Contents

JavaScript ROOT

*** Sergey Linev GSI, Darmstadt ***

Chapter 1

JavaScript ROOT

The JSROOT project allows: - reading of binary and JSON ROOT files in JavaScript; - drawing of different ROOT classes in web browsers; - reading and drawing TTree data; - using in node.js.

1.1 Installing JSROOT

In most practical cases it is not necessary to install JSROOT - it can be used directly from project web sites https://root.cern/js/ and https://jsroot.gsi.de/.

When required, there are following alternatives to install JSROOT on other web servers:

- download and unpack provided packages (recommended)
- use npm package manager and invoke `npm install jsroot`
- clone master branch from repository

1.2 Drawing objects in JSROOT

The main page of the JSROOT project provides the possibility to interactively open ROOT files and draw objects like histogram or canvas.

To automate files loading and objects drawing, one can provide number of URL parameters in address string like:

- file - name of the file, which will be automatically open with page loading
- files - array of file names for loading
- json - name of JSON file with stored ROOT object like histogram or canvas
- item - item name to be displayed
- opt - drawing option for the item
- items - array of items name to be displayed
- opts - array of drawing options for the items
- expand - item name(s) to be expanded in the hierarchy browser
- focus - item name to be focused on in the hierarchy browser
- title - set browser title
- dir - list files in directory on http server, see https://github.com/root-project/jsroot/issues/283
- layout - can be 'simple', 'flex', 'tabs', 'gridNxM', 'horizNMK', 'vertNMK'
- browser - layout of the browser 'fix' (default), 'float', 'no' (hidden), 'off' (fully disabled)
- nobrowser - do not display file browser (same as browser=no)
- float - display floating browser (same as browser=float)
- status - configure status line 'no' (default), 'off' (completely disable), 'size'
- inject - name of extra JavaScript to load, see several examples in demo/ subdir
- optimize - drawing optimization 0:off, 1:only large histograms (default), 2:always
- palette - id of default color palette, 51..121 - new ROOT6 palette (default 57)
- interactive - enable/disable interactive functions 0 - disable all, 1 - enable all
- noselect - hide file-selection part in the browser (only when file name is specified)
- mathjax - use MathJax for latex output
- latex - 'off', 'symbols', 'normal', 'mathjax', 'alwaysmath' control of TLatex processor
- style - name of TStyle object to define global JSROOT style
- toolbar - show canvas tool buttons 'off', 'on' and 'popup', 'left' or 'right' for position, 'vert' for vertical
- divsize - fixed size in pixels for main div element like &dvisize=1500x800
- canvsize - default canvas size in pixels like &canvsize=1200x800

- optstat - settings for stat box, default 1111 (see TStyle::SetOptStat)
- optfit - fit parameters settings for stat box, default 0 (see TStyle::SetOptFit)
- statfmt - formatting for float values in stat box, default 6.4g (see TStyle::SetStatFormat)
- fitfmt - formatting for fit values in stat box, default 5.4g (see TStyle::SetFitFormat)
- optdate - plot specified date on the canvas, 1 - current time, 2 - file creation date, 3 - file modification date
- utc - select timeZone to 'UTC'
- datex - X position of date
- datey - Y position of date
- optfile - plot file name on the canvas, 1 - file name, 2 - full file URL, 3 - object item name
- opttitle - disable/enable drawing of object title in the canvas
- nomenu - disable context menu
- notouch - disable touch events handling
- progress - switch progress display mode between 'off', 'on' and 'modal'

For instance:

- https://root.cern/js/latest/?file=../files/hsimple.root&item=hpx;1
- https://root.cern/js/latest/?file=../files/hsimple.root&nobrowser&item=hpxpy;1&opt=colz
- https://root.cern/js/latest/?file=../files/hsimple.root&noselect&layout=grid2x2&item=hprof;1

Following layouts are supported:

- simple - available space used for single object (default)
- flex - creates as many frames as necessary, each can be individually moved/enlarged
- tabs - tabs for each object drawing
- gridNxM - fixed-size grid with NxM frames
- vertN - N frames sorted in vertical direction (like gridi1xN)
- horizN - N frames sorted in horizontal direction (like gridiNx1)
- vert121 - 3 frames sorted in vertical direction, second frame divided on two sub-frames
- horiz32_12 - 2 horizontal frames with 3 and 2 subframes, and 1/3 and 2/3 as relative size

When specifying `files`, `items` or `opts` parameters, array of strings could be provided like `files=['file1.root','file2.root']`. One could skip quotes when specifying elements names `items=[file1.root/hpx,file2.root/hpy]` or `opts=['',colz]`.

As item name, URL to existing image can be provided like `item=img:http://server/image.png`. Such image will be just inserted in the existing layout. One could specify option `"scale"` to automatically scale image to available space.

Many examples of URL string usage can be found on JSROOT API examples page.

One can very easy integrate JSROOT graphic into arbitrary HTML pages using a **iframe** tag:

```
<iframe width="700" height="400"
        src="https://root.cern/js/latest/?nobrowser&file=https://root.cern/js/files/hsimple.root&item=hpxp
</iframe>
```

1.3 Supported ROOT classes by JSROOT

List of supported classes and draw options:

- TH1 : hist, p, p0, *, l, lf2, a, e, e0, e1, e1x0, e3, e4, lego, text, X+Y+
- TH2 : scat, col, colz, box, box1, text, lego, arr, cont, cont1, cont2, cont3, cont4, surf, surf1, surf2, surf3, surf4, surf6, surf7, lego, lego0, lego1, lego2, lego3, lego4
- TH2Poly : col, lego, europe, usa
- TH3 : scat, box, box1
- TProfile : dflt, e, e1, pe2, hist, text, texte
- TProfile2D : example
- THStack : example
- TF1 : example
- TF2 : example
- TSpline : example
- TGraph : dflt, L, P, *, B, RX, RY
- TGraphErrors : dflt, l, lx, z, >, |>, ||, [], 0, 2, 3, 4, 5
- TGraphAsymmErrors : dflt,
- TGraphMultiErrors : docu, z and other from TGraphErrors
- TGraphPolar : example
- TMultiGraph : example, exclusion
- TGraph2D : example

- TEfficiency : docu
- TLatex : example
- TMathText : example
- TCanvas : example
- TPad : example
- TRatioPlot : example
- TLegend : example
- TTree : single-branch draw
- TPolyLine : dflt
- TGaxis : dflt
- TEllipse : dflt
- TArrow : dflt
- TPolyMarker3D: dflt

More examples of supported classes can be found on: https://root.cern/js/latest/examples.htm

There are special JSROOT draw options which only can be used with for `TCanvas` or `TPad` objects:

- logx - enable log10 scale for X axis
- logy - enable log10 scale for Y axis
- logz - enable log10 scale for Z axis
- log - enable log10 scale for X,Y,Z axes
- log2x - enable log2 scale for X axis
- log2y - enable log2 scale for Y axis
- log2z - enable log2 scale for Z axis
- log2 - enable log2 scale for X,Y,Z axes
- gridx - enable grid for X axis
- gridy - enable grid for X axis
- grid - enable grid for X and Y axes
- tickx - enable ticks for X axis
- ticky - enable ticks for X axis
- tick - enable ticks for X and Y axes
- rx - reverse X axis
- ry - reverse Y axis
- rotate - rotate frame
- fixframe - disable interactive moving of the frame
- nozoomx - disable zooming on X axis
- nozoomy - disable zooming on Y axis
- cpXY - create palette XY for the canvas like cp50
- nopalette - ignore palette stored with TCanvas
- nocolors - ignore colors list stored with TCanvas
- lcolors - use only locally colors list stored with TCanvas
- nomargins - clear frame margins

1.4 Superimposing draw objects

In the URL string one could use “+” sign to specify objects superposition:

- item=hpx+hprof

With similar syntax one could specify individual draw options for superimposed objects

- item=hpx+hprof&opt=logy+hist

Here “logy” option will be used for “hpx1” item and “hist” option for “hprof;1” item.

While draw option can include “+” sign itself, for superposition one could specify arrays of items and draw options like:

- item=[hpx;1,hprof;1]&opt=[logy,hist]

1.5 TTree draw

JSROOT provides possibility to display TTree data, using TTree::Draw syntax:

- opt=px
- opt=px:py
- opt=px:py:pz

It is also possible to use branch by id number specifying name like “br_0”, “br_1” and so on:

- opt=br_0:br_1

Histogram ranges and binning defined after reading first 1000 entries from the tree. Like in ROOT, one could configure histogram binning and range directly:

- opt=px:py>>h(50,-5,5,50,-5,5)

One and two dimensional draw expressions can be resulted into TGraph object, using “>>Graph” as output:

- opt=px:py>>Graph

For any integer value one can accumulate histogram with value bits distribution, specifying as output “>>bits(16)” or “>>bits”:

- opt=event.fTracks.fBits>>bits

There is special handling of TBits objects:

- opt=event.fTriggerBits

It is allowed to use different expressions with branch values:

- opt=px+py:px-py

Such expression can include arithmetical operations and all methods, provided in JavaScript Math class:

- opt=Math.abs(px+py)

In the expression one could use “Entry”*and*”*Entries*” variables.

One also could specify cut condition, separating it with “::” from the rest draw expression like:

- opt=px:py::pz>5

Contrary to the normal ROOT, JSROOT allows to use “(expr?res1:res2)” operator (placed into brackets):

- opt=px:py::(pz>5?2:1)

It is possible to “dump” content of any branch (by default - first 10 entries):

- item=ntuple/px&opt=dump

Or one could dump values produced with draw expression (also first 10 entries by default):

- opt=px:py::pz>>dump

Working with array indexes is supported. By default, all elements in array are used for the drawing. One could specify index for any array dimension (-1 means last element in the array). For instance, dump last element from `event.fTracks` array:

- opt=event.fTracks[-1].fBits>>dump

For any array or collection kind one could extract its size with expression:

- opt=event.fTracks.@size

At the end of expression one can add several parameters with the syntax:

`<draw_expession>;par1name:par1value;par2name:par2value`

Following parameters are supported: - “first” - id of the first entry to process - “entries” - number of entries to process - “monitor” - periodically show intermediate draw results (interval in milliseconds) - “maxrange” - maximal number of ranges in single HTTP request - “accum” - number of accumulated values before creating histogram - “htype” - last letter in histogram type like “I”, “F”, “D”, “S”, “L”, “C” - “hbins” - number of bins on each histogram axis - “drawopt” - drawing option for produced histogram - “graph” - draw into TGraph object

Example - opt=event.fTracks[].fTriggerBits;entries:1000;first:200;maxrange:25

1.6 Geometry viewer

JSROOT implements display of TGeo objects like:

- file=rootgeom.root&item=simple1
- file=building.root&item=geom&opt=z

Following classes are supported by geometry viewer: - TGeoVolume - TGeoNode - TGeoManager (master volume will be displayed) - TEveGeoShapeExtract (used in EVE)

Following draw options could be specified (separated by semicolon or ';'): - axis - draw axis coordinates - z - set z axis direction up (normally y axis is up and x looks in user direction) - clipx/clipy/clipz - enable correspondent clipping panel - clip or clipxyz - enable all three clipping panels - ssao - enable Smooth Lighting Shader (or Screen Space Ambient Occlusion) - wire - instead of filled surfaces only wireframe will be drawn - vislvlN - maximal hierarchy depth of visible nodes (like vislvl6) - moreN - show N times more volumes as usual (normally ~10000 nodes and ~200000 elementary faces are shown) - all - try to display all geometry volumes (may lead to browser hanging) - maxnodesN - configure maximal number of rendered nodes (like maxnodes100K) - maxfacesN - configure maximal number of rendered faces (like maxfaces3M) - highlight - force highlighting of selected volume, normally activated for moderate-size geometries - nohighlight - disable volumes highlighting (can be activated via context menu) - hscene - enable highlight of extra objects like tracks or hits - hsceneonly - enable only highlight of extra objects like tracks or hits - nohscene - disable highlight of extra objects like tracks or hits - macro:name.C - invoke ROOT configuration macro - dflt - set default volumes colors as TGeoManager::DefaultColors() does - transpXY - set global transparency value (XY is number between 1 and 99) - zoomFACTOR - set initial zoom factor (FACTOR is integer value from 1 to 10000, default is 100) - rotyANGLE - set Y rotation angle in degrees (like roty10) - rotzANGLE - set Z rotation angle in degrees (like rotz20) - rotate - enable automatic rotation of the geometry - trzVALUE - set transformation along Z axis (like trz50) - trrVALUE - set radial transformation (like trr100) - ortho_camera - use THREE.OrthographicCamera without possibility to rotate it - ortho_camera_rotate - use THREE.OrthographicCamera and enable it rotation - ctrl - show control UI from the beginning - tracks - show tracks from TGeoManager - showtop - show top-level volume of TGeoManager (default off) - no_screen - let ignore kVisOnScreen bits for nodes visibility - dray - calculate rendering order using raytracing (extensive calculations) - dbox - use distance to nearest point from bounding box for rendering order (default) - dpnt - use distance to shape center as rendering order - dsize - use volume size as rendering order - ddflt - let three.js to calculate rendering order - comp - show left and right components of TGeoCompositeShape - compx - show all sub-components of TGeoCompositeShape

In the URL string several global settings can be changed:

- geosegm - grads per segment is cylindrical shapes, default is 6
- geocomp - compress results of composite shape production, default is true

It is possible to display only part of geometry model. For instance, one could select sub-item like:

- file=rootgeom.root&item=simple1/TOP/REPLICA_1

Or one can use simple selection syntax (work only with first-level volumes):

- item=simple1&opt=-bar1-bar2

Syntax uses '+' sign to enable visibility flag of specified volume and '-' sign to disable visibility. One could use wildcard symbol like '+TUBE1*'.

Another way to configure visibility flags is usage of ROOT macros, which typically looks like:

```
{
   TGeoManager::Import("http://root.cern/files/alice2.root");
   gGeoManager->DefaultColors();
   //   gGeoManager->SetVisLevel(4);
   gGeoManager->GetVolume("HALL")->InvisibleAll();
   gGeoManager->GetVolume("ZDCC")->InvisibleAll();
   gGeoManager->GetVolume("ZDCA")->InvisibleAll();
   //  ...
   gGeoManager->GetVolume("ALIC")->Draw("ogl");
   new TBrowser;
}
```

Example of such macro can be found in root tutorials.

From provided macro only following calls will be executed in JSROOT:

- `gGeoManager->DefaultColors()`
- `gGeoManager->GetVolume("HALL")->InvisibleAll()`
- `gGeoManager->GetVolume("HALL")->SetTransparency(30)`
- `gGeoManager->GetVolume("HALL")->SetLineColor(5)`
- `gGeoManager->GetVolume("ALIC")->Draw("ogl")`

All other will be ignored.

Example of major LHC detectors: * ALICE: full * ATLAS: full, cryo, sctt * CMS: cmse, calo * LHCb: full

Other detectors examples: * HADES: full, preselected * BABAR: full, emca * STAR: full, svtt * D0: full * NA47: full * BRAHMS: full * SLD: full

Together with geometry one could display tracks (TEveTrack) and hits (TEvePointSet, TPolyMarker3D) objects. Either one do it interactively by drag and drop, or superimpose drawing with + sign like:

- item=simple_alice.json.gz+tracks_hits.root/tracks+tracks_hits.root/hits

There is a problem of correct rendering of transparent volumes. To solve problem in general is very expensive (in terms of computing power), therefore several approximation solution can be applied: * **dpnt**: distance from camera view to the volume center used as rendering order * **dbox**: distance to nearest point from bonding box used as rendering order (**default**) * **dsize**: volume size is used as rendering order, can be used for centered volumes with many shells around * **dray**: use raycasting to sort volumes in order they appear along rays, coming out of camera point * **ddflt**: default three.js method for rendering transparent volumes For different geometries different methods can be applied. In any case, all opaque volumes rendered first.

1.7 Reading ROOT files from other servers

In principle, one could open any ROOT file placed in the web, providing the full URL to it like:

- https://jsroot.gsi.de/latest/?file=https://root.cern/js/files/hsimple.root&item=hpx

But one should be aware of Same-origin policy, when the browser blocks requests to files from domains other than current web page. To enable CORS on Apache web server, hosting ROOT files, one should add following lines to `.htaccess` file:

```
<IfModule mod_headers.c>
  <FilesMatch "\.root">
     Header set Access-Control-Allow-Origin "*"
     Header set Access-Control-Allow-Headers "range"
     Header set Access-Control-Expose-Headers "content-range,content-length,accept-ranges"
     Header set Access-Control-Allow-Methods "GET"
  </FilesMatch>
</IfModule>
```

More details about configuring of CORS headers can be found here.

Alternative - enable CORS requests in the browser. It can be easily done with CORS Everywhere plugin for the Firefox browser or Allow CORS plugin for the Chrome browser.

Next solution - install JSROOT on the server hosting ROOT files. In such configuration JSROOT does not issue CORS requests, therefore server and browsers can be used with their default settings. A simplified variant of such solution - copy only the top index.htm file from JSROOT package and specify the full path to `modules/gui.mjs` script like:

```
<script type="module">
   import { openFile, draw } from 'https://root.cern/js/latest/modules/gui.mjs';
   // ...
</script>
```

In the main `<div>` element one can specify many custom parameters like one do it in URL string:

```
<div id="simpleGUI" path="files/path" files="userfile1.root;subdir/usefile2.root">
   loading scripts ...
</div>
```

1.8 Reading local ROOT files

JSROOT can read files from local file system using HTML5 FileReader functionality. Main limitation here - user should interactively select files for reading. There is button **"..."** on the main JSROOT page, which starts file selection dialog. If valid ROOT file is selected, JSROOT will be able to normally read content of such file.

1.9 JSROOT with THttpServer

THttpServer provides http access to objects from running ROOT application. JSROOT is used to implement the user interface in the web browsers.

The layout of the main page coming from THttpServer is very similar to normal JSROOT page. One could browse existing items and display them. A snapshot of running server can be seen on the demo page.

One could also specify similar URL parameters to configure the displayed items and drawing options.

It is also possible to display one single item from the THttpServer server like:

https://root.cern/js/latest/httpserver.C/Files/job1.root/hpxpy/draw.htm?opt=colz

1.10 Data monitoring with JSROOT

1.10.1 Monitoring with http server

The best possibility to organize the monitoring of data from a running application is to use THttpServer. In such case the client can always access the latest changes and request only the items currently displayed in the browser. To enable monitoring, one should activate the appropriate checkbox or provide **monitoring** parameter in the URL string like:

https://root.cern/js/latest/httpserver.C/Files/job1.root/hprof/draw.htm?monitoring=1000

The parameter value is the update interval in milliseconds.

1.10.2 JSON file-based monitoring

Solid file-based monitoring (without integration of THttpServer into application) can be implemented in JSON format. There is the TBufferJSON class, which is capable to convert any (beside TTree) ROOT object into JSON. Any ROOT application can use such class to create JSON files for selected objects and write such files in a directory, which can be accessed via web server. Then one can use JSROOT to read such files and display objects in a web browser.

There is a demonstration page showing such functionality: https://root.cern/js/latest/demo/update_draw.htm. This demo page reads in cycle 20 json files and displays them.

If one has a web server which already provides such JSON file, one could specify the URL to this file like:

https://root.cern/js/latest/demo/update_draw.htm?addr=../httpserver.C/Canvases/c1/root.json.gz

Here the same problem with Cross-Origin Request can appear. If the web server configuration cannot be changed, just copy JSROOT to the web server itself.

1.10.3 Binary file-based monitoring (not recommended)

Theoretically, one could use binary ROOT files to implement monitoring. With such approach, a ROOT-based application creates and regularly updates content of a ROOT file, which can be accessed via normal web server. From the browser side, JSROOT could regularly read the specified objects and update their drawings. But such solution has three major caveats.

First of all, one need to store the data of all objects, which only potentially could be displayed in the browser. In case of 10 objects it does not matter, but for 1000 or 100000 objects this will be a major performance penalty. With such big amount of data one will never achieve higher update rate.

The second problem is I/O. To read the first object from the ROOT file, one need to perform several (about 5) file-reading operations via http protocol. There is no http file locking mechanism (at least not for standard web servers), therefore there is no guarantee that the file content is not changed/replaced between consequent read operations. Therefore, one should expect frequent I/O failures while trying to monitor data from ROOT binary files. There is a workaround for the problem - one could load the file completely and exclude many partial I/O operations by this. To achieve this with JSROOT, one should add “+” sign at the end of the file name. Of course, it only could work for small files.

If somebody still wants to use monitoring of data from ROOT files, could try link like:

- https://root.cern/js/latest/?nobrowser&file=../files/hsimple.root+&item=hpx;1&monitoring=2000

In this particular case, the histogram is not changing.

1.11 JSROOT API

JSROOT can be used in arbitrary HTML pages to display data, produced with or without ROOT-based applications.

Many different examples of JSROOT API usage can be found on JSROOT API examples page.

1.11.1 Import JSROOT functionality

Major JSROOT functions are located in `main.mjs` module and can be imported like:

```
<script type='module'>
   import { openFile, draw } from 'https://root.cern/js/latest/modules/main.mjs';
   let filename = "https://root.cern/js/files/hsimple.root";
   let file = await openFile(filename);
   let obj = await file.readObject("hpxpy;1");
   await draw("drawing", obj, "colz");
</script>
```

Here the default location `https://root.cern/js/latest/` is specified. One always can install JSROOT on private web server. When JSROOT is used with THttpServer, the address looks like:

```
<script type='module'>
   import { httpRequest, draw } from 'http://your_root_server:8080/jsrootsys/modules/main.mjs';
   let obj = await httpRequest('http://your_root_server:8080/Objects/hist/root.json','object');
   await draw('drawing', obj, 'hist');
</script>
```

Loading main module is enough to get public JSROOT functionality - reading files and drawing objects. One also can load some special components directly like:

```
<script type='module'>
   import { HierarchyPainter } from 'https://root.cern/js/latest/modules/gui.mjs';

   let h = new HierarchyPainter("example", "myTreeDiv");

   // configure 'simple' in provided <div> element
   // one also can specify "grid2x2" or "flex" or "tabs"
   h.setDisplay("simple", "myMainDiv");

   // open file and display element
   await h.openRootFile('../../files/hsimple.root');
   await h.display('hpxpy;1',"colz');
</script>
```

After script loading one can configure different parameters in `gStyle` object. It is instance of the `TStyle` object and behaves like `gStyle` variable in ROOT. For instance, to change stat format using to display value in stats box:

```
import { gStyle } from 'https://root.cern/js/latest/modules/main.mjs';
gStyle.fStatFormat = '7.5g';
```

There is also `settings` object which contains all other JSROOT settings. For instance, one can configure custom format for different axes:

```
import { settings } from 'https://root.cern/js/latest/modules/main.mjs';
settings.XValuesFormat = '4.2g';
settings.YValuesFormat = '6.1f';
```

One also can use `build/jsroot.js` bundle to load all functionality at one and access it via `JSROOT` global handle:

```
<script src="https://root.cern/js/latest/build/jsroot.js"></script>
<script>
   // getting json string from somewhere
   let obj = JSROOT.parse(root_json);
   JSROOT.draw('plain', obj, 'colz');
</script>
```

1.11.2 Use of JSON

It is strongly recommended to use JSON when communicating with ROOT application. THttpServer provides a JSON representation for every registered object with an url address like:

```
http://your_root_server:8080/Canvases/c1/root.json
```

Such JSON representation generated using the TBufferJSON class. One could create JSON file for any ROOT object directly, just writing in the code:

```
obj->SaveAs("file.json");
```

To access data from a remote web server, it is recommended to use the `httpRequest` method. For instance to receive object from a THttpServer server one could do:

```
import { httpRequest } from 'https://root.cern/js/latest/modules/main.mjs';
let obj = await httpRequest("http://your_root_server:8080/Canvases/c1/root.json", "object")
console.log('Read object of type', obj._typename);
```

Function returns Promise, which provides parsed object (or Error in case of failure).

If JSON string was obtained by different method, it could be parsed with `parse` function:

```
import { parse } from 'https://root.cern/js/latest/modules/main.mjs';
let obj = parse(json_string);
```

1.11.3 Objects drawing

After an object has been created, one can directly draw it. If HTML page has `<div>` element:

```
<div id="drawing"></div>
```

One could use the `draw` function:

```
import { draw } from 'https://root.cern/js/latest/modules/main.mjs';
draw("drawing", obj, "colz");
```

The first argument is the id of the HTML div element, where drawing will be performed. The second argument is the object to draw and the third one is the drawing option.

Here is complete running example ans source code:

```
import { httpRequest, draw, redraw, resize, cleanup } from 'https://root.cern/js/latest/modules/main.mjs';
let filename = "https://root.cern/js/files/th2ul.json.gz";
let obj = await httpRequest(filename, 'object');
draw("drawing", obj, "lego");
```

In very seldom cases one need to access painter object, created in `draw()` function. This can be done via handling Promise results like:

```
let painter = await draw("drawing", obj, "colz");
console.log('Object type in painter', painter.getClassName());
```

One is also able to update the drawing with a new version of the object:

```
// after some interval request object again
redraw("drawing", obj2, "colz");
```

The `redraw` function will call `draw` if the drawing was not performed before.

In the case when changing of HTML layout leads to resize of element with JSROOT drawing, one should call `resize()` to let JSROOT adjust drawing size. One should do:

```
resize("drawing");
```

As second argument one could specify exact size for draw elements like:

```
resize("drawing", { width: 500, height: 200 });
```

To correctly cleanup JSROOT drawings from HTML element, one should call:

```
cleanup("drawing");
```

1.11.4 File API

JSROOT defines the TFile class, which can be used to access binary ROOT files. One should always remember that all I/O operations are asynchronous in JSROOT. Therefore promises are used to retrieve results when the I/O operation is completed. For example, reading an object from a file and displaying it will look like:

```
import { openFile, draw } from 'https://root.cern/js/latest/modules/main.mjs';
let filename = "https://root.cern/js/files/hsimple.root";
let file = await openFile(filename);
let obj = await file.readObject("hpxpy;1");
await draw("drawing", obj, "colz");
console.log('drawing completed');
```

Here is running example and source code

1.11.5 TTree API

Simple TTree::Draw operation can be performed with following code:

```
import { openFile } from 'https://root.cern/js/latest/modules/io.mjs';
import { draw } from 'https://root.cern/js/latest/modules/draw.mjs';
let file = await openFile("https://root.cern/js/files/hsimple.root");
let tree = await file.readObject("ntuple;1");
draw("drawing", tree, "px:py::pz>5");
```

To get access to selected branches, one should use `TSelector` class:

```
import { openFile } from 'https://root.cern/js/latest/modules/io.mjs';
import { draw } from 'https://root.cern/js/latest/modules/draw.mjs';
import { TSelector, treeProcess } from 'https://root.cern/js/latest/modules/tree.mjs';

let file = await openFile("https://root.cern/js/files/hsimple.root");
let tree = await file.readObject("ntuple;1");
let selector = new TSelector();

selector.AddBranch("px");
selector.AddBranch("py");

let cnt = 0, sumpx = 0, sumpy = 0;

selector.Begin = function() {
   // function called before reading of TTree starts
}

selector.Process = function() {
   // function called for every entry
   sumpx += this.tgtobj.px;
   sumpy += this.tgtobj.py;
   cnt++;
}

selector.Terminate = function(res) {
   if (!res || (cnt === 0)) return;
   let meanpx = sumpx/cnt, meanpy = sumpy/cnt;
   console.log(`Results meanpx = ${meanpx} meanpy = ${meanpy}`);
}

await treeProcess(tree, selector);
```

Here is running example and source code

This examples shows how read TTree from binary file and create `TSelector` object. Logically it is similar to original TSelector class - for every read entry `TSelector::Process()` method is called. Selected branches can be accessed from **tgtobj** data member. At the end of tree reading `TSelector::Terminate()` method will be called.

As third parameter of treeProcess() function one could provide object with arguments

```
let args = { numentries: 1000, firstentry: 500 };
treeProcess(tree, selector, args);
```

1.11.6 TGeo API

Any supported TGeo object can be drawn directly with normal `draw()` function.

If necessary, one can create three.js model for supported object directly and use such model separately. This can be done with the function:

```
import { build } from './path_to_jsroot/modules/geom/TGeoPainter.mjs';
let opt = { numfaces: 100000 };
let obj3d = build(obj, opt);
scene.add( obj3d );
```

Following options can be specified:

- numfaces - approximate maximal number of faces in three.js model (default 100000)
- numnodes - approximate maximal number of meshes in three.js model (default 1000)
- doubleside - use double-side material (default only front side is set)
- wireframe - show wireframe for created object (default - off)
- dflt_colors - assign default ROOT colors for the volumes

When transparent volumes appeared in the model, one could use `produceRenderOrder()` function to correctly set rendering order. It should be used as:

```
import { produceRenderOrder } from './path_to_jsroot/modules/geom/TGeoPainter.mjs';
produceRenderOrder(scene, camera.position, 'box');
```

Following methods can be applied: “box”, “pnt”, “size”, “ray” and “dflt”. See more info in draw options description for TGeo classes.

Here is running example and source code.

1.11.7 Custom user class

There is code example how custom user class can be implemented. It shows usage of different draw options for the class and ability to access sub-elements of the object using specialized `expand` function.

1.11.8 Use with Node.js

To install latest JSROOT release, just do:

```
[shell] npm install jsroot
```

To use in the Node.js scripts, one should add following line:

```
import { httpRequest, makeSVG } from 'jsroot';
```

Using JSROOT functionality, one can open binary ROOT files (local and remote), parse ROOT JSON, create SVG output. For example, to create SVG image with lego plot, one should do:

```
import { openFile, makeSVG } from 'jsroot';
import { writeFileSync } from 'fs';

let file = await openFile("https://root.cern/js/files/hsimple.root");
let obj = await file.readObject("hpx;1");
let svg = await makeSVG({ object: obj, option: "lego2", width: 1200, height: 800 });
writeFileSync("lego2.svg", svg);
```

It is also possible to convert any JavaScript object into ROOT JSON string, using `toJSON()` function. Like:

```
import { toJSON, openFile, makeSVG } from 'jsroot';
import { writeFileSync } from 'fs';

let file = await openFile("https://root.cern/js/files/hsimple.root");
let obj = await file.readObject("hpx;1");
let json = await toJSON(obj);
writrFileSync("hpxpy.json", json);
```

Such JSON string could be parsed by any other JSROOT-based application.

When WebGL rendering is used (lego plots or TGeo drawing), on the Linux one need to have `DISPLAY` correctly set to make it working. To run JSROOT on headless machine, one have to use `xvfb-run` utility, see also here:

```
[shell] xvfb-run -s "-ac -screen 0 1280x1024x24" node geomsvg.js
```

1.11.9 Use with OpenUI5

OpenUI5 is a web toolkit for developers to ease and speed up the development of full-blown HTML5 web applications. JSROOT provides `loadOpenui5` function to load supported OpenUI5:

```
<script type="module">
   import { loadOpenui5 } from 'path_to_jsroot/modules/main.mjs';
   let sap = await loadOpenui5();
   sap.registerModulePath("NavExample", "./");
   new sap.m.App ({
      pages: [
```

```
        new sap.m.Page({
          title: "Nav Container",
          enableScrolling : true,
          content: [ new sap.ui.core.ComponentContainer({ name : "NavExample" })]
        })
      ]
   }).placeAt("content");
</script>
```

JSROOT uses https://openui5.hana.ondemand.com/1.128.0/ when no other source is specified.

There are small details when using OpenUI5 with THttpServer. First of all, location of JSROOT modules should be specified as `/jsrootsys/modules/main.mjs`. And then trying to access files from local disk, one should specify `/currentdir/` folder:

```
jQuery.sap.registerModulePath("NavExample", "/currentdir/");
```

JSROOT provides example showing usage of JSROOT drawing in the OpenUI5, source code can be found in repository.

1.11.10 Migration v6 -> v7

- Core functionality should be imported from `main.mjs` module like:

```
import { create, parse, createHistogram, redraw } from 'https://root.cern/js/7.0.0/modules/main.mjs';
```

- It is still possible to use `JSRoot.core.js` script, which provides very similar (but not identical!) functionality as with `v6` via global `JSROOT` object
- `JSROOT.define()` and `JSROOT.require()` functions only available after `JSRoot.core.js` loading
- Support of `require.js` and `openui5` loaders was removed
- Global hierarchy painter `JSROOT.hpainter` no longer existing, one can use `getHPainter` function:

```
import { getHPainter } from 'https://root.cern/js/7.0.0/modules/main.mjs';
let hpainter = getHPainter();
```

- All math functions previously available via `JSROOT.Math` should be imported from `base/math.mjs` module:

```
import * as math from 'https://root.cern/js/7.0.0/modules/base/math.mjs';
```

- Indication of batch mode `JSROOT.batch_mode` should be accessed via functions:

```
import { isBatchMode, setBatchMode } from 'https://root.cern/js/7.0.0/modules/main.mjs';
let was_batch = isBatchMode();
if (!was_batch) setBatchMode(true);
```

- `JSROOT.extend()` function was removed, use `Object.assign()` instead

1.11.11 Migration v5 -> v6

- Main script was renamed to `JSRoot.core.js`. Old `JSRootCore.js` was deprecated and removed in v6.2. All URL parameters for main script ignored now, to load JSROOT functionality one should use `JSROOT.require` function. To create standard GUI, `JSROOT.buildGUI` function has to be used.
- Instead of `JSROOT.JSONR_unref()` one can use `JSROOT.parse()`. If object is provided to `JSROOT.parse()` it just replaces all references which were introduced by `TBufferJSON::ToJSON()` method.
- Instead of `JSROOT.console()` one should use `console.log()`. Instead of `JSROOT.alert()` one should use `console.error()`.
- Many settings were moved from `JSROOT.gStyle` to `JSROOT.settings` object. It was done to keep only TStyle-related members in `JSROOT.gStyle`.
- Basic painter classes were renamed and made public:
  - `JSROOT.TBasePainter` -> `JSROOT.BasePainter`
  - `JSROOT.TObjectPainter` -> `JSROOT.ObjectPainter`
- Internal `ObjectPainter.DrawingReady` api was deprecated. Draw function has to return `Promise` if object drawing postponed. As argument of returned promise object painter has to be used.
- Many function names where adjusted to naming conventions. Like:
  - `JSROOT.CreateHistogram` -> `JSROOT.createHistogram`

- JSROOT.CreateTGraph -> JSROOT.createTGraph
- JSROOT.Create -> JSROOT.create