Article
Iryna Mykhailova · Aug 21, 2021 9m read

Open Exchange

Transferring Files via REST to Store in a Property, Part 3

The first installment of this article series discussed how to read a big chunk of data from the raw body of an HTTP POST method and save it to a database as a stream property of a class. The second installment discussed how to send files and their names wrapped in a JSON format.

Now let ’ s look closer at the idea of sending large files in parts to the server. There are several approaches we can use to do this. This article discusses using the Transfer-Encoding header to indicate chunked transfer.
The HTTP/1.1 specification introduced the Transfer-Encoding header, and the RFC 7230 section 4.1 described it, but it ’ s absent from the HTTP/2 specification.

Transfer-Encoding Header

The objective of the Transfer-Encoding header is to specify the form of encoding used to transfer the payload body to the user safely. You use this header primarily to delimit a dynamically generated payload accurately and to distinguish payload encodings for transport efficiency or security from the characteristics of the selected resource.

You can use the following values in this header:

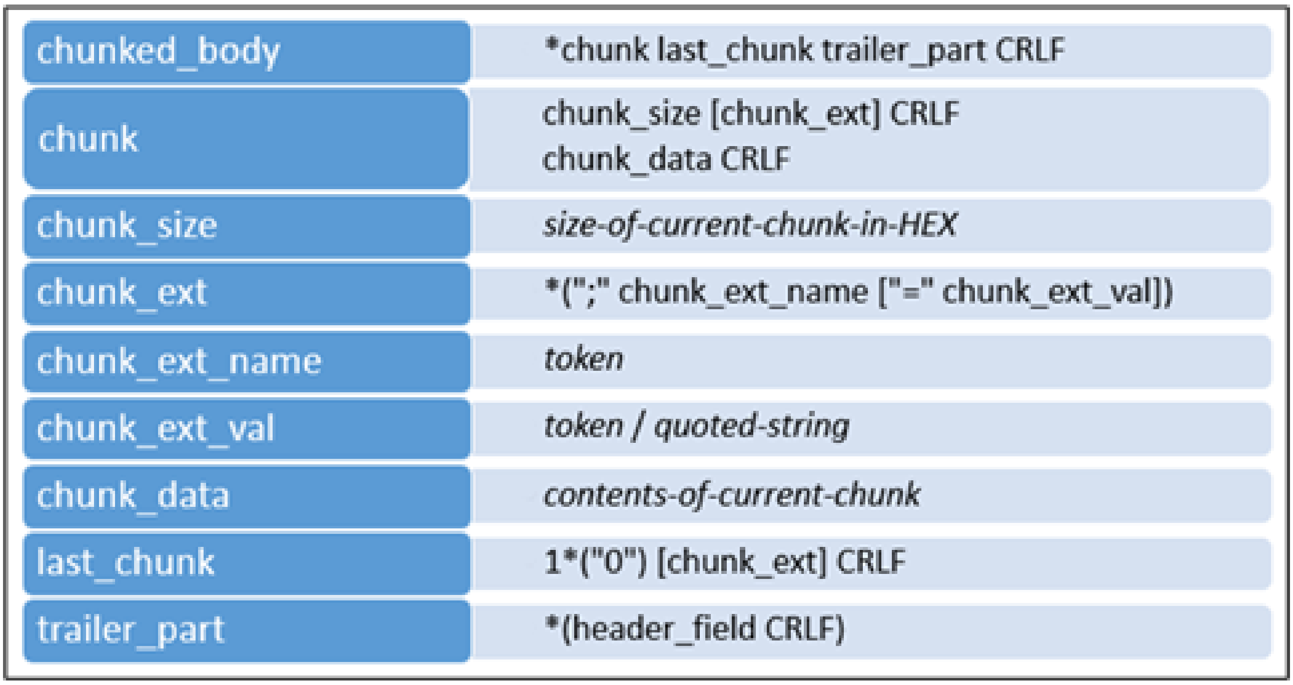
- Chunked
- Compress
- Deflate
- gzip

Transfer-Encoding Equals Chunked

When you set transfer encoding to chunked, the body of the message will consist of an unspecified number of regular chunks, a terminating chunk, a trailer part, and a final carriage return line feed (CRLF) sequence.

Each part starts with a chunk size represented by a hexadecimal number followed by an optional extension and CRLF. After that comes the body of the chunk with CRLF at the end of it. The extensions contain the metadata of the chunk. For example, metadata could include a signature, a hash, mid-message control information, and so on. The terminating chunk is a regular chunk with zero length. A trailer, which consists of (possibly empty) header fields, follows the terminating chunk.

To make it all easier to imagine, here is the structure of a message with Transfer-Encoding = chunked:



chunked_body	*chunk last_chunk trailer_part CRLF
chunk	chunk_size [chunk_ext] CRLF chunk_data CRLF
chunk_size	*size-of-current-chunk-in-HEX*
chunk_ext	*(";" chunk_ext_name ["=" chunk_ext_val])
chunk_ext_name	*token*
chunk_ext_val	*token / quoted-string*
chunk_data	*contents-of-current-chunk*
last_chunk	1*("0") [chunk_ext] CRLF
trailer_part	*(header_field CRLF)

An example of a short, chunked message looks like this:

```
13\r\n
Transferring Files \r\n
4\r\n
on\r\n
1A\r\n
community.intersystems.com
0\r\n
\r\n
```

This message body consists of three meaningful chunks. The first chunk has a length of nineteen octets, the second has four, and the third has twenty-six. You can see that the trailing CRLFs that mark the ends of the chunks don ’ t count toward the chunk size. But, if you use CRLF as the end of line (EOL) marker, then the CRLF does count as a part of a message and takes two octets. The decoded message looks like this:

```
Transferring Files on
community.intersystems.com
```

Forming Chunked Messages in IRIS

For this tutorial, we ’ ll use the method on the server created in the first article. This means that we are going to send the contents of the file directly to the body of the POST method. Since we are sending the contents of the file in the body, we send the POST to http://webserver/RestTransfer/file.

Now, let's look at how we can form a chunked message in IRIS. As specified in Sending HTTP Requests, under the section Sending a Chunked Request, you can send an HTTP request in chunks if you are using HTTP/1.1. The best part of this process is that %Net.HttpRequest automatically computes the content length of the entire message body on the server side so there is no need to change server side at all. Therefore, to send a chunked request, you need to follow these steps in the client only.

The first step is to create a subclass of %Net.ChunkedWriter and implement the OutputStream method. This method should get a stream of data, examine it, decide whether to split it into parts or not, how to split it, and invoke the inherited methods of the class to write the output. In our case, we ’ ll call the class RestTransfer.ChunkedWriter.

Next, in the client-side method responsible for sending data (called “ SendFileChunked ” here), you must create an instance of RestTransfer.ChunkedWriter class and fill it with the requested data you want to send. Since we are sending files, we ’ ll do all the heavy lifting in the RestTransfer.ChunkedWriter class. We add a property named Filename As %String and a parameter named “ MAXSIZEOFCHUNK = 10000. ” Of course, you can decide to set a maximum allowed size for the chunk as a property and set it for each file or message.

Finally, set the EntityBody property of %Net.HttpRequest to be equal to the created instance of the RestTransfer.ChunkedWriter class and you ’ re good to go.

These steps are just the new code you must write and replace in your existing method that sends files to a server.

The method looks like this:

```
ClassMethod SendFileChunked(aFileName) As %Status
{
  Set sc = $$$OK
  Set request = ..GetLink()
  set cw = ##class(RestTransfer.ChunkedWriter).%New()
  set cw.Filename = aFileName
  set request.EntityBody = cw
  set sc = request.Post("/RestTransfer/file")
  Quit:$System.Status.IsError(sc) sc
  Set response=request.HttpResponse
  do response.OutputToDevice()
  Quit sc
}
```

The %Net.ChunkedWriter class is an abstract stream class that provides an interface and has some implemented methods and properties. Here, we use the following property and methods:

- Property TranslateTable as %String forces automatic translation of the chunks when writing them into the output stream (EntityBody). We expect to receive raw data, so we must set TranslateTable to “ RAW ” .
- Method OutputStream is an abstract method overridden by a subclass to do all the chunking.
- Method WriteSingleChunk(buffer As %String) writes the Content-Length HTTP header followed by the entity-body as a single chunk. We check to see if the size of the file is smaller than the MAXSIZEOFCHUNK method, in which case, we use this method.
- Method WriteFirstChunk(buffer As %String) writes the Transfer-Encoding header followed by the first chunk. It should always be present. Zero or more calls to write more chunks may follow it, a compulsory call to write the last chunk with the empty string follows. We check that the length of the file is greater than the MAXSIZEOFCHUNK method and call this method.
- Method WriteChunk(buffer As %String) writes consequent chunks. Check to see if the rest of the file after the first chunk is still greater than MAXSIZEOFCHUNK then use this method to send data. We keep doing it until the size of the last part of the file is less than MAXSIZEOFCHUNK.
- Method WriteLastChunk(buffer As %String) writes the last chunk followed by a zero-length chunk to mark the end of the data.

Based on everything above, our class RestTransfer.ChunkedWriter looks like this:

```
Class RestTransfer.ChunkedWriter Extends %Net.ChunkedWriter
{
  Parameter MAXSIZEOFCHUNK = 10000;
  Property Filename As %String;
  Method OutputStream()
  {
    set ..TranslateTable = "RAW"
    set cTime = $zdatetime($Now(), 8, 1)
    set fStream = ##class(%Stream.FileBinary).%New()
    set fStream.Filename = ..Filename
    set size = fStream.Size
    if size < ..#MAXSIZEOFCHUNK {
      set buf = fStream.Read(.size, .st)
      if $$$ISERR(st)
      {
        THROW st
      } else {
        set ^log(cTime, ..Filename) = size
        do ..WriteSingleChunk(buf)
      }
    } else {
      set ^log(cTime, ..Filename, 0) = size
      set len = ..#MAXSIZEOFCHUNK
      set buf = fStream.Read(.len, .st)
      if $$$ISERR(st)
      {
        THROW st
      } else {
        set ^log(cTime, ..Filename, 1) = len
        do ..WriteFirstChunk(buf)
      }
      set i = 2
      While 'fStream.AtEnd {
        set len = ..#MAXSIZEOFCHUNK
        set temp = fStream.Read(.len, .sc)
    if len<..#MAXSIZEOFCHUNK
    {
      do ..WriteLastChunk(temp)
    } else {
          do ..WriteChunk(temp)
        }
        set ^log(cTime, ..Filename, i) = len
        set i = $increment(i)
      }
    }
  }
}
```

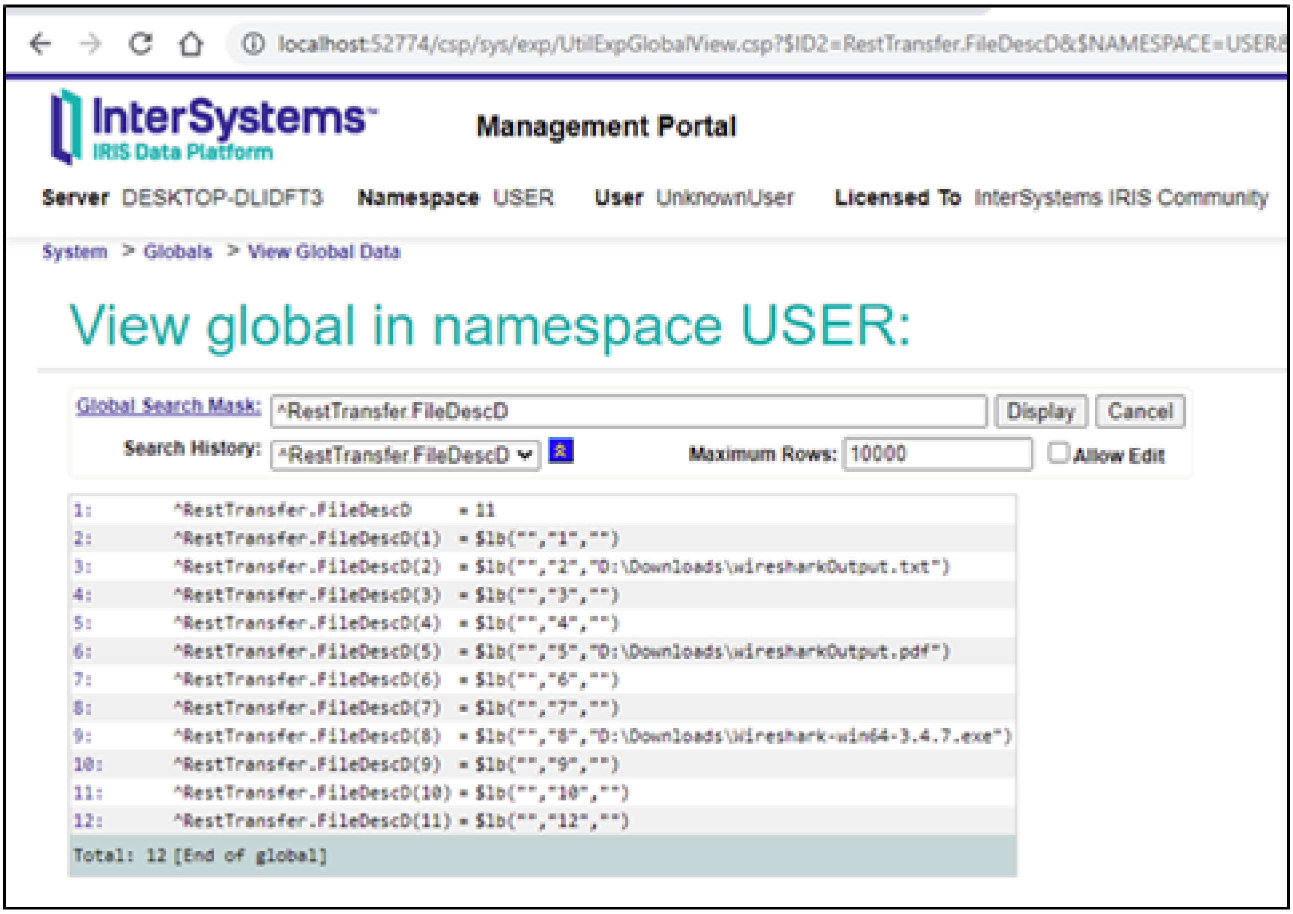
To see how these methods split the file into parts, we add a global ^log with the following structure:

```
//for transfer in a single chunk
^log(time, filename) = size_of_the_file
//for transfer in several chunks
^log(time, filename, 0) = size_of_the_file
^log(time, filename, idx) = size_of_the_idx’s_chunk
```

Now that the programming is complete, let ’ s see how all three approaches work for different files. We write a simple class method to make calls to the server:

```
ClassMethod Run()
{
  // First, I am deleting globals.
  kill ^RestTransfer.FileDescD
  kill ^RestTransfer.FileDescS
  // Then I form a list of files I want to send
  for filename = "D:\Downloads\wiresharkOutput.txt", // 856 bytes
     "D:\Downloads\wiresharkOutput.pdf", // 60 134 bytes
     "D:\Downloads\Wireshark-win64-3.4.7.exe", // 71 354 272 bytes
     "D:\Downloads\IRIS_Community-2021.1.0.215.0-win_x64.exe" //542 370 224 bytes
  {
    write !, !, filename, !, !
    // And call all three methods of sending data to server side.
    set resp1=##class(RestTransfer.Client).SendFileChunked(filename)
    if $$$ISERR(resp1) do $System.OBJ.DisplayError(resp1)
    set resp1=##class(RestTransfer.Client).SendFile(filename)
    if $$$ISERR(resp1) do $System.OBJ.DisplayError(resp1)
    set resp1=##class(RestTransfer.Client).SendFileDirect(filename)
    if $$$ISERR(resp1) do $System.OBJ.DisplayError(resp1)
  }
}
```

After running the class method Run, in the output for the first three files, the status was okay. But for the last file, while the first and last calls worked, the middle one returned an error: 5922, Timed out waiting for response. If we look in our globals method, we see that the code didn ' t save the eleventh file. This means that ##class(RestTransfer.Client).SendFile(filename) failed — or to be precise, the method that unwraps data from JSON didn ' t succeed.


localhost:52774/csp/sys/exp/UtilExpGlobalView.csp?$ID2=RestTransfer.FileDescD&$NAMESPACE=USER&
InterSystems
IRIS Data Platform
Management Portal
System > Globals > View Global Data
Display
Cancel
Allow Edit

Server DESKTOP-DLIDFT3 Namespace USER User UnknownUser Licensed To InterSystems IRIS Community

View global in namespace USER:

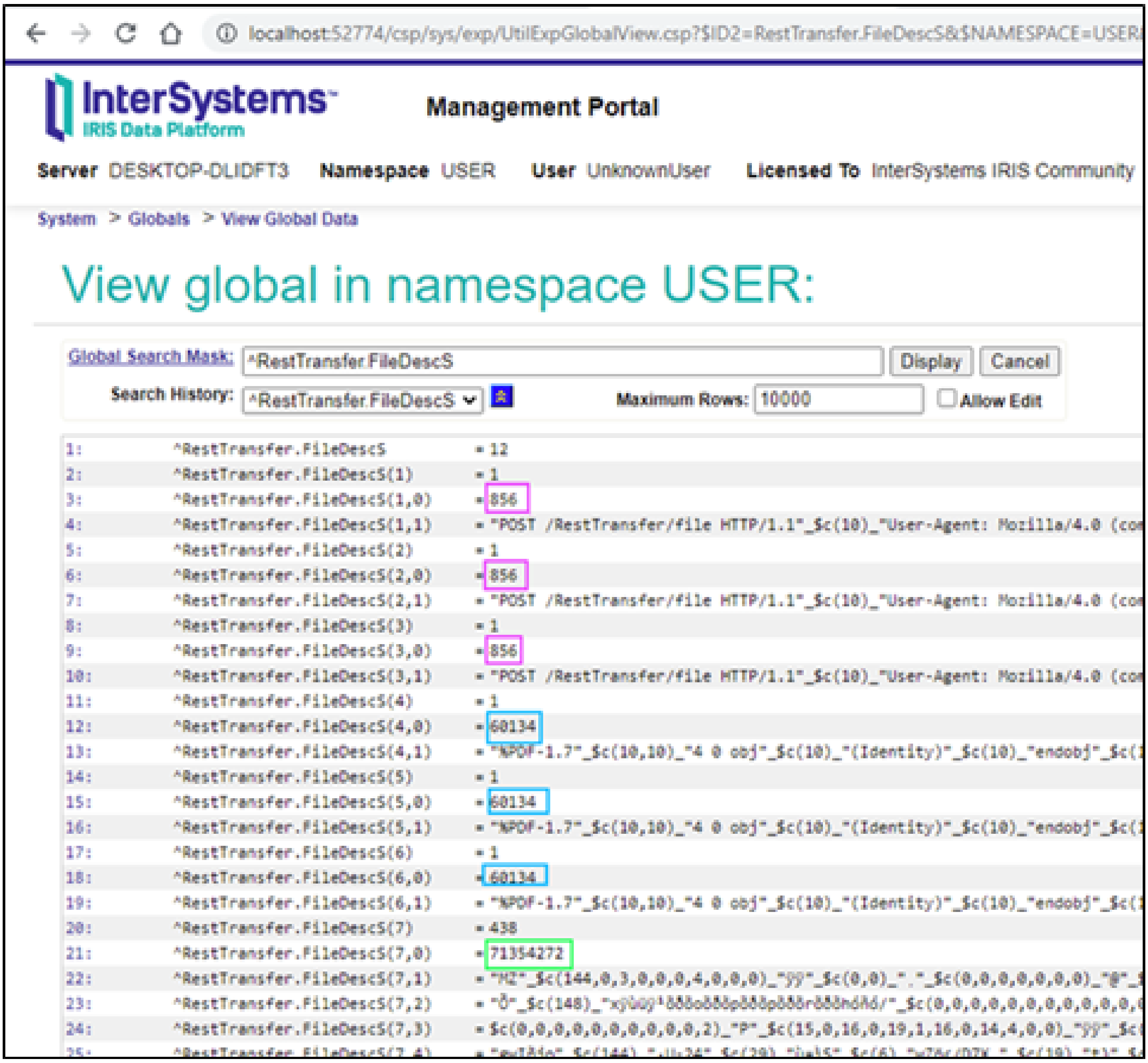
Global Search Mask: ^RestTransfer.FileDescD

Search History: ^RestTransfer.FileDescD

Maximum Rows: 10000

```
1:      ^RestTransfer.FileDescD     = 11
2:      ^RestTransfer.FileDescD(1)  = $lb("","1","")
3:      ^RestTransfer.FileDescD(2)  = $lb("","2","D:\Downloads\wiresharkOutput.txt")
4:      ^RestTransfer.FileDescD(3)  = $lb("","3","")
5:      ^RestTransfer.FileDescD(4)  = $lb("","4","")
6:      ^RestTransfer.FileDescD(5)  = $lb("","5","D:\Downloads\wiresharkOutput.pdf")
7:      ^RestTransfer.FileDescD(6)  = $lb("","6","")
8:      ^RestTransfer.FileDescD(7)  = $lb("","7","")
9:      ^RestTransfer.FileDescD(8)  = $lb("","8","D:\Downloads\Wireshark-win64-3.4.7.exe")
10:     ^RestTransfer.FileDescD(9)  = $lb("","9","")
11:     ^RestTransfer.FileDescD(10) = $lb("","10","")
12:     ^RestTransfer.FileDescD(11) = $lb("","12","")
Total: 12 [End of global]
```

Now, if we look at our streams, we see that all the successfully saved files have the correct sizes.



localhost:52774/csp/sys/exp/UtilExpGlobalView.csp?$ID2=RestTransfer.FileDescS&$NAMESPACE=USER&
InterSystems
IRIS Data Platform
Management Portal
System > Globals > View Global Data
Display
Cancel

Server DESKTOP-DLIDFT3 **Namespace** USER **User** UnknownUser **Licensed To** InterSystems IRIS Community

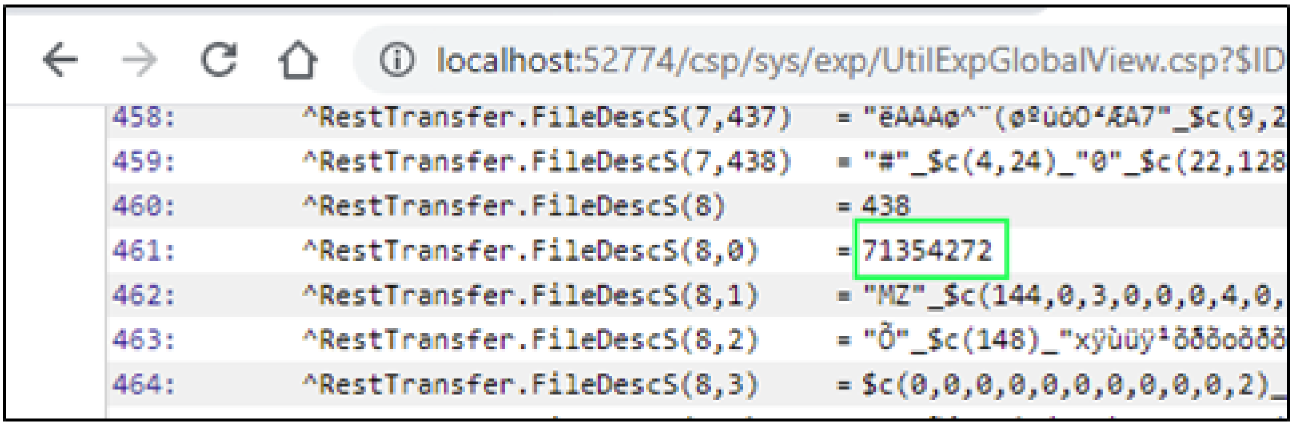
View global in namespace USER:

Global Search Mask: ^RestTransfer.FileDescS

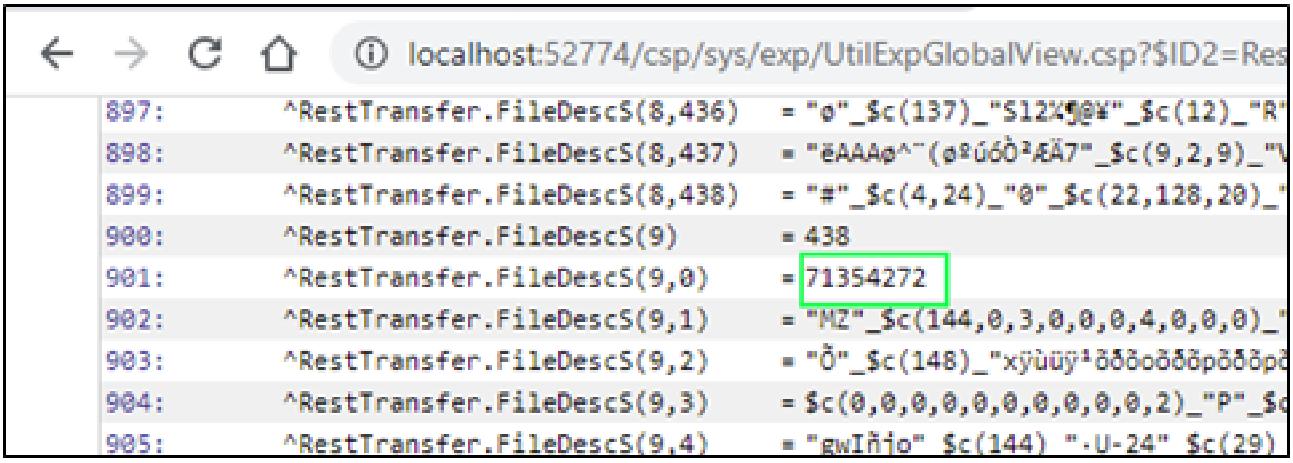
Search History: ^RestTransfer.FileDescS

Maximum Rows: 10000 ☐ Allow Edit

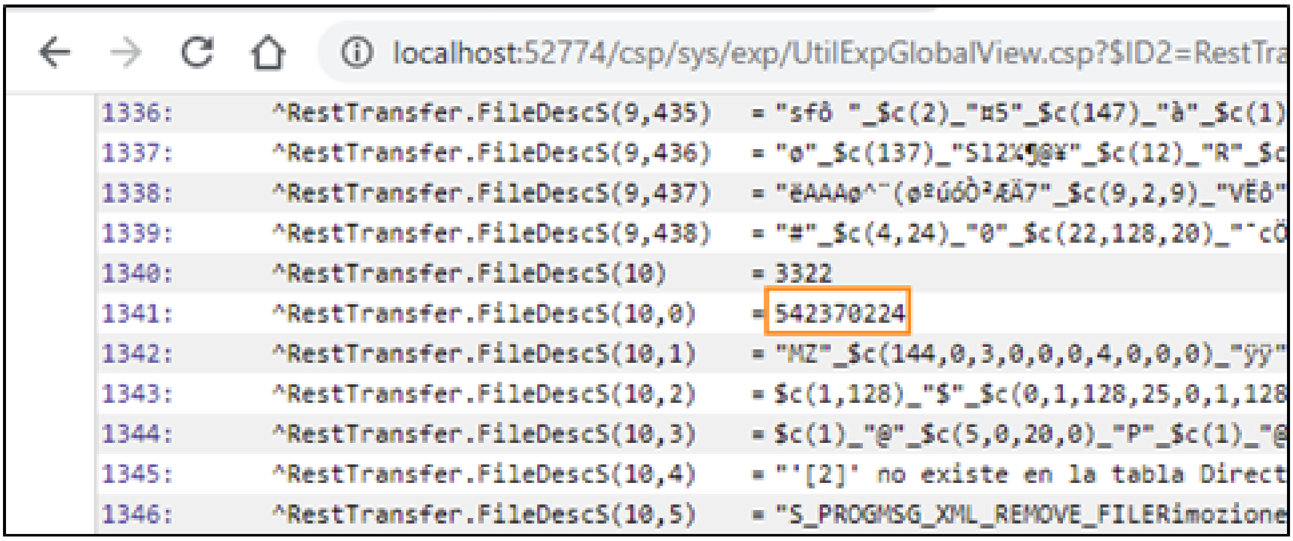
```
1:          ^RestTransfer.FileDescS          = 12
2:          ^RestTransfer.FileDescS(1)       = 1
3:          ^RestTransfer.FileDescS(1,0)     = 856
4:          ^RestTransfer.FileDescS(1,1)     = "POST /RestTransfer/file HTTP/1.1"_$c(10)_"User-Agent: Mozilla/4.0 (com
5:          ^RestTransfer.FileDescS(2)       = 1
6:          ^RestTransfer.FileDescS(2,0)     = 856
7:          ^RestTransfer.FileDescS(2,1)     = "POST /RestTransfer/file HTTP/1.1"_$c(10)_"User-Agent: Mozilla/4.0 (com
8:          ^RestTransfer.FileDescS(3)       = 1
9:          ^RestTransfer.FileDescS(3,0)     = 856
10:         ^RestTransfer.FileDescS(3,1)     = "POST /RestTransfer/file HTTP/1.1"_$c(10)_"User-Agent: Mozilla/4.0 (com
11:         ^RestTransfer.FileDescS(4)       = 1
12:         ^RestTransfer.FileDescS(4,0)     = 60134
13:         ^RestTransfer.FileDescS(4,1)     = "%PDF-1.7"_$c(10,10)_"4 0 obj"_$c(10)_"(Identity)"_$c(10)_"endobj"_$c(1
14:         ^RestTransfer.FileDescS(5)       = 1
15:         ^RestTransfer.FileDescS(5,0)     = 60134
16:         ^RestTransfer.FileDescS(5,1)     = "%PDF-1.7"_$c(10,10)_"4 0 obj"_$c(10)_"(Identity)"_$c(10)_"endobj"_$c(1
17:         ^RestTransfer.FileDescS(6)       = 1
18:         ^RestTransfer.FileDescS(6,0)     = 60134
19:         ^RestTransfer.FileDescS(6,1)     = "%PDF-1.7"_$c(10,10)_"4 0 obj"_$c(10)_"(Identity)"_$c(10)_"endobj"_$c(1
20:         ^RestTransfer.FileDescS(7)       = 438
21:         ^RestTransfer.FileDescS(7,0)     = 71354272
22:         ^RestTransfer.FileDescS(7,1)     = "MZ"_$c(144,0,3,0,0,0,4,0,0,0)_"ÿÿ"_$c(0,0)_"¸"_$c(0,0,0,0,0,0,0)_"@"_$
23:         ^RestTransfer.FileDescS(7,2)     = "Õ"_$c(148)_"xÿùüÿ¹ððõoððþððþðððrððñdÑd/"_$c(0,0,0,0,0,0,0,0,0,0,0,0,
24:         ^RestTransfer.FileDescS(7,3)     = $c(0,0,0,0,0,0,0,0,0,0,0,2)_"P"_$c(15,0,16,0,19,1,16,0,14,4,0,0)_"ÿÿ"_$c
25:         ^RestTransfer.FileDescS(7,4)
```



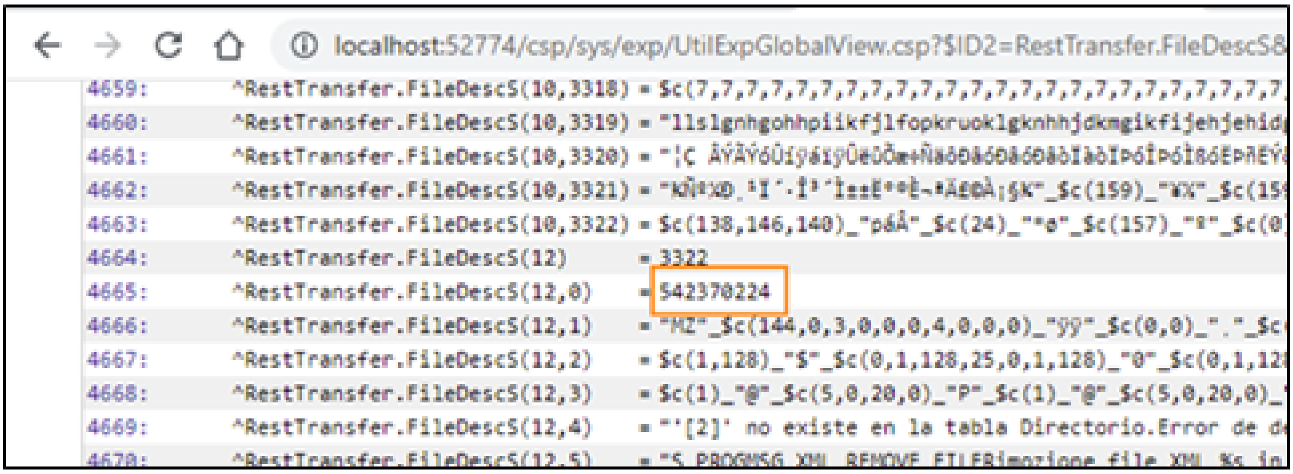
```
458:        ^RestTransfer.FileDescS(7,437)   = "ëAAAø^¨(ø²ùóO⁴ÆA7"_$c(9,2
459:        ^RestTransfer.FileDescS(7,438)   = "#"_$c(4,24)_"0"_$c(22,128
460:        ^RestTransfer.FileDescS(8)       = 438
461:        ^RestTransfer.FileDescS(8,0)     = 71354272
462:        ^RestTransfer.FileDescS(8,1)     = "MZ"_$c(144,0,3,0,0,0,4,0,
463:        ^RestTransfer.FileDescS(8,2)     = "Õ"_$c(148)_"xÿùüÿ¹ðððoððð
464:        ^RestTransfer.FileDescS(8,3)     = $c(0,0,0,0,0,0,0,0,0,0,0,2)_
```



```
897:        ^RestTransfer.FileDescS(8,436)   = "ø"_$c(137)_"S12X¶@¥"_$c(12)_"R"
898:        ^RestTransfer.FileDescS(8,437)   = "ëAAAø^¯(øºúóÒ²ÆÄ7"_$c(9,2,9)_"V
899:        ^RestTransfer.FileDescS(8,438)   = "#"_$c(4,24)_"0"_$c(22,128,20)_"
900:        ^RestTransfer.FileDescS(9)       = 438
901:        ^RestTransfer.FileDescS(9,0)     = 71354272
902:        ^RestTransfer.FileDescS(9,1)     = "MZ"_$c(144,0,3,0,0,0,4,0,0,0)_"
903:        ^RestTransfer.FileDescS(9,2)     = "Õ"_$c(148)_"xÿùüÿ¹ðððoðððpðððp
904:        ^RestTransfer.FileDescS(9,3)     = $c(0,0,0,0,0,0,0,0,0,0,2)_"P"_$c
905:        ^RestTransfer.FileDescS(9,4)     = "gwIñjo"_$c(144)_"·U-24"_$c(29)
```

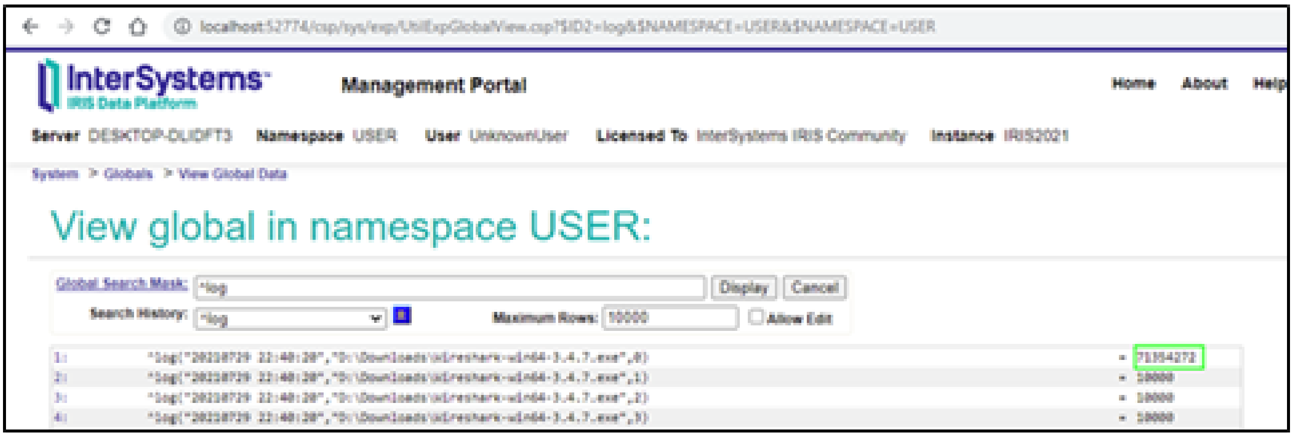

localhost:52774/csp/sys/exp/UtilExpGlobalView.csp?$ID2=RestTra

```
1336:       ^RestTransfer.FileDescS(9,435)   = "sfô "_$c(2)_"¤5"_$c(147)_"à"_$c(1)
1337:       ^RestTransfer.FileDescS(9,436)   = "ø"_$c(137)_"S12X¶@¥"_$c(12)_"R"_$c
1338:       ^RestTransfer.FileDescS(9,437)   = "ëAAAø^¯(øºúóÒ²ÆÄ7"_$c(9,2,9)_"VËô"
1339:       ^RestTransfer.FileDescS(9,438)   = "#"_$c(4,24)_"0"_$c(22,128,20)_"¯cÖ
1340:       ^RestTransfer.FileDescS(10)      = 3322
1341:       ^RestTransfer.FileDescS(10,0)    = 542370224
1342:       ^RestTransfer.FileDescS(10,1)    = "MZ"_$c(144,0,3,0,0,0,4,0,0,0)_"ÿÿ"
1343:       ^RestTransfer.FileDescS(10,2)    = $c(1,128)_"$"_$c(0,1,128,25,0,1,128
1344:       ^RestTransfer.FileDescS(10,3)    = $c(1)_"@"_$c(5,0,20,0)_"P"_$c(1)_"@
1345:       ^RestTransfer.FileDescS(10,4)    = "'[2]' no existe en la tabla Direct
1346:       ^RestTransfer.FileDescS(10,5)    = "S_PROGMSG_XML_REMOVE_FILERimozione
```


localhost:52774/csp/sys/exp/UtilExpGlobalView.csp?$ID2=RestTransfer.FileDescS&

```
4659:       ^RestTransfer.FileDescS(10,3318) = $c(7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7
4660:       ^RestTransfer.FileDescS(10,3319) = "1lslgnhgohhpiikfjlfopkruoklgknhhjdkmgikfijehjehid
4661:       ^RestTransfer.FileDescS(10,3320) = "¦Ç ÀŸÀŸóÛíýáíýÛèûÕæ+ÑàóÐàóÐàóÐàòÏàòÏÞóÏÞóÎßóÉÞñÉŸ
4662:       ^RestTransfer.FileDescS(10,3321) = "KÑ¤XÐ¸¹I´·Î³´Î±±£*ªË¬*Ä£ØÀ¡§K"_$c(159)_"¥X"_$c(15
4663:       ^RestTransfer.FileDescS(10,3322) = $c(138,146,140)_"pàÅ"_$c(24)_"*ø"_$c(157)_"®"_$c(0
4664:       ^RestTransfer.FileDescS(12)      = 3322
4665:       ^RestTransfer.FileDescS(12,0)    = 542370224
4666:       ^RestTransfer.FileDescS(12,1)    = "MZ"_$c(144,0,3,0,0,0,4,0,0,0)_"ÿÿ"_$c(0,0)_"."_$c
4667:       ^RestTransfer.FileDescS(12,2)    = $c(1,128)_"$"_$c(0,1,128,25,0,1,128)_"0"_$c(0,1,12
4668:       ^RestTransfer.FileDescS(12,3)    = $c(1)_"@"_$c(5,0,20,0)_"P"_$c(1)_"@"_$c(5,0,20,0)_"
4669:       ^RestTransfer.FileDescS(12,4)    = "'[2]' no existe en la tabla Directorio.Error de d
4670:       ^RestTransfer.FileDescS(12,5)    = "S_PROGMSG_XML_REMOVE_FILERimozione file XML %s in
```

If we look at the ^log global, we see how many chunks the code created for each file:

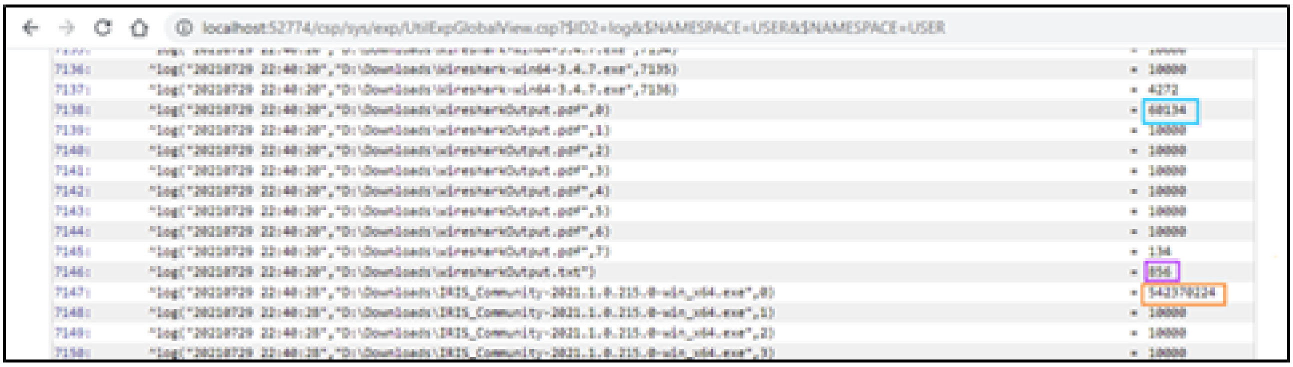


Server DESKTOP-OLIDFT3 Namespace USER User UnknownUser Licensed To InterSystems IRIS Community Instance IRIS2021

View global in namespace USER:

Global Search Mask: ^log
Search History: ^log
Maximum Rows: 10000

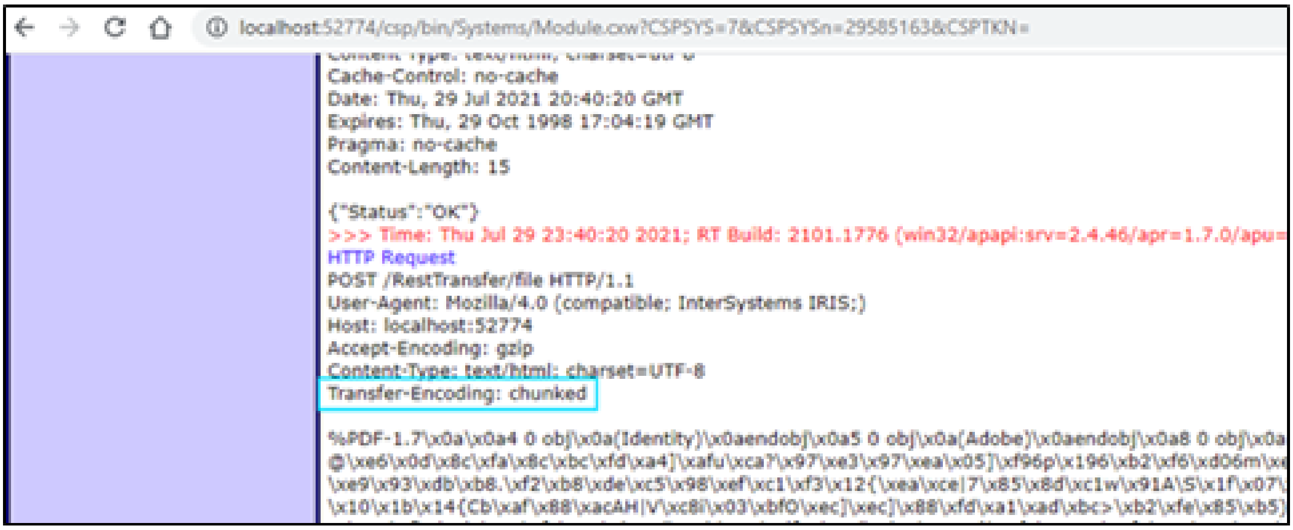
```
1:   ^log("20210729 22:40:20","D:\Downloads\wireshark-win64-3.4.7.exe",0)   = 71354272
2:   ^log("20210729 22:40:20","D:\Downloads\wireshark-win64-3.4.7.exe",1)   = 10000
3:   ^log("20210729 22:40:20","D:\Downloads\wireshark-win64-3.4.7.exe",2)   = 10000
4:   ^log("20210729 22:40:20","D:\Downloads\wireshark-win64-3.4.7.exe",3)   = 10000
```


localhost:52774/csp/sys/exp/UtilExpGlobalView.csp?$ID2=log&$NAMESPACE=USER&$NAMESPACE=USER

```
7136:   ^log("20210729 22:40:20","D:\Downloads\wireshark-win64-3.4.7.exe",7135)   = 10000
7137:   ^log("20210729 22:40:20","D:\Downloads\wireshark-win64-3.4.7.exe",7136)   = 4272
7138:   ^log("20210729 22:40:20","D:\Downloads\wiresharkOutput.pdf",0)   = 60134
7139:   ^log("20210729 22:40:20","D:\Downloads\wiresharkOutput.pdf",1)   = 10000
7140:   ^log("20210729 22:40:20","D:\Downloads\wiresharkOutput.pdf",2)   = 10000
7141:   ^log("20210729 22:40:20","D:\Downloads\wiresharkOutput.pdf",3)   = 10000
7142:   ^log("20210729 22:40:20","D:\Downloads\wiresharkOutput.pdf",4)   = 10000
7143:   ^log("20210729 22:40:20","D:\Downloads\wiresharkOutput.pdf",5)   = 10000
7144:   ^log("20210729 22:40:20","D:\Downloads\wiresharkOutput.pdf",6)   = 10000
7145:   ^log("20210729 22:40:20","D:\Downloads\wiresharkOutput.pdf",7)   = 134
7146:   ^log("20210729 22:40:20","D:\Downloads\wiresharkOutput.txt")   = 856
7147:   ^log("20210729 22:40:20","D:\Downloads\IRIS_Community-2021.1.0.215.0-win_x64.exe",0)   = 543270224
7148:   ^log("20210729 22:40:20","D:\Downloads\IRIS_Community-2021.1.0.215.0-win_x64.exe",1)   = 10000
7149:   ^log("20210729 22:40:20","D:\Downloads\IRIS_Community-2021.1.0.215.0-win_x64.exe",2)   = 10000
7150:   ^log("20210729 22:40:20","D:\Downloads\IRIS_Community-2021.1.0.215.0-win_x64.exe",3)   = 10000
```

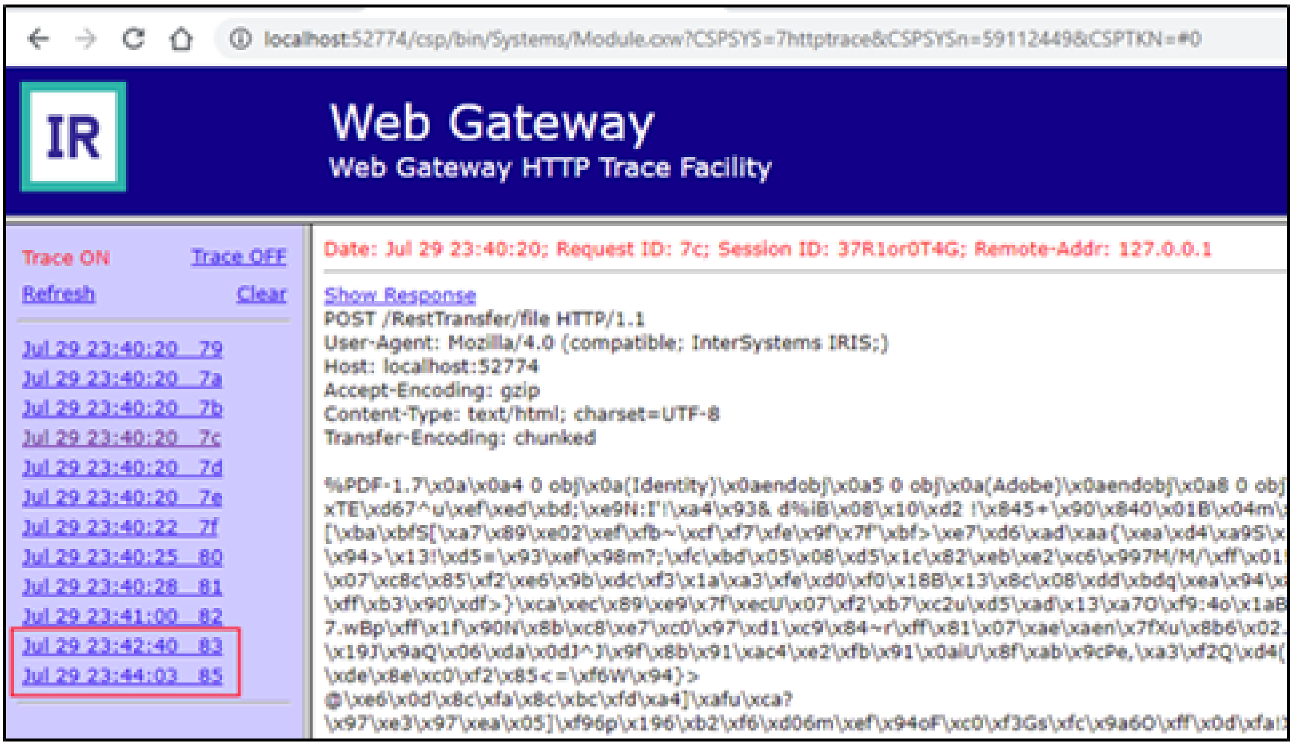
You ’ d probably like to see the bodies of the actual messages. Eduard Lebedyuk suggested in the article Debugging Web that it ’ s possible to use CSP Gateway Logging and Tracing.

If we look in the Event Log for the second chunked file, we see that the value of the Transfer-Encoding header is indeed “ chunked. ” Unfortunately, the server has already glued the message together, so we don ’ t see the actual chunking.


localhost:52774/csp/bin/Systems/Module.cxw?CSPSYS=7&CSPSYSn=29585163&CSPTKN=

```
Cache-Control: no-cache
Date: Thu, 29 Jul 2021 20:40:20 GMT
Expires: Thu, 29 Oct 1998 17:04:19 GMT
Pragma: no-cache
Content-Length: 15

{"Status":"OK"}
>>> Time: Thu Jul 29 23:40:20 2021; RT Build: 2101.1776 (win32/apapi:srv=2.4.46/apr=1.7.0/apu=
HTTP Request
POST /RestTransfer/file HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; InterSystems IRIS;)
Host: localhost:52774
Accept-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked

%PDF-1.7\x0a\x0a4 0 obj\x0a(Identity)\x0aendobj\x0a5 0 obj\x0a(Adobe)\x0aendobj\x0a8 0 obj\x0a
@\xe6\x0d\x8c\xfa\x8c\xbc\xfd\xa4]\xafu\xca?\x97\xe3\x97\xea\x05]\xf96p\x196\xb2\xf6\xd06m\x
\xe9\x93\xdb\xb8.\xf2\xb8\xde\xc5\x98\xef\xc1\xf3\x12{\xea\xce|7\x85\x8d\xc1w\x91A\S\x1f\x07\
\x10\x1b\x14{Cb\xaf\x88\xacAH|V\xc8i\x03\xbfO\xec]\xec]\x88\xfd\xa1\xad\xbc>\xb2\xfe\x85\xb5]
```

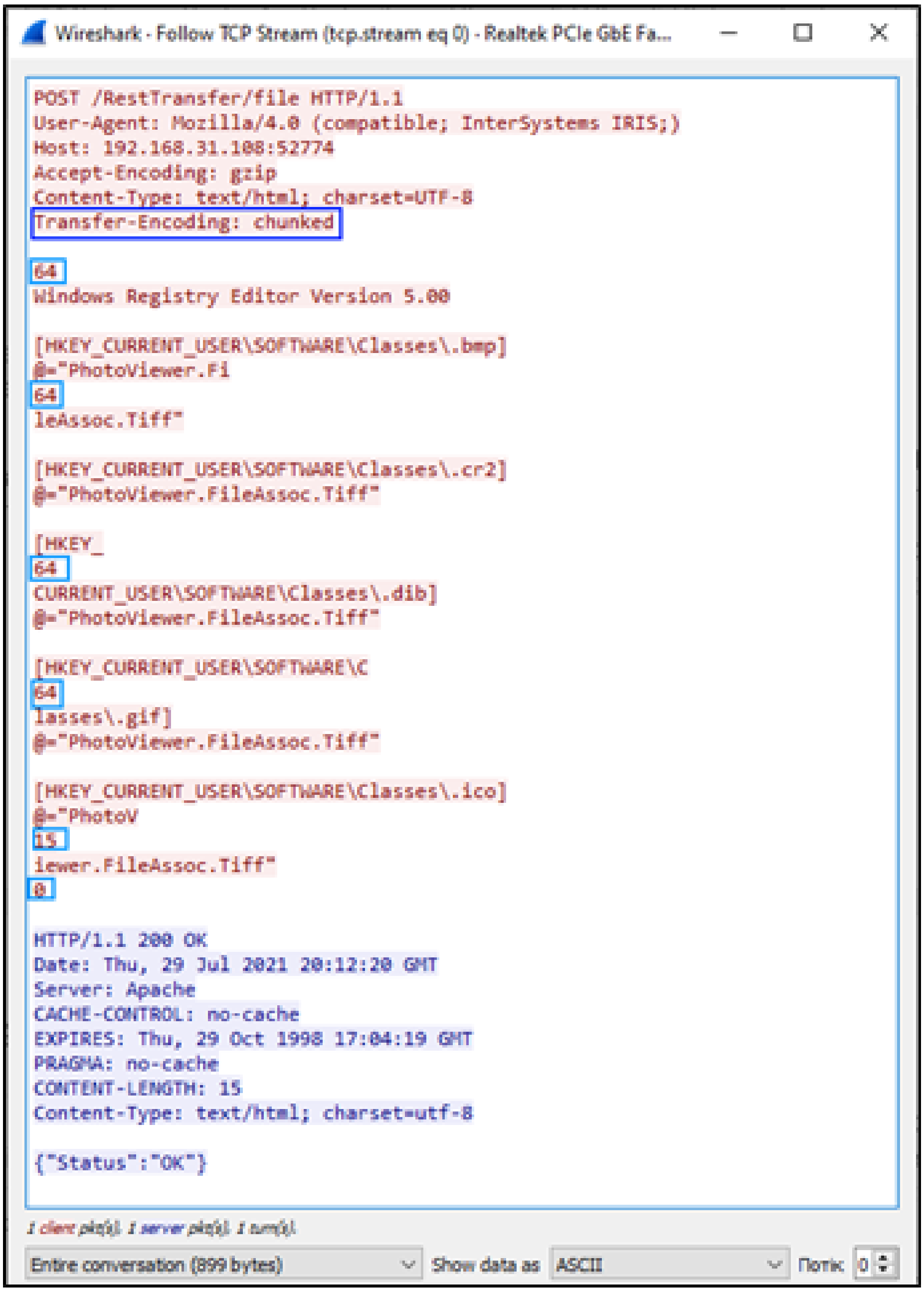
Using the Trace feature doesn ’ t show a lot more information, but it clarifies that there is a gap between the penultimate and the last request.



Date: Jul 29 23:40:20; Request ID: 7c; Session ID: 37R1or0T4G; Remote-Addr: 127.0.0.1

```
POST /RestTransfer/file HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; InterSystems IRIS;)
Host: localhost:52774
Accept-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked

%PDF-1.7\x0a\x0a4 0 obj\x0a(Identity)\x0aendobj\x0a5 0 obj\x0a(Adobe)\x0aendobj\x0a8 0 obj
xTE\xd67^u\xef\xed\xbd;\xe9N:I'!\xa4\x93& d%iB\x08\x10\xd2 !\x845+\x90\x840\x01B\x04m\
[\xba\xbfS[\xa7\x89\xe02\xef\xfb~\xcf\xf7\xfe\x9f\x7f'\xbf>\xe7\xd6\xad\xaa{\xea\xd4\xa95\x
\x94>\x13!\xd5=\x93\xef\x98m?;\xfc\xbd\x05\x08\xd5\x1c\x82\xeb\xe2\xc6\x997M/M/\xff\x01
\x07\xc8c\x85\xf2\xe6\x9b\xdc\xf3\x1a\xa3\xfe\xd0\xf0\x18B\x13\x8c\x08\xdd\xbdq\xea\x94\x
\xff\xb3\x90\xdf>}\xca\xec\x89\xe9\x7f\xecU\x07\xf2\xb7\xc2u\xd5\xad\x13\xa7O\xf9:4o\x1aB
7.wBp\xff\x1f\x90N\x8b\xc8\xe7\xc0\x97\xd1\xc9\x84~r\xff\x81\x07\xae\xaen\x7fXu\x8b6\x02.
\x19J\x9aQ\x06\xda\x0dJ^J\x9f\x8b\x91\xac4\xe2\xfb\x91\x0alU\x8f\xab\x9cPe,\xa3\xf2Q\xd4(
\xde\x8e\xc0\xf2\x85<=\xf6W\x94}>
@\xe6\x0d\x8c\xfa\x8c\xbc\xfd\xa4]\xafu\xca?
\x97\xe3\x97\xea\x05]\xf96p\x196\xb2\xf6\xd06m\xef\x94oF\xc0\xf3Gs\xfc\x9a6O\xff\x0d\xfa!
```

To see the actual parts of the messages, we copy the client to another computer to use a network sniffer. Here we ’ ve chosen to useWireshark because it is free and it has the necessary functions. To better show you how the code splits the file into chunks, we can change the value of MAXSIZEOFCHUNK to 100 and chose to send a small file. So now, we can see the following result:

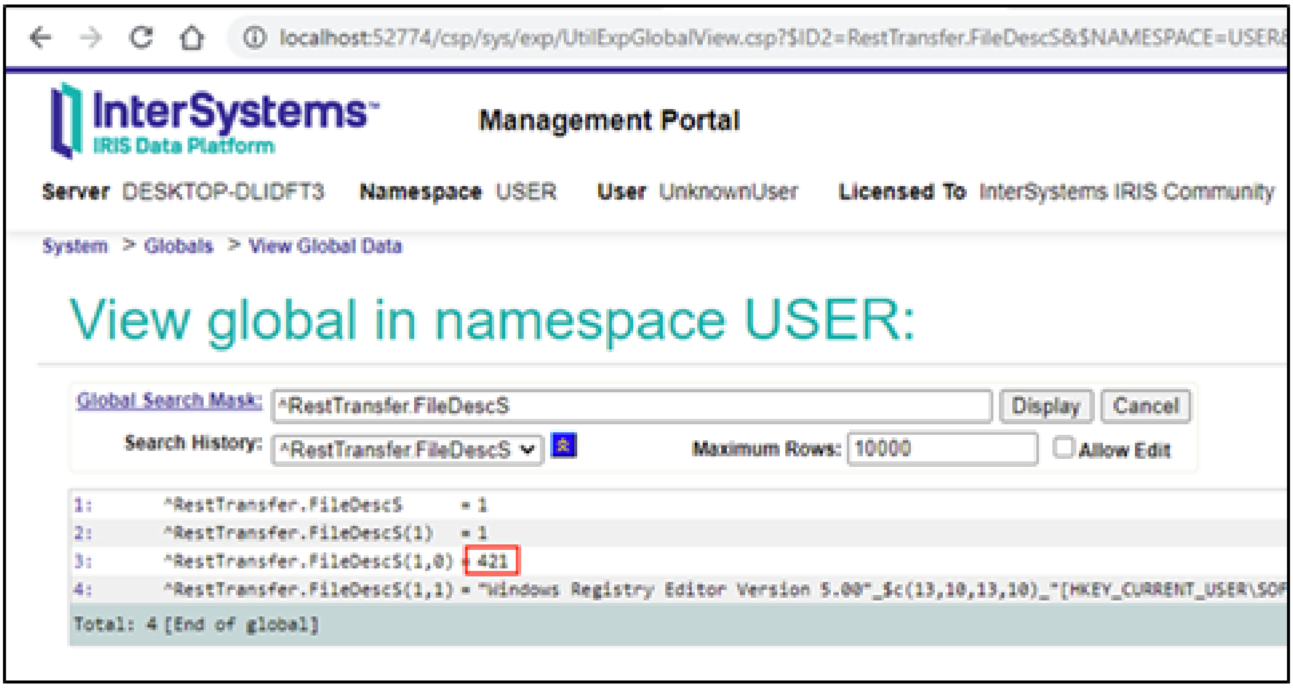


Wireshark - Follow TCP Stream (tcp.stream eq 0) - Realtek PCIe GbE Fa...

```
POST /RestTransfer/file HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; InterSystems IRIS;)
Host: 192.168.31.108:52774
Accept-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked

64
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\SOFTWARE\Classes\.bmp]
@="PhotoViewer.Fi
64
leAssoc.Tiff"

[HKEY_CURRENT_USER\SOFTWARE\Classes\.cr2]
@="PhotoViewer.FileAssoc.Tiff"

[HKEY_
64
CURRENT_USER\SOFTWARE\Classes\.dib]
@="PhotoViewer.FileAssoc.Tiff"

[HKEY_CURRENT_USER\SOFTWARE\C
64
lasses\.gif]
@="PhotoViewer.FileAssoc.Tiff"

[HKEY_CURRENT_USER\SOFTWARE\Classes\.ico]
@="PhotoV
15
iewer.FileAssoc.Tiff"
0

HTTP/1.1 200 OK
Date: Thu, 29 Jul 2021 20:12:20 GMT
Server: Apache
CACHE-CONTROL: no-cache
EXPIRES: Thu, 29 Oct 1998 17:04:19 GMT
PRAGMA: no-cache
CONTENT-LENGTH: 15
Content-Type: text/html; charset=utf-8

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

1 client pkt(s), 1 server pkt(s), 1 turn(s).
Entire conversation (899 bytes)
Show data as
ASCII
Поток 0

We see that the lengths of all but the last two chunks equal 64 in HEX (100 in DEC), the final chunk with data equals 21 DEC (15 in HEX), and we can see the size of the last chunk is zero. Everything looks OK and accords with the specification. The overall length of the file equals 421 (4x100+1x21), which we can also see in globals:


localhost:52774/csp/sys/exp/UtilExpGlobalView.csp?$ID2=RestTransfer.FileDescS&$NAMESPACE=USER&
InterSystems
IRIS Data Platform
Management Portal
Server DESKTOP-DLIDFT3
Namespace USER
User UnknownUser
Licensed To InterSystems IRIS Community
System > Globals > View Global Data
View global in namespace USER:
Global Search Mask:
Display
Cancel
Search History:
Maximum Rows:
Allow Edit

Global Search Mask: ^RestTransfer.FileDescS
Search History: ^RestTransfer.FileDescS
Maximum Rows: 10000
□ Allow Edit

```
1:      ^RestTransfer.FileDescS      = 1
2:      ^RestTransfer.FileDescS(1)   = 1
3:      ^RestTransfer.FileDescS(1,0) = 421
4:      ^RestTransfer.FileDescS(1,1) = "Windows Registry Editor Version 5.00"_$c(13,10,13,10)_"[HKEY_CURRENT_USER\SOF
Total: 4 [End of global]
```

Wrapping Up

Overall, we can see that this approach works and enables sending large files without problems to the server. Additionally, if you ’ re sending large amounts of data to a client, you might want to familiarize yourself with the Web Gateway Operation and Configuration, section Application Path Configuration Parameters, parameter Response Size Notification. It specifies Web Gateway behavior when sending large amounts of data depending on the version of HTTP used.

The code for this approach is added to the previous version of this example on GitHub and InterSystems Open Exchange.

While on the topic of sending files in chunks, it is also possible to use the Content-Range header with or without the Transfer-Encoding header to indicate which exact part of the data is being transferred. Furthermore, you can use a completely new concept of streams available with the HTTP/2 specification.

As always, if you have any questions or suggestions, 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-3