On Demand Secure Isolation

D3.3
ODSI Management Reference Implementation (public)

Project acronym	ODSI
Project title	On Demand Secure Isolation
Advantage	
Deliverable number	D3.3
Deliverable name	ODSI Management Reference Implementation
Version	V 0.7
Work package	WP 3 – ODSI system Management
Lead beneficiary	ORANGE
Authors	NEXTEL S.A.
Nature	R – Report
Dissemination level	PU – Public
Delivery date	16/03/2018

Executive Summary

This document describes the solution implemented as an ODSI compliant system. It is designed to offer a secure working model of a solution offered to an end user to manage remote devices.

The solution implements the basic elements of ODSI, isolation, secure communications and management capabilities. All the elements have been described and include a technical description of the interfaces they offer so integration is achieved.

Table of Contents

1 Introduction

The current proposal aims to offer a dynamic environment where different applications can be securely managed and run in a single device by means of Isolation and secure communications.

The goal is to offer a complex architecture where a device can run different processes, isolated one from the others, so whenever one is compromised or malfunctioning it does not affect the others. This is going to be offered in a managed environment.

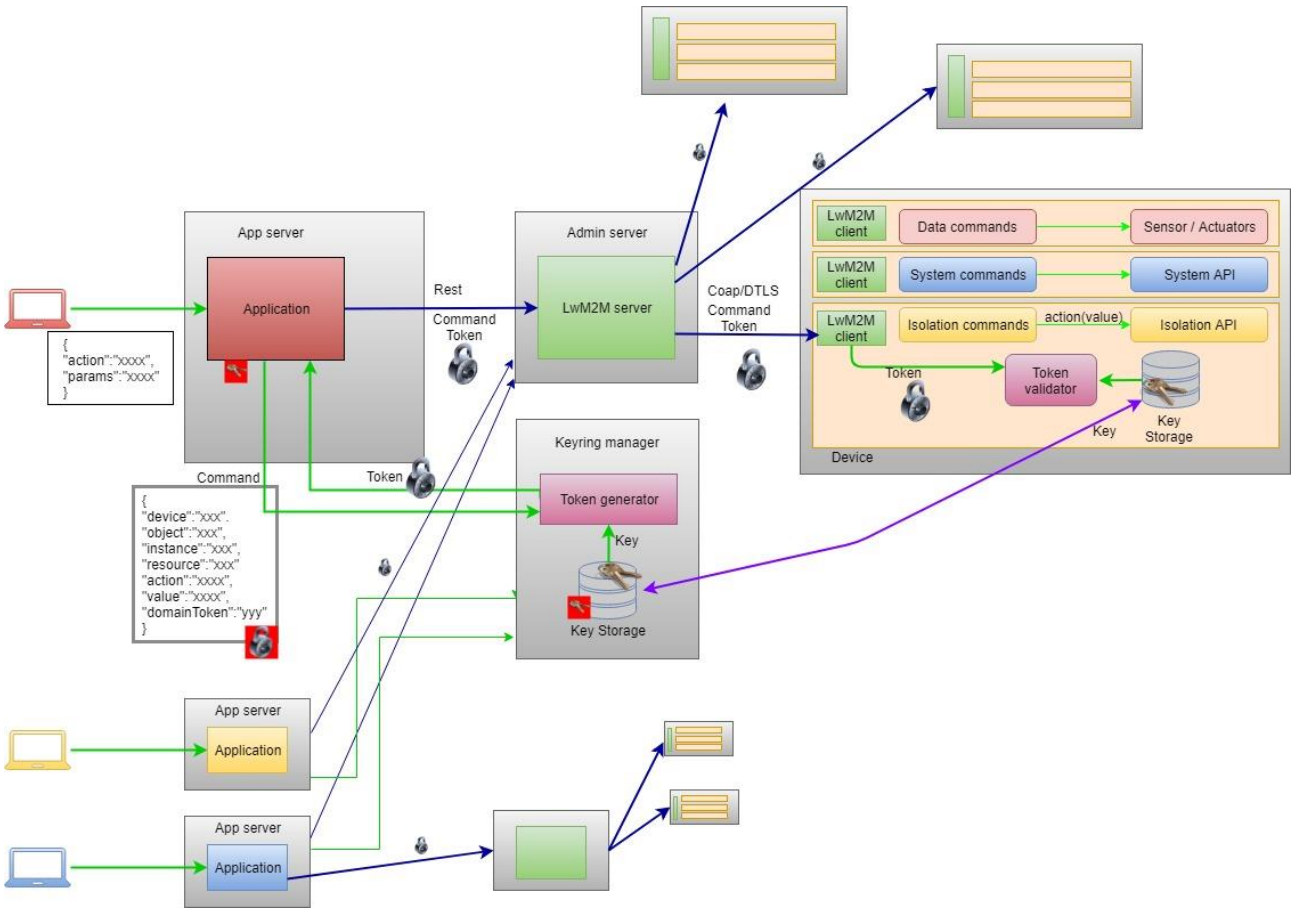
On the other side, the management environment is designed to interact with different devices, extending the isolation to a network of hosts that offer a service in a coordinated way, but isolated from other elements that share network or even hardware.

The document has been structured to describe the complete solution, first by describing the general architecture as it has been developed. The second part offers a more detailed description of each element and its components and functions. The last part offers a technical approach to the interfaces offered in the different levels.

2 General Architecture to Management Security Isolation

2.1 Description

The proposed architecture is formed by a series of elements that will interact in a chain so the function is completed. The main function for the design is the management of the isolated domains, but the model is the same to be set for business specific applications.



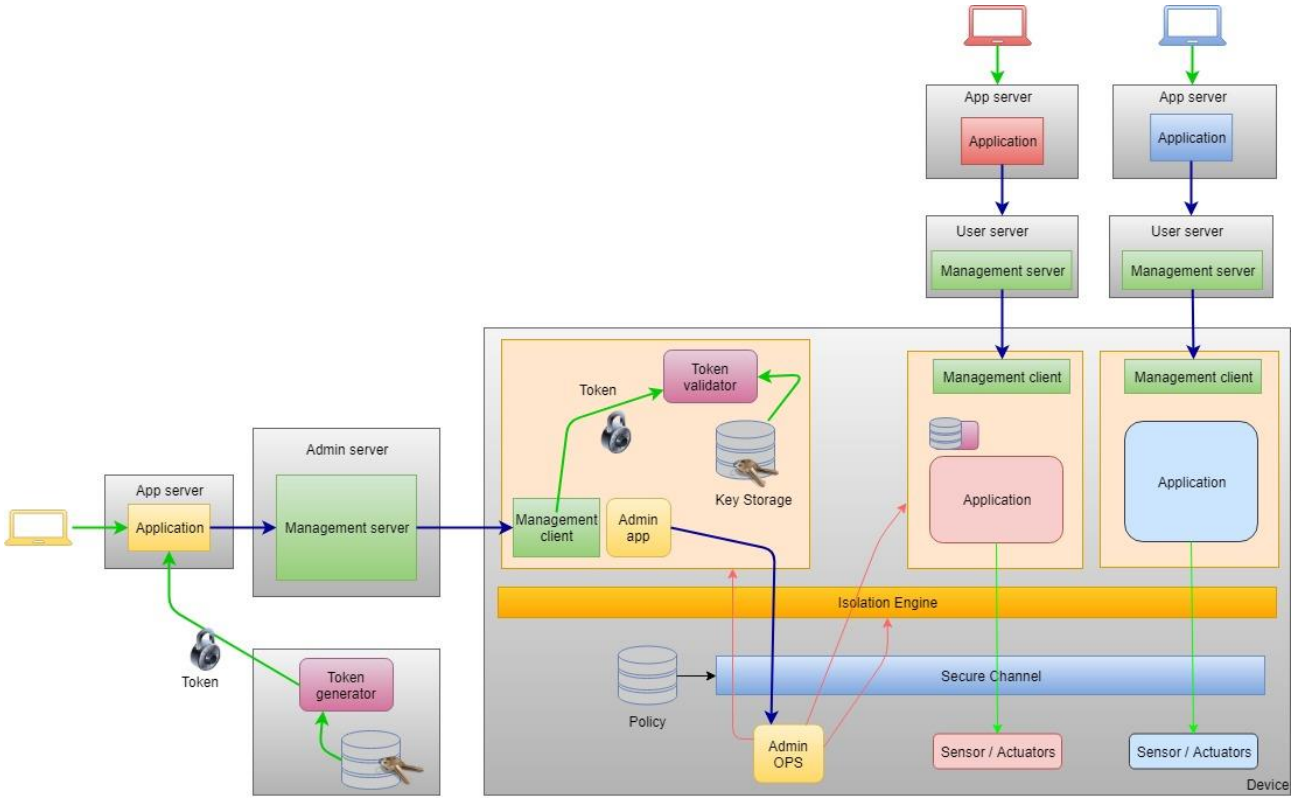
General architecture

Starting from the end user side, the model stars in a management application. This application manages its own data and users. At one point, the interaction of user and application will result on the need of requesting an action on the device. The action requested will have the form of a read or write operation over a known element. This could be a physical sensor or actuator to interact with or a data element in the system.

For this request to be accepted by the gateway server it must be first signed. So the application must contact the Keyring manager and request that the operation is signed for the destination device. This implies that a rest service is called and the data structure is send, signed by the application. Once the signature is validated the token generator will generate a token that includes signature for the end device.

The data structure that contains the command, the value and the token will be the content of the LwM2M request to be redirected through the Gateway server to the client. On reception, the client in the device will validate the token so the operation is authorized.

If the token is correct, the client will call the internal API to perform the requested operation and, if necessary, return a result. This will be done through a dedicated API on the device, specific to a function, say system management, isolation management or business dedicated functions.



System components

This architecture takes advantage of the Lightweight Machine to Machine Architecture [1] as a reference to manage multiple devices in a distributed environment.

2.2 Evaluation of requisites implementation

As collected in D3.2[2], requisites that apply to the secure management are compiled in the following table:

Requirement	Threat ID (Report T4.4) [6]
Prevent unauthorized introduction of software that might affect services performance in a trusted executed environment	T1: Unauthorized access to a domain T3: Manipulation of application data T7: Waste/abuse of system resources
Provide mechanisms to manage (add/modify/remove) secure domains	T1: Unauthorized access to a domain T4: manipulation of configuration data T5: manipulation of privilege credentials
Provide mechanisms to manage (add/modify/remove) keys	T1: Unauthorized access to a domain T4 : Manipulation of configuration data T5: Manipulation of privilege credentials
Prevent users from performing unauthorized actions	T1: Unauthorized to a domain
Ensure that message have not been tampered with during transit	T3: manipulation of application data T4: manipulation of configuration data T5: manipulation of privilege credentials T6: manipulation of audit data
Determine whether the message comes from a valid source	T1: Unauthorized access to a domain
Protect messages from eavesdropping	T1: Unauthorized access to a domain Unauthorized information flow between system modules

To complete this requirements the proposed architecture offers a series of security measures and functionalities that solve what is required.

Requirement	Resolution
Prevent unauthorized introduction of software that might affect services performance in a trusted executed environment	Managed administration will be the only entry point for software to run.
Provide mechanisms to manage	Administration operations module offers a secured

(add/modify/remove) secure domains	interface with the isolation mechanism.
Provide mechanisms to manage (add/modify/remove) keys	Secure Channel controls the applications that can perform privileged operations and offers an interface to manage the identity of the requesters.
Prevent users from performing unauthorized actions	Secure Channel as a secured intermediate channel that allows only certain operations.
Ensure that message have not been tampered with during transit	Token that accompanies every communication grants that no interceptor can modify the message.
Determine whether the message comes from a valid source	Token is signed by the requester so only valid sender will be accepted.
Protect messages from eavesdropping	Transport level protection mechanisms, (https, DTLS...) will ensure that no unauthorized listener can obtain the data.

3 Components

3.1 Application server

Both for system administration or dedicated functions inside an isolated domain, the system relays on an application that should offer an interface and functionality to the user, or work as an autonomous smart system. This is the edge of the ODSI proposal defined in the present document. This system is the one that will generate request to be performed in the device.

In case of user oriented applications, this application is responsible of managing users and permissions.

This application server is out of the scope of the ODSI project and is only provided for demonstration purposes.

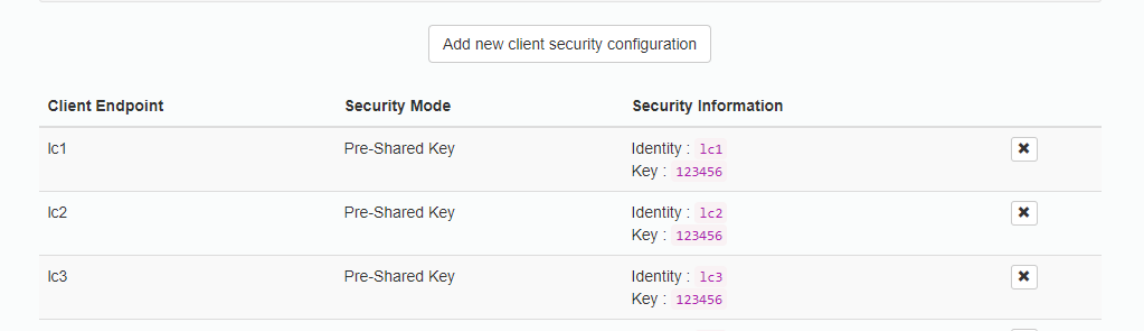
3.2 Admin server

Admin server contains a LwM2M server that will process every request sent by the application server. The LwM2M[3] [4] [5] [6] standard defines a protocol to register, communicate and manage distributed devices.

This will have the form of a PUT request where the value will contain a JSON[6] representation of the operation to be processed. It will include the following:

- Destination element, Device, domain, object, instance…
- Command. Read, write, exec, delete.
- Value
- Token. Signature of the command

LwM2M server offers an interface to define device connection credentials so the communication between client and server can be established securely.



LwM2M Identity management

3.3 Keystore Manager

The Keystore Manager is the module responsible for the administration of the keys used for cryptographic purposes, and also for signatures. Each domain alias must have associated it's corresponding key to:

- Sign a token request to be sent to the Token Generator
- Verify the token request by the Token Generator
- Cipher the ODSI access token
- Verify the ODSI access token by the Token Validator

It is accessible by a REST api interface. The creation and distribution of the keys should be done at configuration time of each of the actors that require one.

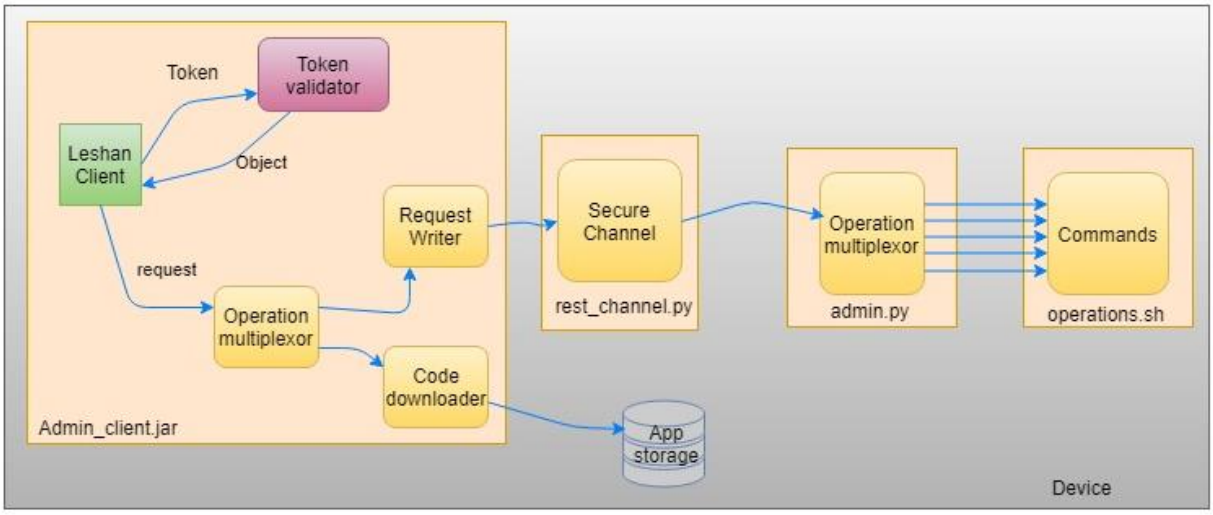
3.4 Admin client

On the device, requests are processed by a LwM2M client that will be subscribed to the server. This subscription is managed as a standard process in the LwM2M protocol. The request comes in the form of a PUT request where the content is a JSON data structure that contains all the data needed to define the operation.

The first step is to validate the token. For this operation, the Token Validator will select the key from the internal storage and validate the signature.

Once the operation is validated, the Operation Multiplexor will convert the read, write, exec, delete operations to commands to be requested in the device. Some may be data management and others may refer to system calls to perform certain operations. The request writer will create this system calls and send them to the channel.

The code downloader manages the deployment of the application code for each instance to be run. It makes the compiled code available in a known folder so the isolation environment can publish it.



Admin client internal architecture

3.5 Secure channel

Secure Channel allows the interaction of the process to be run in the docker with the external resources that require privileged access. It is a Rest interface that will link services, in the form of executable commands, and applications authorized to run them, defined as credentials to access. All this will be linked in a policy that will state which application can run which services.

A policy that relates a service and an application will define:

- Path to the executable program or script that performs the privileged operations.
- id and key for the application access.

The Rest interface will only be accessible from the local device. The url to run a command will be the following:

http://localhost:5000/service/gpio?identity=<id>&key=<key>
Method: POST
Body:
{

```
    "service":"<service name>",
    "params":"<parameters>”
}
```

The result of the execution will be included in the response.

3.6 Admin operations

The isolated system offers a command line interface that works as an API for the system calls to be executed.

3.7 Managed client and operations

The model described for the administrator is designed as a general purpose exchange. It will offer as a result the deployment of an business oriented application. This application is supposed to work against different resources in the system. In order to offer the same security levels as the ones described in this model the same architecture can be deployed, using an application server, gateway server, application client and io interface. Optionally, the user can deploy a single autonomous application that will run isolated but not making use of the secure channel and token security.

4 Interfaces

4.1 Keystore Manager Interface

The Keystore Manager interface is a REST service that administers the keys used for ciphering and/or signing by the different actors. It offers the actions:

Title	List domain aliases with an associated key
URL	/keys
Method	GET
Success Response	**Code:** 200 **MediaType:** application/json **Content:** { "aliases": [[string]] } **Example:** { "aliases": ["orange", "nextel", "device_1"] }
Error Response	**Code:** 500

Title	Generate a key for the specified domain alias
URL	/keys/{alias}
Method	POST
Success Response	**Code:** 201
Error Response	**Code:** 403 **MediaType:** text/plain **Content:** The associated key already exist OR **Code:** 500

Title	Obtains the key of the specified domain alias
URL	/keys/{alias}

Method	GET
Success Response	**Code:** 200 OR **Code:** 204
Error Response	**Code:** 500

Title	Verify that service is running
URL	/verify
Method	GET
Success Response	**Code:** 200 **MediaType:** text/plain **Content:** Keystore Manager RESTApi successfully started

4.2 Gateway Interface

Gateway in this proposal is using LwM2M as a communication model. Once the ODSI device is registered in the system, the objects defined for isolation management are the following:

- App Name/26261/0/0
- Identity/26261/0/1
- Access key/26261/0/2
- Local Port/26261/0/3
- Code file name/26261/0/4
- Created?/26261/0/5
- Running?/26261/0/6
- Domain creation/26261/0/7
- Domain deletion/26261/0/8
- Start Application/26261/0/9
- Stop Application/26261/0/10
- Refresh data/26261/0/11

All these objects will support PUT operations, so commands are encapsulated in the request. The content of the request is as the following:

```
{
	"id": 0,
	"value": "{
		\"domain\":\"DOMAIN NAME\",
		\"device\":\"DEVICE NAME\",
		\"operation\":{
			\"code\":\"COMMAND\",
			\"objectId\":26261,
			\"objectInstanceId\":0,
			\"resourceId\":0,
			\"resourceInstanceId\":0
		},
		\"value\":\"VALUE\",
		\"token\":\"XXXXXXXXX\"
	}"
}
```

4.3 Token generator interface

The token generator interface is a REST service that generates an ODSI Security Token. It offers the actions:

Title	Generate ODSI Security Token
URL	/token
Method	POST

Data Params	**MediaType:** application/json
	Content:
	{
	"domain": [string],
	"cipheredCommand": [alphanumeric]
	}
	Example:
	{
	"domain": "orange",

	"cipheredCommand": "Ppr5fFAgtTWe-JAhRGvLrui-wCaJm_gFsTfDAJVS9anbjY2tvYmYffjV_ae2-7pOosEtD0Bj_vlw7zCbheg9O5zgtiOKrf134V6fNBRF3-89M40NNLA6zY4OXcsga9DMOoAF29zKHrUgE8yy2OM_dIHg_mfZe0KxTWGkUldKoYfsAEcakHU2_zjKZy2jxGVR2xkwyac4CCbOt1qNF_Jffg==" }
Success Response	**Code:** 200 **MediaType:** application/json **Content:** { "token": [alphanumeric] } **Example:** { "token": "QwnpfhEcsOoRvJjFbnybsqGPWvdeRxyQXoNSJvfxnAePbeFEhsPfW42f3BjMi3qt6YVI_90RTs4tAHx8o1hxarAGKDkju62yUsKNsavhbeIk9q31wNp2gWfTbQKhO--w" }
Error Response	**Code:** 400 **MediaType:** application/json **Content:** { "error": "Invalid token request" } OR **Code:** 500 **MediaType:** application/json **Content:** { "error": "Unsuccessful token generation request" }

Title	Verify that service is running
URL	/verify
Method	GET
Success Response	**Code:** 200 **MediaType:** text/plain **Content:** Token Generator RESTApi successfully started

4.4 Token validator interface

The token validator interface is a REST service that generates an ODSI Security Token. It offers the actions:

Title	Generate ODSI Security Token
URL	/token
Method	POST

Data Params	**MediaType:** application/json
	Content:
	{
	"domain": [string],
	"cipheredCommand": [alphanumeric]
	}
	Example:
	{
	"domain": "orange",
	"cipheredCommand": "Ppr5fFAgtTWe-JAhRGvLrui-wCaJm_gFsTfDAJVS9anbjY2tvYmYffjV_ae2-7pOosEtD0Bj_vlw7zCbheg9O5zgtiOKrf134V6fNBRF3-89M40NNLA6zY4OXcsga9DMOoAF29zKHrUgE8yy2OM_dIHg_mfZe0KxTWGkUldKoYfsAEcakHU2_zjKZy2jxGVR2xkwyac4CCbOt1qNF_Jffg=="
	}

Success Response	**Code:** 200
	MediaType: application/json
	Content: { "token": [alphanumeric] }
	Example:
	{
	"token": "QwnpfhEcsOoRvJjFbnybsqGPWvdeRxyQXoNSJvfxnAePbeFEhsPfW42f3BjMi3qt6YVI_90RTs4tAHx8o1hxarAGKDkju62yUsKNsavhbeIk9q31wNp2gWfTbQKhO--w"
	}

Error Response	**Code:** 400 **MediaType:** application/json **Content:** { "error": "Invalid token request" } OR **Code:** 500 **MediaType:** application/json **Content:** { "error": "Unsuccessful token generation request" }

Title	Verify that service is running
URL	/verify
Method	GET
Success Response	**Code:** 200 **MediaType:** text/plain **Content:** Token Generator RESTApi successfully started

4.5 Secure channel

Title	List services
URL	/service?identity=<id>&key=<key>
Method	GET
Success Response	**Code:** 200 **MediaType:** application/json **Content:** { "service": [[string]] } **Example:** { "service": ["admin", "gpio"] }
Error Response	**Code:** 200 **Content:**{"error":"Error description"}

Title	Add new service
URL	/service?identity=<id>&key=<key>
Method	PUT
Data Params	**MediaType:** application/json **Content:** { "service": [string], "command": [string]

	}
	Example:
	{
	"service":"sname",
	"command":"/opt/odsi/opmux/opmux.py"
	}

Success Response	**Code:** 200
Error Response	**Code:** 200 **Content:**{"error":"Error description"}

Title	Get service data - Lists existing applications
URL	/service/<servicename>?identity=<id>&key=<key>
Method	GET
Success Response	**Code:** 200 **MediaType:** application/json **Content:** { "applications": [[string]] } **Example:** { "applications": ["gpioreader", "gpioadmin"] }
Error Response	**Code:** 200 **Content:**{"error":"Error description"}

Title	Add application to service
URL	/service/<servicename>?identity=<id>&key=<key>
Method	PUT

Data Params	**MediaType:** application/json
	Content:
	{
	"application":[string],
	"identity":[string],
	"key":[string]
	}

	Example:
	{
	"application":"aname",
	"identity":"aident",
	"key":"akey"
	}

Success Response	**Code:** 200
Error Response	**Code:** 200 **Content:**{"error":"Error description"}

Title	Run command on service
URL	/service/<servicename>?identity=<id>&key=<key>
Method	POST

Data Params	**MediaType:** application/json
	Content:
	{
	"service":[string],
	"params":[string]
	}
	Example:
	{
	"service":"gpio",
	"service":"led1 on"
	}

Success Response	**Code:** 200
Error Response	**Code:** 200 **Content:**{"error":"Error description"}

Title	Delete service

URL	/service/<servicename>?identity=<id>&key=<key>
Method	DELETE
Success Response	**Code:** 200
Error Response	**Code:** 200 **Content:**{"error":"Error description"}

Title	Get application data
URL	/service/<servicename>/<appname>?identity=<id>&key=<key>
Method	GET

Success Response	**Code:** 200
	MediaType: application/json
	Content:
	{
	"application":[string],
	"identity":[string],
	"key":[string]
	}
	Example:
	{
	"application":"aname",
	"identity":"aident",
	"key":"akey"
	}

Error Response	**Code:** 200 **Content:**{"error":"Error description"}

Title	Delete app
URL	/service/<servicename>/<appname>?identity=<id>&key=<key>
Method	DELETE

Success Response	**Code:** 200
Error Response	**Code:** 200 **Content:**{"error":"Error description"}

4.6 Admin operations interface

The interface for the administration operations is a command line api where all the information is passed as parameters and the operation will be launched.

The available commands are this:

- create.- Creates an isolated environment.
- delete.- Cleans the isolated domain from the system.
- run.- Launches the existing domain
- stop.- Stops the domain
- list.- Lists existing domains.
- exists.- Checks existence of a domain
- isrun.- Checks whether domain is running.

The syntax of the command line is the following:

```
Usage:    opmux.sh COMMAND DOMAIN [PARAMS]
      API to manage domains in docker environments

Management Commands:
  create  Create new domain
          opmux.sh create DOMAIN IDENT KEY LPORT CODE
  delete  Delete domain
          opmux.sh delete DOMAIN IDENT
  run     Run domain application
          opmux.sh run DOMAIN IDENT LPORT
  stop    Stop domain application
          opmux.sh stop DOMAIN IDENT
  list    List domains and data
```

```
        opmux.sh list DOMAIN
exists  Check domain exists
        opmux.sh exists DOMAIN
            0: Exists
            1: Not exists
            2: Incomplete: Missing volume folder
            3: Incomplete: Missing dockerfile
            4: Incomplete: Missing container
isrun   Check domain is running
        opmux.sh isrun DOMAIN
            0: not running
            1: Running
```

The parameters are this:

- DOMAIN.- Unique (in the device) name for the domain
- IDENT.- Key identificator, as configured in the gateway server.
- KEY.- Secret key related to the identificator.
- LPORT.- Local port for gateway communication. Must be unique in the device.
- CODE.- Code filename to run. Must be present in the configured folder.

5 Conclusion

ODSI is defining a solution that is supposed to be valid in a very wide range of devices and architectures. Isolation, secure communications and management are the basis of a trusted environment, and any solution aiming to be compatible should fulfill them in a valid manner.

In this document we have described what the actual implementation of an ODSI environment that covers all the requisites of such a system. Isolation, management and secure communications among elements have been described.

From an end user that requests a service from an application to the device it is supposed to be the destination of that actions, we have defined a set of intermediate agents that will offer the functionalities that the project requires.

A token that gives end to end data security, an internal data channel that performs data access and a isolation manager that allows a complete control of the environment are the main solutions demonstrated.

The document also includes a detailed description of the interfaces offered by the different elements.

6 Glossary and terminology

Acronym	Description
LwM2M	Lightweight machine to machine
JSON	JavaScript Object Notation
REST	Representational State Transfer

7 References

[1] Lightweight Machine to Machine Architecture. http://www.openmobilealliance.org/release/LightweightM2M/V1_0-20170208-A/OMA-AD-LightweightM2M-V1_0-20170208-A.pdf
[2] ODSI 3.1 – ODSI platform constraints for system management
[3] Lightweight Machine to Machine Technical Specification. http://www.openmobilealliance.org/release/LightweightM2M/V1_0-20170208-A/OMA-TS-LightweightM2M-V1_0-20170208-A.pdf
[4] OMA. Open Mobile Alliance http://openmobilealliance.org/wp/index.html
[5] Complete index documentation for LwM2M http://openmobilealliance.org/release/LightweightM2M/
[6] Leshan OMA Lightweight M2M server and client in Java http://www.eclipse.org/leshan/
[7] https://www.json.org/