

Artículo

[Ricardo Paiva](#) · 16 jul, 2021 · Lectura de 7 min

[Open Exchange](#)

## Cómo transferir archivos a través de REST para almacenar en una propiedad. Parte 1

Surgió una pregunta en la Comunidad de Desarrolladores de InterSystems sobre la posibilidad de [crear una interfaz TWAİN para una aplicación Caché](#). Hubo varias sugerencias excelentes sobre cómo obtener datos de un dispositivo de imágenes en un cliente web a un servidor y después almacenar estos datos en una base de datos.

Sin embargo, para implementar cualquiera de estas sugerencias, se debe poder transferir datos desde un cliente web a un servidor de base de datos y almacenar los datos recibidos en una propiedad de clase (o una celda de tabla, como fue el caso en la pregunta). Esta técnica puede ser útil no solo para transferir datos de imágenes recibidos desde un dispositivo TWAİN, sino también para otras tareas como organizar un archivo, compartir una imagen, etc.

Por lo tanto, el objetivo de este artículo es mostrar cómo escribir un servicio RESTful para obtener datos del cuerpo de un comando HTTP POST, ya sea en un estado sin procesar o envuelto en una estructura JSON.

## Conceptos básicos de REST

Antes de ir a los detalles, empezaremos hablando sobre REST en general y sobre cómo se crean los servicios RESTful en IRIS.

Una Transferencia de Estado Representacional (REST) es un estilo arquitectónico para sistemas hiper-media distribuidos. La abstracción clave de información en REST es un recurso que tiene su identificador adecuado y se puede representar en JSON, XML u otro formato conocido tanto por el servidor como por el cliente.

Por lo general, HTTP se usa para transferir datos entre el cliente y el servidor. Cada operación CRUD (crear, leer, actualizar, eliminar) tiene su propio método HTTP (POST, GET, PUT, DELETE), mientras que cada recurso o colección de recursos tiene su propio URI.

En este artículo, usaré solo el método POST para insertar un nuevo valor en la base de datos, por lo que necesito conocer sus restricciones.

POST no tiene ningún límite en el tamaño de los datos almacenados en su cuerpo de acuerdo con la especificación [IETF RFC7231 4.3.3 Post](#). Pero los distintos navegadores y servidores web imponen sus propios límites, normalmente de 1 MB a 2 GB. Por ejemplo, [Apache](#) permite un máximo de 2 GB. En cualquier caso, si necesitas enviar un archivo de 2 GB, quizá deberías reconsiderar tu enfoque.

## Requisitos para un servicio RESTful en IRIS

En IRIS, para implementar un servicio RESTful, debes:

- Crear un intermediario de clases que amplíe la clase abstracta `%CSP.REST`. Esto, a su vez, extiende `%CSP.Page`, y hace posible acceder a diferentes métodos, parámetros y objetos útiles, en particular `%request`).

- Especificar el UriMap para definir rutas.
- Opcionalmente, configurar el parámetro UseSession para especificar si cada llamada REST se ejecuta en su propia sesión web o comparte una sola sesión con otras llamadas REST.
- Proporcionar métodos de clase para realizar las operaciones definidas en rutas.
- Definir la aplicación web CSP y especificar su seguridad en la página de la aplicación web (Administración del sistema> Seguridad> Aplicaciones> Aplicaciones web), donde la clase Dispatch debe contener el nombre de la clase de usuario y el Nombre, la primera parte de la URL para la llamada REST.

En general, hay varias formas de enviar grandes cantidades de datos (archivos) y sus metadatos de cliente a servidor, como:

- Codificación en Base64 del archivo y los metadatos y añadir una sobrecarga de procesamiento tanto al servidor como al cliente para la codificación / decodificación.
- Primero enviar el archivo y devolver una identificación (ID) al cliente, que luego enviará los metadatos con la identificación. El servidor vuelve a asociar el archivo y los metadatos.
- Enviar primero los metadatos y devolver una identificación (ID) al cliente, que luego envía el archivo con la identificación, y el servidor vuelve a asociar el archivo y los metadatos.

En este primer artículo, básicamente tomaré el segundo enfoque, pero sin devolver la identificación (ID) al cliente y sin añadir los metadatos (el nombre del archivo para almacenar como otra propiedad) porque no hay nada excepcional en esta tarea. En un segundo artículo, usaré el primer enfoque, pero empaquetaré mi archivo y metadatos (el nombre del archivo) en una estructura JSON antes de enviarlo al servidor.

## Implementación del servicio RESTful

Ahora vayamos a los detalles. Primero, vamos a definir la clase, cuyas propiedades configuraremos:

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

Por supuesto, normalmente tendrás más metadatos para el archivo, pero esto debería ser suficiente para nuestros propósitos.

A continuación, necesitamos crear un intermediario de clase, que luego expandiremos con rutas y métodos:

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

Y, por último, para la configuración preliminar, necesitamos especificar esta aplicación en una lista de aplicaciones web:

Ahora que la configuración preliminar está hecha, podemos escribir métodos para guardar un archivo recibido de un cliente REST (usaré el [Advanced REST Client](#)) en una base de datos como una propiedad de archivo de una instancia de la clase RestTransfer.FileDesc.

Comenzaremos trabajando con información almacenada como datos sin procesar en el cuerpo del método POST. Primero, añadiremos una nueva ruta a UrlMap:

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

Esta ruta especifica que cuando el servicio recibe un comando POST con URL / RestTransfer / file, debe llamar al método de clase InsertFileContents. Para facilitar las cosas, dentro de este método crearé una nueva instancia de una clase RestTransfer.FileDesc y estableceré su propiedad File en los datos recibidos. Esto devuelve el estado y un mensaje con formato JSON indicando éxito o error. Aquí está el método de la clase:

```
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
}
```

Primero, crea una nueva instancia de la clase RestTransfer.FileDesc y comprueba que se creó correctamente y tenemos un OREF. Si el objeto no se pudo crear, formamos una estructura JSON:

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

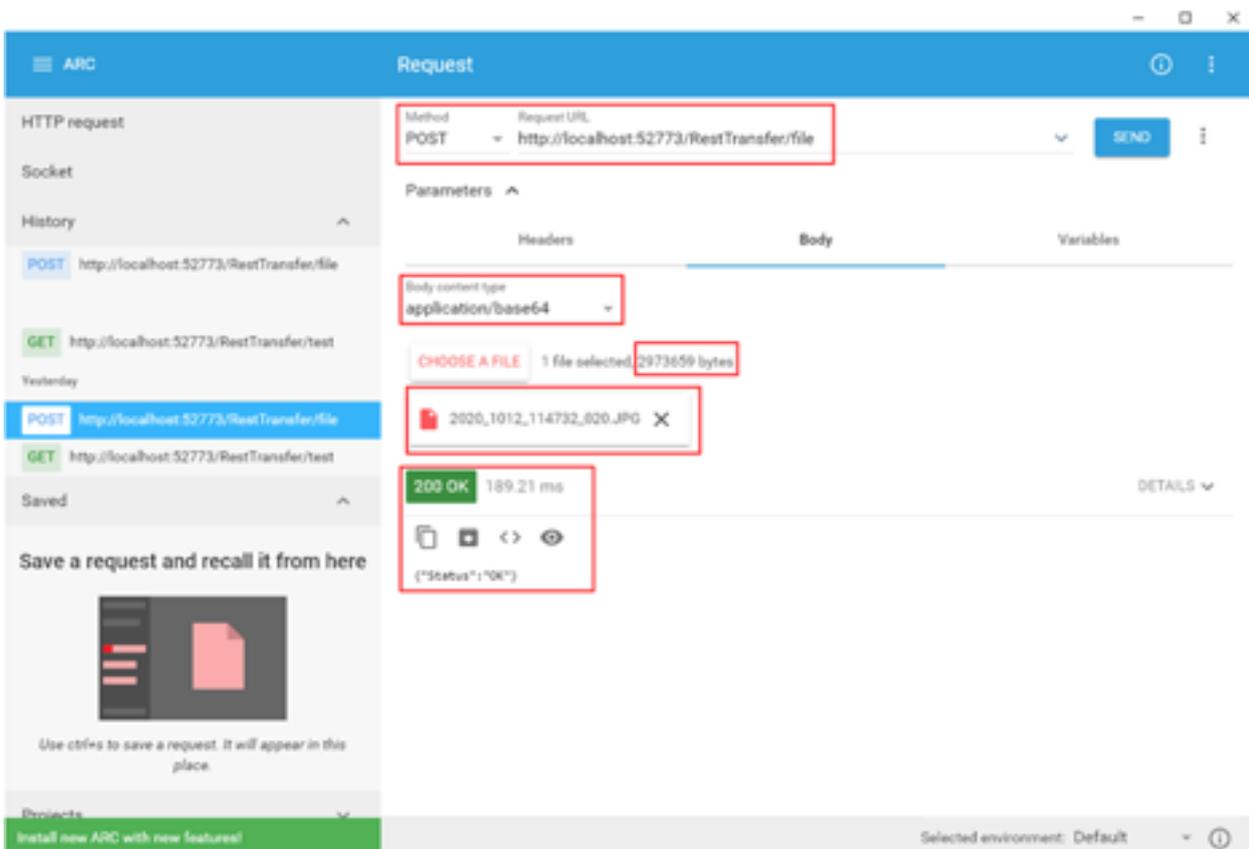
Si se creó el objeto, el contenido de la solicitud se copia en la propiedad Archivo. Si la copia no fue exitosa, formamos una estructura JSON con una descripción del error:

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

Si el contenido fue copiado, guardamos el objeto, y si se guarda de forma correcta, formamos un JSON {"Status", "OK"}. Si no, el JSON devuelve una descripción del error.

Finalmente, escribimos el JSON en una respuesta y devolvemos el estado st.

Este es un ejemplo de cómo transferir una imagen:



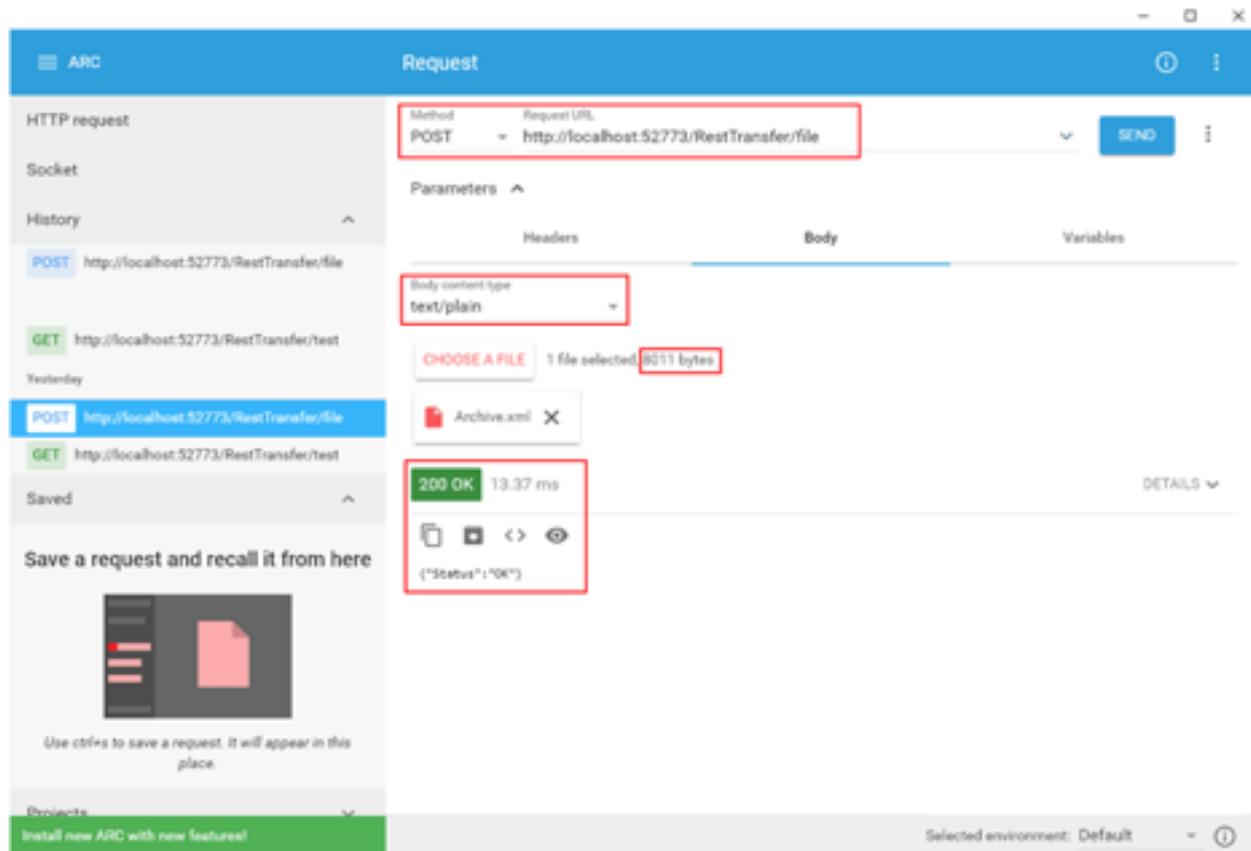
Cómo se guarda en la base de datos:

```
^RestTransfer.FileDescS(4) = 92
^RestTransfer.FileDescS(4,0) = 2973659
^RestTransfer.FileDescS(4,1) = "y0yá%8Exif"_$c(0,0)_"II*"_$c(
^RestTransfer.FileDescS(4,2) = "%±ü"_$c(144)_"-"_$c(127)_"ð¿"
```

Podemos guardar esta secuencia en un archivo y ver que se transfirió sin cambios:

```
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)
```

Lo mismo se puede hacer con un archivo de texto:



Esto también será visible en los globals:

```
383: ^RestTransfer.FileDescS(5) = 1
384: ^RestTransfer.FileDescS(5,0) = 8011
385: ^RestTransfer.FileDescS(5,1) = "<?xml version=""1.0"" encoding=""UTF-8"">"_$c(13,10)_"<Export generator=""Cache"" ver
```

Y podemos almacenar ejecutables o cualquier otro tipo de datos:

