Yag rest http API documentation

Version of this document

If you found this document in your Yag installation, you might need to get the last version here: http://yagame.fr/documentation/

Généralités

Yag has an internal http server which will be referred to as “web server”, “http server”, or even simply “Yag” when the context is obvious.
We will therefore say “Yag is processing the request” to say exactly “Yag's internal http server is processing the request”.

This server is complete and can therefore manage a website.

Why a web server ?

Integrating a complete web server offers 2 important possibilities:

- We can create any site

- We can interface this server directly with Yag

The first possibility allows the use of standard web technologies (html / css / javascript ...) which greatly enhances the automation possibilities of Yag.

The second possibility is to query Yag to modify / retrieve internal game data such as pawns, character sheets, dice, etc.

These two combined possibilities make it possible to control certain features of Yag with JavaScript.

REST

Yag's query must of course follow a format.
The standard format for http requests is REST (REpresentational State Transfer).
In our context, in practice, this simply means that request URLs should be organized and readable in tree form.
For example, all API requests will start with /yagapi/
We will then specify the general context, for example:

- Pawn queries will start with /yagapi/pawn
- Queries regarding dice will start with /yagapi/dice

The more we advance in the request, the more precise we are.

For example the request /yagapi/dice/roll/init/current means:

- That we go to the Yag API (yagapi)
- That we are interested in the dice (dice)
- That we will ask for a roll
- That we will launch the initiative (init)

Purpose of the document

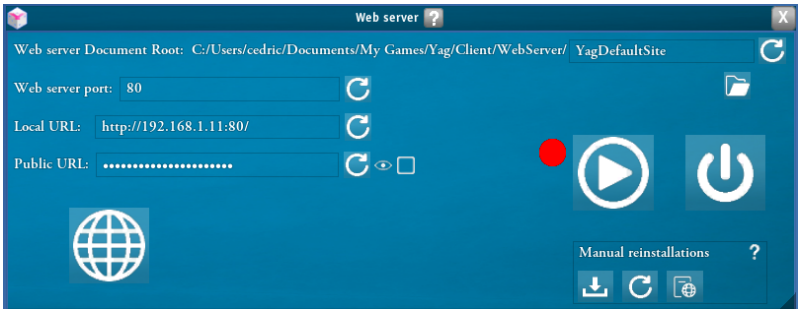
This document describes:

- the use of Yag's web server
- Yag's full REST API specifications

Web server

This server is complete and can therefore manage a website.
It's an integration of the mongoose web server.

It is managed in the "web server" window:

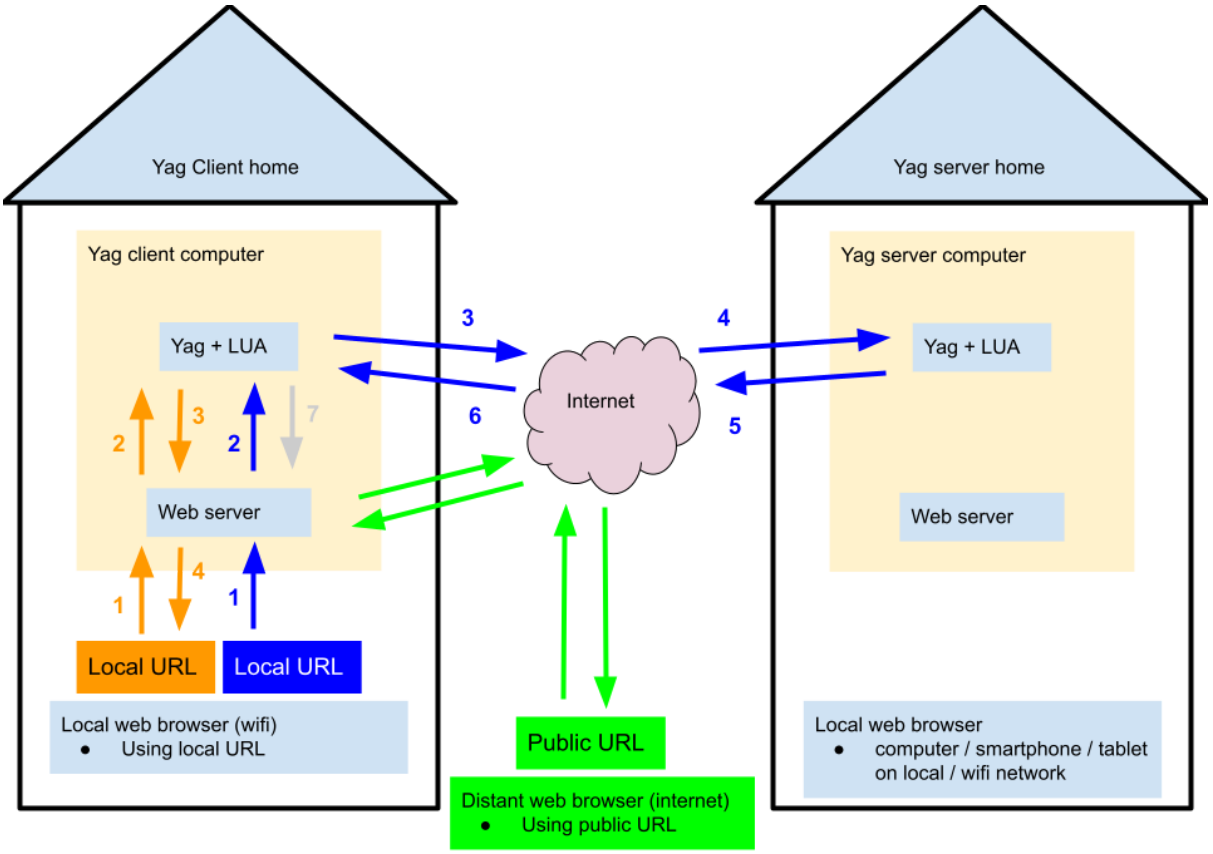


Consult the documentation pdf of this window for details of operation and use:

https://yagame.fr/documentation/

Global architecture

It might be helpful to understand the global architecture of how Yag works networkwise.
This section goes a bit into technical details and, while it's helpful to understand what's going on internally, it can be skipped with no problem if you're not that much into technique.



Without the web server

The essential thing to understand is that a Yag client doesn't store all the information about the game.
In particular, all the character sheets are kept on the server and the client can only get one at a time.
This is because the volume of data in the character sheets is too big to be shared entirely (this leads to crashes when there are too many character sheets).
So when a Yag client needs a character sheet, it sends an internal request to the Yag server which sends the CS back to the client (blue arrows 3, 4, 5, 6 on the picture).
All this is transparent to the user when not using the web server: Yag is quietly doing its internal cooking and the user is never aware of it.

With the web server

Things get a bit more complicated when using the web server.
This is because http is a stateless protocol and cannot maintain a connection.

Requesting local info

When the browser is requesting local data there is no problem.
This is illustrated in the orange arrows 1 to 4.
In the following, O1 means “Orange arrow 1”, etc.
O1- browser sends a REST API request to web server
O2- web server request the info to Yag
O3- Yag can answer instantly because the info is stored locally
O4- browser get its response in the same connection
This is why the REST API requests querying local info can return their results in the same request (and the results can be processed directly in the callback function of the request)

So in practice requesting local data takes only one web request.

Requesting distant info

When the browser is requesting distant data things are more complicated.
This is illustrated in the blue arrows 1 to 6
In the following, B1 means “Blue arrow 1”, etc.
B1- browser sends a REST API request to web server
B2- web server request the info to Yag
B3- Yag doesn’t have the info and sends a request to the Yag server
Because B3 is going through the net, many game frames can pass and the connexion B1 is lost: B8 doesn’t exist and B7 is useless (hence grayed).
B4, B5, B6- Yag server receives the request and sends the data back to Yag client.

At this point, Yag client has the info locally but can’t send it back to browser because B8 doesn’t exist anymore.

And because Yag cannot initiate a connection to the web browser, the web browser has to create a new request to get the (now) current info through a local cycle (orange arrows).

So in practice requesting distant data takes two web requests.

That is why when javascript needs a character sheet, it must be done in two requests:

- /yagapi/charactersheet/select/guid to select the target sheet and get its data from the server (blue arrows)
- /yagapi/charactersheet/get/current to get the (now) current sheet from the client (orange arrows)

See the javascript example to see how it works in practice.

Public and local URL

Incidentally, the above diagram also illustrates the difference in use between the local URL and the public URL in the web server window:

The local URL can be used on the internal network of the house (wifi, etc.), these are the orange arrows and the arrows B1 and B2.
The public URL must be used from the internet, these are the green arrows.

Apart from the fact that the incoming connection must be authorized on the computer (NAT, port forwarding, firewall, etc.), the green arrows are exactly equivalent to the orange arrows: as long as nothing is requested from the Yag server the green connection is not lost and functions as a local connection.

Yag and Javascript

In order to ensure that the defined LUA scripts are applied, Yag calls the LUA functions internally. The structures used for the Rest Javascript API are therefore modeled on those of the LUA.

JSON Objects in the API

The structures used for JavaScript are identical to those used for LUA, but in JSON format.
For example, the ps structure is defined as follows in the LUA API documentation:

```
{
  health = 10,
  initiative = 14,
  round = true,
  righthand = true,
  lefthand = true
}
```

In the JavaScript API it will therefore appear as a JSON object containing the same fields:

```
{
  “health” : 10,
  “initiative” : 14,
  “round” : true,
  “righthand” : true,
  “lefthand” : true
}
```

And identically for all structures used for the JavaScript API.
Consult the LUA API documentation for a full description of the available structures:
http://yagame.fr/wp-content/uploads/2019/02/yag_lua_api_fr.pdf

AJAX requests

Querying Yag from JavaScript is done asynchronously, with AJAX (Asynchronous Javascript And Xml) requests.

There is therefore no connected state: each request creates a new connection which disappears after the request is processed.

This operation imposes an important constraint: Yag cannot create a connection to the browser, it is always the browser that must initiate a request, and wait for Yag's response on the connection created for this request.

Yag cannot therefore signal to the browser to update itself outside the context of the connection created by a request.

For example for the journal, Yag cannot signal to the browser that it has been updated, it is up to the browser to create periodic requests (for example every second) to check if it needs to update its display from the journal.

Another example: if you select a pawn by clicking on it in Yag, you have no way to automatically update the display in the browser, you will have to either provide an update button or perform a periodic query, for example every second, so that the browser always knows which pawn is currently selected and possibly keeps its display up to date.

When you modify a result, you must call the update display function in the script. For example, when we launch the initiative in javascript, we must request the update of the pawn in the callback:

```
function DiceRollCSInit() {

    // new ajax request
    var xhr = new XMLHttpRequest();

    // callback preparation
    xhr.onreadystatechange = function () {
        if (xhr.readyState === 4 && xhr.status === 200) {
            PawnSheetGetById(document.getElementById("CSPawnId").value)
        }
    };

    // launching the ajax request
    xhr.open('POST', '/yagapi/dice/roll/init/current');
    xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded", "charset=UTF-8");
    var myFormData = "Guid=" + encode(document.getElementById("CSGuid").value);
    xhr.send(myFormData);
}
```

This is also the reason why the display of the character sheets is done in 2 steps. It is technically impossible to make a single request, like the following, to retrieve the records:

- /yagapi/charactersheet/get/guid

The retrieval of the character sheet is done on the Yag server, and it is also an asynchronous request between the Yag client and the Yag server. When the client requests the file from the server, the http connection ends and the Yag client can no longer send the file it just retrieved to the browser.
So we have 2 different requests:

- /yagapi/charactersheet/select/guid
- /yagapi/charactersheet/get/current

The first selects the form (behind the scenes, the Yag client requests it from the Yag server).
The second returns the current file (it is now present locally on the Yag client, which no longer needs to request it from the server).

In short, for anything that may change during the game (list of character sheets, diary, etc.), manual (with a dedicated button) or periodic (automated in Javascript) updates must be provided because Yag has no way to contact the browser, it can only respond.

Typical javascript function

Here is an example of a typical javascript function using the API

The function takes the pawn identifier (pid) as a parameter, generates an AJAX request with the corresponding Yag REST API (/yagapi/pawn/get/id), and updates the display by setting up a callback function.

```
function PawnSheetGetById(pid) {

   // creating the ajax request
   var xhr = new XMLHttpRequest();

   // preparing the callback
   xhr.onreadystatechange = function () {
      if (xhr.readyState === 4 && xhr.status === 200) {
         // decoding and analyzing the response
         var DecodedResponse = decode(xhr.responseText);
         var PawnInfoJson = JSON.parse(DecodedResponse);
         var PSJson = PawnInfoJson["ps"];

         // filling the HTML fields
         document.getElementById("CSRound").checked = PSJson.round;
         document.getElementById("CSCurrentHealth").value = PSJson.health;
         document.getElementById("CSInitResult").value = PSJson.initiative;
         document.getElementById("CSShowLeftHand").checked = PSJson.lefthand;
         document.getElementById("CSShowRightHand").checked = PSJson.righthand;

         document.getElementById("CSPawnId").value = pid;
      }
   };

   // preparing and launching the request
   var myFormData = "id=" + encode(pid);
   xhr.open('GET', '/yagapi/pawn/get/id?' + myFormData);
   xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded", "charset=UTF-8");
   xhr.send();
```

}

Processing chain of a request

With the above function, the complete processing of an API request is as follows:

1. *Browser (javascript):* sending the Rest API request /yagapi/pawn/get/id?id=#3
2. *Yag internal http server (c++)*: reception of the request and transfer to the Yag code (c++ and LUA API)
3. *Yag code (c++ and LUA API)*: processing of the request
    a. getting the ps for pawn #3
    b. sending the result back to the internal http server
4. *Yag internal http server (c++)*: transforming the result into a JSON object and sending it back to the browser
5. *Browser (javascript)*: receiving, processing and displaying the result

URL encoding

In order to be able to handle special characters in requests, Yag only works with URL-encoded data. Care should therefore be taken to encode the data before sending them, and to decode them after reception.
In the above function:

- Encoding
    - var myFormData = "id=" + **encode**(pid);
- Decoding
    - var DecodedResponse = **decode**(xhr.responseText);

This is also the reason why AJAX requests should always indicate in their header that their content is encoded. The following line should therefore always be used:
xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded", "charset=UTF-8");

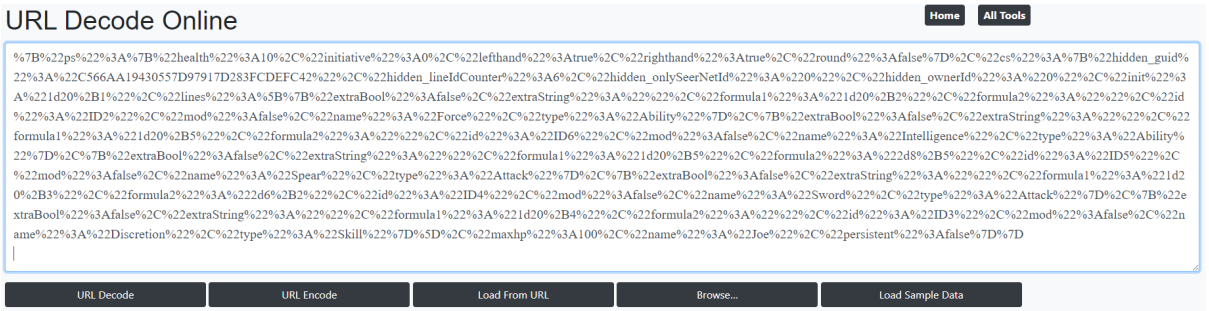
Manual example

For example to retrieve the ps of the pawn identified by #3, you should in principle send Yag the following request:
http://localhost/yagapi/pawn/get/id?id=#3
But to manage the special character #, data must be URL-encoded.

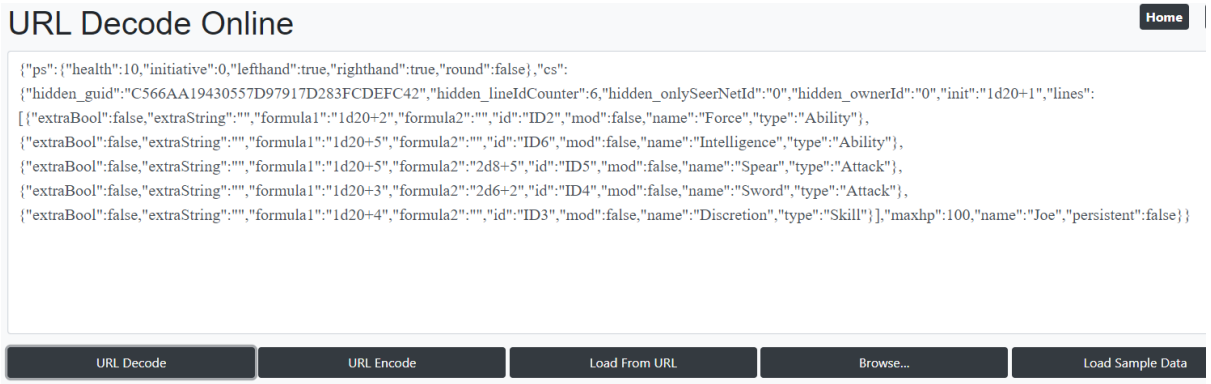
- non URL-encoded: #3
- URL-encoded: %233

If we want to interrogate Yag manually, we must therefore send it the following request:
http://localhost/yagapi/pawn/get/id?id=%233

Yag then returns the result of the request in URL-encoded form:



Which can be decoded with any encoding / decoding site to get a human readable JSON object:



Of course, in practice, we will always use the following encode() and decode() javascript functions:

```
function encode(TargetStr) {
   return encodeURIComponent(TargetStr).replace(/'/g, "%27").replace(/"/g, "%22");
}

function decode(TargetStr) {
   return decodeURIComponent(TargetStr.replace(/\+/g, " "));
}
```

REST API

Memory aid

These acronyms were invented for Yag, with the exception of the GUID which is an official acronym.
The GUID is a global unique identifier.
It's a long string of characters that looks like this:
3F2504E0-4F89-11D3-9A0C-0305E82C3301
Yag internally uses some guid to identify character sheets, tokens, etc.
The javascript code may need it in its requests.
So there are requests that return a guid, which can then be used in other requests.

Acronym	**Signification**
ps	Pawn Sheet
cs	Character Sheet
guid	Global Unique IDentifier
pguid	Pawn Guid
csguid	Cs Guid
id	IDentifier
sp	Selected Pawns

dr	Dice Result

Pawns

/yagapi/pawn/get/guid

- It's a GET request
- It returns the ps of the target pawn, its id, and the csguid identifier of its cs.

Call

- It only takes one parameter: pguid
    - var myFormData = "pguid=" + encode(pGuid);
    - xhr.open('GET', '/yagapi/pawn/get/guid?' + myFormData);
    - xhr.send();
- pGuid is the guid of the target pawn

Return

- It returns a JSON string containing the ps, id, and csguid of the target pawn.
    - {"ps" : <pawn's ps>,"id" : <pawn's id>, "csguid" : <pawn's cs guid>}

/yagapi/pawn/get/id

- It's a GET request
- It returns the ps, the pguid identifier of the target pawn, and the csguid identifier of its cs.

Call

- It only takes one parameter: id
    - var myFormData = "id=" + encode(pid);
    - xhr.open('GET', '/yagapi/pawn/get/id?' + myFormData);
    - xhr.send();
- id is the identifier of the target pawn

Return

- It returns a JSON string containing the ps, pguid, and csguid of the target pawn.
    - {"ps" : <pawn's ps>,"pguid" : <pawn's guid>, "csguid" : <pawn's cs guid>}

/yagapi/pawn/get/selected/firstvisible

- It's a GET request
- It returns the ps, the pguid identifier of the first selected visible pawn, the csguid identifier of its cs, and its id.

Call

- It takes no parameter
    - xhr.open('GET', '/yagapi/pawn/get/selected/firstvisible');
    - xhr.send();

Return

- It returns a JSON string containing the ps and the cs of the first selected pawn.
    - {"ps" : <ps du pion>,"pguid" : <guid du pion>, "csguid" : <guid de la cs du pion>, "id" : <id du pion>}

/yagapi/pawn/get/selected/firstfromcurrentcs

- It's a GET request
- It returns the ps, the pguid identifier of the first selected pawn of the current cs, and its id.

Call

- It takes no parameter
    - xhr.open('GET', '/yagapi/pawn/get/selected/firstfromcurrentcs');
    - xhr.send();

Return

- It returns a JSON string containing the ps and the id of the first selected pawn of the current cs.
    - {"ps" : <ps du pion>,"pguid" : <guid du pion>, "id" : <id du pion>}

/yagapi/pawn/set/ids

- It's a POST request
- It updates the ps of the pawns in the ids list.

Call

- It takes two parameters: the target ps and the list ids of the id of target pawns
    - var PSJson = {};
    - PSJson["health"] = document.getElementById("CSCurrentHealth").value;
    - [...]
    - var IDSJson = ["#2", "#4", "JaneID"];
    -
    - xhr.open('POST', '/yagapi/pawn/set/ids');
    - var myFormData = "ps=" + encode(JSON.stringify(PSJson)) + "&" + "ids=" + encode(JSON.stringify(IDSJson));
    - xhr.send(myFormData);

Return

- It returns nothing

/yagapi/pawn/set/current

- It's a POST request
- It updates the ps and id of the current pawn.

Call

- It takes 3 parameters: the target ps , the pawn's pguid and its new id.
    - xhr.open('POST', '/yagapi/pawn/set/current');
    - var myFormData = "ps=" + encode(JSON.stringify(PSJson)) + "&" + "id=" + encode(pid) + "&" + "pguid=" + encode(pGuid);
    - xhr.send(myFormData);
- PSJson is the target ps in JSon format
- pGuid is the pawn's guid. It is necessary so Yag can check that the currently selected pawn is really the target pawn.
- pid is the target pawn's new id

Return

- It returns nothing

/yagapi/pawn/attack

- It's a POST request
- It triggers the attack animation of the pawns from the ids list.

Call

- It takes one parameter: the list ids of the id of target pawns
    - var IDSJson = ["#2", "#4", "JaneID"];
    - xhr.open('POST', '/yagapi/pawn/attack);
    - var myFormData = "ids=" + encode(JSON.stringify(IDSJson));
    - xhr.send(myFormData);

Return

- It returns nothing

/yagapi/pawn/get/selected/list

- It's a GET request
- It returns the list of selected pawns

Call

- It takes no parameter
    - xhr.open('GET', '/yagapi/pawn/get/selected/all');
    - xhr.send();

Return

- It returns a JSON string containing the list sp of the selected pawns.
    - {"sp":["#1","#2","#3"]}

/yagapi/pawn/select/nextincharactersheet

- It's a GET request
- It selects the next pawn from the current character sheet.

Call

- It takes no parameter
    - xhr.open('GET', '/yagapi/pawn/select/nextincharactersheet');
    - xhr.send();

Return

- It returns nothing

/yagapi/pawn/select/nextvisible

- It's a GET request
- It selects the next visible pawn.

Call

- It takes no parameter
    - xhr.open('GET', '/yagapi/pawn/select/nextvisible');

    - xhr.send();

Return

- It returns nothing

/yagapi/pawn/select/id

- It's a GET request
- It selects the pawn identified by id.

Call

- It only takes one parameter: id
    - var myFormData = "id=" + encode(pid);
    - xhr.open('GET', '/yagapi/pawn/select/id?' + myFormData);
    - xhr.send();
- id is the id of the target pawn

Return

- It returns nothing

/yagapi/pawn/preview/id

- It's a GET request
- It previews the pawn identified by id.

Call

- It only takes one parameter: id
    - var myFormData = "id=" + encode(pid);
    - xhr.open('GET', '/yagapi/pawn/preview/id?' + myFormData);
    - xhr.send();
- id is the id of the target pawn

Return

- It returns nothing

/yagapi/pawn/preview/firstselected

- It's a GET request
- It previews the first selected pawn.

Call

- It takes no parameter

Return

- It returns nothing

/yagapi/pawn/goto/id

- It's a GET request
- It sends the player to the pawn identified by id.

Call

- It only takes one parameter: id
    - var myFormData = "id=" + encode(pid);
    - xhr.open('GET', '/yagapi/pawn/select/id?' + myFormData);
    - xhr.send();
- id is the id of the target pawn

Return

- It returns nothing

/yagapi/pawn/goto/firstselected

- It's a GET request
- It sends the player to the first selected pawn.

Call

- It takes no parameter

Return

- It returns nothing

Character sheets

/yagapi/charactersheet/select/guid

- It's a GET request
- It selects the cs identified by the csguid given as parameter.

Call

- It only takes one parameter: the guid csguid of the target cs
    - var myFormData = "csguid =" + csguid;
    - xhr.open('GET', '/yagapi/charactersheet/select/guid?' + myFormData);
    - xhr.send(myFormData);
- csguid is the identifier of the target character sheet

Return

- It returns nothing

/yagapi/charactersheet/get/list

- It's a GET request
- It returns the list of character sheets visible to the player

Call

- It takes no parameter
    - xhr.open('GET', '/yagapi/charactersheet/get/list');
    - xhr.send(myFormData);

Return

- It returns a JSON string containing the CharacterSheets list of the {"Guid", "Name"} pairs of visible character sheets:
    - {"CharacterSheets":[{"Guid":"4D0E6A7F49A961770FCC749C3DB72A5D","Name":"Jane"},{"Guid":"C566AA19430557D97917D283FCDEFC42","Name":"Joe"}]}

- In each returned JSON object:
    - Guid is the (internal) identifier of the perso sheet (necessary to create in javascript a button that can select this perso sheet)
    - Name is the name of the character sheet

/yagapi/charactersheet/get/current

- It's a GET request
- It returns the current cs and useful associated data.

Call

- It takes no parameter
    - xhr.open('GET', '/yagapi/charactersheet/get/current);
    - xhr.send(myFormData);

Return

- It returns a JSON string containing the current cs, its csguid, the cs id counter cslineidcounter, the list ids of the pawns attached to the cs, and the list sp of the selected pawns.
    - {"cs" : <current cs>, "csguid": <cs guid>, "cslineidcounter": <lines id counter>, "ids" : <list of the cs pawns' id>, "sp" : <selected pawns id>}

/yagapi/charactersheet/set/current

- It's a POST request
- It updates the cs given as a parameter

Call

- It take 3 parameters: cs, target csguid, and cslineidcounter.
    - xhr.open('POST', '/yagapi/charactersheet/set/current');
    - var myFormData = "cs=" + encode(JSON.stringify(CSJson)) + "&" + "csguid=" + encode(CsGuid) + "&" + "cslinecounter=" + encode(LineIdCounter);
    - xhr.send(myFormData);
- CSJson is the target cs in JSON format
- CsGuid is the target csguid. It is necessary to send it so Yag can check that the target cs is the currently selected one.
- LineIdCounter (integer) is the current value of the id counter

Return

- It returns nothing

/yagapi/charactersheet/duplicate/guid

- It's a GET request
- It duplicates the cs identifid by the csguid given as a parameter.

Call

- Elle n'attend qu'un seul paramètre: CSGuid
    - var myFormData = "CSGuid=" + csguid;
    - xhr.open('GET', '/yagapi/charactersheet/duplicate/guid?' + myFormData);
    - xhr.send(myFormData);
- CSGuid is the identifier of the target character sheet

Return

- It returns nothing

/yagapi/charactersheet/create

- It's a GET request
- It creates a new cs.

Call

- It takes no parameter

Return

- It returns nothing

Dice

/yagapi/dice/clear

- It's a GET request
- It stops the current dice roll and removes the dice rolled (emergency dice roll stopping procedure)

Call

- It takes no parameter
    - xhr.open(GET, '/yagapi/charactersheet/set');
    - xhr.send();

Return

- It returns nothing

/yagapi/dice/roll/panel

- It's a POST request
- It rolls the dice for the formulas in the dice panel

Call

- It takes two parameters TryFormula and EffectFormula:
    - var myFormData1 = "TryFormula=" + encode(document.getElementById("TryFormula").value);
    - var myFormData2 = "EffectFormula=" + encode(document.getElementById("EffectFormula").value);
    -
    - xhr.open('POST', '/yagapi/dice/roll/panel');
    - var myFormData = myFormData1 + "&" + myFormData2;
    - xhr.send(myFormData);
- TryFormula and EffectFormula are respectively the formulas for the test and the result (typically hit/damage)

Return

- It returns an object roll containing the 2 results Try and Effect:
    - roll = {"Effect" : <effect dr>, "Try" : <try dr>}

- dr is the LUA structure of a dice result

/yagapi/dice/roll/init/all

- It's a GET request
- It rolls the initiative for all pawns participating to the round

Call

- It takes no parameter
    - xhr.open(GET, '/yagapi/dice/roll/init/all');
    - xhr.send();

Return

- It returns nothing

/yagapi/dice/roll/init/current

- It's a GET request
- It rolls the initiative for the current character sheet.

Call

- It takes no parameter
    - xhr.open(GET, '/yagapi/dice/roll/init/current');
    - xhr.send();

Return

- It returns an object roll containing the 2 results Try and Effect:
    - roll = {"Effect" : <effect dr>, "Try" : <try dr>}
- dr is the LUA structure of a dice result

/yagapi/dice/roll/line/current

- It's a POST request
- It rolls the dice for the line LineId in the current character sheet.

Call

- It takes one parameter LineId:
    - xhr.open('POST', '/yagapi/dice/roll/line/current');
    - var myFormData = "LineId=" + encode(TargetLineId);
    - xhr.send(myFormData);
- LineId is the identifier of the line in the character sheet

Return

- It returns an object roll containing the 2 results Try and Effect:
    - roll = {"Effect" : <effect dr>, "Try" : <try dr>}
- dr is the LUA structure of a dice result

/yagapi/dice/load

- It's a POST request

- It loads a set of dice.

Call

- It takes one parameter ds:
  - xhr.open('POST', '/yagapi/dice/load');
  - var myFormData = "ds=" + encode(JSON.stringify(DSJson));
  - xhr.send(myFormData);
- DSJson is the name of the dice set to load

Return

- It returns nothing

The journal

/yagapi/journal/get/all

- It's a GET request
- It returns the full content of the journal.

Call

- It takes no parameter
  - xhr.open('GET', '/yagapi/journal/get/all');
  - xhr.send();

Return

- It returns an array entries containing the full journal
  - entries = : ["entry 1", "entry 2", ...]

/yagapi/journal/get/last

- It's a GET request
- It returns the last last lines of the journal.

Appel

- It takes one parameter: the number last of lines to return
  - var myFormData = "last=42";
  - xhr.open('GET', '/yagapi/journal/get/last?' + myFormData);
  - xhr.send();

Retour

- It returns an array entries containing the last last lines:
  - entries = : ["entry -42", "entry -41", ...]

/yagapi/journal/get/at

- It's a GET request
- It returns the content of the journal starting at line at.

Call

- It takes one parameter: the number at of the first line to return
  - var myFormData = "at=42";
  - xhr.open('GET', '/yagapi/journal/get/at?' + myFormData);
  - xhr.send();

Return

- It returns an array entries containing all the lines following the line at:
    - entries = : [“entry 43”, “entry 44”, ...]

/yagapi/journal/send/public

- It’s a POST request
- It displays a public message in the journal

Call

- It takes 1 parameter Message:
    - xhr.open('POST', '/yagapi/journal/send/public');
    - var myFormData = "Message=" + document.getElementById("JournalMessage").value;
    - xhr.send(myFormData);

Return

- It returns nothing

/yagapi/journal/send/local

- It’s a POST request
- It displays a private message in the journal

Call

- It takes 1 parameter Message:
    - xhr.open('POST', '/yagapi/journal/send/local);
    - var myFormData = "Message=" + document.getElementById("JournalMessage").value;
    - xhr.send(myFormData);

Return

- It returns nothing

Turn order

/yagapi/initiative/get/list

- It’s a GET request
- It returns the list of pawns participating to the turn with their initiative.

Call

- It takes no parameter
    - xhr.open('GET', '/yagapi/initiative/get/list');
    - xhr.send();

Return

- It returns an array lines containing the round info for each participating pawn:
    - lines = : [{“id”: “id1”, “acting”: bActing1, “init”: init1, “csguid”, “csguid1” }, ...]
    - The array contains 1 JSON object per pawn, with:

    - id = pawn identifier (string)
    - acting = is it the pawn's turn to play ? (boolean)
    - init = pawn's initiative (integer)
    - csguid = guid of the pawn's cs (string)

/yagapi/initiative/down

- It's a GET request
- It gives the turn to the pawn whose initiative is immediately lesser than the current one.

Call

- It takes no parameter
    - xhr.open('GET', '/yagapi/initiative/down');
    - xhr.send();

Return

- It returns nothing

/yagapi/initiative/up

- It's a GET request
- It gives the turn to the pawn whose initiative is immediately greater than the current one.

Call

- It takes no parameter
    - xhr.open('GET', '/yagapi/initiative/up');
    - xhr.send();

Return

- It returns nothing

/yagapi/initiative/reset

- It's a GET request
- It resets the initiative for this round.

Call

- It takes no parameter
    - xhr.open('GET', '/yagapi/initiative/reset');
    - xhr.send();

Return

- It returns nothing