Article
Iryna Mykhailova · Feb 22, 2021 6m read

Open Exchange

Transferring Files via REST to Store in a Property, Part 1

A question came up in the InterSystems developer community concerning the possibility of creating a TWAIN interface to a Caché application. There were several great suggestions on how to get data from an imaging device on a web client to a server, then store this data in a database.

However, in order to implement any of these suggestions, you need to be able to transfer data from a web client to a database server and store the received data in a class property (or a table cell, as was the case in the question). This technique can be useful not only for transferring imaging data received from a TWAIN device, but also for other tasks such as organizing a file archive, an image share, and so forth.

Thus, the main objective of this article is to show how to write a RESTful service to get data from the body of an HTTP POST command either in a raw state or wrapped in a JSON structure.

REST Basics

Before getting to the specifics, let ’ s start with a few words about REST in general and how RESTful services are created in IRIS.

Representational state transfer (REST) is an architectural style for distributed hypermedia systems. The key abstraction of information in REST is a resource which has its proper identifier and can be represented in a JSON, XML, or other format known to both server and client.

Usually, HTTP is used to transfer data between client and server. Each CRUD (create, read, update, delete) operation has its own HTTP method (POST, GET, PUT, DELETE), while each resource or collection of resources has its own URI.

In this article, I ’ ll be using only the POST method to insert a new value into the database, so I need to know its constraints.

POST doesn ’ t have any limit on the size of data stored in its body according to the IETF RFC7231 4.3.3 Post specification. But different web servers and browsers impose their own limits, typically from 1MB to 2GB. For example, Apache allows a maximum of 2GB. In any case, if you need to send a 2GB file, you might want to rethink your approach.

Requirements for a RESTful Service in IRS

In IRIS, in order to implement a RESTful service, you have to:

- Create a class broker that extends the abstract class %CSP.REST. This, in turn, extends %CSP.Page, and this makes it possible to access different useful methods, parameters,

and objects, in particular %request).
- Specify UrlMap to define routes.
- Optionally set the UseSession parameter to specify whether each REST call is executed in its own web session or shares a single session with other REST calls.
- Provide class methods to perform operations defined in routes.
- Define the CSP web application and specify its security on the web application page (System Administration > Security > Applications > Web Applications), where the Dispatch class should hold the name of the user class and Name, the first part of the URL for the REST call.

In general, there are several ways to send big chunks of data (files) and their metadata from client to server, such as:

- Base64-encode the file and the metadata and add processing overhead to both the server and the client for encoding/decoding.
- Send the file first and return an ID to the client, which then sends the metadata with the ID. The server reassociates the file and the metadata.
- Send the metadata first and return an ID to the client, which then sends the file with the ID, and the server reassociates the file and the metadata.

In this first article, I ’ m going to basically take the second approach, but without returning ID to the client and adding the metadata (the filename to store as another property) because there ’ s nothing exceptional about this task. In a second article, I ’ ll use the first approach, but I ’ ll package my file and metadata (the filename) in a JSON structure before sending it to the server.

Implementing the RESTFul Service

Now let ’ s get to the specifics. First, let ’ s define the class, properties of which we ’ ll be setting

```
Class RestTransfer.FileDesc Extends %Persistent
{
  Property File As %Stream.GlobalBinary;
  Property Name As %String;
}
```

Of course, you ’ ll usually have more metadata to go with the file, but this should suffice for our purposes.

Next, we need to create a class broker, which we ’ ll expand later with routes and methods:

```
Class RestTransfer.Broker Extends %CSP.REST
{
  XData UrlMap
  {
    <Routes>
    </Routes>
  }
}
```

And lastly, for the preliminary setup we need to specify this application in a list of web applications:

Now that the preliminary setup is done, we can write methods to save a file received from a REST client (I ’ ll use the Advanced REST Client) into a database as a File property of an instance of class RestTransfer.FileDesc.

We ’ ll start by working with information stored as raw data in the body of POST method. First, let ’ s add a new route to UrlMap:

```
<Route Url="/file" Method="POST" Call="InsertFileContents"/>
```

This route specifies that when the service receives a POST command with URL /RestTransfer/file, it should call the InsertFileContents class method. To make things easier, inside this method I ’ ll create a new instance of a class RestTransfer.FileDesc and set its File property to the received data. This returns the status and a JSON-formatted message indicating either success or error. Here ’ s the class method:

```
ClassMethod InsertFileContents() As %Status
{
  Set result={}
  Set st=0
  set f = ##class(RestTransfer.FileDesc).%New()
  if (f = $$$NULLOREF) {
    do result.%Set("Message","Couldn't create an instance of the class")
  } else {
    set st = f.File.CopyFrom(%request.Content)
    If $$$ISOK(st) {
      set st = f.%Save()
      If $$$ISOK(st) {
        do result.%Set("Status","OK")
      } else {
        do result.%Set("Message",$system.Status.GetOneErrorText(st))
      }
    } else {
      do result.%Set("Message",$system.Status.GetOneErrorText(st))
    }
  }
  write result.%ToJSON()
  Quit st
}
```

First, it creates a new instance of class RestTransfer.FileDesc and checks that it was created successfully and we have an OREF. If the object couldn ’ t be created, we form a JSON structure:

```
{"Message", "Couldn't create an instance of class"}
```

If the object was created, the contents of the request are copied into the property File. If copying was not successful, we form a JSON structure with a description of the error:

```
{"Message",$system.Status.GetOneErrorText(st)}
```

If the contents were copied, we save the object, and if the save is successful, we form a JSON {"Status","OK"}. If not, the JSON returns an error description.

Finally, we write the JSON to a response and return status st.

Here ’ s an example of transferring an image:

How it ’ s saved in the database:

We can save this stream to a file and see that it was transferred without change:

```
set f = ##class(RestTransfer.FileDesc).%OpenId(4)
set s = ##class(%Stream.FileBinary).%New()
set s.Filename = "D:\Downloads\test1.jpg"
do s.CopyFromAndSave(f.File)
```

The same can be done with a text file:

This will also be visible in globals:

And we can store executables or any other type of data:

We can check the size in globals, and it ’ s correct:

Now that this part works as it should, let ’ s take a look at the second approach — getting the contents of the file and its metadata (the filename) stored in JSON format and transferred in the body of a POST method.

You can read more about creating REST services in the InterSystems documentation.

The example code for the both approaches is on GitHub and InterSystems Open Exchange. If you have any questions or suggestions pertaining to either, please don ’ t hesitate to write them in the comments section.

#REST API #InterSystems IRIS
Check the related application on InterSystems Open Exchange

Source URL:https://community.intersystems.com/post/transferring-files-rest-store-property-part-1