Remote Interoperability Protocol
for
online laboratories

Prepared by Luis de la Torre, Jesus Chacon and Dictino Chaos

Universidad Nacional de Educacion a Distancia (UNED)
ETS. Ingenieria Informatica
Madrid
Spain

18th November 2018

Contents

Revision History **1**

1 Introduction **2**

1.1 Purpose . . . 2

1.2 Document Conventions . . . 2

1.3 Intended Audience and Reading Suggestions . . . 2

1.4 Product Scope . . . 3

2 Overall Description **4**

2.1 Protocol Perspective . . . 4

2.1.1 Server Implementation Perspective . . . 5

2.1.2 Client Implementation Perspective . . . 6

2.2 Protocol Functions (High-Level API) . . . 6

2.2.1 Internal Functions . . . 6

2.2.2 External Functions . . . 7

2.3 Protocol Communication Methods (Low-Level API) . . . 7

2.4 Operating Environment . . . 7

2.5 Design and Implementation Constraints . . . 8

2.6 Assumptions and Dependencies . . . 8

2.7 User Documentation . . . 8

2.7.1 Method info . . . 8

2.7.2 Method connect . . . 18

2.7.3 Method set . . . 18

2.7.4 Method get . . . 19

2.8 Developer Documentation . . . 20

3 External Interface Requirements **21**

3.1 User Interfaces . . . 21

3.2 Software Interfaces . . . 21

3.3 Communications Interfaces . . . 21

4 Protocol Features **22**

4.1 Defining Experiences . . . 22
4.2 Obtaining Meta-data . . . 22
4.3 Obtaining Readable and Writable Variables . . . 22
4.4 Obtaining Interface Methods . . . 22
4.5 Invoking Interface Methods to Read and Write Variables . . . 22
4.6 Defining Server Events . . . 22
4.7 Subscribing to Server Events . . . 22

Appendices **23**

A Glossary **24**

B Analysis Models **25**

References **26**

Revision History

Revision	Date	Author(s)	Description
0.1	18.11.2018	L. de la Torre	Chapter 1 - Introduction and document structure
0.21	24.11.2018	L. de la Torre	Chapter 2 - Overall Description (Sections 2.1-2.5)
0.22	25.11.2018	L. de la Torre	Chapter 2 - Overall Description (Updated Sections 2.1-2.5 and added Sections 2.6 and 2.7.1)
0.23	29.11.2018	L. de la Torre	Chapter 2 - Overall Description (Finished Section 2.7.1)
0.24	30.11.2018	L. de la Torre	Chapter 2 - Overall Description (Updated Sections 2.2, 2.3 and 2.7 and added Sections 2.7.2-2.7.4)

Chapter 1

Introduction

1.1 Purpose

This document is a specification of the Remote Interoperability Protocol (RIP), which was conceived at UNED for the remote operation of online laboratories (OLs). Instructions on how to correctly implement both a server and a client that talk RIP are also given.

1.2 Document Conventions

For the purpose of this document, we consider that an OL can either be a virtual laboratory (VL) or a remote laboratory (RL).

VLs are simulations and offer experimentation possibilities based on mathematical models.

RLs use lab equipment and perform the experiments in real life, just remotely.

A *simulation model* is understood as software that includes mathematical models that simulates a system for virtual experimentation purposes.

Control program is a term used in this document to refer to the software in charge of controlling and monitoring lab equipment.

In this sense, we consider that a VL always has an associated *simulation model* that is hosted and run in some computer, while a RL always has an associated *control program* that is also hosted and run in some computer.

Finally, we define an *experience* as each of the lab activities that can be carried out with an OL implementation, either through a RL or through a VL.

1.3 Intended Audience and Reading Suggestions

Audiences that may be interested in this document are educators, researchers and industry stakeholders that want or need to remotely communicate either with hardware devices or mathematical models from a web application.

More specifically, this document aims at anyone who is interested in one or more of the following points:

1. Implementing a RIP server and/or a RIP client to use RIP as the communication protocol for operating OLs.
2. Using or modifying an existing RIP server and/or RIP client implementation.

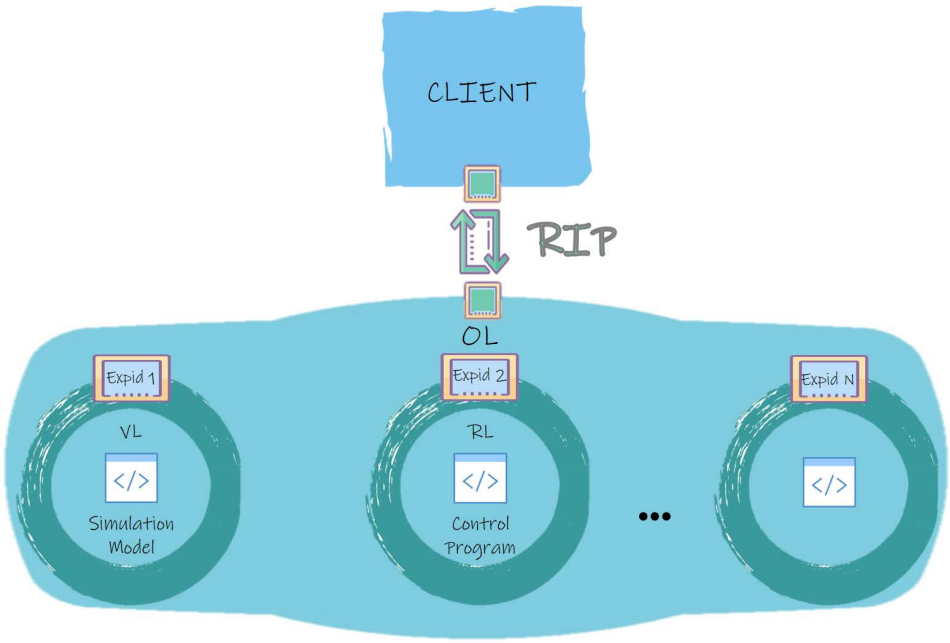

Figure 1.1: RIP protocol is used by a RIP client and a RIP server to communicate both. The RIP server is implemented in an OL and the RIP client is implemented in a web browser application.

3. Making modifications on the RIP protocol itself.

In any of the above cases, it is advised to read the present document in order. Before reading this document, it is reccommended to have some notions about TCP [1], HTTP [2], SSE [3] and JSON-RPC [4].

1.4 Product Scope

The objective of RIP is to offer a simple, yet powerful, communication solution usable from web clients. As such, RIP only uses pure HTTP standard protocols, supported by all major web browsers.

RIP is designed to communicate web clients with OLs; either VLs or RLs. When used to communicate with a VL, RIP exposes meta-data and input and output methods and variables related to a simulation model that is hosted and runs on a computer (usually, a remote server). When used to communicate with a RL, RIP does the same thing with a *control program* defined in a computer (usually, a remote server) to monitor and manipulate the lab equipment.

Figure 1.1 shows the usage of the RIP protocol implemented in a RIP client and a RIP server to communicate a web client with an OL. The figure represents how an OL can implement either a VL, a RL, or any combination of both, each one defined as an independent *experience*, referenced through a certain *expId* parameter.

Chapter 2

Overall Description

2.1 Protocol Perspective

The protocol is an open source, under the GNU general Public License. It is a communication protocol to be used in the client-server model, especially designed for OLs in which the client runs within a web browser. RIP provides a simple mechanism for users and client machines/programs to acquire information about the lab *experiences* defined in the server and about each *experience's* inputs and outputs. The protocol also defines methods for reading and writing the values of these inputs and outputs, respectively.

The main features of RIP are the following:

1. Defining *experiences* on the OL.
2. Obtaining meta-data related to each defined *experience*.
3. Obtaining a list of readable and writable variables for each *experience*.
4. Obtaining a list of methods to read and write variables in each *experience*.
5. Invoking methods to read and write variables in each *experience*.
6. Defining server events to send data either periodically or based on any other triggering condition defined in an *experience*.
7. Subscribing a client to any server event declared in an *experience*.

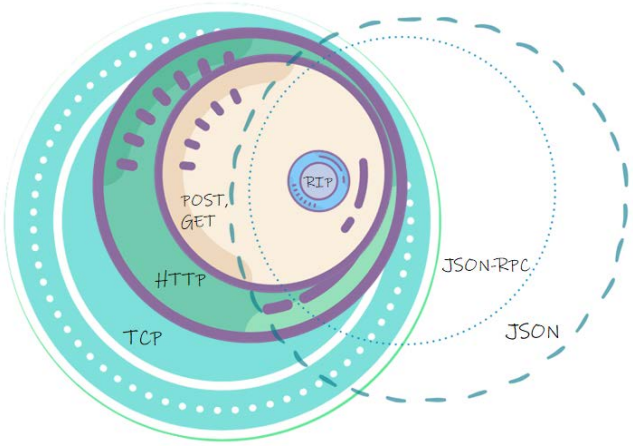

Figure 2.1: RIP is based on POST and GET HTTP methods and on the JSON-RPC format.

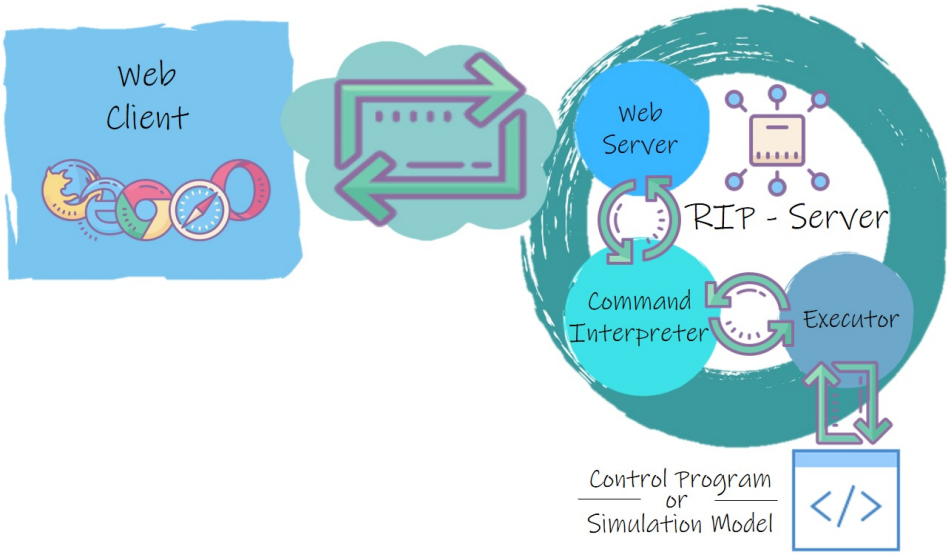

Figure 2.2: Architectural view of a RIP Server

RIP is based on two cornerstones: POST and GET methods for the communications transport and JSON-RPC for formatting messages. POST and GET are HTTP methods, which, in turn, is based on TCP communications. On the other hand, JSON-RPC is based on the JSON format. Figure 2.1 represents these ideas.

Communication between two software entities is possible when they talk the same protocol. Therefore, a RIP implementation is needed in both the client and the server.

2.1.1 Server Implementation Perspective

Figure 2.2 depicts the architecture of a *RIP Server* that implements the RIP protocol. To sum it up, there are three functional subsystems: the *Web Server*, that handles client connections, user sessions, etc., the *Command Interpreter*, that speaks RIP, and the *Executor*, that controls the execution of the laboratory *control programs* or *simulation models*.

The *Web Server* component admits (and handles in different ways) three types of requests: GET (used to retrieve *experiences'* meta-data), SSE (used to get server-to-client data updates) and POST (used to send client-to-server updates or client-to-server requests for data updates). These different methods are each associated with the three basic cases of use, namely:

- *Meta-data* - A client, wanting to obtain information about the laboratory, launches an HTTP GET request to the URL associated with the laboratory. The RIP server responds with a JSON-RPC structure that informs the client, depending on the request's parameters, with one of the following:
  1. General information about the OL: what are the *experiences* defined and how they can be accessed.
  2. Detailed information of a particular *experience* (when the request includes the experience id as a parameter).
- *Observer* - A client, that desires to receive updates on the state of the plant, subscribes to an SSE event stream associated to the *experience* of interest.
- *Operator* - A client, wanting to act over the OL or to receive an update on demand, sends a POST request with the command codified as a RIP-JSON-RPC structure.

An *experience* represents a lab activity associated with an OL. In the case of RLs, each *experience* is implemented as a *control program*, which in general is responsible of managing the physical connections with the

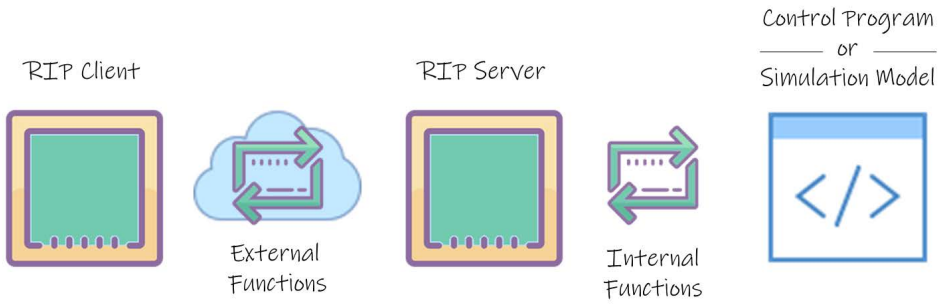

Figure 2.3: Internal and external functions implementation in RIP clients and servers

hardware, safe measures, and any other functionality the lab designer has considered appropriate to include. In the case of VLs, each *experience* is implemented as a *simulation model* that represents a real system. The *experience* abstraction is useful for two purposes: 1) to publish information about OLs in a standard and structured way and 2) to allow for hosting and running several different *control programs* or *simulation models* in the same computer.

2.1.2 Client Implementation Perspective

A RIP client must simply implement the required communication protocol methods (that is, POST, GET and SSE) with the appropriate format for reading and writing the messages content (that is, JSON-RPC) and the structure and functions defined by RIP (detailed in later sections).

2.2 Protocol Functions (High-Level API)

The functions defined and used in the protocol are divided in two types: internal and external.

The internal functions are private functions, used internally by RIP server implementations to communicate either with the *simulation model* or the *control program* used in the OL.

The external functions are client-side methods. These functions are used by the RIP clients to both get meta-data about the defined *experiences* and to write and read data to and from the OL.

In this way, RIP servers must implement the internal functions, while RIP clients must implement the external functions (see Figure 2.3). Client-based methods of the external functions in a RIP client communicate with the *Web Server* of a RIP server, get translated by the *Command Interpreter* into a series of two or more internal functions, and are finally executed by the *Executor* component (see Figure 2.2).

2.2.1 Internal Functions

The list of existing internal functions a RIP server must implement is:

1. *readablelist*, *writablelist* = **open**(*expId*): Returns the list of readable and writable variables (in two different variables) defined in the *experience* associated to the input *expId* parameter.
2. **close**(*expId*): Closes the control program (if it is a RL) or the simulation model (if it is a VL) of the OL *experience* defined by the input *expId* parameter.
3. **start**(*expId*): Starts the execution of the *control program* or *simulation model* associated to the OL *experience* defined by the input *expId* parameter.
4. **stop**(*expId*): Stops the execution of the *control program* or *simulation model* associated to the OL *experience* defined by the input *expId* parameter.

5. *readvariablenames*, *readvariablevalues* = **get**(*expId*, *variablenames*): Retrieves the current values of the variables (*readvariablevalues*) specified by the *variablenames* input parameter. For this to happen, these variables must exist in the *control program* or *simulation model* associated to the *experience* defined by the *expId* input parameter. When the **get()** method is called, *readvariablenames* contains only the names of the variables that were successfully read, not all requested ones in *variablenamess*.
6. **set**(*expId*, *variablenames*, *variablevalues*): Writes the received values (*variablevalues*) in the specified variables (*variablenames*) of the appropiate OL *experience*. For this to happen, these variables must exist in the *control program* or *simulation model* associated to the (*expId*) input parameter.

2.2.2 External Functions

The list of existing external functions a RIP server must implement so that a RIP client may use is:

1. *result* = **info**(*expId* = null): Retrieves the meta-data information about the *experience* defined by the input *expId* parameter. This parameter is optional and if it is not specified, the method then returns meta-data information about all the *experiences* defined.
2. *result* = **connect**(*expId*): Connects to an *experience* defined by the input *expId* parameter and establishes connection to the SSE to start receiving server data updates.
3. *result* = **get**(*expId*, *variablenames*): Retrieves the current values of the variables (*readvariablevalues*) specified by the *variablenames* input parameter. For this to happen, these variables must exist in the *control program* or *simulation model* associated to the *experience* defined by the *expId* input parameter.
4. *result* = **set**(*expId*, *variablenames*, *variablevalues*): Writes the received values (*variablevalues*) in the specified variables (*variablenames*) of the appropiate OL *experience*. For this to happen, these variables must exist in the *control program* or *simulation model* associated to the (*expId*) input parameter.

Where:

Variables *variablenames* and *variablevalues* are arrays of text. For example: *variablenamess* = ["x", "y", ...], *variablevalues* = ["10", "a", ...].

Sections 2.7 and 2.8 give more information about the external and internal functions, respectively.

2.3 Protocol Communication Methods (Low-Level API)

Three HTTP communication methods are available in RIP for communicating the client with the server:

1. **GET** - To obtain OL meta-data.
2. **POST** - To send client-to-server requests for (i) writing OL variables' values and (ii) reading OL variables' values.
3. **SSE** - To subscribe the client to data streams so that it receives server-to-client OL variables' values updates.

Table 2.1 shows the correspondence between the external protocol functions of the high-level API and the HTTP methods of the low-level API:

2.4 Operating Environment

RIP uses only HTTP methods for the communication. Therefore, it works in any major web browser. However, RIP also relies on the use of SSEs, which, up to date, are not supported by Microsoft Internet Explorer nor

HTTP Method (Low-Level API)	RIP function (High-Level API)
GET	info(), info(expId)
POST	set(expId, variablenames, variablevalues), get(expId, variablenames)
SSE	connect(expId)

Table 2.1: RIP functions - HTTP methods correspondence

Microsoft Edge. Nevertheless, there are numerous poly-fill solutions for implementing SSE so that they work on these browsers that do not support them natively.

2.5 Design and Implementation Constraints

RIP is designed to only use pure HTTP methods on purpose, with the aim of guaranteeing its correct functioning from web browsers. Therefore, the only hard implementation constraint is that HTTP is used for implementing RIP communications.

2.6 Assumptions and Dependencies

RIP depends solely on the use of HTTP POST and GET methods and on the JSON format for exchanging data.

A common and easy way of implementing RIP in web clients is through the use of the EventSource object [5] (for the SSEs) and the XMLHTTPRequest object [6] or the Fetch API [7] (for the POST messages), which are both supported by all major web browsers. Still, implementations without the use of such APIs are possible, just more laborious.

RIP server implementations may differ a lot depending on the language used to make the implementation, and so do their possible dependencies.

2.7 User Documentation

This section provides a deeper insight on the external functions of the high-level API, which is the only knowledge required by a user to use a RIP client implementation to communicate with an OL running a RIP server.

A RIP client must implement a class with the methods described in Section 2.2.2. Next, detailed information of each of them is given.

2.7.1 Method info

This method can be called with or without the *expId* input parameter.

When called without this parameter, **info()** returns meta-data with the list of experience identifiers associated to all *experiences* defined in the OL. It also returns more information about the low-level API method call for retrieving meta-data. The result is a JSON object containing one single top-level member:

(1) **experiences**: A JSON object with the following top-level members:

(I) **list**: An array of JSON objects, each of which contains one single top-level member:
  (A) **id**: A string with the experience identifier (expId) that is unmistakably related to one and only one *experience* defined in the OL.

(II) **methods**: An array of JSON objects, each of which contains the following top-level members:
  (A) **url**: A string with the URL to call the method.
  (B) **type**: A string specifying the HTTP method to use.
  (C) **description**: A string describing the method.
  (D) **params**: An array of JSON objects detailing the required and optional parameters for the method:
    (i) **name**: A string with the name of the parameter
    (ii) **required**: A string ("yes" or "no") specifying whether the parameter is required ("yes") or optional ("no").
    (iii) **location**: A string referencing the location for the parameter ("query", "header" or "body").
    (iv) **value**: [Optional] A string with the required value for the parameter.
    (v) **type**: [Optional] A string detailing the type of the parameter.
    (vi) **elements**: [Optional] An array of JSON objects with the following top-level members:
      (a) **description**: A string with a description of the parameter element.
      (b) **type**: A string specifying the type of the parameter element.
      (c) **subtype**: [Optional] A string detailing the type of the elements inside an array parameter element when they are all of the same type.
    (vii) **subtype**: [Optional] A string detailing the type of the elements inside an array parameter when they are all of the same type.
  (E) **returns**: A string with the MIME-type of the method's result.
  (F) **example**: A JSON object with the following top-level members:
    (i) **url**: A string with the URL to call the method.
    (ii) **headers**:[Optional] A JSON object with the parameters to be placed on the HTTP request headers with as many top-level members as headers are set:
      (a) **HeaderName**: HeaderValue
    (iii) **body**: [Optional] A JSON object with the parameters to be placed on the HTTP request body and with the following top-level members:
      (a) **jsonrpc**: "2.0"
      (b) **method**: A string with the RIP method to invoke ("get", "set" or "connect").
      (c) **params**: An array with the following elements:
        (1) **expId**: A string with the *eperience* identifier.
        (2) **VariablesName**: An array of strings in which each element has the name of a variable in the *control program* or in the *simulation model*.
        (3) **VariablesValue**: [Optional] An array of strings with the values to be written in the variables specified in the *VariablesName* member, following the same order.
      (d) **id**: A string with a number that indicates how many request have been sent till now, including this one.

Example:

Request:

The *info()* RIP client method sends a GET request to: `http(s)://BASEURL/RIP`

Response:

The RIP server responds to the GET message with the following body:

```
{
  "experiences": {
```

```
    "list": [
             {
             "id": "Test1"
             },
             {
             "id": "Test2"
             }
            ],
    "methods": [
                {
                "url": "10.192.38.68:8080/RIP",
                "type": "GET",
                "description": "Retrieves information (variables and methods) of the experiences in the
                    server",
                "params": [
                           {
                           "name": "Accept",
                           "required": "no",
                           "location": "header",
                           "value": "application/json"
                           },
                           {
                           "name": "expId",
                           "required": "no",
                           "location": "query",
                           "type": "string"
                           }
                          ],
                "returns": "application/json",
                "example": {
                           "url": "10.192.38.68:8080/RIP?expId=Test1"
                           }
                }
              ]
    }
}
```

When called with the *expId* parameter, **info(expId)** returns meta-data of the referenced *experience*. The result is a JSON object containing three top-level members:

(1) **info**: A JSON object with the following top-level members:

- (I) **name**: A string with the name given to the experience.
- (II) **description**: A string with a description of the experience.
- (III) **authors**: A string with the name of the authors of the experience, separated by commas.
- (IV) **keywords**: An array of strings, each with a keyword related to the experience.

(2) **readables**: A JSON object with the following top-level members:

- (I) **list**: An array of JSON objects with the following top-level members:
  - (A) **name**: A string with the name of the variable.
  - (B) **description**: A string with a description of the variable.
  - (C) **type**: A string with the type of the variable: *"string"*, *"int"*, *"float"* or *"boolean"*.
  - (D) **min**: A string with the minimum value the variable can have. In case of boolean variables, *"false"*; in case of string variables, *""*; in any other case, any number in string format.
  - (E) **max**: A string with the minimum value the variable can have. In case of boolean variables, *"true"*; in case of string variables, *""*; in any other case, any number in string format.

(F) **precision**: A string with the minimum value the variable can have. In case of boolean or string variables, *""*; in any other case, any number in string format.

(II) **methods**: EXACTLY AS the *methods* top-level member from the *experiences* JSON object returned by the *info()* method when called without parameters: an array of JSON objects, each of which contains the following top-level members:

(A) **url**: A string with the URL to call the method.

(B) **type**: A string specifying the HTTP method to use.

(C) **description**: A string describing the method.

(D) **params**: An array of JSON objects detailing the required and optional parameters for the method:

(i) **name**: A string with the name of the parameter

(ii) **required**: A string ("yes" or "no") specifying whether the parameter is required ("yes") or optional ("no").

(iii) **location**: A string referencing the location for the parameter ("header", "query" or "body").

(iv) **value**: [Optional] A string with the required value for the parameter.

(v) **type**: [Optional] A string detailing the type of the parameter.

(vi) **elements**: [Optional] An array of JSON objects with the following top-level members:

(a) **description**: A string with a description of the parameter element.

(b) **type**: A string specifying the type of the parameter element.

(c) **subtype**: [Optional] A string detailing the type of the elements inside an array parameter element when they are all of the same type.

(vii) **subtype**: [Optional] A string detailing the type of the elements inside an array parameter when they are all of the same type.

(E) **returns**: A string with the MIME-type of the method's result.

(F) **example**: A JSON object with the following top-level members:

(i) **url**: A string with the URL to call the method.

(ii) **headers**:[Optional] A JSON object with the parameters to be placed on the HTTP request headers with as many top-level members as headers are set:

(a) **HeaderName**: HeaderValue

(iii) **body**: [Optional] A JSON object with the parameters to be placed on the HTTP request body and with the following top-level members:

(a) **jsonrpc**: "2.0"

(b) **method**: A string with the RIP method to invoke ("get", "set" or "connect").

(c) **params**: An array with the following elements:

(1) **expId**: A string with the *eperience* identifier.

(2) **VariablesName**: An array of strings in which each element has the name of a variable in the *control program* or in the *simulation model*.

(3) **VariablesValue**: [Optional] An array of strings with the values to be written in the variables specified in the *VariablesName* member, following the same order.

(d) **id**: A string with a number that indicates how many request have been sent till now, including this one.

(3) **writables**: A JSON object with the following top-level members:

(I) **list**: EXACTLY AS the *list* top-level member from *readables*: an array of JSON objects with the following top-level members:

(A) **name**: A string with the name of the variable.

(B) **description**: A string with a description of the variable.

(C) **type**: A string with the type of the variable: *"string"*, *"int"*, *"float"* or *"boolean"*.

(D) **min**: A string with the minimum value the variable can have. In case of boolean variables, *"false"*; in case of string variables, *""*; in any other case, any number in string format.

(E) **max**: A string with the minimum value the variable can have. In case of boolean variables, *"true"*; in case of string variables, *""*; in any other case, any number in string format.

(F) **precision**: A string with the minimum value the variable can have. In case of boolean or string variables, *""*; in any other case, any number in string format.

(II) **methods**: EXACTLY AS the *methods* top-level member from the *experiences* JSON object returned by the *info()* method when called without parameters: an array of JSON objects, each of which contains the following top-level members:

(A) **url**: A string with the URL to call the method.

(B) **type**: A string specifying the HTTP method to use.

(C) **description**: A string describing the method.

(D) **params**: An array of JSON objects detailing the required and optional parameters for the method:

(i) **name**: A string with the name of the parameter

(ii) **required**: A string ("yes" or "no") specifying whether the parameter is required ("yes") or optional ("no").

(iii) **location**: A string referencing the location for the parameter ("header", "query" or "body").

(iv) **value**: [Optional] A string with the required value for the parameter.

(v) **type**: [Optional] A string detailing the type of the parameter.

(vi) **elements**: [Optional] An array of JSON objects with the following top-level members:

(a) **description**: A string with a description of the parameter element.

(b) **type**: A string specifying the type of the parameter element.

(c) **subtype**: [Optional] A string detailing the type of the elements inside an array parameter element when they are all of the same type.

(vii) **subtype**: [Optional] A string detailing the type of the elements inside an array parameter when they are all of the same type.

(E) **returns**: A string with the MIME-type of the method's result.

(F) **example**: A JSON object with the following top-level members:

(i) **url**: A string with the URL to call the method.

(ii) **headers**:[Optional] A JSON object with the parameters to be placed on the HTTP request headers with as many top-level members as headers are set:

(a) **HeaderName**: HeaderValue

(iii) **body**: [Optional] A JSON object with the parameters to be placed on the HTTP request body and with the following top-level members:

(a) **jsonrpc**: "2.0"

(b) **method**: A string with the RIP method to invoke ("get", "set" or "connect").

(c) **params**: An array with the following elements:

(1) **expId**: A string with the *eperience* identifier.

(2) **VariablesName**: An array of strings in which each element has the name of a variable in the *control program* or in the *simulation model*.

(3) **VariablesValue**: [Optional] An array of strings with the values to be written in the variables specified in the *VariablesName* member, following the same order.

(d) **id**: A string with a number that indicates how many request have been sent till now, including this one.

Example:

Request:

The *info(expId)* RIP client method sends a GET request to: `http(s)://BASEURL/RIP?expId=Test1`

Response:

The RIP server responds to the GET message with the following body:

```
{
"info": {
       "name": "Test1",
       "description": "Test1",
       "authors": "L. de la Torre",
       "keywords": ["Test", "Example"]
       },
"readables": {
    "list": [
              {
              "name": "intout",
              "description": "Integer output",
              "type": "int",
              "min": "-20",
              "max": "10",
              "precision":"1"
              },
              {
              "name": "stringout",
              "description": "String output",
              "type": "string",
              "min": "",
              "max": "",
              "precision": ""
              },
              {
              "name": "booleanout",
              "description": "Boolean output",
              "type": "boolean",
              "min": "false",
              "max": "true",
              "precision":""
              },
              {
              "name": "doubleout",
              "description": "Double output",
              "type": "float",
              "min": "-Inf",
              "max": "Inf",
              "precision": "0"
              }
           ],
    "methods": [
                {
                "url": "10.192.38.68:8080/RIP/SSE",
                "type": "GET",
                "description": "Subscribes to an SSE to get regular updates on the servers' variables",
                "params": [
                           {
                           "name": "Accept",
                           "required": "no",
                           "location": "header",
                           "value": "application/json"
                           },
                           {
                           "name": "expId",
                           "required": "yes",
                           "location": "query",
                           "type": "string"
                           },
```

```
                        {
                        "name": "variables",
                        "required": "no",
                        "location": "query",
                        "type": "array",
                        "subtype": "string"
                        }
                    ],
                "returns": "text/event-stream",
                "example": "10.192.38.68:8080/RIP/SSE?expId=TestOK"
                },
                {
                "url": "10.192.38.68:8080/RIP/POST",
                "type": "POST",
                "description": "Sends a request to retrieve the value of one or more servers' variables
                    on demand",
                "params": [
                        {
                        "name": "Accept",
                        "required": "no",
                        "location": "header",
                        "value": "application/json"
                        },
                        {
                        "name": "Content-Type",
                        "required": "yes",
                        "location": "header",
                        "value": "application/json"
                        },
                        {
                        "name": "jsonrpc",
                        "required": "yes",
                        "type": "string",
                        "location": "body",
                        "value": "2.0"
                        },
                        {
                        "name": "method",
                        "required": "yes",
                        "type": "string",
                        "location": "body",
                        "value": "get"
                        },
                        {
                        "name": "params",
                        "required": "yes",
                        "type": "array",
                        "elements": [
                            {
                            "description": "Experience id",
                            "type":"string"
                            },
                            {
                            "description":"Name of variables to be retrieved",
                            "type": "array",
                            "subtype": "string"
                            }
                         ],
                        "location": "body"
                        },
                        {
```

```
                        "name": "id",
                        "required": "yes",
                        "type": "int",
                        "location":"body"
                        }
                    ],
                "returns": "application/json",
                "example": {
                        "url": "10.192.38.68:8080/RIP/POST",
                        "headers": {
                                "Accept": "application/json",
                                "Content-Type": "application/json"
                                },
                        "body": {
                               "jsonrpc": "2.0",
                               "method": "get",
                               "params": [
                                          "Test1",
                                          ["intout","booleanout"]
                                        ],
                               "id":"1"
                               }
                        }
                },
                {
                "url": "http://camera1_ip/axis-cgi/mjpg/video.cgi",
                "type": "GET",
                "description": "Retrieve an image of the lab captured from the camera: 'Camera 1'",
                "params": [
                           {
                           "name": "resolution",
                           "required": "no",
                           "location": "query",
                           "type": "string"
                           },
                           {
                           "name": "user",
                           "required": "no",
                           "location": "query",
                           "type": "string"
                           },
                           {
                           "name": "password",
                           "required": "no",
                           "location": "query",
                           "type": "string"
                           }
                        ],
                "returns": "video/x-motion-jpeg",
                "example": {
                        "url": "http://camera1_ip/axis-cgi/mjpg/video.cgi"
                        }
            ]
          },
"writables": {
    "list": [
            {
            "name": "intin",
            "description": "Integer input",
            "type": "int",
            "min": "-2147483648",
```

```
            "max": "2147483647",
            "precision":"1"
            },
            {
            "name": "stop",
            "description": "Stops the control program",
            "type": "boolean",
            "min": "false",
            "max": "true",
            "precision":""
            },
            {
            "name": "booleanin",
            "description": "Boolean input",
            "type": "boolean",
            "min": "false",
            "max": "true",
            "precision": ""
            },
            {"name": "stringin",
            "description": "String input",
            "type": "string",
            "min": "",
            "max": "",
            "precision":""
            },
            {
            "name": "doublein",
            "description": "Double input",
            "type": "float",
            "min": "-Inf",
            "max": "Inf",
            "precision": "0"
            }
        ],
    "methods": [
              {
              "url": "10.192.38.68:8080/RIP/POST",
              "type": "POST",
              "description": "Sends a request to write the value of one or more servers' variables on
                  demand",
              "params": [
                        {
                        "name": "Accept",
                        "required": "no",
                        "location": "header",
                        "value": "application/json"
                        },
                        {
                        "name": "Content-Type",
                        "required": "yes",
                        "location": "header",
                        "value": "application/json"
                        },
                        {
                        "name": "jsonrpc",
                        "required": "yes",
                        "type": "string",
                        "location": "body",
                        "value": "2.0"
                        },
```

```
                        {
                        "name": "method",
                        "required": "yes",
                        "type": "string",
                        "location": "body",
                        "value": "set"
                        },
                        {
                        "name": "params",
                        "required": "yes",
                        "type": "array",
                        "elements": [
                                    {
                                    "description": "Experience id",
                                    "type": "string"
                                    },
                                    {
                                    "description": "Name of variables to write",
                                    "type": "array",
                                    "subtype":"string"
                                    },
                                    {
                                    "description": "Value for variables",
                                    "type": "array",
                                    "subtype":"mixed"
                                    }
                                ],
                        "location": "body"
                        },
                        {
                        "name": "id",
                        "required": "yes",
                        "type": "int",
                        "location": "body"
                        }
                    ],
            "returns": "application/json",
            "example": {
                      "url: "10.192.38.68:8080/RIP/POST",
                      "headers": {
                                "Accept": "application/json",
                                "Content-Type": "application/json"
                                },
                      "body": {
                             "jsonrpc": "2.0",
                             "method": "set",
                             "params": [
                                       "Test1",
                                       ["intin","stringin"],
                                       ["2","hello"]
                                      ],
                             "id": "1"
                             }
                      }
            }
          ]
        }
}
```

2.7.2 Method connect

This method has one input parameter (*expId*), for specifying the *experience* the client is going to connect to.

When this method is called, an SSE communication is established with the RIP server. The result is a stream of data (which periodicity or aperiodicity is defined in the server) that can be captured by the onmessage() and the addEventListener() methods of the EventSource object, for example [5]. The structure of the data obtained through the established SSE follows the SSE standard:

(1) **retry**: A number specifying the time (in miliseconds) to wait before the client tries to automatically reconnect to the SSE. This field appear only once per connection.

(2) **event**: The name of the event that follows

(3) **id**: A unique number that identifies the event.

(4) **data**: A JSON object with the following top-level members:

  (I) **data**: An array with the following elements:
    (A) **VariablesName**: An array of strings with the names of the retrieved variables.
    (B) **VariablesValue**: A mixed array with the values of the retrieved variables.

Example:

Request:

The *connect("Test1")* RIP client method established an SSE connection to: `http(s)://URL/RIP/SSE?expId=Test1`.

Response:

The RIP server responds with a stream of data that follows this format:

```
retry: 2000

event: periodiclabdata
id: 1
data:
{
"result": [
            [
              "intout",
              "doubleout",
              "stringout",
              "booleanout"
            ],
            [
              -2,
              3.5,
              "testing",
              true
            ]
          ]
}
```

A RIP client implementation must automatically capture these events and assign the received values to their corresponding variables in the client side.

2.7.3 Method set

This method has three input parameters: *expId*, *VariablesName* and *VariablesValue*.

When this method is called, the values specified in *VariablesValue* are assigned to the *control program's* or *simulation model's* variables named as the strings in *VariablesName*. The *control program* or *simulation model* is determined with the *expId* parameter. This method returns a JSON object with information about the result of the operation. The resulting JSON object contains the following top-level members:

(1) **jsonrpc**: "2.0"

(2) **result**: Either *true*, if the operation was completed succesfully by the RIP server, or *false*, if it was not.

(3) **id**: A string with a number that indicates how many request have been sent till now, including this one.

Example:

Request:

The *set("Test1", ["doublein", "intin"], [0.5, -1])* RIP client method sends a POST request to: `http(s)://URL/RIP/POST?expId=Test1` with body:

```
{
"jsonrpc": "2.0",
"method": "set",
"params": [
           "Test1",
           ["doublein", "intin"],
           [0.5, -1]
         ],
"id": "2"
}
```

Response:

The RIP server responds to the POST message with the following body:

```
{
"jsonrpc": "2.0",
"result": true,
"id": "2"
}
```

2.7.4 Method get

This method has two input parameters: *expId* and *VariablesName*.

When this method is called, the values of the *control program's* or *simulation model's* variables named as the strings in *VariablesName* are returned in a JSON object. The *control program* or *simulation model* is determined with the *expId* parameter. The resulting JSON object that contains the following top-level members:

(1) **jsonrpc**: "2.0"

(2) **result**: An array with the following elements:

  (I) **VariablesName**: An array of strings with the names of the retrieved variables.

  (II) **VariablesValue**: A mixed array with the values of the retrieved variables.

(3) **id**: A string with a number that indicates how many request have been sent till now, including this one.

Example:

Request:

The *get("Test1", ["doubleout", "intout"])* RIP client method sends a POST request to: `http(s)://URL/RIP/POST?expId=Test1` with body:

```
{
"jsonrpc": "2.0",
"method": "get",
"params": [
           "Test1",
           ["doubleout", "intout"]
         ],
"id": "3"
}
```

Response:

The RIP server responds to the POST message with the following body:

```
{
"jsonrpc": "2.0",
"result": [
           [
             "doubleout",
             "intout"
           ],
           [
             0.5,
             -1
           ]
         ],
"id": "3"
}
```

2.8 Developer Documentation

This section provides more details on the internal functions of the high-level API and on the low-level API, which, along with the information provided in last section for the external functions of the high-level API, is the knowledge required to either implement a new RIP client or a new RIP server, or to modify existing ones.

TODO.

Each method returns an error indicator when the operation is not completed successfully.

At the moment, only numbers, text and booleans are supported as valid types for *variablevalues* and *readvariablevalues* in the **set**() and **get**() methods, respectively.

Chapter 3

External Interface Requirements

3.1 User Interfaces

TODO

3.2 Software Interfaces

TODO

3.3 Communications Interfaces

TODO

Chapter 4

Protocol Features

4.1 Defining Experiences

TODO

4.2 Obtaining Meta-data

TODO

4.3 Obtaining Readable and Writable Variables

TODO

4.4 Obtaining Interface Methods

TODO

4.5 Invoking Interface Methods to Read and Write Variables

TODO

4.6 Defining Server Events

TODO

4.7 Subscribing to Server Events

TODO

Appendices

Appendix A

Glossary

Control program - A software program which purpose is to control and monitor the hardware equipment used to in a laboratory activity.

Experience - Any lab activity that can be performed in an OL.

OL - Online laboratory.

RIP - Remote Interoperability Protocol.

RL - Remote laboratory.

Simulation model - A mathematical simulation that models a certain system with which laboratory activities can be carried out.

SSE - Server-sent events.

VL - Virtual laboratory.

Appendix B

Analysis Models

TODO

References

[1] U. o. S. C. Information Sciences Institute. (1981) Transmission control protocol specification. [Online]. Available: https://tools.ietf.org/html/rfc793

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. (1999) Http 1.1 specification. [Online]. Available: https://tools.ietf.org/html/rfc2616.html

[3] (2009) Server-sent events specification. [Online]. Available: https://www.w3.org/TR/eventsource

[4] J.-R. W. Group. (2010) Json-rpc 2.0 specification. [Online]. Available: https://www.jsonrpc.org/specification

[5] (2015) Server-sent events. [Online]. Available: https://www.w3.org/TR/eventsource/

[6] (2018) Xmlhttprequest standard. [Online]. Available: https://xhr.spec.whatwg.org/

[7] (2018) Fetch standard. [Online]. Available: https://fetch.spec.whatwg.org/