Contents

HTTP Server

*** Sergey Linev   GSI, Darmstadt ***

Chapter 1

HTTP server in ROOT

The idea of THttpServer is to provide remote http access to running ROOT application and enable HTML/JavaScript user interface. Any registered object can be requested and displayed in the web browser. There are many benefits of such approach:

- standard http interface to ROOT application
- no any temporary ROOT files to access data
- user interface running in all browsers

1.1   Starting the HTTP server

To start the http server, at any time, create an instance of the THttpServer class like:

```
serv = new THttpServer("http:8080");
```

This will start a civetweb-based http server on the port 8080. Then one should be able to open the address "http://localhost:8080" in any modern browser (IE9, Firefox, Chrome, Opera) and browse objects created in application. By default, the server can access files, canvases, and histograms via the gROOT pointer. All those objects can be displayed with JSROOT graphics.

There is a snapshot (frozen copy) of such server, running in tutorials/http/httpserver.C macro from ROOT tutorial.

<iframe width="800" height="500" src="https://root.cern.ch/js/3.6/httpserver.C/?layout=simple&item=Canvases/c1">

One could specify several options when creating http server. They could be add as additional URL parameters to the constructor arguments like:

```
serv = new THttpServer("http:8080?loopback&thrds=2");
```

Following parameters are supported:

- thrds=N - number of threads used by the civetweb (default is 5)
- top=name - configure top name, visible in the web browser
- auth_file=filename - authentication file name, created with htdigets utility
- auth_domain=domain - authentication domain
- loopback - bind specified port to loopback 127.0.0.1 address
- debug - enable debug mode, server always returns html page with request info

If necessary, one could bind http server to specific IP address like:

```
new THttpServer("http:192.168.1.17:8080")
```

1.2   Registering objects

At any time, one could register other objects with the command:

```
TGraph* gr = new TGraph(10);
gr->SetName("gr1");
serv->Register("graphs/subfolder", gr);
```

One should specify sub-folder name, where objects will be registered. If sub-folder name does not starts with slash /, than top-name folder /Objects/ will be prepended. At any time one could unregister objects:

```
serv->Unregister(gr);
```

THttpServer does not take ownership over registered objects - they should be deleted by user.

If the objects content is changing in the application, one could enable monitoring flag in the browser - then objects view will be regularly updated.

1.3   Command interface

THttpServer class provide simple interface to invoke command from web browser. One just register command like:

```
serv->RegisterCommand("/DoSomething","SomeFunction()");
```

Element with name DoSomething will appear in the web browser and can be clicked. It will result in gROOT->ProcessLineSync("SomeFunction()") call. When registering command, one could specify icon name which will be displayed with the command.

```
serv->RegisterCommand("/DoSomething","SomeFunction()", "/rootsys/icons/ed_execute.png");
```

In example usage of images from $ROOTSYS/icons directory is shown. One could prepend button; string to the icon name to let browser show command as extra button. In last case one could hide command element from elements list:

```
serv->Hide("/DoSomething");
```

One can find example of command interface usage in tutorials/http/httpcontrol.C macro.

1.4   Configuring user access

By default, the http server is open for anonymous access. One could restrict the access to the server for authenticated users only. First of all, one should create a password file, using the **htdigest** utility.

```
[shell] htdigest -c .htdigest domain_name user_name
```

It is recommended not to use special symbols in domain or user names. Several users can be add to the ".htdigetst" file. When starting the server, the following arguments should be specified:

```
root [0] new THttpServer("http:8080?auth_file=.htdigest&auth_domain=domain_name");
```

After that, the web browser will automatically request to input a name/password for the domain "domain_name"

Based on authorized accounts, one could restrict or enable access to some elements in the server objects hierarchy, using THttpServer::Restrict() method.

For instance, one could hide complete folder from 'guest' account:

```
root [6]  serv->Restrict("/Folder",  "hidden=guest");
```

Or one could hide from all but 'admin' account:

```
root [7]  serv->Restrict("/Folder",  "visible=admin");
```

Hidden folders or objects can not be accessed via http protocol.

By default server runs in readonly mode and do not allow methods execution via 'exe.json' or 'exe.bin' requests. To allow such action, one could either grant generic access for all or one could allow to execute only special method:

```
root [8]  serv->Restrict("/Folder/histo1",  "allow=all");
root [9]  serv->Restrict("/Folder/histo1",  "allow_method=GetTitle");
```

One could provide several options for the same item, separating them with '&' sign:

```
root [10]  serv->Restrict("/Folder/histo1",  "allow_method=GetTitle&hide=guest");
```

Complete list of supported options could be found in TRootSniffer:Restrict() method documentation.

1.5 Using FastCGI interface

FastCGI is a protocol for interfacing interactive programs with a web server like Apache, lighttpd, Microsoft ISS and many others.

When starting THttpServer, one could specify:

```
serv = new THttpServer("fastcgi:9000");
```

In fact, the FastCGI interface can run in parallel to http server. One can just call:

```
serv = new THttpServer("http:8080");
serv->CreateEngine("fastcgi:9000");
```

One could specify a debug parameter to be able to adjust the FastCGI configuration on the web server:

```
serv->CreateEngine("fastcgi:9000?debug=1");
```

All user access will be ruled by the main web server - for the moment one cannot restrict access with fastcgi engine.

1.5.1 Configure fastcgi with Apcahe2

First of all, one should compile and install mod_fastcgi module. Then *mod_fastcgi* should be specified in httpd.conf to load it when Apache server is started. Finally in host configuration file one should have following lines:

```
 <IfModule mod_fastcgi.c>
    FastCgiExternalServer "/srv/www/htdocs/root.app" -host rootapp_host_name:9000
 </IfModule>
```

Here is supposed that directory "/srv/www/htdocs" is root directory for web server. Than one should be able to open address:

```
 http://apache_host_name/root.app/
```

1.5.2   Configure fastcgi with lighttpd

An example of configuration file for *lighttpd* server is:

```
server.modules += ( "mod_fastcgi" )
fastcgi.server = (
   "/root.app" =>
     (( "host" => "192.168.1.11",
        "port" => 9000,
        "check-local" => "disable",
        "docroot" => "/"
     ))
)
```

Be aware, that with *lighttpd* one should specify IP address of the host, where ROOT application is running. Address of the ROOT application will be following:

```
http://lighttpd_host_name/root.app/
```

1.6   Integration with existing applications

In many practical cases no change of existing code is required. Opened files (and all objects inside), existing canvas and histograms are automatically scanned by the server and will be available to the users. If necessary, any object can be registered directly to the server with a **THttpServer::Register()** call.

Central point of integration - when and how THttpServer get access to data from a running application. By default it is done during the gSystem->ProcessEvents() call - THttpServer uses a synchronous timer which is activated every 100 ms. Such approach works perfectly when running macros in an interactive ROOT session.

If an application runs in compiled code and does not contain gSystem->ProcessEvents() calls, two method are available.

1.6.1   Asynchronous timer

The first method is to configure an asynchronous timer for the server, like for example:

```
serv->SetTimer(100, kFALSE);
```

Then, the timer will be activated even without any gSystem->ProcessEvents() method call. The main advantage of such method is that the application code can be used without any modifications. But there is no control when access to the application data is performed. It could happen just in-between of **TH1::Fill()** calls and an histogram object may be incomplete. Therefore such method is not recommended.

1.6.2   Regular calls of THttpServer::ProcessRequests() method

The second method is preferable - one just inserts in the application regular calls of the THttpServer::ProcessRequests() method, like:

```
serv->ProcessRequests();
```

In such case, one can fully disable the timer of the server:

```
serv->SetTimer(0, kTRUE);
```

1.7 Data access from command shell

The big advantage of the http protocol is that it is not only supported in web browsers, but also in many other applications. One could use http requests to directly access ROOT objects and data members from any kind of scripts.

If one starts a server and register an object like for example:

```
root [1]  serv = new THttpServer("http:8080");
root [2]  TNamed* n1 = new TNamed("obj", "title");
root [3]  serv->Register("subfolder", n1);
```

One could request a JSON representation of such object with the command:

```
[shell] wget http://localhost:8080/Objects/subfolder/obj/root.json
```

Then, its representation will look like:

```
{
   "_typename" : "TNamed",
   "fUniqueID" : 0,
   "fBits" : 50331656,
   "fName" : "obj",
   "fTitle" : "title"
}
```

The following requests can be performed:

- `root.bin` - binary data produced by object streaming with TBufferFile
- `root.json` - ROOT JSON representation for object and objects members
- `root.xml` - ROOT XML representation
- `root.png` - PNG image (if object drawing implemented)
- `root.gif` - GIF image
- `root.jpeg` - JPEG image
- `exe.json` - method execution in the object
- `exe.bin` - method execution, return result in binary form
- `cmd.json` - command execution
- `item.json` - item (object) properties, specified on the server
- `multi.json` - perform several requests at once
- `multi.bin` - perform several requests at once, return result in binary form

All data will be automatically zipped if '.gz' extension is appended. Like:

```
[shell] wget http://localhost:8080/Objects/subfolder/obj/root.json.gz
```

If the access to the server is restricted with htdigest, it is recommended to use the **curl** program since only curl correctly implements such authentication method. The command will look like:

```
[shell] curl --user "accout:password" http://localhost:8080/Objects/subfolder/obj/root.json --digest -o ro
```

1.7.1 Objects data access in JSON format

Request `root.json` implemented with TBufferJSON class. TBufferJSON generates such object representation, which could be directly used in JSROOT for drawing. `root.json` request returns either complete object or just object member like:

```
[shell] wget http://localhost:8080/Objects/subfolder/obj/fTitle/root.json
```

The result will be: "title".

For the `root.json` request one could specify the 'compact' parameter, which allow to reduce the number of spaces and new lines without data lost. This parameter can have values from '0' (no compression) till '3' (no spaces and new lines at all).

Usage of `root.json` request is about as efficient as binary `root.bin` request. Comparison of different request methods with TH1 object shown in the table:

Request	Size
root.bin	1658 bytes
root.bin.gz	782 bytes
root.json	7555 bytes
root.json?compact=3	5381 bytes
root.json.gz?compact=3	1207 bytes

One should remember that JSON representation always includes names of the data fields which are not present in the binary representation. Even then the size difference is negligible.

`root.json` used in JSROOT to request objects from THttpServer.

1.7.2   Generating images out of objects

For the ROOT classes which are implementing Draw method (like TH1 or TGraph) one could produce images with requests: `root.png`, `root.gif`, `root.jpeg`. For example:

```
wget "http://localhost:8080/Files/hsimple.root/hpx/root.png?w=500&h=500&opt=lego1" -O lego1.png
```

For all such requests one could specify following parameters:

- `h` - image height
- `w` - image width
- `opt` - draw options

1.7.3   Methods execution

By default THttpServer starts in monitoring (read-only) mode and therefore forbid any methods execution. One could specify read-write mode when server is started:

```
serv = new THttpServer("http:8080;rw");
```

Or one could disable read-only mode with the call:

```
serv->SetReadOnly(kFALSE);
```

Or one could allow access to the folder, object or specific object methods with:

```
serv->Restrict("/Histograms", "allow=admin"); // allow full access for user with 'admin' accout
serv->Restrict("/Histograms/hist1", "allow=all"); // allow full access for all users
serv->Restrict("/Histograms/hist1", "allow_method=Rebin"); // allow only Rebin method
```

'exe.json' accepts following parameters: - `method` - name of method to execute - `prototype` - method prototype (see TClass::GetMethodWithPrototype for details) - `compact` - compact parameter, used to compress return value - `_ret_object_` - name of the object which should be returned as result of method execution (used together with remote TTree::Draw call)

Example of retrieving object title:

```
[shell] wget 'http://localhost:8080/Objects/subfolder/obj/exe.json?method=GetTitle' -O title.json
```

Example of TTree::Draw method execution:

```
[shell] wget 'http://localhost:8080/Files/job1.root/ntuple/exe.json?method=Draw&prototype="Option_t*"&opt=
```

One also used `exe.bin` method - in this case results of method execution will be returned in binary format. In case when method returns temporary object, which should be delete at the end of command execution, one should specify `_destroy_result_` parameter in the URL string:

```
[shell] wget 'http://localhost:8080/Objects/subfolder/obj/exe.json?method=Clone&_destroy_result_' -O clone
```

If method required object as argument, it could be posted in binary or XML format as POST request. If binary form is used, one should specify following parameters:

```
[shell] wget 'http://localhost:8080/hist/exe.json?method=Add&h1=_post_object_&_post_class_=TH1I&c1=10' --p
```

Here is important to specify post object class, which is not stored in the binary buffer. When used XML form (produced with TBufferXML::ConvertToXML) method, only string with XML code could be specified:

```
[shell] wget 'http://localhost:8080/hist/exe.json?method=Add&h1=_post_object_xml_&c1=10' --post-file=h.xml
```

To get debug information about command execution, one could submit `exe.txt` request with same arguments.

1.7.4 Commands execution

If command registered to the server:

```
serv->RegisterCommand("/Folder/Start", "DoSomthing()");
```

It can be invoked with `cmd.json` request like:

```
[shell] wget http://localhost:8080/Folder/Start/cmd.json -O result.txt
```

If command fails, `false` will be returned, otherwise result of gROOT->ProcessLineSync() execution.

1.7.5 Performing multiple requests at once

To minimize traffic between sever and client, one could submit several requests at once. This is especially useful when big number of small objects should be requestsed simultaneosely. For this purposes `multi.bin` or `multi.json` requests could be used. Both require string as POST data which format as:

```
subfolder/item1/root.json\n
subfolder/item2/root.json\n
subfolder/item1/exe.json?method=GetTitle\n
```

If such requests saved in 'req.txt' file, one could submit it with command:

```
[shell] wget http://localhost:8080/multi.json?number=3 --post-file=req.txt -O result.json
```

For `multi.json` request one could use only requests, returning JSON format (like `root.json` or `exe.json`). Result will be JSON array. For `multi.bin` any kind of requests can be used. It returns binary buffer with following content:

```
[size1 (little endian), 4 bytes] + [request1 result, size1 bytes]
[size2 (little endian), 4 bytes] + [request2 result, size2 bytes]
[size3 (little endian), 4 bytes] + [request3 result, size3 bytes]
```

While POST data in request used to transfer list of multiple reqeusts, it is not possible to submit such kind of requests, which themselvs require data from POST block.