POSIX time
            set pResponse.Sunrise = $ZT($PIECE($ZDTH(tProxy.sys.sunrise, -2),",",2),3)
            set pResponse.Sunset = $ZT($PIECE($ZDTH(tProxy.sys.sunset, -2),",",2),3)
          }
       }
   }catch{
       Set tSC=$$$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 Test.REST.WeatherRequest Extends (%Persistent, Ens.Util.MessageBodyMethods)
{
   Property City As %String;
}
```

This operation calls the **JSONStreamToObject()** method and returns a Caché 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 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;
 }
```

If you don't have a business process running you can run and test this and other operations in **[Ensemble > Production Configuration]** by selecting your operation, navigating to **Actions** in the **Production Settings** menu and clicking **Test**.

4

Walkthrough the REST Samples in ENSDEMO

The Demo.REST.DirectoryService implements a REST service that provides access to a Caché database provided in the SAMPLES namespace.

In order to call a REST service using the CSP port, you must first define a web application to handle the class of the service. For example, to call the Demo.REST.MathService or the Demo.REST.DirectoryService in ENSDEMO, you could define a web application named /RestServices in the ENSDEMO namespace. You must allow access to the ENSDEMO database and the SAMPLES database. See "Configuring Ensemble for REST Services" for a walkthrough of configuring Ensemble and defining a web application.

If you are running Demo.Rest.Production, then you can enter following REST call in a web browser. This sends an HTTP GET request that queries for the people in the database whose name begins with "j":

```
http://localhost:57772/RestServices/Demo.REST.DirectoryService/directory/person/name/j*
```

The browser sends this URL as an HTTP Get command. The URLMap sends this request to the retrievePerson method with the following parameters:

```
retrievePerson("person","name","j*")
```

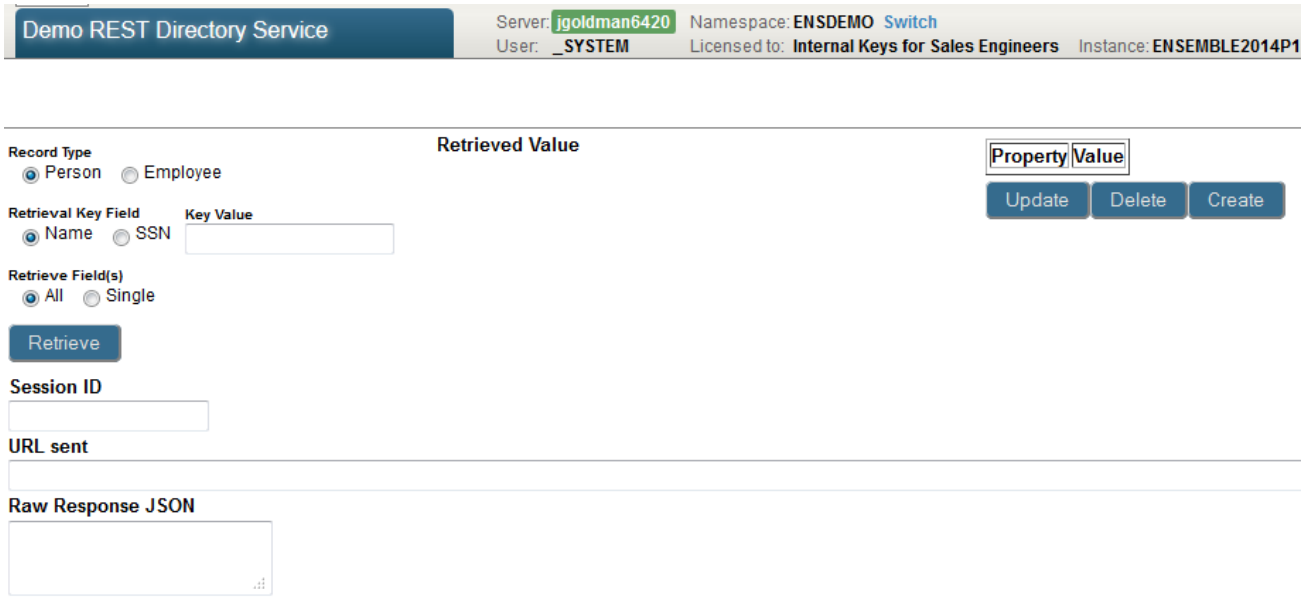
This method builds an SQL query statement that requests the information from the SAMPLES database and executes the query. It then calls the **ObjectToJSONStream()** method and returns the following JSON to the REST caller:

```
[{
    "Age":38,
    "Company":"InterTron Holdings Inc.",
    "DOB":"1975-06-05",
    "FavoriteColors":,
    "Home": {
        "City":"Islip",
        "State":"WY",
        "Street":"1199 Maple Place",
        "Zip":38998
    },
    "ID":179,
    "Name":"Jung,Jules G.",
    "Office": {
        "City":"Youngstown",
        "State":"OH",
        "Street":"803 Madison Place",
        "Zip":70937
    },
    "SSN":"584-94-9346",
    "Spouse":"Underman,Diane Z."
},{
    "Age":24,
    "Company":"MacroNet Associates",
    "DOB":"1989-06-05",
    "FavoriteColors":["Black","Purple"
```

```
    ],
    "Home": {
        "City":"Jackson",
        "State":"VT",
        "Street":"9371 Elm Drive",
        "Zip":93057
    },
    "ID":186,
    "Name":"Jung,Sam B.",
    "Office": {
        "City":"Gansevoort",
        "State":"SD",
        "Street":"6284 First Place",
        "Zip":84332
    },
    "SSN":"173-48-2772",
    "Spouse":"Fripp,Hannah D."
}]
```


Although you can create an HTTP GET command by just entering a URL in a browser, it is harder to generate an HTTP POST, PUT, or DELETE. You can use a utility such as curl to generate these HTTP commands or you can use the DirectoryPage and DirectoryOperation provided in the ENSDEMO REST sample. Since the DirectoryOperation calls the REST service from the Ensemble DirectoryService, it is not an ideal example showing how to call an external REST service, but it is a very useful tool to demonstrate calling the DirectoryService REST services.

To use the DirectoryPage and Directory Operation, in Studio select the ENSDEMO namespace and open the class Demo.REST.DirectoryPage. Select **View Web Page** and Studio displays the following web page in your default browser:



If you enter `j*` as the Key Value and click **Retrieve**, the web page and DirectoryOperation sends the same HTTP GET command that you entered in the browser. The web page displays the URL sent to the DirectoryService and the JSON returned by it. The page displays a table with two columns from the returned data. If you click on one of the rows, the page displays all of the properties and values for that person. You can then alter some of the values and click **Update**, which generates an HTTP POST command. If you click **Delete**, it deletes the record with the specified ID value with an HTTP DELETE command. Finally if you update the SSN field with a unique value and click **Create**, it creates a new record with an HTTP PUT command.

The DirectoryPage web page displays the JSON value returned by DirectoryService but does not display any JSON values sent to the service. One way to view these values is to use a TCP trace utility. Simply modify the DirectoryOperation configuration page to specify another port, such as 9989 and then use the TCP trace utility to transfer the message from port 9989 to 9988.

Remember, if you use this page to update the data, you are changing the data in the Caché SAMPLES database.

It is also possible to not use URLMap and implement the lower-level **OnProcessInput()** method. The following shows to process the call in this method:

```
Class Demo.REST.DirectoryService Extends EnsLib.REST.Service
{

Parameter ADAPTER = "EnsLib.HTTP.InboundAdapter";

/// The EnsServicePrefix parameter identifies the URL prefix that this component handles
Parameter EnsServicePrefix = "/directory";

Method OnProcessInput(
    pInput As %Library.AbstractStream,              /// Contains the HTTP command
    Output pOutput As %Stream.Object) As %Status    /// Output to return to REST call
  {
   /// Get the HTTP operation: GET, POST, PUT, or DELETE
   Set tCmd=$ZConvert(pInput.Attributes("HttpRequest"),"U")

   /// Get the URL
   Set tURL=$ZConvert(pInput.Attributes("URL"),"I","URL")

   /// Get the 2nd part of the URL which contains the service,
   /// Test that it matches EnsServicePrefix, "/directory"
   Set tService="/"_$ZConvert($Piece(tURL,"/",2),"L")  Quit:..#EnsServicePrefix'=tService

   /// Get the 3rd part of the URL either person or employee
   Set tType=$ZConvert($Piece(tURL,"/",3),"L")

   /// Get the 4th part of the URL in this case name
   Set tKeyIn=$Piece(tURL,"/",4), tKey=$ZConvert(tKeyIn,"L")

   /// Get the 5th part of the URL, the value to query for
   Set tKeyVal=$Replace($ZConvert($Piece(tURL,"/",5),"I","URL"),"'","''")
```

A

Configuring Ensemble for REST Services

This appendix briefly discusses how to configure your system so that you can use REST services through the Ensemble CSP port. This information is intended to help you set up a development or test system for these services. Complete information about these topics is provided in the Caché documentation. See "Configuring Caché" in the Caché System Administration Guide for more details.

To set up an Ensemble development or test system for REST services, follow these steps:

1. If you have installed Ensemble in a locked down installation, Studio access is disabled. Open the Management Portal and enable Studio access:
   a. Start the Management Portal from the Ensemble cube. You will have to use your Windows login username rather than _system to access the portal. Enter the password that you specified during installation.
   b. Select **System Administration**, **Security**, and **Services** to get to the Services portal page.
   c. The `%Services_Bindings` service is disabled by default. Select the service name and check the **Service Enabled** checkbox and save the setting.
2. If you are not using an existing Ensemble namespace, create a new namespace:
   a. Select **System Administration**, **Configuration**, **System Configuration**, and **Namespaces** to get to the Namespaces portal page.
   b. Click the **Create New Namespace** button, specify a name for the namespace, such as SERVICESNS.
   c. Click the **Create New Database** button for the globals database.
   d. In the Database Wizard, enter a name for the globals database, such as SERVICES_GDB. The wizard uses the name to create a directory for the database.
   e. Click the **Next** button twice to get to the Database Resource form. Select the **Create a new resource** radio button. The wizard displays a Create New Resource form. Accept the suggested name, such as %DB_SERVICES_GDB and ensure that Public Permissions Read and Write checkboxes are *not* checked. Click the **Save** button on the Database Resource form and the **Finish** button on the Database Wizard form.
   f. Repeat steps c through e for the routines database.
   g. Click the **Save** button to complete creating the namespace.
   h. Click **Close** to close the log.
3. Create an empty role and assign it to the unknown user:
   a. Select **System Administration**, **Security**, and **Roles** to display the Roles portal page.
   b. Click the **Create New Role** button and name the role, for example, Services_Role, and click the **Save** button.

c. Select the **Members** tab, select the Unknown User, click the right arrow, and click the **Assign** button.

4. Define a web application that will handle calls to the Ensemble CSP port. The web application name defines the root of the URL that will call the service. A single web application can support multiple business services but they must all have a class that is the same or a subclass of the web application dispatch class.

   a. Select **System Administration**, **Security**, **Applications**, and **Web Applications** to display the **Web Applications** portal page. Click the **Create New Web Application** button.

   b. Name the web application, such as `/weatherapp` or `/math/sum`. You must start the name with a / (slash) character.

   c. Set the **Namespace** to the namespace that the production is running in, such as SERVICESNS. Leave the Namespace Default Application unchecked.

   d. You can check the Application, CSP/ZEN, and Inbound Web Services checkboxes.

   e. Leave the Resource Required and Group By ID fields empty.

   f. Check the Unauthenticated checkbox on the **Allowed Authentication Methods** line.

   g. Set the Dispatch Class to the component class, such as EnsLib.REST.Service.

   h. Click Save.

   i. Select the Matching Roles tab.

   j. In the **Select a Matching Role:** field, select the role that you created in the previous step.

   k. In the **Select target roles to add to the selected matching role** field, select the role or roles associated with the namespace globals and routines. The globals and routines may be in the same database or in separate databases. If your service, accesses another Caché database, you should also select its role. For example, if you are defining a web application for the Demo.REST.DirectoryService class in ENSDEMO, you must also select the %DB_SAMPLES role. You can select multiple roles while holding the Ctrl key.

      Note: The globals database also may have a secondary database and a corresponding role, such as %DB_GDBSECONDARY. This secondary database is used to store passwords. You don't need access to this database for pass-through services and operations, but if you create a custom web service that uses password access, you should also add the secondary database role to the target database.

   l. After the roles are highlighted, click the right-arrow key to move them to the **Selected** text box.

   m. Then click the **Assign** button.

This completes the system configuration.