DELIVERABLE

Project Acronym:	**F4W**
Grant Agreement nbr:	**000000**
Project Title:	**FACTS4WORKERS**

RESTdesc demo input/output

Revision:

Authors:

Joachim Van Herwegen (iMinds – Ugent – Multimedia Lab)

REVISION HISTORY

Revision	Date	Author	Organisation	Description

Table of Contents

1 Introduction

This document describes the complete input/output flow generated by the demo use case, giving developers a clearer picture of what is required to make use of the RESTdesc server.

2 API flow

2.1 Worker → RESTdesc: Clean start

```
{}
```

This is the very first call to the server so an empty POST call to demo/next is sufficient. In the future when there are multiple use cases it will be necessary to identify which goals needs to be completed. This could be solved by posting a JSON object containing a ‘goal’ field that contains the goal ID, or the contents of the goal file depending on where these get stored.

2.2 RESTdesc → Worker: Ask the worker initial start data

```
{
    "http:body": {
        "message": "Please input the starting information. 'id' corresponds to the calibration ID."
    },
    "http:resp": {
        "http:body": {
            "id": "_:b1_sk4_2"
        }
    },
    "http:methodName": "GET",
    "http:requestURI": "http://askTheWorker/start",
    "output": "...",
    "data": [...]
}
```

Since the first ‘API’ call is a call to the worker, RESTdesc immediately returns a response. “http:body” always corresponds to the input to the ‘API’, so in this case we simply included a message for the worker. “http:resp”:”http:body” corresponds to the expected response. The worker input should be sent in the same JSON format.

2.3 Worker → RESTdesc: Send initial worker response

```
{
    "json": { "id": 1 },
    "eye": {
        "http:body": {
            "message": "Please input the starting information. 'id' corresponds to the calibration ID."
        },
        "http:resp": {
            "http:body": {
                "id": "_:b1_sk4_2"
            }
        },
        "http:methodName": "GET",
        "http:requestURI": "http://askTheWorker/start"
        "output": "...",
        "data": [...],
    }
}
```

A response to the RESTdesc server should always be a JSON object containing 2 fields. The first field, 'json', should contain the worker input in the correct JSON format. The 'eye' field should contain the RESTdesc message this is a response to. The server will use this to make sure all input values get assigned to the correct variables.

2.4 RESTdesc: Next step: Call API 1

```
{
    "http:resp": {
        "http:body": {
            "machine_parameters": [
                "_:b2_sk9_2",
                "_:b2_sk10_2",
                "_:b2_sk11_2",
                "_:b2_sk12_2"
            ],
            "tolerances": [
                { "min": "_:b2_sk5_2", "max": "_:b2_sk6_2" },
                { "min": "_:b2_sk7_2", "max": "_:b2_sk8_2" }
            ],
            "part_number": "_:b2_sk4_2",
            "operator": "_:b2_sk3_2"
        }
    },
    "http:methodName": "GET",
    "data": [...],
    "http:requestURI": "http://pacific-shore-4503.herokuapp.com/calibrations/1"
}
```

This is the response generated after receiving the previous input. Since the request URI is not a worker call, this response does not need to be sent anywhere. The response JSON here corresponds to the expected output format of the API. Since this is a GET request, there is no body. Currently our system automatically calls the next API if no user input is required. It is always possible to change this and just send a response again that indicates which API needs to be called next should this be preferred.

2.5 RESTdesc → API 1: Get calibration information

This is a simple GET call to `http://pacific-shore-4503.herokuapp.com/calibrations/1` so no body is required.

2.6 API 1 → RESTdesc: Send calibration information

```
{
    "id": 2,
    "operator": "Gianni",
    "part_number": "123",
    "machine_parameters": [ 1200.25, 0.0024, 13.7, 270 ],
    "tolerances": [
        { "min": 134.3, "max": 134.35 },
        { "min": 0.37, "max": 0.4 }
    ],
    "geometrical_dimension": [ 134.33, 0.38 ],
    "result": "ok",
    "suggested_parameters": [ 1200.25, 0.0024, 13.7, 270 ]
}
```

This is the full response from the API. It actually contains too much information since it already has a result value for example. This is not a problem since RESTdesc only takes the fields that correspond to the mapping in “http:resp”:”http:body” previously generated in section 2.4. All other values get ignored.

2.7 RESTdesc → Worker: Ask measurements

```
{
    "http:body": {
        "machine_parameters": [ "1200.25", "0.0024", "13.7", "270" ],
        "part_number": "123",
        "message": "Please measure a new part with the following settings."
    },
    "http:resp": {
        "http:body": {
            "geometrical_dimension": [ "_:b3_sk4_2", "_:b3_sk5_2" ]
        }
    },
    "http:methodName": "GET",
    "http:requestURI": "http://askTheWorker/doMeasurement",
    "output": "...",
    "data": [...]
}
```

The next API call is a worker API, so this information gets sent back to the client again. The “http:body” here contains all the information the worker needs to do his measurements: the machine parameters and the part number. Again it is indicated in which format the client should respond.

2.8 Worker → RESTdesc: Send measurements

```
{
    "json": {
        "geometrical_dimension": [ 134.34, 0.37 ]
    },
    "eye": {...}
}
```

This is similar to what is done in step 2.3.

2.9 RESTdesc: Next step: Call API 2

```
{
    "http:body": {
        "geometrical_dimension": [ 134.34, 0.37 ],
        "tolerances": [
            { "min": 134.3, "max": 134.35 },
            { "min": 0.37, "max": 0.4 }
        ],
        "machine_parameters": [ 1200.25, 0.0024, 13.7, 270 ],
        "part_number": "123",
        "operator": "Gianni"
    },
    "http:resp": {
        "http:body": {
            "suggested_parameters": [
                "_:b4_sk6_1",
                "_:b4_sk7_1",
                "_:b4_sk8_1",
                "_:b4_sk9_1"
            ],
            "result": "_:b4_sk5_4",
            "id": "_:b4_sk4_1"
        }
    },
    "http:methodName": "POST",
    "http:requestURI": "http://pacific-shore-4503.herokuapp.com/calibrations",
    "data": [...]
}
```

This is the EYE result after the second user input round. This is quite similar to the result of 2.4. Since this API requires a POST call, a body is included. This is the exact JSON the server will POST to the API URI.

2.10 RESTdesc → API 2: POST measurements

```
{
    "geometrical_dimension": [ 134.34, 0.37 ],
    "tolerances": [
        { "min": 134.3, "max": 134.35 },
        { "min": 0.37, "max": 0.4 }
    ],
    "machine_parameters": [ 1200.25, 0.0024, 13.7, 270 ],
    "part_number": "123",
    "operator": "Gianni"
}
```

The data that gets sent here is the body part shown in 2.9. Since it is a normal API, the server does the call again.

2.11 API 2 → RESTdesc: POST results

```
{
    "id": 3,
    "operator": "Gianni",
    "part_number": "123",
    "machine_parameters": [ 1200.25, 0.0024, 13.7, 270 ],
    "tolerances": [
        { "min": 134.3, "max": 134.35 },
        { "min": 0.37, "max": 0.4 }
    ],
    "geometrical_dimension": [ 134.34, 0.37 ],
    "result": "ok",
    "suggested_parameters": [ 1200.25, 0.0024, 13.7, 270 ]
}
```

This is the JSON response of API 2. When we add this information to EYE, it will indicate our goal has been reached (result == “ok”). If the result is “recalibrate” instead of “ok”, the RESTdesc server will return a response similar to the one in 2.7.