

Article

[Iryna Mykhailova](#) · Juin 29, 2022 8m de lecture

[Open Exchange](#)

## Transmission de fichiers via REST pour les stocker en Property, partie 2

Dans le premier article de cette série, nous avons vu comment lire un "gros" volume de données dans le corps brut d'une méthode HTTP POST et l'enregistrer dans une base de données en tant que propriété de flux d'une classe. Voyons maintenant comment enregistrer de telles données et métadonnées au format JSON.

Malheureusement, Client REST avancé (Advanced REST Client) ne permet pas de composer des objets JSON avec des données binaires comme valeur d'une clé (ou peut-être que je n'ai simplement pas trouvé comment le faire), j'ai donc décidé d'écrire un client simple en JavaScript pour envoyer des données au serveur.

J'ai créé une nouvelle classe appelée RestTransfer.Client et je lui ai ajouté les paramètres Server = "localhost" and Port = 52773 pour décrire mon serveur web. Et j'ai créé une méthode de classe GetLink dans laquelle je crée une nouvelle instance de la classe %Net.HttpRequest et je configure ses propriétés avec les paramètres susmentionnés.

Pour envoyer réellement la requête POST à un serveur, j'ai créé une méthode de classe SendFileDirect, qui lit le fichier que je souhaite envoyer au serveur et écrit son contenu dans la propriété EntityBody de ma requête.

Ensuite, je lance la méthode Post("/RestTransfer/file") et, si elle se termine avec succès, la réponse se trouvera dans la propriété HttpResponse.

Pour voir le résultat renvoyé par le serveur, je lance la méthode OutputToDevice de la réponse.

Voici la méthode de classe :

```
Class RestTransfer.Client
{
  Parameter Server = "localhost";
  Parameter Port = 52773;
  Parameter Https = 0;

  ClassMethod GetLink() As %Net.HttpRequest
  {
    set request = ##class(%Net.HttpRequest).%New()
    set request.Server = ..#Server
    set request.Port = ..#Port
    set request.ContentType = "application/octet-stream"
    quit request
  }

  ClassMethod SendFileDirect(aFileName) As %Status
  {
    set sc = $$$OK
    set request = ..GetLink()
    set s = ##class(%Stream.FileBinary).%New()
    set s.FileName = aFileName
    While 's.AtEnd
```



## Présentation de JSON

JSON est un format léger de stockage et de transport de données, dans lequel les données se présentent sous la forme de paires nom/valeur séparées par des virgules. Dans ce cas, le JSON ressemblera à quelque chose comme ceci :

```
{
  "Name": "test.txt",
  "File": "Hello, world!"
}
```

IRIS dispose de plusieurs classes qui vous permettent de travailler avec JSON. Je vais utiliser les suivantes :

%JSON.Adaptor permet de sérialiser un objet compatible JSON en un document JSON et vice versa.  
%Library.DynamicObject permet d'assembler dynamiquement des données qui peuvent être facilement transmises entre un client Web et un serveur.

Pour en savoir plus sur ces classes et d'autres qui vous permettent de travailler avec JSON, consultez la section Référence des classes de la documentation et des manuels [Application Development Guides](#).

Le principal avantage de ces classes est de permettre de créer facilement du JSON à partir d'un objet IRIS ou de créer un objet à partir de JSON.

Je vais montrer deux approches de la façon dont vous pouvez travailler avec JSON pour envoyer/recevoir des fichiers, à savoir une qui utilise %Library.DynamicObject pour créer un objet du côté client et le sérialiser en un document JSON et %JSON.Adaptor pour le désérialiser en RestTransfer.FileDesc du côté serveur, et une autre où je forme manuellement une chaîne à envoyer et à analyser du côté serveur. Pour rendre cela plus lisible, je vais créer deux méthodes différentes sur le serveur pour chaque approche.

Pour l'instant, concentrons-nous sur la première méthode.

## Création de JSON avec IRIS

Pour commencer, héritons de la classe RestTransfer.FileDesc de %JSON.Adaptor. Cela nous permet d'utiliser la méthode d'instance %JSONImport() pour désérialiser un document JSON directement dans un objet. Maintenant, cette classe aura l'aspect suivant :

```
Class RestTransfer.FileDesc Extends (%Persistent, %JSON.Adaptor)
{
  Property File As %Stream.GlobalBinary;
  Property Name As %String;
}
```

Ajoutons un nouvel itinéraire à UriMap dans le service broker de la classe :

```
<Route Url="/jsons" Method="POST" Call="InsertJSONSmall"/>
```

Ceci spécifie que, lorsque le service reçoit une commande POST avec l'URL /RestTransfer/jsons, il doit appeler la méthode de classe InsertJSONSmall.

Cette méthode s'attend à recevoir un texte formaté en JSON avec deux paires clé-valeur où le contenu du fichier sera inférieur à la longueur maximale d'une chaîne. Je vais définir les propriétés de l'objet Name and File, le sauvegarder dans la base de données, et renvoyer l'état et un message formaté en JSON indiquant un succès ou une erreur.

Voici la méthode de classe.

```
ClassMethod InsertJSONSmall() As %Status
{
  Set result={}
  Set st=0

  set f = ##class(RestTransfer.FileDesc).%New()
  if (f = $$$NULLOREF) {
    do result.%Set("Message", "Couldn't create an instance of class")
  } else {
    set st = f.%JSONImport(%request.Content)
    If $$$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
}
```

Que fait cette méthode ? Elle récupère la propriété Content de l'objet %request et, à l'aide de la méthode %JSONImport héritée de la classe %JSON.Adaptor, la convertit en un objet RestTransfer.FileDesc .

S'il y a des problèmes pendant la conversion, nous formons un JSON avec la description de l'erreur :

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

Autrement, nous enregistrons l'objet. S'il est enregistré sans problème, nous créons un message JSON {"Status","OK"}. Sinon, un message JSON avec la description de l'erreur.

Enfin, nous écrivons le JSON dans une réponse et renvoyons l'état st.

Du côté client, j'ai ajouté une nouvelle méthode de classe SendFile pour envoyer des fichiers au serveur. Le code est très similaire à celui de SendFileDirect mais, au lieu d'écrire le contenu du fichier directement dans la propriété EntityBody, je crée une nouvelle instance de la classe %Library.DynamicObject, je fixe sa propriété Name égale au nom du fichier, et je code et copie le contenu du fichier dans la propriété File.

Pour encoder le contenu du fichier, j'utilise la méthode Base64EncodeStream() proposée par [Vitaliy Serdtsev](#).

Ensuite, en utilisant la méthode %ToJSON de la classe %Library.DynamicObject, je sérialise mon objet en un document JSON, je l'écris dans le corps de la requête et je lance la méthode Post("/RestTransfer/jsons").

Voici la méthode de classe :

```
ClassMethod SendFile(aFileName) As %Status
{
  Set sc = $$$OK
  Set request = ..GetLink()
  set s = ##class(%Stream.FileBinary).%New()
  set s.Filename = aFileName
  set p = {}
  set p.Name = s.Filename
  set sc = ..Base64EncodeStream(s, .t)
  Quit:$System.Status.IsError(sc)

  While 't.AtEnd {
    set p.File = p.File_t.Read(.len, .sc)
    Quit:$System.Status.IsError(sc)
  }
  do p.%ToJSON(request.EntityBody)
  Quit:$System.Status.IsError(sc)

  set sc = request.Post("/RestTransfer/jsons")
  Quit:$System.Status.IsError(sc)

  Set response=request.HttpResponse
  do response.OutputToDevice()
  Quit sc
}
```

Nous lançons cette méthode et la méthode SendFileDirect pour transférer plusieurs petits fichiers :

```
do ##class(RestTransfer.Client).SendFile("D:\Downloads\Outline Template.pdf")
do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\Outline Template.pdf")

do ##class(RestTransfer.Client).SendFile("D:\Downloads\pic3.png")
do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\pic3.png")
do ##class(RestTransfer.Client).SendFile("D:\Downloads\Archive (1).xml")
do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\Archive (1).xml")
```

Nous obtiendrons les résultats suivants :

Comme vous pouvez le constater, les longueurs sont les mêmes et, si nous enregistrons ces fichiers sur un disque dur, nous verrons qu'ils sont toujours les mêmes.

## Formation manuelle de JSON

Maintenant, nous allons nous concentrer sur la deuxième approche, qui consiste à former le JSON manuellement.

Pour ce faire, ajoutons un nouveau itinéraire à UriMap dans le service broker de classe :

```
<Route Url="/json" Method="POST" Call="InsertJSON"/>
```

Cela spécifie que lorsque le service reçoit une commande POST avec l'URL /RestTransfer/json, celui-ci doit lancer la méthode de classe InsertJSON. Dans cette méthode, je m'attends à recevoir le même JSON, mais je n'impose aucune limite à la longueur du fichier. Voici la méthode de classe :

```
ClassMethod InsertJSON() As %Status
{
  Set result={}
  Set st=0

  set t = ##class(%Stream.TmpBinary).%New()
  While '%request.Content.AtEnd
  {
    set len = 32000
    set temp = %request.Content.Read(.len, .sc)
    set:len<32000 temp = $extract(temp,1,*-2)
    set st = t.Write($ZCONVERT(temp, "I", "RAW"))
  }
  do t.Rewind()

  set f = ##class(RestTransfer.FileDesc).%New()
  if (f = $$$NULLOREF)
  {
    do result.%Set("Message", "Couldn't create an instance of class")
  } else {
    set str = t.Read()
    set pos = $LOCATE(str,"","")
    set f.Name = $extract(str, 10, pos-1)
    do f.File.Write($extract(str, pos+11, *))
    While 't.AtEnd {
      do f.File.Write(t.Read(.len, .sc))
    }

    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
}
```

Comment cette méthode fonctionne-t-elle ? Tout d'abord, je crée une nouvelle instance d'un flux

temporaire %Stream.TmpBinary, et j'y copie le contenu de la requête.

Comme je vais l'utiliser comme une chaîne de caractères, je dois me débarrasser d'un guillemet de fin de chaîne (") et d'une accolade (}). Pour ce faire, dans le dernier segment du flux, je laisse de côté les deux derniers caractères : \$extract(temp,1,\*-2).

En même temps, je convertis ma chaîne en utilisant le tableau de traduction "RAW" : \$ZCONVERT(temp, "I", "RAW").

Ensuite, je crée une nouvelle instance de ma classe RestTransfer.FileDesc et je fais les mêmes vérifications que dans les autres méthodes. Je connais la structure de ma chaîne de caractères, donc j'extrais le nom du fichier et le fichier lui-même et je les place dans les propriétés correspondantes.

Du côté client, j'ai modifié la méthode SendFile de ma classe et, avant de former le JSON, je vérifie la longueur du fichier. S'il est inférieur à 2 000 000 d'octets (limite apparente pour la méthode %JSONImport), je lance Post("/RestTransfer/jsons"). Sinon, je lance Post("/RestTransfer/json").

Now the method looks like this:

```
ClassMethod SendFile(aFileName) As %Status
{
  Set sc = $$$OK
  Set request = ..GetLink()
  set s = ##class(%Stream.FileBinary).%New()
  set s.FileName = aFileName
  if s.Size > 2000000 //3641144 max length of the string in IRIS
  {
    do request.EntityBody.Write("{\"Name\":\"_s.FileName_\", \"File\":\"\"")
    While 's.AtEnd
    {
      set temp = s.Read(.len, .sc)
      do request.EntityBody.Write($ZCONVERT(temp, "O", "RAW"))
      Quit:$System.Status.IsError(sc)
    }

    do request.EntityBody.Write("}")
    set sc = request.Post("/RestTransfer/json")
  } else {
    set p = {}
    set p.Name = s.FileName
    set sc = ..Base64EncodeStream(s, .t)
    Quit:$System.Status.IsError(sc)
    While 't.AtEnd {
      set p.File = p.File_t.Read(.len, .sc)
      Quit:$System.Status.IsError(sc)
    }
    do p.%ToJSON(request.EntityBody)
    Quit:$System.Status.IsError(sc)
    set sc = request.Post("/RestTransfer/jsons")
  }

  Quit:$System.Status.IsError(sc)
  Set response=request.HttpResponse
  do response.OutputToDevice()
  Quit sc
}
```

Pour lancer la deuxième méthode, je crée manuellement la chaîne avec JSON. Et pour transférer le fichier lui-même, côté client, je convertis également le contenu en utilisant le tableau de conversion "RAW"  
: (\$ZCONVERT(temp, "O", "RAW").

Nous lançons maintenant cette méthode et la méthode SendFileDirect pour transférer plusieurs fichiers de dimensions différentes :

```
do ##class(RestTransfer.Client).SendFile("D:\Downloads\pic3.png")
do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\pic3.png")
do ##class(RestTransfer.Client).SendFile("D:\Downloads\Archive (1).xml")
do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\Archive (1).xml")
do ##class(RestTransfer.Client).SendFile("D:\Downloads\Imagine Dragons-
Thunder.mp3")
do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\Imagine Dragons-
Thunder.mp3")
do ##class(RestTransfer.Client).SendFile("D:\Downloads\ffmpeg-win-2.2.2.exe")
do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\ffmpeg-win-2.2.2.exe")
```

Nous obtiendrons les résultats suivants :

Nous pouvons voir que les longueurs sont les mêmes, donc tout fonctionne comme prévu.

## Conclusion

Ici encore, vous pouvez en savoir plus sur la création de services [dans la documentation](#). Le code exemplaire pour les deux approches se trouve sur [GitHub](#) et [InterSystems Open Exchange](#).

Quant au transfert des résultats du module TWAIN dont nous avons parlé dans le premier article de cette série, il dépend de la taille des données que vous recevrez. Si la résolution maximale est limitée et que les fichiers sont de petite taille, vous pouvez utiliser les classes de système %JSON.Adaptor and %Library.DynamicObject. Mais pour plus de sécurité, il est préférable d'envoyer un fichier directement dans le corps d'une commande POST ou de former le JSON manuellement. Il peut également être judicieux d'utiliser des requêtes par plage et de diviser les gros fichiers en plusieurs parties, de les envoyer séparément et de les consolider en un seul fichier du côté serveur.

En tout cas, si vous avez des questions ou des suggestions concernant l'une ou l'autre, n'hésitez pas à les écrire dans la section des commentaires.

[#REST API #InterSystems IRIS](#)  
[Voir l'application sur InterSystems Open Exchange](#)

URL de la  
source: <https://fr.community.intersystems.com/post/transmission-de-fichiers-rest-pour-les-stocker-en-property-partie-2>