First Look: Developing REST Interfaces in InterSystems Products

Version 2018.1
2018-11-30

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

First Look: Developing REST Interfaces in InterSystems Products
InterSystems IRIS Data Platform Version 2018.1 2018-11-30

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

First Look: Developing REST Interfaces in InterSystems Products

This *First Look* helps you develop REST interfaces. You can use these REST interfaces with UI tools, such as Angular, to provide access to databases and interoperability productions. You can also use them to enable external systems to access InterSystems IRIS Data Platform™ applications.

1 Why Provide a REST Interface

If you need to access the data in an InterSystems IRIS™ database from an external system or if you want to provide a user interface for that data, you can do it by defining a REST interface. REST — or REpresentational State Transfer — is a way to retrieve, add, update, or delete data from another system using an exposed URL. REST is based on HTTP and uses the HTTP verbs POST, GET, PUT, and DELETE to map to the common Create, Read, Update, and Delete (CRUD) functions of database applications. You can also use other HTTP verbs, such as HEAD, PATCH, and OPTIONS, with REST.

REST is one of many ways to share data between applications, so you may not always need to set up a REST service if you choose to communicate directly using another protocol, such as TCP, SOAP, or FTP. But using REST has the following advantages:

- REST typically has a small overhead. 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. REST is typically much simpler to use than SOAP, which uses XML and has more overhead.
- By defining an interface in REST, it is easy to minimize the dependencies between the client and database server. This allows you to update your user interface without impacting your database implementation. You can also update the database implementation without impacting the user interface or any external applications that access the REST API.

2 How to Define REST Interfaces in InterSystems IRIS

Before defining REST interfaces, you should understand how a REST call flows through InterSystems IRIS. First consider the parts of a REST call such as:

```
GET http://localhost:52773/rest/coffeemakerapp/coffeemakers
```

This consists of the following parts:

- GET — this is the http verb.
- http: — this specifies the communication protocol.
- //localhost:52773 — this specifies the server and port number.
- /rest/coffeemakerapp/coffeemakers — this is the main part of the URL that identifies the resource that the REST call is directed to. In the following discussion, URL refers to this part of the REST call.

Note: Although this *First Look* uses the web server installed with InterSystems IRIS (using port 52773), you should replace it with a commercial web server for any code that you deploy. The web server installed with InterSystems IRIS is intended only for temporary use in developing code and does not have the robust features of a commercial web server.

When a client application makes a REST call:

1. InterSystems IRIS directs it to the web application that corresponds to the URL. For example, a URL starting with /coffeemakerapp would be sent to the application handling coffee makers and a URL starting with /api/docdb would be sent to the web application handling the Document Data Model.
2. The web application directs the call to a method based on the HTTP verb and any part of the URL after the section that identifies the web application. It does this by comparing the verb and URL against a structure called the URLMap.
3. The method uses the URL to identify the resource that the REST call is specifying and performs an action based on the verb. For example, if the verb is GET, the method returns some information about the resource; if the verb is POST, the method creates a new instance of the resource; and if the verb is DELETE, the method deletes the specified resource. For POST and PUT verbs, there is typically a data package, which provides more information.
4. The method performs the requested action and returns a response message to the client application.

Defining a REST interface requires:

- Defining the web application — you typically do this using the Management Portal. The section "Trying to Define a REST Interface for Yourself" describes how to do this.
- Creating a subclass of %CSP.REST and defining the UrlMap.
- Coding the methods that handle the REST call.

This *First Look* uses a sample application, coffeemakerapp, that accesses a database of coffee makers. Each record describes a coffee maker, including information such as its name, brand, and price. The coffeemakerapp provides REST interfaces to get information about the coffee makers, create a new record in the database, update an existing record, or delete a record. In a later section, this *First Look* tells you how to get the sample code from https://github.com/intersystems/FirstLook-REST.

2.1 Creating a Subclass of %CSP.REST and Defining the URLMap

Here is the first part of the demo.CoffeeMakerRESTServer class definition. It extends the %CSP.REST class.

```
Class Demo.CoffeeMakerRESTServer Extends %CSP.REST
 {
  Parameter HandleCorsRequest = 1

  XData UrlMap [ XMLNamespace = "http://www.intersystems.com/urlmap" ]
  {
    <Routes>
      <Route Url="/test" Method="GET" Call="test"/>
      <Route Url="/coffeemakers" Method="GET" Call="GetAll" />
      <Route Url="/coffeemaker/:id" Method="GET" Call="GetCoffeeMakerInfo" />
      <Route Url="/newcoffeemaker" Method="POST" Call="NewMaker" />
      <Route Url="/coffeemaker/:id" Method="PUT" Call="EditMaker" />
      <Route Url="/coffeemaker/:id" Method="DELETE" Call="RemoveCoffeemaker"/>
   </Routes>
  }
```

Look at the Route elements. Each has three properties:

- Url — this identifies the REST URL that can be handled by this Route. Since IRIS directs URLs starting with /rest/coffeemakerapp, this property specifies the part of the URL immediately after that. If the Url property is /coffeemakers, this Route handles URLs starting with /rest/coffeemakerapp/coffeemakers.

- Method — this identifies the verb that the Route handles. Note that the last two lines have the same value for Url, /coffeemaker/:id. The Route with the PUT method will handle PUT verbs with a URL starting with /rest/coffeemaker-app/coffeemaker/:id and the Route with the DELETE method will handle DELETE verbs with the same starting URL.
- Call — specifies the method to call to process this REST call. The method is passed the complete URL and any data so it can base its response on the URL.

The part of the Url value that starts with a : represents a wildcard. That is /coffeemaker/:id will match /coffeemaker/5, /coffeemaker/200, and even /coffeemaker/XYZ. The called method will get passed the value of :id in a parameter. In this case, it identifies the ID of the coffee maker to update (with PUT) or delete.

The Url value has additional options that allow you to forward the REST URL to another instance of a %CSP.REST subclass, but you don't need to deal with that in this *First Look*. The HandleCorsRequest parameter specifies whether browsers should allow Cross-origin Resource Sharing (CORS), which is when a script running in one domain attempts to access a REST service running in another domain, but that is also an advanced topic.

2.2 Coding the Methods

The GetAll method retrieves information about all coffee makers. Here is its code:

```
ClassMethod GetAll() As %Status
{
  Set tArr = []
  Set rs = ##class(%SQL.Statement).%ExecDirect(,"SELECT * FROM demo.coffeemaker")
  While rs.%Next() {
    Do tArr.%Push({
      "img":              (rs.%Get("Img")),
      "coffeemakerID":    (rs.%Get("CoffeemakerID")),
      "name":             (rs.%Get("Name")),
      "brand":            (rs.%Get("Brand")),
      "color":            (rs.%Get("Color")),
      "numcups":          (rs.%Get("NumCups")),
      "price":            (rs.%Get("Price"))
    })
  }

  Write tArr.%ToJSON()
  Quit $$$OK
}
```

Points to note about this method:

- There are no parameters. Whenever this method is called it executes an SQL statement that selects all records from the demo.coffeemaker database.
- For each record in the database, it appends the values to an array as name, value pairs.
- It converts the array to JSON and returns the JSON to the calling application by writing the JSON out to stdout.
- Finally, it quits with a success.

The NewMaker() method has no parameters, but has a JSON payload that specifies the coffee maker to create. Here is its code:

```
ClassMethod NewMaker() As %Status
{
  If '..GetJSONFromRequest(.obj) {
    Set %response.Status = ..#HTTP400BADREQUEST
    Set error = {"errormessage": "JSON not found"}
    Write error.%ToJSON()
    Quit $$$OK
  }

  If '..ValidateJSON(obj,.error) {
    Set %response.Status = ..#HTTP400BADREQUEST
    Write error.%ToJSON()
    Quit $$$OK
  }
```

```
  Set cm = ##class(demo.coffeemaker).%New()
  Do ..CopyToCoffeemakerFromJSON(.cm,obj)

  Set sc = cm.%Save()

  Set result={}
  do result.%Set("Status",$s($$$ISERR(sc):$system.Status.GetOneErrorText(sc),1:"OK"))
  write result.%ToJSON()
  Quit sc
}
```

Points to note about this method:

- First it tests if the payload contains a valid JSON object by calling the GetJSONFromRequest() and ValidateJSON() methods defined later in the class.
- Then it uses the JSON object to create a new demo.coffeemaker and then saves the record in the database.
- It returns the status by writing it to stdout.

Finally, the RemoveCoffeemaker() method shows how the :id part of the Url is passed to the method as a parameter:

```
ClassMethod RemoveCoffeemaker(id As %String) As %Status
{
  Set result={}
  Set sc=0

  if id'="",##class(demo.coffeemaker).%ExistsId(id) {
    Set sc=##class(demo.coffeemaker).%DeleteId(id)
    do result.%Set("Status",$s($$$ISERR(sc):$system.Status.GetOneErrorText(sc),1:"OK"))
  }
  else  {
    do result.%Set("Status","")
  }

  write result.%ToJSON()

    quit sc
}
```

In summary, the methods specified by the Route Call property handles the REST call by:

- Getting any :parameter as a call argument.
- Accessing the payload through obj value.
- Returning the response to the client application by writing it to stdout.

3 Trying to Define a REST Interface for Yourself

This section shows you step-by-step how to use the coffee maker application to handle REST calls. It starts with getting your system ready, including downloading the sample code from github, and takes you through the steps required to build the code and create the environment in the Management Portal.

3.1 Getting Your System Ready

Before creating this example, you should do the following:

- Install a running, licensed instance of InterSystems IRIS.
- Install a REST API application such as Postman, Chrome Advanced REST Client, or cURL. You will use it to generate test REST HTTP commands to show that your REST interfaces are working..
- Clone or download the FirstLook-REST sample code from github: https://github.com/intersystems/FirstLook-REST. This sample demonstrates REST APIs for a coffee maker database. If you're not familiar with github:

1. Go to https://github.com/intersystems/FirstLook-REST in a web browser.
2. Select **Clone or download**.
3. Select Download README-REST-master.zip.
4. Extract the files from the zip archive.
5. Using a text editor, open the README.md file and follow the instructions in it. Details of creating a namespace and web applications are in the following section.

3.2 Build the Sample Code

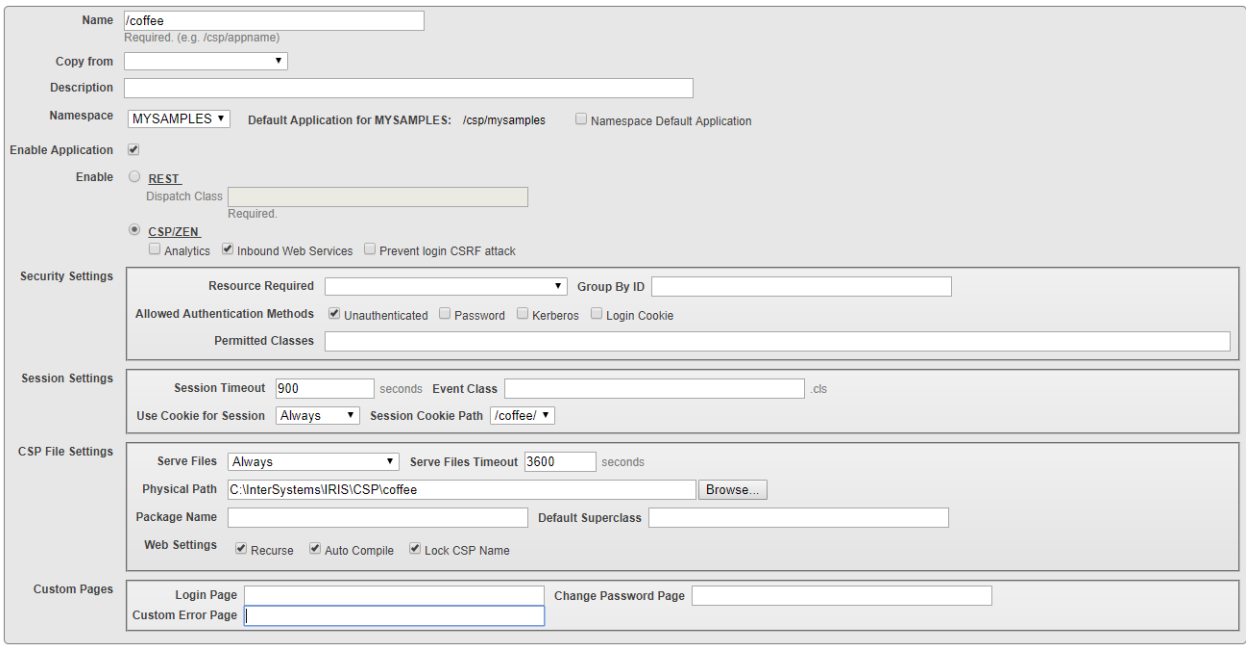
Build the sample code. Details are in the README.md provided in the github repository.

3.3 Defining Web Applications

In the Management Portal, select a namespace, such as a namespace, MYSAMPLES, that you have created. Then you create a web application to contain the application. This is not the web application that will process the REST requests.

1. Select **System Administration** > **Security** > **Applications** > **Web Applications**.
2. Select **Create New Web Application** and enter the following values:
   a. **Name**: `/coffee`
   b. **Namespace**: the name of a namespace
   c. Select **CSP/ZEN**
   d. **Web Application Physical Path** : *install-dir*`\CSP\coffee\`

   Your New Web Application form should be similar to:

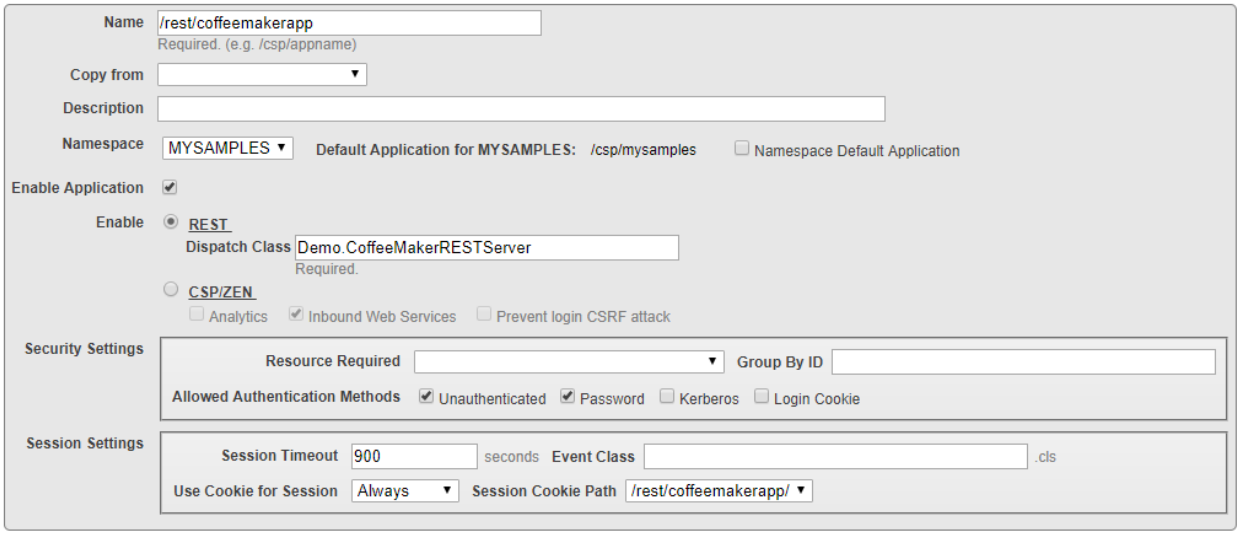


3. Select **Save**.

Now you create a second web application. This is the one that handles the REST calls.

1. Select **System Administration** > **Security** > **Applications** > **Web Applications**.
2. Select **Create New Web Application** and enter the following values:
   a. **Name**: `/rest/coffeemakerapp` — this specifies the URLs that will be handled by this web application. InterSystems IRIS will direct all URLs that begin with `/rest/coffeemakerapp` to this web application.
   b. **Namespace**: the name of the interoperability-enabled namespace
   c. Select **REST**.
   d. **Dispatch Class**: `Demo.CoffeeMakerRESTServer` — this is the class that defines the URLMap.
   e. **Allowed Authentication Methods**: select both **Unauthenticated** and **Password** check boxes.

   Your New Web Application form should be similar to:



3. Select **Save**.

3.4 Accessing the REST Interfaces

The CoffeeMaker REST application is now working. You will enter REST commands to access the coffee maker database. In your REST API tool, such as Postman, follow these steps:

1. Specify the following REST call to add a new coffee maker:
   - HTTP Action: POST
   - URL: http://*server*:*port*/rest/coffeemakerapp/newcoffeemaker
   - Username and password for your InterSystems IRIS account
   - Input data:

     ```
     {"img":"img/coffee3.png","coffeemakerID":"99","name":"Double Dip","brand":"Coffee+","color":"Blue","numcups":2,"price":71.73}
     ```

   Although the data contains a value for coffeemakerID, that is a calculated field and a new value is assigned when the record is addded. The call returns a success status:

   ```
   {"Status":"OK","Message":"New maker saved with ID 1"}
   ```

2. Repeat the previous step to add the following two coffee makers:

{"img":"img/coffee4.png","coffeemakerID":"99","name":"French Press","brand":"Coffee For You","color":"Blue","numcups":4,"price":50.00}
{"img":"img/coffee9.png","coffeemakerID":"99","name":"XPress","brand":"Shiney Appliances","color":"Green","numcups":1,"price":95.00}

3. Specify the following REST call to get a list of coffee makers in the database:
   - HTTP Action: GET
   - URL: http://*server*:*port*/rest/coffeemakerapp/coffeemakers
   - Username and password for your InterSystems IRIS account

   The call returns a list of coffeemakers, such as:

[{"img":"img/coffee3.png","coffeemakerID":"1","name":"Double Dip","brand":"Coffee+","color":"Blue","numcups":2,"price":71.73},
{"img":"img/coffee4.png","coffeemakerID":"2","name":"French Press","brand":"Coffee For You","color":"Blue","numcups":4,"price":50},
{"img":"img/coffee9.png","coffeemakerID":"3","name":"XPress","brand":"Shiney Appliances","color":"Green","numcups":1,"price":95}]

4. Specify the following REST call to delete the coffee maker with ID=2:
   - HTTP Action: DELETE
   - URL: http://*server*:*port*/rest/coffeemakerapp/coffeemaker/2
   - Username and password for your InterSystems IRIS account

   The call returns a success status:

{"Status":"OK"}

5. Specify the following REST call to get a list of coffee makers in the database after the deletion:
   - HTTP Action: GET
   - URL: http://*server*:*port*/rest/coffeemakerapp/coffeemakers
   - Username and password for your InterSystems IRIS account

   The call returns a list of coffeemakers, such as:

[{"img":"img/coffee3.png","coffeemakerID":"1","name":"Double Dip","brand":"Coffee+","color":"Blue","numcups":2,"price":71.73},
{"img":"img/coffee9.png","coffeemakerID":"3","name":"XPress","brand":"Shiney Appliances","color":"Green","numcups":1,"price":95}]

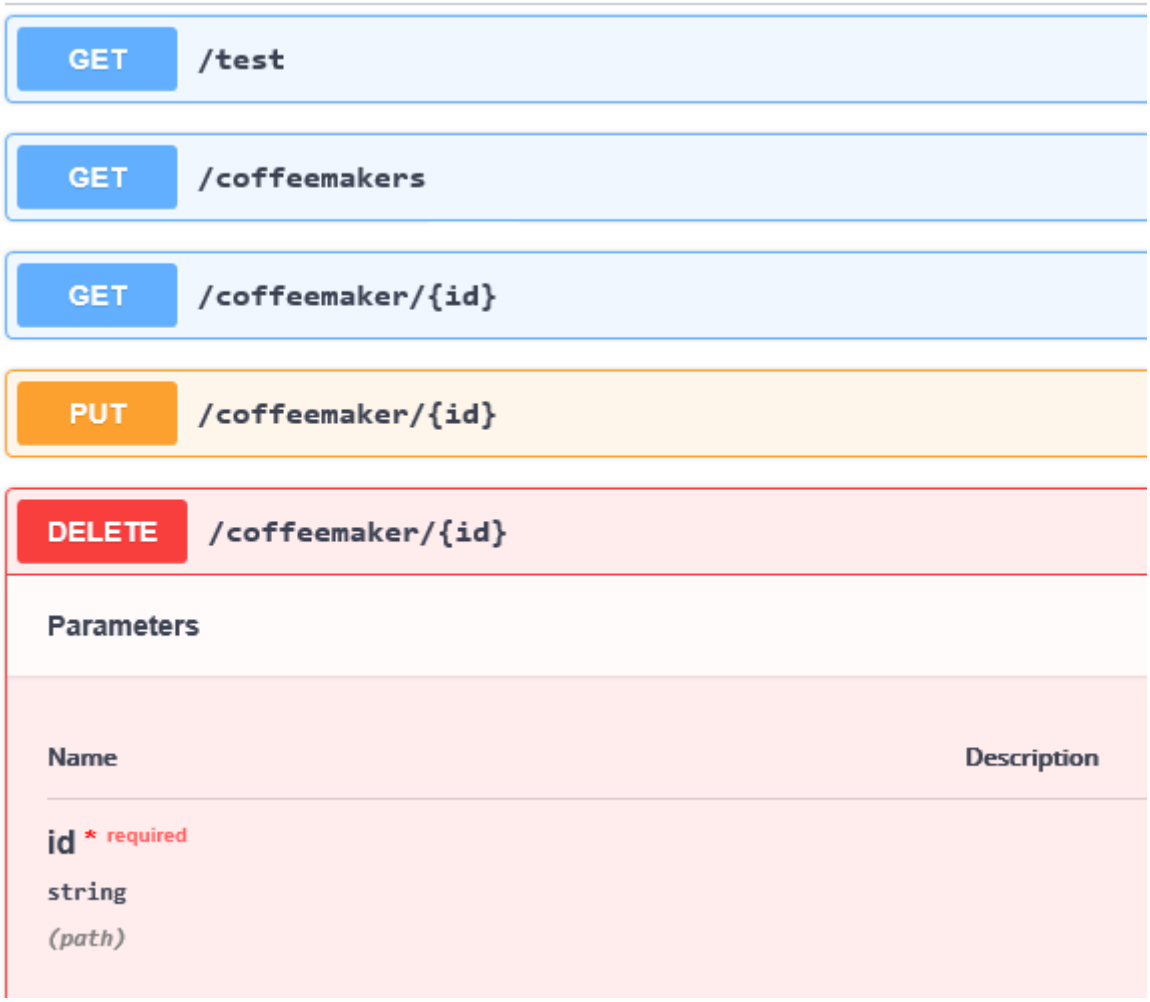
3.5 Documenting REST Interfaces

When you provide REST interfaces to developers you should provide documentation so that they know how to call the interfaces. You can use the Open API Spec to document REST interfaces and a tool, such as Swagger to edit and format the documentation. InterSystems is developing a feature to support this documentation. This release contains a feature in API Management that generates the document framework for your REST APIs. You still need to edit the generated documentation to add comments and additional information, such as content of arguments and HTTP return values.

To generate the documentation for the CoffeeMakerApp REST sample, enter the following REST call:

- HTTP Action: GET
- URL: http://*server*:*port*/api/mgmnt/v1/*namespace*/spec/rest/coffeemakerapp/
- Username and password for your InterSystems IRIS account

You can paste the output from this call into the swagger editor. It converts the JSON to YAML (Yet Another Markup Language) and displays the doc. You can use the swagger editor to add more information to the documentation. The swagger editor displays the documentation as shown in the following:



4 For More Information on InterSystems IRIS and REST

See the following for more information on creating REST services in InterSystems IRIS:

- Setting Up RESTful Services is an InterSystems online class that uses the same coffee maker application as this *First Look*, but goes into more detail. You need to be logged into learning.intersystems.com to take this course. If you don't have an account, you can create one.
- Creating REST Services
- Using REST Services and Operations in Productions