InterSystems
IRIS Data Platform

First Look: Developing REST Interfaces in InterSystems Products

Version 2019.4
2020-01-28

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

First Look: Developing REST Interfaces in InterSystems Products
InterSystems IRIS Data Platform Version 2019.4 2020-01-28
Copyright © 2020 InterSystems Corporation
All rights reserved.

Table of Contents

First Look: Developing REST Interfaces in InterSystems Products

This First Look shows you how to 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.

To browse all of the First Looks, including those that can be performed on a free evaluation instance of InterSystems IRIS, see InterSystems First Looks.

1 Why Provide a REST Interface

If you need access to the data in an InterSystems IRIS database from an external system or 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 REST Calls to 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 of the InterSystems IRIS instance hosting the REST interface.

- /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 (in this case on port 52773 on the local system), 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.

3 How to Define REST Interfaces in InterSystems IRIS

There are two ways to define REST interfaces in InterSystems IRIS:

- Define an OpenAPI 2.0 specification and then use the API Management tools to generate the code for the REST interface.
- Manually code the REST interface, then define a web application in the Management Portal.

This *First Look* shows how to manually code the REST interface, including developing a dispatch class. If you prefer to use the specification-first method, these dispatch classes are generated automatically and should not be edited. Advantages of using the specification-first definition include the ability to automatically generate documentation and client code from the specification, but you can use either way to define REST interfaces. For more information about defining REST interfaces using a specification, see Creating Rest Services.

4 Manually Coding REST Interfaces

Manually coding REST interfaces includes the following steps:

- Creating a subclass of %CSP.REST and defining the UrlMap.
- Coding the methods that handle the REST call.
- Defining the web application — you typically do this using the Management Portal.

This *First Look* uses a sample application, coffeemakerapp, that accesses a database of coffee makers to demonstrate how to manually code REST interfaces. 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. The following sections explore the class definition and some methods of this sample application with annotations to enhance your understanding of the code. You will download the code for the entire application from GitHub when completing the exercise section of this First Look.

4.1 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/coffeemakerapp/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.

4.2 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.

5 Defining a REST Interface for Yourself

This section shows you step-by-step how to use the coffee maker application to handle REST calls. Rather than having you write the code for the REST interfaces, you will download the completed application from GitHub. After building the application, you will define the web application and then test the application by making REST calls.

5.1 Before you Begin

To use the procedure, you will need a system to work on, with an installed REST API application such as Postman, Chrome Advanced REST Client, or cURL installed, and a running InterSystems IRIS instance to connect to. Your choices for InterSystems IRIS include several types of licensed and free evaluation instances; the instance need not be hosted by the system you are working on (although they must have network access to each other). For information on how to deploy each type of instance if you do not already have one to work with, see Deploying InterSystems IRIS in *InterSystems IRIS Basics: Connecting an IDE*. For the information needed to connect your REST API application to your InterSystems IRIS instance, see InterSystems IRIS Connection Information in the same document.

5.2 Downloading the Sample Application

Start the exercise by cloning or downloading the completed coffeemakerapp application, which includes all of the REST interfaces needed to test REST calls to InterSystems IRIS. This FirstLook-REST sample code is available at: https://github.com/intersystems/FirstLook-REST. The contents downloaded from GitHub must be accessible from your InterSystems IRIS instance.

The procedure for downloading the files depends on the type of instance you are using, as follows:

- If you are using an ICM-deployed instance:
  1. Use the **icm ssh** command with the **-machine** and **-interactive** options to open your default shell on the node hosting the instance, for example:

     ```
     icm ssh -machine MYIRIS-AM-TEST-0004 -interactive
     ```

2. On the Linux command line, use one of the following commands to clone the repo to the data storage volume for the instance. For a configuration deployed on Azure, for example, the default mount point for the data volume is /dev/sdd, so you would use commands like the following:

   ```
   $ git clone https://github.com/intersystems/FirstLook-REST /dev/sdd/FirstLook-REST
   OR
   $ wget -qO- https://github.com/intersystems/FirstLook-REST/archive/master.tar.gz | tar xvz -C
   /dev/sdd
   ```

   The files are now available to InterSystems IRIS in /irissys/data/FirstLook-REST on the container's file system.

- If you are using a containerized instance (licensed or Community Edition) that you deployed by other means:

  1. Open a Linux command line on the host. (If you are using Community Edition on a cloud node, connect to the node using SSH, as described in *Getting Started with InterSystems IRIS Community Edition*.)
  2. On the Linux command line, use either the **git clone** or the **wget** command, as described above, to clone the repo to a storage location that is mounted as a volume in the container.
     - For a Community Edition instance, you can clone to the instance's durable %SYS directory (where instance-specific configuration data is stored). On the Linux file system, this directory is /opt/ISC/dur. This makes the files available to InterSystems IRIS in /ISC/dur/FirstLook-REST on the container's file system.
     - For a licensed containerized instance, choose any storage location that is mounted as a volume in the container (including the durable %SYS directory if you use it). For example, if your **docker run** command included the option **-v /home/user1:/external**, and you clone the repo to /home/user1, the files are available to InterSystems IRIS in /external/FirstLook-REST on the container's file system.

- If you are using an InterSystems Learning Labs instance:

  1. Open the command-line terminal in the integrated IDE.
  2. Change directories to /home/project/shared and use the **git clone** command to clone the repo:

     ```
     $ git clone https://github.com/intersystems/FirstLook-REST
     ```

  The folder is added to the Explorer panel on the left under **Shared**, and the directory is available to InterSystems IRIS in /home/project/shared.

- If you are using an installed instance:
  - If the instance's host is a Windows system with GitHub Desktop and GitHub Large File Storage installed:
    1. Go to https://github.com/intersystems/FirstLook-REST in a web browser on the host.
    2. Select **Clone or download** and then choose **Open in Desktop**.

    The files are available to InterSystems IRIS in your GitHub directory, for example in C:\Users\User1\Documents\GitHub\FirstLook-REST.
  - If the host is a Linux system, simply use the **git clone** command or the **wget** command on the Linux command line to clone the repo to the location of your choice.

5.3 Building the Sample Code

To build the sample code, follow this procedure

1. Open the InterSystems IRIS Terminal, using the procedure described for your instance in *InterSystems IRIS Basics: Connecting an IDE*.

2. Enter the following commands, replacing `<path>` with the full path of the buildsample/Build.RESTSample.cls file that you downloaded:

   ```
   do $system.OBJ.Load("<path>/Build.RESTSample.cls","ck")
   
   do ##class(Build.RESTSample).Build()
   ```

3. When prompted, enter the full path of the directory to which you downloaded this sample. The method then loads and compiles the code and performs other needed setup steps.

Note: The FirstLook-REST / README.md file contains a version of these instructions.

5.4 Defining a Web Application

Now that you have built the sample application with the REST interfaces, you need to define a web application:

1. Open the Management Portal for your instance in your browser, using the URL described for your instance in *InterSystems IRIS Basics: Connecting an IDE*.
2. Create an interoperability-enabled namespace by navigating to the Namespaces page (**System Administration** > **Configuration** > **System Configuration** > **Namespaces**) and clicking the **Create New Namespace** button following the instructions for using the New Namespace page in Create/Modify a Namespace in the "Configuring InterSystems IRIS" chapter of the *System Administration Guide*.
3. Select **System Administration** > **Security** > **Applications** > **Web Applications**.
4. Select **Create New Web Application** and enter the following settings
   - **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.
   - **Namespace**: The name of the interoperability-enabled namespace you created.
   - **Enable**: Select **REST**.
   - **Dispatch Class**: `Demo.CoffeeMakerRESTServer` — This is the class that defines the URLMap.
   - **Security Settings/Allowed Authentication Methods**: Select both the **Unauthenticated** and **Password** check boxes.
5. Select **Save**.

5.5 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 a REST POST request to add a new coffee maker, using the information specific to your InterSystems IRIS instance where needed
   - HTTP Action: POST
   - URL: http://*server*:*port*/rest/coffeemakerapp/newcoffeemaker, where *server* and *port* are the host identifier and web server port for your instance.
   - Login credentials for your instance.
   - 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 added. 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":"Shiny Appliances", \
"color":"Green","numcups":1,"price":95.00}
```

3. Specify a REST GET request, using the same instance-specific information, to get a list of coffee makers in the database:

   - HTTP Action: GET
   - URL: http://*server*:*port*/rest/coffeemakerapp/coffeemakers
   - Login credentials for your instance.

   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":"shiny 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
   - Login credentials for your instance.

   The call returns a success status:

```
{"Status":"OK"}
```

5. Repeat the REST GET request. 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":"Shiny Appliances", \
"color":"Green","numcups":1,"price":95}]
```

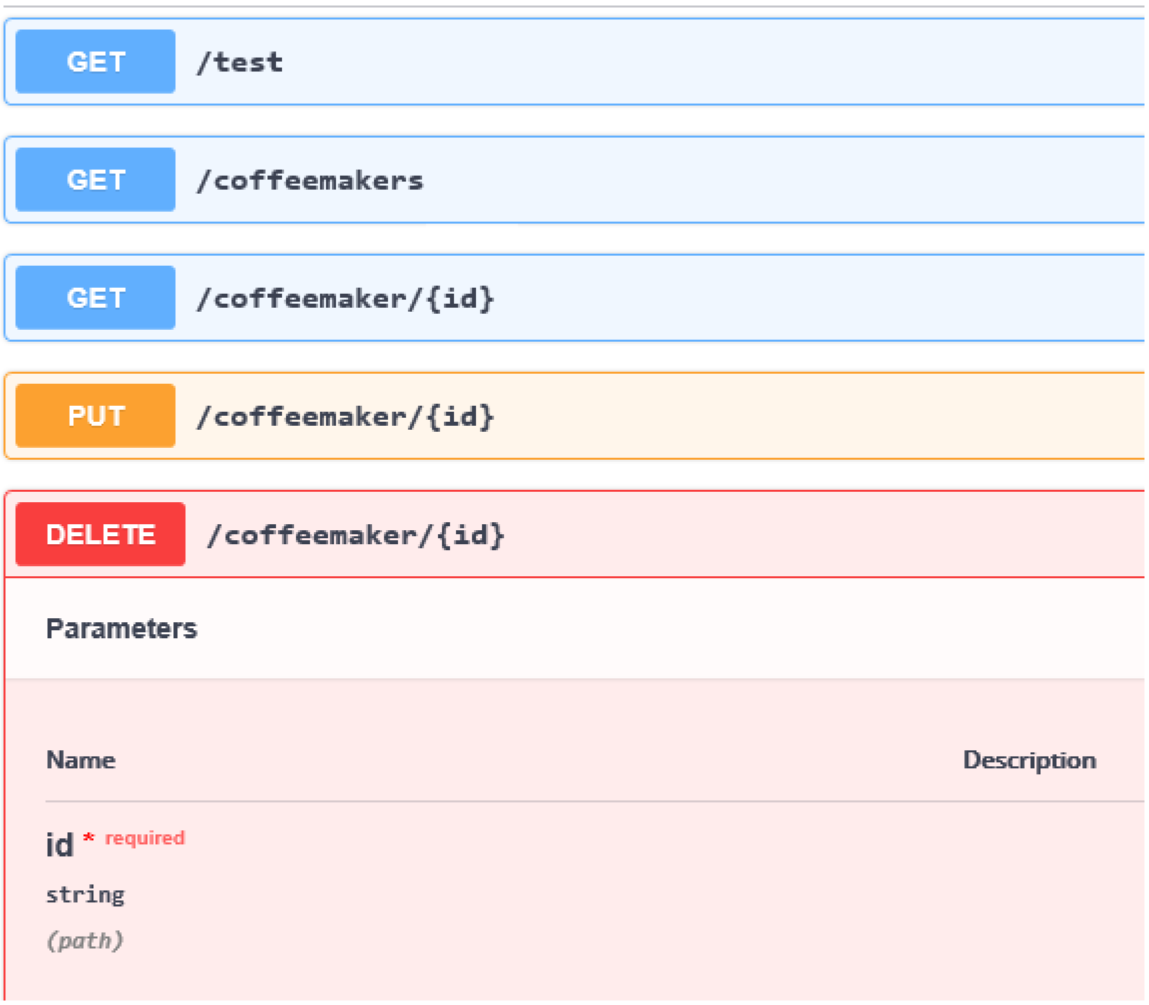
5.6 Documenting REST Interfaces

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, using the information specific to your InterSystems IRIS instance and the name of the namespace you created:

- HTTP Action: GET
- URL: http://*server*:*port*/api/mgmnt/v1/*namespace*/spec/rest/coffeemakerapp/
- Login credentials for your InterSystems IRIS instance.

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:


GET /test

GET /coffeemakers

GET /coffeemaker/{id}

PUT /coffeemaker/{id}

DELETE /coffeemaker/{id}

Parameters

Name	Description
id * required string (path)	

6 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