OpenIO Labs - System

OpenIO Labs System Architecture

UG101 (v2017.1) April 3, 2017

Revision History

Date	Version	Revisions
03/04/2017	2017.1	*Initial release.*

Table of Contents

Chapter 1: Introduction

The OpenIO Labs system represents a leap forward in the manner in which users can create and control experimental systems within a laboratory environment. The system comprises edge devices, cloud servers with the user accessing the cloud server using standard internet browsers. Within the OpenIO Labs system, the edge devices are referred to as Input Output Interfaces (IOIs). The user attaches a range of external devices and instruments to this edge device. Using the OpenIO Labs ScriptML programming system, the user can interact with these external devices using applications that they create in the language of their choice (e.g. C, Python, Java etc.).

The OpenIO Labs system was designed to address the requirements introduced by advanced Machine-to-Machine Heterogeneous Networks (M2M Het-Nets). Such networks are particularly prevalent in cases where both data retrieval and control are simultaneously required. There are a multitude of example deployments that can be considered, with two examples presented below.

1. Laboratory research environments which are requiring many types of sensors and actuators to be controlled, and for data to be collected, tagged and stored.
2. Industrial process control systems in which real time-process data is analysed and stored, and, based on the analysis certain actions and events are triggered.

If we consider the example illustrated in Figure 1: Typical M2M Het-Net Deployment we can see a flavour of the types of sensors and devices that we need to interconnect. The sensors and devices will range from low complexity to high complexity, and in addition, the interfaces to these sensors and devices will range from relatively simple to complex.

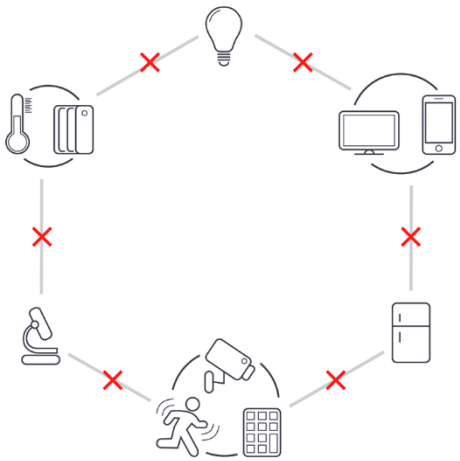

Figure 1: Typical M2M Het-Net Deployment

We can consider these sensors and devices to lie within different categories and accessed via different interfaces, as illustrated in Table 1: Examples of Sensors and Devices and Interfaces below.

Sensor / Device Type	***Example Interfaces***
Temperature, Humidity and other sensors	I2C, SPI and GPIO interfaces
Motors, Actuators, Switches and Relays	I2C, SPI and GPIO interfaces
ADC and DAC converters	I2C, SPI and parallel interfaces
Simple Instrumentation: Balances, pH meters	RS232 and USB interfaces
Standard Lab Equipment	USB interfaces with USBTMC stack
Complex Lab Equipment	USB interfaces with proprietary interfaces

Table 1: Examples of Sensors and Devices and Interfaces

What is clear from both the likely topology of the network and also the variety of the sensors and devices attached to it, is that the M2M-Het-Net needs to have the flexibility to deal with a range of devices and a range of complexity. In these and in all other similar deployments, the requirements can be reduced to a simple set:

1. Allow the system to interface to a range of sensors, instruments and devices via a range of different interfaces and different interface protocols.
2. Place intelligence at the network edges to allow for rapid processing of data, reduction in the transfer of data across the network, accommodate temporary network outages.
3. Utilise a common and industry standard protocol to connect the network edges to the network core.
4. Deploy high reliability cloud servers to offer high performance data acquisition, guaranteed availability, with full network scalability.

The approach adopted by OpenIO Labs to meet these requirements is to connect devices and sensors via an Input/Output Interface (IOI). The IOI is an intelligent edge node that interfaces to the sensors and devices via the range of interfaces outlined in Table 1. The IOI is connected to OpenIO Labs platform via the industry standard Light Weight M2M (LWM2M) communications protocol. The interconnection between these devices is illustrated in Figure 2: OpenIO Labs Base Architecture below.

The structure and operation of these nodes and the interface protocol are considered in more detail below.

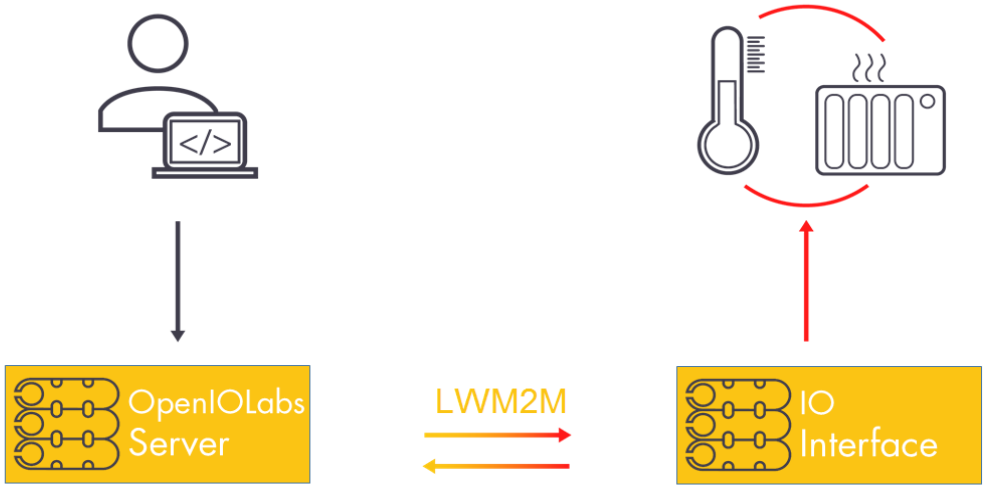

Figure 2: OpenIO Labs Base Architecture

OpenIO Labs Nodes and Protocols

The OpenIO Labs physical architecture is comprised of the three main entities illustrated in Figure 2. The Edge device (IOI), the interconnection communications network (using LWM2M) and the OpenIO Labs platform.

The OpenIO Labs IOIs are a family of edge node devices that are designed to interface to external devices and sensors. Each IOI provides a range of interfaces, with some examples of the interface types presented in Table 2: Example IOI Interface Types below.

Interface	Description
ADC	Many sensors and devices such as temperature sensors or humidity sensors interface via an analogue signal. The IOI provides an analogue interface via an ADC, processes the input signal and passes data to the OpenIO Labs platform via the LWM2M interface.
DAC	Some devices such as motors, LEDs and heaters require an analogue signal to allow control of their operation. The IOI provides analogue control of the DACs via the LWM2M interface from the OpenIO Labs platform.

I2C	There are many devices and sensors that utilize the simple I2C serial interface between the IOI and the device. The I2C interface links the sensor or device to the IOI and then the LWM2M protocol links the IOI to the server.
SPI	The Serial Peripheral Interface (bus) is another form of simple interface from the sensors and devices to the IOI. The IOI implements the client part of the SPI interface, interacts with the device and passes data to/from the server.
GPIO	The General Purpose IO interface provides a simple binary interface between the IOI and the sensors and devices. Output devices as simple as an LED or as complex as a motor can be controlled via the GPIO. Input devices such as light detectors can also be interfaced to the IOI via a GPIO interface. The GPIO interface can be used in a single bit (wire) or multiple bit (bus) configuration.
UART	Universal Asynchronous Receiver / Transmitter is a well established serial interface protocol that can be used with a range of input and output devices ranging from printers, GPS modules and PCs. The IOI provides the UART interface to the edge devices and links them to the server. These edge devices can be used for either input (data extraction) or output (device control).
USB	The USB interface connects the IOI to a range of USB compatible peripherals. For complex instruments such as oscilloscopes or signal analysers, the USBTMC protocol can be used to allow the IOI to control and monitor these complex instruments.

Table 2: Example IOI Interface Types

In addition to the various IO protocols, each IOI also includes high speed processors based on a multi-core ARM CPU. The CPU allows for the control and management of the edge devices and sensors, provides edge processing as well as the interface protocol back to the OpenIO Labs platform.

This User Guide is an introduction to the OpenIO Labs system and it's architecture. Within the following sections of this guide we will further consider:

1. Cloud server
2. Interconnection protocol between the server and the edge devices
3. Edge devices (IOIs)
4. User access to the system through a standard Internet browser.
5. ScriptML Programming System

Chapter 2: OpenIO Lab Book Server

The OpenIO Labs Cloud server can be hosted on a range of cloud server platforms. The server OS is Linux and implements most of its functionality with the Node.js family of libraries and tools. The server includes both an SQL and a NoSQL data base to store the various forms of data that are to be collected and saved. The server implements a range of security mechanisms to ensure the integrity of both the server, the user and the user's data. Data security is additionally guaranteed with regular database backups. Access to the server is via the LWM2M Machine-to-Machine protocol. In this chapter we will review these different aspects of the server's structure and operation.

Server Platform

The server architecture is designed to be portable across a range of physical hosting platforms. A standard Linux distribution is used as the OS for the server. Currently the OpenIO Lab Book server is hosted on the Amazon AWS Cloud server system. AWS together with Amazon's Elastic Compute Cloud (EC2) allow for a highly scalable architecture in which CPU resources, memory and data storage can be scaled according to the requirements of the deployment.

The key benefits for the use of cloud servers are scalability and reliability. The scalability allows for CPU and memory resources to be dynamically activated based on current user loads. This ensures that a consistent performance is offered to all users.

The server reliability allows for hot-fail-over for the servers. This means that in the event of a server failure, the processing load will be switched to a paired server ensuring a minimum of disruption to services provided by the server.

Security

A key part of any system such as the OpenIO Labs system is the security of the system. A number of security features have been built into the system and include:

1. Device authentication
2. User Authentication
3. Data encryption
4. DoS Attacks

Device authentication ensures that only valid OpenIO Labs approved devices can be connected to the server. The device authentication is achieved through the use of X.509 certificates. These certificates are signed by a trusted certificate vendor, and from which the individual device

certificates are signed. This creates a chain of trust that ensures only valid devices can be used.

User authentication ensures that only valid users can access the system. The user authentication in the form of user identity and passwords prevents unauthorised users accessing the system. With user authentication, the user can be reassured that only they can access the system and that their data is secure.

Data encryption is used between the IOI and the cloud server. The LWM2M DTLS protocol is used[1] to ensure that all packets sent between the IOI and the server are encrypted. This will ensure that threats such as man-in-the-middle attacks are prevented, as are the eavesdropping of data.

The Denial of Service attacks are prevented through the use of the Amazon AWS Shield technology. Should a DOS attack occur, the Amazon Shield will limit the impact that this has on the server performance.

Server Databases

The user can ensure that all data sent to the server is saved in a database. To support this, two different database technologies are used.

First, there is an SQL based database, which in the OpenIO Labs system is MariaDB, and in second there is a NoSQL database which in the OpenIO Labs system is based on the MongoDB database.

The SQL database is the main database for storing the objects and resources sent from the IOI to the server, and the NoSQL database is used to store large data objects such as images and streamed data. Both databases are seamlessly integrated together, so the user does not need to worry about how and where the data is stored.

Server Access Protocols

Access to the server from the external devices is normally via the LWM2M protocol. This protocol has been designed to inter-work with low complexity edge devices. The LWM2M is based on CoAP, a light weight alternative to HTTP that uses UDP in place of TCP. This ensures that there is a fast and efficient path between the edge devices and the server, whilst ensuring transfer reliability.

In addition to the LWM2M access protocol, there is a RESTful interface to the server. The RESTful interface allows users to access the server from software applications such as Matlab. This allows the user to mix both measured data collected via the LWM2M interface with data that has been computed with applications such as Matlab.

1 DTLS support will be available in the OpenIO Labs Phase 2 system release.

Chapter 3: IOI to Server Protocol

The IOIs and the OpenIO Labs platform communicate using the industry standard LWM2M protocol. The LWM2M protocol is responsible for managing the communications between the IOI and the server. These communications include Registration, Data Transfer, Control and Notification messages.

One of the key components of the LWM2M protocol is the definition of the Objects and Resources it uses. The client and server communicate using these Objects and Resources to exchange data. The Objects can be considered as the "Thing" we wish to exchange between client and server, and the Resources are the details and attributes for this "Thing".

The LWM2M specification defines a protocol which regulates the interactions between a client and a server. The details of the specification are beyond the scope of this document, but are fully defined in the LWM2M Technical Specification. The OpenIO Labs system implements both the client and the server end of these exchange, and uses the LWM2M protocol to support these exchanges.

The OpenIO Labs client further divides the structure for the use of the LWM2M protocol into two parts. These two parts are the client specific functions and the second the ScriptML functions (ScriptML is discussed in more detail later).

The client functions are responsible for managing the relationship between the client and the server using the LWM2M protocol (Registration, De-registration, Read, Write, Observe etc.). These functions are managed in such a way that the end user (ScriptML script writer) is not troubled by the details for these aspects of the LWM2M protocol.

The ScriptML part is what the user will see and interact with. The client exposes a number of simple APIs that the user will use to create, manage and control objects and resources attached to the client (or device to use the LWM2M term).

By structuring the system in this way, the end user is presented with a very simple programming environment in which they can choose from a range of languages (such as C, Python, C++ etc.) to create complex experimental systems. The user will focus on the logic of the application that they wish to develop, but not be troubled with the complexity of interacting with the server using LWM2M.

LWM2M Objects and Resources

LWM2M is based on the concept of Objects and Resources. The Objects are the things that we may want to measure or control, and the Resources can be thought of as the attributes of the Object that define the details of the Object.

Within LWM2M an Object can be a range of things, from a physical device such as a temperature

measurement object, to the LWM2M security object. Pretty much anything that we need to interact with is defined as an object.

If we consider an example, such as the temperature measurement object. The object has a defined identification number (Object Id), which in the case of the temperature object is 3303. This temperature object has a number of resources associated with it. Similar to the Object Identifier, each resource also has an identifier to allow the client and server to exchange data relating to that specific resource. For ease of understanding, the resources that the Temperature sensor may support are presented in the following table.

Resource	ID	Description
Sensor Value	5700	Last or Current Measured Value from the Sensor
Sensor Units	5701	Measurement Units Definition e.g. 'Cel' for Temperature in Celsius.
Min Measured Value	5601	The minimum value measured by the sensor since power ON or reset
Max Measured Value	5602	The maximum value measured by the sensor since power ON or reset
Min Range Value	5603	The minimum value that can be measured by the sensor
Max Range Value	5604	The maximum value that can be measured by the sensor
Reset Min Max to Current	5605	Reset the Min and Max Measured Values to Current Value

Resources Used by the Temperature Object (Object ID=3303)

In addition to the information presented in the table above, each of the resources has a number of associated parameters such as whether it is Read only, Write only, or Read and Write. The data type is also defined for the resource (such as Integer, Float, String, Boolean etc.). The number of instances of the resource associated with a specific object are defined, and finally, whether a specific resource is mandatory or optional for that specific object.

It should be noted that the Resource ID is unique for a resource type, but that resource type could be used in a range of objects. So, for example, Resource ID 5700 for the sensor value, will also be used in sensors such as Barometers, Magnetometers and Ammeters, amongst many others.

So within the client we can have multiple objects of multiple types. Each object will contain multiple resources of multiple types.

The Internet Protocol for Smart Objects (IPSO) have defined a number of Objects and Resources that may be used with an LWM2M system. An example set of these Objects and Resources can be found in the literature.

Chapter 4: IOI Edge Devices

IOIs are the edge devices that attach to the server. In this section we will consider the role and use of the Edge Device with reference to a specific edge device implemented by Open IO Labs, namely the GP2.

The OpenIO Labs system allows sensor and control devices to be connected to an edge device such as the GP2, and from the GP2 to the cloud server. The GP2 allows the user to create a range of control scripts that they can use to control the external devices attached to the GP2, collect data from these devices, process the data and then push the data to the cloud server for presentation and storage. The connection between the GP2 and the cloud server uses the LWM2M protocol described previously, and which is designed as a low latency, scalable protocol for Machine-to-Machine communications.

The OpenIO Labs GP2 is a very flexible edge processing device. With the GP2, its various interfaces, and the use of ScriptML, the user is able to connect a large number of devices and instruments to the GP2 for control, data collection and processing.

With the LWM2M interface to the OpenIO Labs Cloud server, the user can create scripts via a web browser, download these scripts to the GP2 where they can control devices and sensors attached to the GP2 as well as collect data from these devices. This data can be further processed within the GP2 before pushing the data to the OpenIO Labs Cloud server for further processing, presentation and storage.

GP2 Specifications

The specifications for the OpenIO Labs GP2 IOI are presented in the table below. The GP2 consists of a high specification embedded processor with a range of peripherals such as USB, I2C bus, GPIO and UART. The user has full access to these peripherals via the OpenIO Labs ScriptML programming system, in which the users can create applications using languages of their choice such as in C or in Python.

Feature	Description
CPU	1.2 GHz Quad core ARM Cortex-A53 processor
RAM	1 GByte SDRAM
USB	4 Port USB 2.0 supporting maximum transfer rates of 480 Mbit/s
Ethernet	10/100 Mbit/s via standard RJ-45 connector
WiFi	IEEE 802.11n support
I2C	Support for I2C bus with a number of devices allowed on the same bus

SPI	Support for the Serial Peripheral Interface bus
GPIO	16 General Purpose IO connections
UART	Support for RS232-like serial interface protocol
LWM2M	Support for the LWM2M protocol from the GP2 to the cloud server
ScriptML	Support for the OpenIO Labs ScriptML programming system for user defined applications

Table 3: Summary of GP2 Features

GP2 Interfaces and Connections

The GP2 front panel includes:

- Micro-USB power connection and LED power indication
- 10/100 Mbps RJ45 Ethernet interfaces
- 4 x USB 2.0 type A connectors
- 40 Pin IDC IO interface connector.

The power connector is a standard micro-USB interface connector providing power to the OpenIO Labs GP2 IOI. The power LED is illuminated when power is applied. The RJ-45 Ethernet provides a wired access for the GP2 to external IP networks. The quad USB interface allows for the connection of a range of USB devices such as external test equipment like oscilloscopes and spectrum analysers.

Chapter 5: User Access

The user access to the OpenIO Labs system is via a web browser using a secure connection to the server. A user identity and password are required to allow the user server access. On the server, the user is able to perform a range of tasks. These tasks include:

- Install and configure an IOI
- Establish tagging sessions
- View collected data
- Upload ScriptML applications
- Display data in a variety of 2D and 3D plots
- Apply simple mathematical algorithms to the database

When an authorised IOI is attached to the internet, it will immediately connect to the cloud server and register. The user will see the presence of the IOI displayed in their browser. When the IOI is connected, the user can then configure any parameters that are required for that specific IOI.

The OpenIO Labs system uses the concepts of tags to allow data to be collected and later searched for using meaningful words and phrases. There can be multiple tags applied to the system at different levels of granularity. There could be a tag that applies to a whole project, one that applies to a specific aspect of the project, one that applies to a specific day or week, and one that applies to a specific experiment. Using these tags the user can select what aspects of the data they wish to review, plot or download.

Data that is collected via the IOI can be viewed through a tabular interface on the user interface. If the data is a large stream of collected information, it can also be downloaded for viewing or post processing with applications such as Excel.

One of the key aspects of the system as already described is the use of ScriptML to allow the user to create applications that control and process the data collected from an IOI at the edge of the network. This significantly improves the power and response time for this data processing activity. The user downloads and executes these ScriptML applications from the browser. The applications can be stopped and re-started using simple button controls.

In addition to viewing the data in a tabular format, the data can also be plotted using 2D and 3D plots, depending on the nature of the data.

Finally, simple mathematical functions can be applied to the data. This will allow, for instance, the scaling of the data, or alternatively providing things such as logical expressions to additionally process the data.

Chapter 6: ScriptML

The OpenIO Labs system, like all M2M-Het-Nets, is the interconnection of a large number of devices. Each of these devices is used to either control experiments or processes, or to collect, process, and push data back to a server for either further analysis or storage.

In many example deployments, there is a need to add edge processing to the system. By edge processing, we mean the ability to process raw data and make process decisions on that data at the edges of the network (in the IOI devices), rather than pushing the data back to the server and having the server perform the processing.

Within the OpenIO Labs system, we refer to this edge processing as the "application logic", the functions and processes that we need to perform on the data as it is collected, or to drive the edge devices based on the collected data or a set of algorithms defined to achieve a specific system objective. To achieve these objectives, it is recognised that a flexible method of implementing the application logic is the key to bringing the whole system together.

The specification of a framework for defining and implementing the application logic through the use of standardised languages, protocols and procedures is seen as a complex and difficult task to orchestrate across the whole M2M industry. The approach, taken by OpenIO Labs, was not to define such a structure and framework for the application logic, but rather a much simpler scheme in which we define an application logic transport protocol.

The application logic transport protocol is a mechanism that allows the application logic defined in one node or system in one language to be transported to a second system and executed on that second system within a completely different execution environment. If we take as an example, a server that is wanting to download an application logic executable to a device somewhere in the system. The server will accept the definition of the application logic written in some high-level language (as an example Python or C). An encoder within the server will convert this high level application into an interim transport language, the language "tokens" are then transported to the edges devices using the LWM2M transport protocol and then executed on the edge device using a language decoder, or interpreter.

To achieve these goals, OpenIO Labs have defined ScriptML. ScriptML is a Meta Language that is an exact encoding of the logic defined in a source language.

The benefit of implementing the Application logic using ScriptML is that the detailed definition of a complex application framework can be avoided and instead the easier problem of defining the syntax, grammar and transportation of the ScriptML tokens can be done.

The simplest example is illustrated in Figure 3: Example of a Simple ScriptML Deployment below, in which we have the Server interacting with the edge devices attached to the OpenIO Labs IOI. The application logic encoded in the server is transported to the IOI via the LWM2M transport protocol and where the decoder interprets the application logic and implements the desired functions.

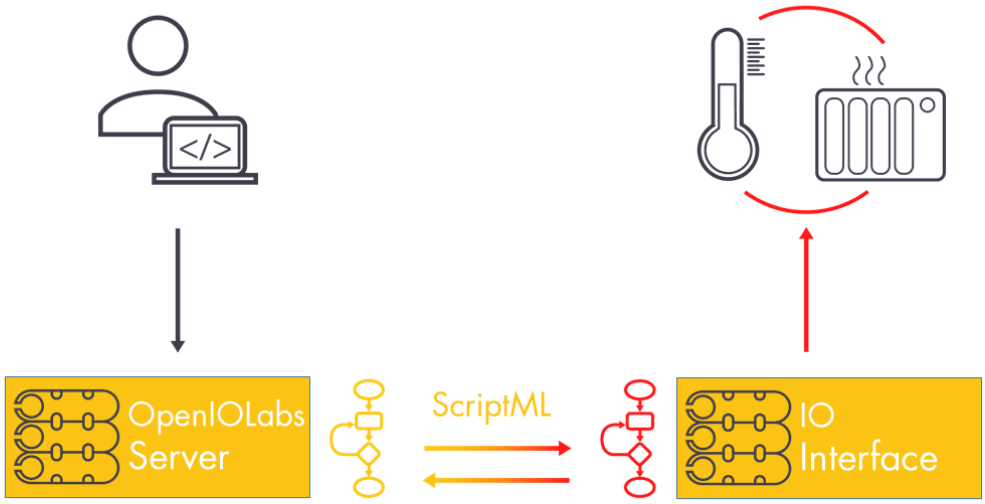

Figure 3: Example of a Simple ScriptML Deployment

A more elaborate deployment is illustrated in Figure 4: Example of a Typical ScriptML Deployment below. In this example we have a number of external edge devices (temperature monitoring, light control, complex instrumentation and devices) connected to a set of IOIs (not shown). The ScriptML application logic encoding system ensures that all of these devices are managed and controlled within the overall performance objectives for the system.

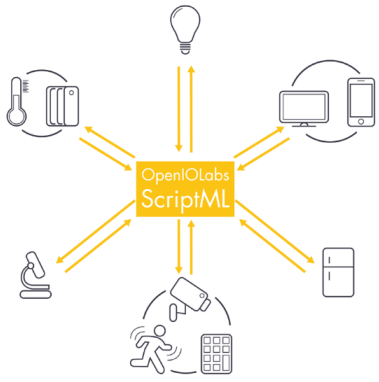

Figure 4: Example of a Typical ScriptML Deployment