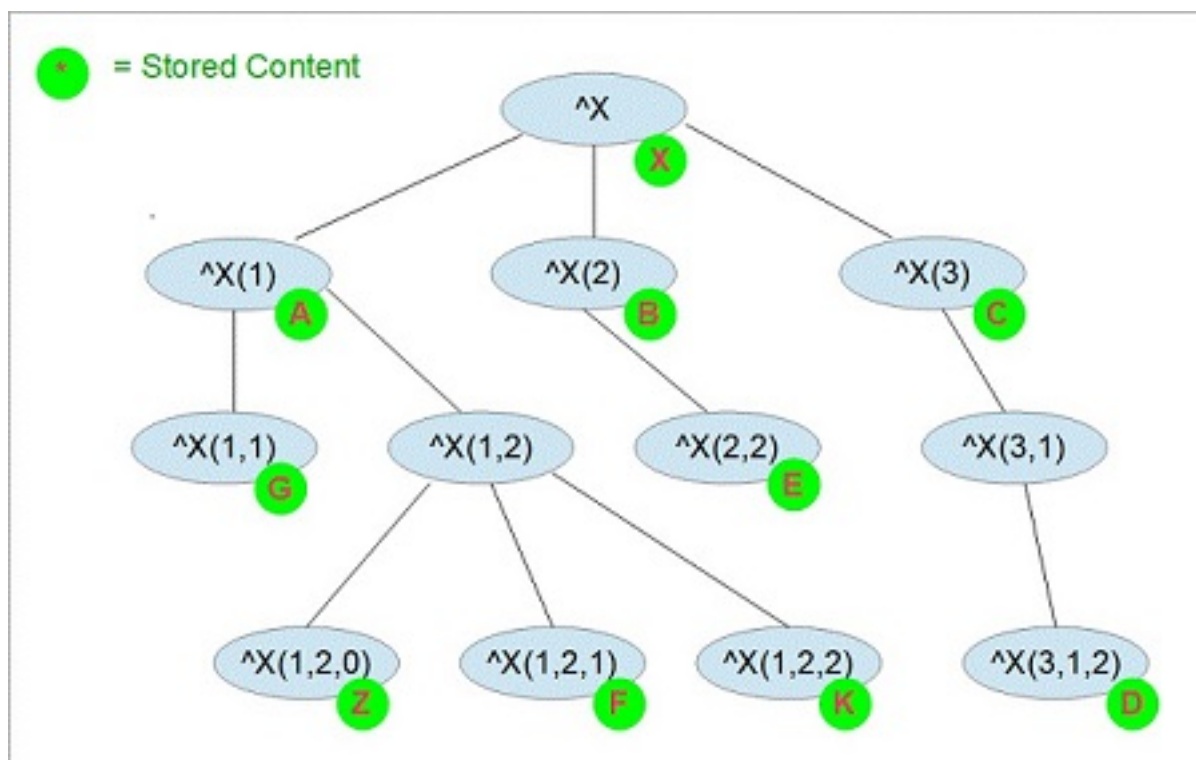


## Article

[Robert Cemper](#) · Mar 24, 2022 3m read

## GlobalToJSON-XL-Academic

This package offers a utility to export an XLarge Global into a JSON object file and to show or import it again. [In a previous example](#), this all was processed in memory. But if this is a large Global you may either experience <MAXSTRING> or an <STORE> error if the generated JSON structure exceeds available memory.



Academic refers to the structure created.

- each node of the Global including the top node is represented as a JSON object
- {"node":<node name>,"val":<value stored>,"sub":[<JSON array of subscript objects>]}
- value and subscript are optional but one of them always exists for a valid node
- the JSON object for the lowest level subscript has only value but no further subscript.

So this is basically a 1:1 image of your global and it's exported to a file (default: gbl.json)

In addition to the export, a show method displays the generated file.

The tricky part is the import from file. It is a customize JSON parser as all others just operate in memory. this fails with a reasonable-sized Global (eg. ^oddDEF with ~1.7 million nodes takes ~78MB JSON file.)

## Prerequisites

Make sure you have [git](#) and [Docker desktop](#) installed.

## Installation

Clone/git pull the repo into any local directory

```
git clone https://github.com/rcemper/GlobalToJSON-XLA.git
```

Run the IRIS container with your project:

```
docker-compose up -d --build
```

## How to Test it


This is the pre-loaded Global ^dc.MultiD for testing.

# View global in namespace USER:

Global Search Mask:

Display

Cancel

Search History:  

Maximum Rows:

☐ Allow Edit

1:	<code>^dc.MultiD</code>	<code>= 5</code>
2:	<code>^dc.MultiD(1)</code>	<code>= \$lb("Braam,Ted Q.",51353)</code>
3:	<code>^dc.MultiD(1,"mJSON")</code>	<code>= "{}"</code>
4:	<code>^dc.MultiD(2)</code>	<code>= \$lb("Klingman,Uma C.",62459)</code>
5:	<code>^dc.MultiD(2,2,"Multi","a")</code>	<code>= 1</code>
6:	<code>^dc.MultiD(2,2,"Multi","rob",1)</code>	<code>= "rcc"</code>
7:	<code>^dc.MultiD(2,2,"Multi","rob",2)</code>	<code>= 2222</code>
8:	<code>^dc.MultiD(2,"Multi","a")</code>	<code>= 1</code>
9:	<code>^dc.MultiD(2,"Multi","rob",1)</code>	<code>= "rcc"</code>
10:	<code>^dc.MultiD(2,"Multi","rob",2)</code>	<code>= 2222</code>
11:	<code>^dc.MultiD(2,"mJSON")</code>	<code>= {"A":"ahahah","Rob":"VIP","Rob2":1111,"Rob3":true}"</code>
12:	<code>^dc.MultiD(3)</code>	<code>= \$lb("Goldman,Kenny H.",45831)</code>
13:	<code>^dc.MultiD(3,"mJSON")</code>	<code>= "{}"</code>
14:	<code>^dc.MultiD(4)</code>	<code>= \$lb("", "")</code>
15:	<code>^dc.MultiD(4,"mJSON")</code>	<code>= {"rcc":122}"</code>
16:	<code>^dc.MultiD(5)</code>	<code>= \$lb("", "")</code>
17:	<code>^dc.MultiD(5,"mJSON")</code>	<code>= "{}"</code>
Total: 17 [End of global]		

There are 3 methods available

- `ClassMethod export(gref As %String = "%%", file = "gbl.json") As %String file = 0 >>> display to terminal`
- `ClassMethod show(file = "gbl.json") As %String`
- `ClassMethod import( file = "gbl.json", test = 0) As %String test = 1 >>> load into a PPG`

Open IRIS terminal

```
$ docker-compose exec iris iris session iris
```

```
USER>write ##class(dc.GblToJson.XLA).export("^dc.MultiD")
File gbl.json created
```

```
USER>write ##class(dc.GblToJson.XLA).export("^dc.MultiD",0)
{"node":"^dc.MultiD"
,"val":5
,"sub":[
{"node":1
,"val": "$lb(\"Braam,Ted Q.\",51353)"
,"sub":[
{"node":"mJSON"
,"val": "{}"
}
}
}
--- truncated ---
```

```
USER>>write ##class(dc.GblToJson.XLA).show()
{"node":"^dc.MultiD"
,"val":5
,"sub":[
```

```
{ "node":1  
  , "val": "$lb(\"Braam, Ted Q.\", 51353) "  
--- truncated ---
```

validated JSON object

```

1  {
2    "node": "^dc.MultiID",
3    "val": 5,
4    "sub": [{
5      "node": 1,
6      "val": "$lb(\\"Braam,Ted Q.\\",51353)",
7      "sub": [{
8        "node": "mJSON",
9        "val": "{}"
10     }]
11   },
12   {
13     "node": 2,
14     "val": "$lb(\\"Klingman,Uma C.\",62459)",
15     "sub": [{
16       "node": 2,
17       "sub": [{
18         "node": "Multi",
19         "sub": [{
20           "node": "a",
21           "val": 1
22         },
23         {
24           "node": "rob",
25           "sub": [{
26             "node": 1,
27             "val": "rcc"
28           },
29           {
30             "node": 2,
31             "val": 2222
32           }
33         ]
34       }
35     ]
36   },
37   {
38     "node": "Multi",
39     "sub": [{
40       "node": "a",
41       "val": 1
42     },
43     {
44       "node": "rob",
45       "sub": [{
46         "node": 1,
47         "val": "rcc"
48       },
49       {
50         "node": 2,
51         "val": 2222
52       }
53     ]
54   }
55 ],
56 ],
57 {
58   "node": "mJSON",
59   "val": "{\\"A\\":\\"ahahah\\",\\"Rob\\":\\"VIP\\",\\"Rob2\\":1111,\\"Rob3\\":true}"
60 }
61 ],
62 {
63   "node": 3,
64   "val": "$lb(\\"Goldman,Kenny H.\",45831)",
65   "sub": [{
66     "node": "mJSON",
67     "val": "{}"
68   }]
69 },
70 {
71   "node": 4,
72   "val": "$lb(\\"\\",\\"\\")",
73   "sub": [{
74     "node": "mJSON",
75     "val": "{\\"rcc\\":122}"
76   }]
77 },
78 {
79   "node": 5,
80   "val": "$lb(\\"\\",\\"\\")",
81   "sub": [{
82     "node": "mJSON",
83     "val": "{}"
84   }]
85 }
86 ]
87 }
88 ]
89 }

```

Now we want to verify the load function as a test into a PPG

```
USER>write ##class(dc.GblToJSON.XLA).import(,1)
Global ^|dc.MultiD loaded
```

```
USER>zwrite ^|dc.MultiD
^|dc.MultiD=5
^|dc.MultiD(1)=$lb("Braam,Ted Q.",51353)
^|dc.MultiD(1,"mJSON")="{ }"
^|dc.MultiD(2)=$lb("Klingman,Uma C.",62459)
^|dc.MultiD(2,2,"Multi","a")=1
^|dc.MultiD(2,2,"Multi","rob",1)="rcc"
^|dc.MultiD(2,2,"Multi","rob",2)=2222
^|dc.MultiD(2,"Multi","a")=1
^|dc.MultiD(2,"Multi","rob",1)="rcc"
^|dc.MultiD(2,"Multi","rob",2)=2222
^|dc.MultiD(2,"mJSON")="{ \"A\":\"ahahah\", \"Rob\":\"VIP\", \"Rob2\":1111, \"Rob3\":true}"
^|dc.MultiD(3)=$lb("Goldman,Kenny H.",45831)
^|dc.MultiD(3,"mJSON")="{ }"
^|dc.MultiD(4)=$lb("","")
^|dc.MultiD(4,"mJSON")="{ \"rcc\":122}"
^|dc.MultiD(5)=$lb("","")
^|dc.MultiD(5,"mJSON")="{ }"
```

USER>

q.a.d.

Code Quality

Do not wonder about some strange code constructs.

They were required as CodeQuality neither understands the NEW command, nor the scope of %variables !



[Video](#)

[Online Demo Terminal](#)

[Online Demo SMP](#)

[Previous article in DC](#)

[GitHub](#)

[#Globals](#) [#JSON](#) [#ObjectScript](#) [#InterSystems](#) [IRIS](#)

---

Source URL: <https://community.intersystems.com/post/globaltojson-xl-academic>