LC2 Specification

Michael Franzen, Artem Chirkin, Lukas Treyer

August 8, 2016

Contents

Changelog

- 2016-08-05: First draft

1 Introduction

LUCI2 can be perceived as an IaaS provider, mediating between clients and *cloud* services for computational architecture analysis. Like most IaaS providers Luci's services fall into three different categories: ==storage==, ==networking== and ==computing==. We will broadly outline the main purpose of these functional units within the LC2 framework.

1.1 Notes

The words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are used in accordance to RFC2119.

1.2 Understanding the document

Current document may have not up-to-date action syntax specification. In order to get the most recent specification one can use "generate specification" Lightweight Urban Computation Interchange (Luci) command. This document is not intended to be a replacement to that feature.

The structure of the document is organized as follows: Section 1.3 gives an information how to participate in editing of the current document. Section **??** explains the syntax of Luci actions with some commentaries. Section **??** is the main informational part of the document: it discusses the conventions used between Luci and its services.

1.3 Editing and understanding the document

The document source resides in git repository `https://bitbucket.org/treyerl/lucy.git`, the `.tex` file is `spec/lpsg/LPS-guidelines.tex`, compiled with `texlive pdflatex` tool.

The document contains a number of JSON or GeoJSON listings representing content of Luci actions. In the listings we use the following coloring scheme:

- Key names are shown in black (e.g. `action`);
- Reserved keywords, such as value types are shown in blue (e.g. `string`);
- Fixed strings constants are shown in red (e.g. `"create_scenario"`);
- Additional structural keywords are shown in grey (e.g. `OPT` means the key-value pair is optional, `XOR` before several key-value pairs means exactly one alternative).
- Comments are in purple, separated by double slash (e.g. `// comment`)

If there is a missing reserved keyword, you can add it into tex file annotation (`keywords` or `ndkeywords` lists in `lstdefinelanguage` command).

2 Service API

For registering a service in Luci, a set of three parameters have to be implemented into the service and forwarded to Luci using the built-in service `RemoteRegister` (Appendix **??**).

The three required fields are

- `serviceName`: The service name as a string
- `description`: The service description as a string
- `exampleCall`: An example call to the service as json encoded string

If a service accepts input arguments or outputs data, two additional parameters can be provided:

- `input`
- `output`

3 Scenario Services

Luci uses GeoJSON format to represent scenario geometry. The format declares geometry information syntax, but does not declare consistent naming and geometry type mappings for Luci scenario entities, such as buildings, roads, etc. This document aims at providing guidelines on usage of Luci scenarios in Luci clients and services.

A note on jsonGeometry data type The word `geometry` in Luci specification has two different meanings. On the one hand it is a name of the key that occurs from time to time in Luci actions. On the other hand it is a name of a pre-defined data type that represents scenario geometry in JSON format. To resolve this ambiguity, in current document we use word `geometry` to represent the name of the key, and `jsonGeometry` to represent the data type. This differentiation does not introduce any changes to the existing JSON messages.

3.1 Luci scenario actions

To send geometry to Luci, we wrap it in the structure that is shown in listing 1. Special type `jsonGeometry` wraps various types of geometry processed by Luci. It allows to enclose arbitrary number of custom-named geometries (keys `KEY_NAME_N`) inline (GeoJSON) or in separate files (as binary attachments).

```
{
  /* key name is an arbitrary string.
     The convention is to name it by filename of a file,
     or arbitraryname.geojson in case of GeoJSON geometry */
  KEY_NAME_1 :
    XOR { // "in-line" geometry
      format    : string // "GeoJSON", later add also TopoJSON, CurveJSON, ...
      geometry : object // GeoJSON FeatureCollection
      OPT crs   : string // name of a crs
      OPT attributeMap : { // mapping between Luci and foreign types
        LUCI_ATTRIBUTE_NAME : FOREIGN_ATTRIBUTE_NAME,
        ...
      }
    }
    XOR { // "streaminfo" - attachment description
      format      : string // shp | shx | dbf | any other file format?
      streaminfo : {
        checksum : string // MD5 sum of an attached binary data
        length : long // length of the attached binary data
        order : long // number of attachment (starts with 1)
      }
      OPT crs     : string // name of a crs
      OPT attributeMap : { // mapping between Luci and foreign types
        LUCI_ATTRIBUTE_NAME : FOREIGN_ATTRIBUTE_NAME,
        ...
      }
    },
  KEY_NAME_2 : ...,
  ...
}
```

Listing 1: structure of jsonGeometry data type

The type of object geometry is GeoJSON FeatureCollection – the format that is described in GeoJSON specification[1].

According to spec/LuciSpecification.pdf, Luci provides three operations to work with scenarios: create_scenario, update_scenario, and get_scenario. This document covers only GeoJSON geometry manipulation; in this format individual entities are represented as Feature objects in FeatureCollection. Each Feature has a property geomID:long that is given either by a client, or by Luci (in case if client application does not specify property geomID).

Creating a scenario is done via create_scenario action shown in listing 2. The action allows to specify a location (projection) and a geometry to put inside the new scenario.

```
{
  action            : "create_scenario", // constant, represents the action
  name              : string, // name of the scenario
  OPT projection    : {
    XOR crs     : string, // name of a crs
    XOR bbox    : [ [number, number]   // top-left coords [lat, long]
                  , [number, number]], // bottom-right coords [lat, long]
  },
  OPT geometry      : jsonGeometry, // wrapper around various geometry types
  OPT switchLatLon : boolean // switch lat-long to long-lat in geometry
}
```

Listing 2: JSON action structure for creating a scenario in Luci

Listing 3 shows the action to update scenario geometry. The action allows to change the name and the bounding box, as well as the geometry inside.

```
{
  action        : "update_scenario", // constant, represents the action
  ScID          : long, // ID of the scenario in Luci
  OPT name      : string, // set a new name for the scenario
  OPT bbox      : [ [number, number]   // top-left coords [lat, long]
                  , [number, number]], // bottom-right coords [lat, long]
  OPT geometry : jsonGeometry // wrapper around various geometry types
  OPT switchLatLon : boolean // switch lat-long to long-lat in geometry
}
```

Listing 3: JSON action structure for updating a scenario in Luci

- In order to add an entity to the scenario, one adds Feature into FeatureCollection (geometry object).

[1] http://geojson.org/geojson-spec.html

- In order to modify an existing entity, one must specify its property `geomID` (if given `geomID` does not exist, the entity is added to the scenario, otherwise it is edited).
- In order to delete a number of entities from the scenario, one adds an empty `Feature` into `FeatureCollection` that has a property `deleted_geomIDs:[long]` – array of `geomID` for deletion.

Listing 4 shows the action to get scenario geometry. The action allows to specify the format of the data to (Luci does transformation), and get the geometry from the scenario at given time.

```
{
  action              : "get_scenario", // constant, represents the action
  XOR scenarioname    : string // name of the scenario in Luci
  XOR ScID            : long, // ID of the scenario in Luci
  OPT format_request  : string, // maybe we will change this later to "format"
  OPT crs             : string,
  OPT geomIDs         : [long], // select a subset of scenario objects
  OPT timerange       : { // time is a number - timestamp in unix format
    XOR until   : long,
    XOR from    : long,
    XOR between : [long,long],
    XOR exactly : long,
    OPT all     : boolean // include all versions (not only the last)
  }
}
```

Listing 4: JSON action structure for getting a scenario from Luci

3.2 Scenario GeoJSON geometry

Although GeoJSON specification provides all necessary geometry primitives, we need a more structured convention to define one-to-one mapping between the geometry and scenario entities. Luci does not generate an error for an input that does not follow it – the convention only describes what kind of data structures services and clients should expect.

Section **??** describes the rules assumed by Luci when communicating with all services and clients. Section **??** describes the rules assumed by most applications, but not checked in Luci. Section **??** describes the application-specific rules. Any service or client provider (Luci user) may introduce a rule that is used in their application. The providers are encouraged to add these rules into section **??**. By an agreement in our team some of the rules go up from section **??** to section **??**. In case of wide usage they might be enforced in Luci, thus moving one step up to section **??**.

3.2.1 Standardized rules

Object `geometry` in listing 1 is assumed to be of type `FeatureCollection`. Every entity is represented as a `Feature` inside that collection, and has a property `geomID:long` that is given by a client or Luci (if the client omits the property).

3.2.2 Conventional rules

Some services require 3D objects, others use only 2D footprints. To distinguish these two types of geometry, we agreed on using `Feature` property `layer`.

- Object (e.g. Building) – a 3D geometry, represented as `Feature` with `geometry` field of type `Polygon` or `MultiPolygon`. To be processed correctly by Luci services, the object requires property `layer:"buildings"`.
- Footprint – a 2D geometry, represented as `Feature` with `geometry` field of type `Polygon` or `MultiPolygon`. To be processed correctly by Luci services, the footprint requires property `layer:"footprints"`.

3.2.3 Per-application rules

Web geometry modeler The application distinguishes dynamic and static geometry: dynamic geometry can be edited, static geometry is only used for evaluation and visualization. Thus, I propose an optional property `static:boolean`. Absence of a property implies `static:false`.

4 QUA-Compliance

4.1 QUA-View-Compliance

A Built-In Services

A.1 task.Create

Create a task. A task corresponds to a node in the workflow diagram.

Inputs:

- **run**:

```
"task.Create"
```

- **parentID**: the taskID of the workflow in which this task is being created

```
"number"
```

- **position**: x/y from top left

```
{
  "x": "number",
  "y": "number"
}
```

- **serviceName**:

```
"string"
```

- **OPT inputs**: if omitted example values will be inserted

```
"json"
```

- **OPT listensToDone**: a list of taskIDs to which this task generally listens (=not listens to specific outputs)

```
"list"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR result**:

```
{
  "name": "string",
  "taskID": "number"
}
```

A.2 test.Validation

A service to test how type and value constraint validation works.@inputs.key1 normal string@inputs.key2 string values constraint to ’string1’ and ’string2’@inputs.key3 number values constraint to 4 ranges@inputs.key4 optional attachment that must be of type ’jpg’@inputs.key5 geojson geometry@inputs.key6 optional attachment that requires types to be one of ’jpg’ or ’png’@inputs.key7 optional attachment with predefied required keys

Inputs:

- **run**:

```
"test.Validation"
```

- **key1**:

```
"string"
```

- **key2**:

```
{
  "constraints": [
    "string1",
    "string2"
  ],
  "type": "string"
}
```

- **key3**:

```
{
  "constraints": "error",
  "type": "number"
}
```

- **key5**:

```
"jsongeometry/geojson"
```

- **OPT key4**:

```
"attachment/jpg"
```

- **OPT key6**:

```
{
  "constraints": [
    "jpg",
    "png"
  ],
  "type": "attachment"
}
```

- **OPT key7**:

```
"attachment/jpg"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "errorTestStrings": "list",
  "passed": "boolean"
}
```

A.3 test.Randomly

Get either an amount of random numbers or create key-value-pairs from a key list.

Inputs:

- **run**:

```
"test.Randomly"
```

- **XOR amount**:

```
"number"
```

- **XOR keys**:

```
"list"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "XOR keyValuePairs": "json",
  "XOR randomNumbers": "list"
}
```

A.4 test.FileEcho

A service to test sending and receiving attachments. Mainly used by connectionlibrary developers.

Inputs:

- **run**:

```
"test.FileEcho"
```

- **ANY filenames**:

```
"attachment"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR result**:

```
{
  "ANY sameFilenames": "attachment"
}
```

A.5 ServiceHistory

Returns the history of a requested service as in number of calls and availableworkers either since startup time or during the requested period

Inputs:

- **run**:

```
"ServiceHistory"
```

- **serviceName**:

```
"string"
```

- **OPT period**: in seconds

```
"number"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "calls": {
    "{ callID": {
      "callID": "number",
      "idleTime": "number",
      "readingTime": "number",
      "runningTime": "number",
      "timestamp": "number",
      "writingTime": "number"
    }
  },
  "workers": {
    "{ allWorkers": {
      "allWorkers": "number",
      "idleWorkers": "number",
      "timestamp": "number"
    }
  }
}
```

A.6 user.Delete

Deletes a user if it does not own a group or [TODO] own a scenario

Inputs:

- **run**:

```
"user.Delete"
```

- **id**:

```
"number"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "success": "boolean"
}
```

A.7 test.Fibonacci

A service that serves as an (implementation) example.

Inputs:

- **run**:

```
"test.Fibonacci"
```

- **amount**: how many numbers should be calculated

```
"number"
```

- **OPT onlyLast**: return only the last number

```
"boolean"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "fibonacci_sequence": "list"
}
```

A.8 FilterServices

Filters services according to a) its keys at a given recursion level, b) its types at a given recursion level or c) a given json template with values equal to 'null' meaning only keys are compared.

Inputs:

- **run**:

```
"FilterServices"
```

- **XOR jsonMatch**:

```
"json"
```

- **XOR keys**:

```
[
  "string"
]
```

- **XOR types**:

```
[
  "string"
]
```

- **OPT rcrLevel**:

```
"number"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "serviceList": [
    "string"
  ]
}
```

A.9 task.SubscribeTo

Subscribes the current client to a taskID or any call to a service indicated byserviceName

Inputs:

- **run**:

```
"task.SubscribeTo"
```

- **XOR serviceName**: if a client subscribes to a service, it gets a notification upon any output created by a service with that name (technically: the client subscribes to the service factory rather than to a specific service)

```
"string"
```

- **XOR taskIDs**: a list of taskIDs to which the client should be subscribed

```
"list"
```

- **OPT inclResults**: specifies whether the client that subscribes only get a notification (false) or all the results produced by a service (true)

```
"boolean"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "success": "boolean"
}
```

A.10 workflow.ServiceRemove

Remove a service instance from a workflow configuration.

Inputs:

- **run**:

```
"workflow.ServiceRemove"
```

- **id**: the instance id

```
"number"
```

- **taskID**:

```
"number"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "success": "boolean"
}
```

A.11 AttachmentJSON

Returns the json specification of attachments and json geometry objects.

Inputs:

- **run**:

```
"AttachmentJSON"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "attachment": "attachment",
  "jsongeometry": "jsongeometry"
}
```

A.12 user.Logout

resets the connection to an unauthenticated state

Inputs:

- **run**:

```
"user.Logout"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "success": "boolean"
}
```

A.13 RemoteDeregister

Deregisters a client from a service registration. This will cancel any running service call, cancel all subscriptions of this client and, if the service to be deregistered is the last remaining of its kind, its serviceName will be removedfrom the list of available services.

Inputs:

- **run**:

```
"RemoteDeregister"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR result**:

```
{
  "deregisteredName": "string",
  "id": "number",
  "nodeIP": "string",
  "remainsAvailable": "boolean"
}
```

A.14 GetStartupTime

Returns the startup of time of Luci.

Inputs:

- **run**:

```
"GetStartupTime"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "startupTime": "number"
}
```

A.15 user.List

Get a list of users

Inputs:

- **run**:

```
"user.List"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "groups": {
    "{ id": {
      "id": "number",
      "owner": "number"
    }
  },
  "users": {
    "{ email": {
      "email": "string",
      "groupIDs": "list",
      "id": "number"
    }
  }
}
```

A.16 task.UnsubscribeFrom

Unsubscribe from a service or a list of taskIDs.

Inputs:

- **run**:

```
"task.UnsubscribeFrom"
```

- **XOR serviceName**:

```
"string"
```

- **XOR taskIDs**:

```
"list"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "success": "boolean"
}
```

A.17 task.Remove

Remove tasks.

Inputs:

- **run**:

```
"task.Remove"
```

- **XOR serviceName**: removes all tasks representing the service with this name

```
"string"
```

- **XOR taskIDs**: a list of taskIDs to be removed

```
"list"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR result**:

```
{
  "taskIDs": "list"
}
```

A.18 workflow.List

Returns a list of all workflows (not only root workflows).

Inputs:

- **run**:

```
"workflow.List"
```

- **OPT name**: restrict the list to only one entry, e.g. {'1':'New Workflow'}

```
"string"
```

Outputs:

- **result**: {'taskID':'name'}, not a list but a json object; since keys in json cannot be numbers, the taskIDs are converted to string

```
{
  "ANY numberAsString": "string"
}
```

A.19 workflow.UpdateIO

Updates in & outputs of a workflow; difference to task.Update: it does not set the values for in & outputs, but actually creates in & outputs dynamically for a workflow, its key, type and default value

Inputs:

- **run**:

```
"workflow.UpdateIO"
```

- **taskID**:

```
"number"
```

- **OPT inputs**:

```
{
  "ANY keyname": {
    "default": "error",
    "type": "string"
  }
}
```

- **OPT outputs**:

```
{
  "ANY keyname": {
    "default": "error",
    "type": "string"
  }
}
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR result**:

```
{
  "elements": "list",
  "inputSchema": "json",
  "inputs": "json",
  "listensToDone": "list",
  "name": "string",
  "parentID": "number",
  "position": "json",
  "services": "list",
  "taskID": "number"
}
```

A.20 task.Revert

Reverts a task to the last saved state.

Inputs:

- **run**:

```
"task.Revert"
```

- **taskID**:

```
"number"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR result**: refer to ¡a href='#task.get'¿task.Get¡/a¿ for descriptions.

```
{
  "OPT elements": "list",
  "OPT services": "list",
  "inputSchema": "json",
  "inputs": "json",
  "listensToDone": "list",
  "name": "string",
  "parentID": "number",
  "position": "json",
  "taskID": "number"
}
```

A.21 ServiceList

Get a list of all loaded and registered services as a list of service names.

Inputs:

- **run**:

```
"ServiceList"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "installed": "list",
  "serviceNames": "list"
}
```

A.22 ServiceInfo

Returns all known information on selected services such as a short description like this one, in- and output description and an example call.

Inputs:

- **run**:

```
"ServiceInfo"
```

- **OPT inclDescr**: indicates whether descriptions such as this one should be included in the result

```
"boolean"
```

- **OPT serviceNames**: Optional list of selected service names for which information should be returned. If no such list is given, the result will show the structure of general json structures such as 'attachment', 'jsongeometry'

```
"list"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR result**:

```
{
  "ANY serviceName": {
    "OPT numNodes": "number",
    "example": "json",
    "inputs": "json",
    "outputs": "json"
  }
}
```

A.23 task.Get

Returns infos about a task (requesting a taskID) or returns a list if ids representingthe same service

Inputs:

- **run**:

```
"task.Get"
```

- **XOR serviceName**: requests a list of taskID representing the same service

```
"string"
```

- **XOR taskID**: requests infos about a specific task

```
"number"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR result**:

```
{
  "XOR task": {
    "OPT elements": "list",
    "OPT services": "list",
    "inputSchema": "json",
    "inputs": "json",
    "listensToDone": "list",
    "name": "string",
    "parentID": "number",
    "position": "json",
    "taskID": "number"
  },
  "XOR taskIDs": "list"
}
```

A.24 user.Authenticate

Authenticates a user

Inputs:

- **run**:

```
"user.Authenticate"
```

- **email**:

```
"string"
```

- **password**:

```
"string"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "success": "boolean"
}
```

A.25 workflow.Safe

Saves a workflow to disk (in Luci). Luci uses H2's mvstore to serialize workflows to disk. In the future this would also allow to restore earlier versions of the workflow (no version tree, but linear versions).

Inputs:

- **run**:

```
"workflow.Safe"
```

- **taskID**:

```
"number"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR result**:

```
{
  "change": "boolean"
}
```

A.26 workflow.ServiceGet

Gets the services, the instance ids and corresponding startup args associated with this workflow.

Inputs:

- **run**:

```
"workflow.ServiceGet"
```

- **taskID**: the taskID to indentify the workflow

```
"number"
```

- **OPT ids**: filter the instances to show only the ones with the given ids

```
"list"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "ANY IP": {
    "ANY id": {
      "args": "string",
      "isAutoID": "boolean",
      "serviceName": "string"
    }
  }
}
```

A.27 test.Delay

Used to simulate a service with a long calculation time.

Inputs:

- **run**:

```
"test.Delay"
```

- **seconds**:

```
"number"
```

- **ANY inputs**: any inputs are being forwarded

```
"any"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "ANY outputs": "any"
}
```

A.28 test.Error

A service that simulates an error being thrown.

Inputs:

- **run**:

```
"test.Error"
```

- **OPT message**: optional error message

```
"string"
```

Outputs:

- **error**:

```
"string"
```

A.29 user.Permission

Allows to restrict given services only to given userIDs/groupIDs.

Inputs:

- **run**:

```
"user.Permission"
```

- **OPT add**:

```
{
  "ANY serviceName": "list"
}
```

- **OPT remove**:

```
{
  "ANY serviceName": "list"
}
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "ANY serviceName": "list"
}
```

A.30 workflow.ServiceStart

Start up all the services that are configured to run with the given workflow (taskID).

Inputs:

- **run**:

```
"workflow.ServiceStart"
```

- **taskID**:

```
"number"
```

- **OPT serviceIDs**: optionally restrict the instances to be started only to the given IDs.

```
"list"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "started": {
    "ANY IP": {
      "ANY serviceName": "number"
    }
  }
}
```

A.31 user.Create

Create a user by giving username, email, and sha1 encrypted password representedwith hexadecimals

Inputs:

- **run**:

```
"user.Create"
```

- **email**:

```
"string"
```

- **password**:

```
"string"
```

- **OPT city**:

```
"string"
```

- **OPT country**:

```
"string"
```

- **OPT name**:

```
"string"
```

- **OPT organization**:

```
"string"
```

- **OPT street**:

```
"string"
```

- **OPT surname**:

```
"string"
```

- **OPT zip**:

```
"number"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "id": "number"
}
```

A.32 ServiceControl

Used to remotely administrate installed services on a selected machine or on allmachines having a 'ServiceControl' remote service running.

Inputs:

- **run**:

```
"ServiceControl"
```

- **XOR info**: get filtered information about either selected machines (nodes) or about selectedserviceNames on all machines or full information about all machines in case this value is 'null'

```
{
  "\"null\"": "error",
  "{XOR nodes": {
    "OPT threadPool": "boolean",
    "XOR nodes": [
      "string"
    ],
    "XOR serviceNames": [
      "string"
    ]
  }
}
```

- **XOR install**: install files on either selected machines (nodes) or on all machines (services)

```
{
  "XOR nodes": {
    "ANY IP": [
      "attachment"
    ]
  },
  "XOR services": [
    "attachment"
  ]
}
```

- **XOR remove**: remove services identified by their name (serviceName) either on selected machines (nodes) or on all machines (services)

```
{
  "XOR nodes": {
    "ANY IP": [
      "string"
    ]
  },
  "XOR services": [
    "string"
  ]
}
```

- **XOR start**: startup a given number of services either globally or on selected machines (nodes)

```
{
  "XOR nodes": {
    "ANY IP": {
      "ANY serviceName": {
        "\"number\",": "error",
        "{ANY id": {
          "ANY id": "string"
        }
      }
    }
  },
  "XOR services": {
    "ANY serviceName": "number"
  }
}
```

- **XOR stop**: stop a given amount of services either globally or on specified machines (nodes).

```
{
  "XOR nodes": {
    "ANY IP": {
      "ANY serviceName": "number"
    }
  },
  "XOR services": {
    "ANY serviceName": "number"
  }
}
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR result**:

```
{
  "OPT errors": "list",
  "XOR info": {
    "OPT quickPool": {
      "avgWaitingTime": "number",
      "maxPoolSize": "number",
      "numCalls": "number",
      "poolSize": "number",
      "waitingInQueue": "number"
    },
    "OPT slowPool": {
      "avgWaitingTime": "number",
      "maxPoolSize": "number",
      "numCalls": "number",
      "poolSize": "number",
      "waitingInQueue": "number"
    },
    "XOR nodes": {
      "ANY ip": {
        "OPT installedServices": {
          "ANY serviceName": {
            "exec": "string",
            "numRequestedCPUCores": "number"
          }
        },
        "OPT system": {
          "CPU": {
            "archNumBits": "number",
            "endian": "string",
            "numCores": "number"
          },
          "name": "string",
          "osname": "string",
          "osversion": "string"
        },
        "services": {
          "busy": {
            "ANY serviceName": "number"
          },
          "idle": {
            "ANY serviceName": "number"
          },
          "running": {
            "ANY serviceName": "number"
          }
        }
      }
    },
    "XOR services": {
      "ANY serviceName": {
        "OPT available": "number",
        "OPT nodes": "list",
        "OPT running": "number",
        "avgDuration": "number",
        "numCalls": "number"
      }
    },
    "remoteSummary": {
      "busy": "number",
      "idle": "number"
    }
  },
  "XOR installed": {
    "ANY IP": [
      "string"
    ]
  },
  "XOR removed": {
    "ANY IP": [
      "string"
    ]
  },
  "XOR started": {
    "ANY IP": {
      "ANY serviceName": "number"
    }
  },
  "XOR stopped": {
    "ANY IP": {
      "ANY serviceName": "number"
    }
  }
}
```

A.33 workflow.ServiceUpdate

Update a service instance that is coupled to a workflow configuration.

Inputs:

- **run**:

```
"workflow.ServiceUpdate"
```

- **args**: arguments to be used to start the service with

```
"string"
```

- **id**: the instance id

```
"number"
```

- **ip**: the IPv4 of the machine on which the service should run

```
"string"
```

- **serviceName**: the name of the service to start (using exec parameter from the service installation)

```
"string"
```

- **taskID**:

```
"number"
```

- **OPT isAutoID**:

```
"boolean"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "success": "boolean"
}
```

A.34 Exists

Tests whether a service with the given serviceName or a task with given taskID exists.

Inputs:

- **run**:

```
"Exists"
```

- **XOR instanceID**:

```
"number"
```

- **XOR serviceName**: the name of the service to be tested

```
"string"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "exists": "boolean"
}
```

A.35 RemoteRegister

Registers a client as a service.

Inputs:

- **run**:

```
"RemoteRegister"
```

- **description**:

```
"string"
```

- **exampleCall**: Mandatory json object that shows an example call. Since OPT/XOR/ANY modifiers are only allowed in in & output specifications and not in calls, an example call shows very clearly how a call could look like.

```
"json"
```

- **serviceName**: mandatory string that identifies the service

```
"string"
```

- **OPT id**: Optional int to be used to identify a remote service. E.g. for viewer services it might be important to be able to identify remote service instances. If the id is taken already an error will be thrown / registration aborted.

```
"number"
```

- **OPT inputs**: Optional json object holding the input description to be included by 'ServiceInfo' / the API

```
"json"
```

- **OPT outputs**: Optional json object holding the output description to be included by 'ServiceInfo' / the API

```
"json"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR result**:

```
{
  "id": "number",
  "nodeIP": "string",
  "registeredName": "string"
}
```

A.36 user.Update

Udpate user information and does sanity checks on some of the properties.

Inputs:

- **run**:

```
"user.Update"
```

- **id**:

```
"number"
```

- **OPT city**:

```
"string"
```

- **OPT country**:

```
"string"
```

- **OPT email**:

```
"string"
```

- **OPT name**:

```
"string"
```

- **OPT organization**:

```
"string"
```

- **OPT password**:

```
"string"
```

- **OPT street**:

```
"string"
```

- **OPT surname**:

```
"string"
```

- **OPT zip**:

```
"number"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "success": "boolean"
}
```

A.37 Download

Downloads a file from a given URL and stores is in Luci’s attachment folder as &lt;md5 checksum&gt;.&lt;format&gt;

Inputs:

- **run**:

```
"Download"
```

- **url**: the URL to be used

```
"string"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR progress**:

```
"json"
```

- **XOR result**:

```
{
  "checksum": "string",
  "format": "string",
  "length": "number"
}
```

A.38 task.Update

Update a the input values / subscriptions of a task.

Inputs:

- **run**:

```
"task.Update"
```

- **taskID**:

```
"number"
```

- **OPT inputs**: inputs to be changed; subscriptions are represented by ’property’:{’taskID’:’number’,’key’:’s
(output key)’}

```
"json"
```

- **OPT listensToDone**: a list of taskIDs to which the task should listen generally (=not to a specific output)

```
"list"
```

- **OPT name**: the name of the task (can be different to service name)

```
"string"
```

- **OPT outputs**: if the task to be updated is a workflow that is contained by another workflow (e.g. a group) the workflow can have outputs that might be linked to outputs of some task it contains

```
"json"
```

- **OPT position**:

```
{
  "x": "number",
  "y": "number"
}
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR result**:

```
{
  "OPT unknownKeys": "list",
  "taskID": "number"
}
```

A.39 workflow.Create

Create a workflow.

Inputs:

- **run**:

```
"workflow.Create"
```

- **OPT group**: a list of taskIDs that should be grouped together into a workflow

```
"list"
```

- **OPT name**:

```
"string"
```

Outputs:

- **XOR error**:

```
"string"
```

- **XOR result**:

```
{
  "elements": "list",
  "inputSchema": "json",
  "inputs": "json",
  "listensToDone": "list",
  "name": "string",
  "parentID": "number",
  "position": "json",
  "services": "list",
  "taskID": "number"
}
```