Francis Fongang Namaya

Development of a mobile control system for Teklab Multipurpose Workstations on Android

Helsinki Metropolia University of Applied Sciences
Master of Engineering
Information Technology
Development of a mobile control system for TEKALAB Multipurpose Workstations on Android
This thesis was done for Helsinki Metropolia University of Applied Sciences at the Department of Information and Communication Technology, Piha Lab. The aim of the thesis was to develop a mobile control system for TEKLAB Multipurpose Workstations on Android. The objective was to be able to monitor and control the TEKLAB ELP100NET laboratory tables from a mobile device through a graphical user interface. Another objective was to be able to view the status of all TEKLAB ELP100NET laboratory tables with the application.

A Python program that runs on ConnectPort X2 via iDigi Device Cloud was developed along with the Android application developed to provide an interface for controlling and monitoring TEKLAB ELP100NET laboratory tables in the Piha lab Ethernet network. HTTP POST was used for sending and receiving RCI XML data string wrapped over SCI XML. The Digi ConnectPort X2 gateway was configured and connected to the Piha Lab Ethernet network interfacing with iDigi Device Cloud.

Keywords

TEKLAB ELP100NET, Android, Digi ConnectPort X2, iDigi Device Cloud, RCI, HTTP.
## Abstract

Abbreviations and Terms

1. **INTRODUCTION**

2. **THEORETICAL FRAMEWORK**
   - 2.1 TEKLAB ELP100NET Laboratory tables
   - 2.2 Platform
   - 2.3 Communication protocols
   - 2.4 Security

3. **SYSTEM DESCRIPTION**
   - 3.1 Digi ConnectPort X2 Overview
   - 3.2 Application

4. **IMPLEMENTATION**
   - 4.1 Development and Design

### Table of Contents:

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td></td>
</tr>
<tr>
<td>Abbreviations and Terms</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2 THEORETICAL FRAMEWORK</td>
<td>2</td>
</tr>
<tr>
<td>2.1 TEKLAB ELP100NET Laboratory tables</td>
<td>2</td>
</tr>
<tr>
<td>2.2 Platform</td>
<td>5</td>
</tr>
<tr>
<td>2.3 Communication protocols</td>
<td>8</td>
</tr>
<tr>
<td>2.4 Security</td>
<td>14</td>
</tr>
<tr>
<td>3 SYSTEM DESCRIPTION</td>
<td>18</td>
</tr>
<tr>
<td>3.1 Digi ConnectPort X2 Overview</td>
<td>18</td>
</tr>
<tr>
<td>3.2 Application</td>
<td>21</td>
</tr>
<tr>
<td>4 IMPLEMENTATION</td>
<td>26</td>
</tr>
<tr>
<td>4.1 Development and Design</td>
<td>26</td>
</tr>
<tr>
<td>4.1.1 Python program</td>
<td>26</td>
</tr>
<tr>
<td>4.1.2 Android program</td>
<td>27</td>
</tr>
<tr>
<td>4.1.2.1 The Piha Mobile Control program</td>
<td>27</td>
</tr>
<tr>
<td>4.1.2.2 Piha Mobile Login screen</td>
<td>27</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Term</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>ADT</td>
<td>Android Development Tools</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CD</td>
<td>Compact Disc</td>
</tr>
<tr>
<td>DIA</td>
<td>Device Integration Application</td>
</tr>
<tr>
<td>EDP</td>
<td>Embedded Display Port</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>ID</td>
<td>Identification of Device</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standard Organization</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>J2ME</td>
<td>Java Micro Edition</td>
</tr>
<tr>
<td>M2M</td>
<td>Machine-to-Machine</td>
</tr>
<tr>
<td>OS</td>
<td>Operating system</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>POP</td>
<td>Post Office Protocol</td>
</tr>
<tr>
<td>RCI</td>
<td>Remote Command Interface</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>SCI</td>
<td>Server Command Interface</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UIDs</td>
<td>UNIX user identifiers</td>
</tr>
<tr>
<td>UTF-8</td>
<td>Universal Character Set Transformation Format -8bit</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>WAN</td>
<td>Wireless Area Network</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XML-RPC</td>
<td>XML Remote Procedure Call</td>
</tr>
</tbody>
</table>
Used terminology

**Android**

Operating system for mobile devices and tablets developed by Open Handset Alliance (OHA). OHA is led by Google.

**Ethernet**

It is a family of computer networking technologies for local area networks (LANs).

**Telnet**

It is a network protocol used on the Internet or local area networks to provide a bidirectional interactive text-oriented communication facility using a virtual terminal connection.

**An XML element tag**

It is dest in this example: `<dest index="23">`

**An XML attribute**

It is the index="23" in this example: `<dest index="23">`, where index is the name of the attribute, and "23" is the value of the attribute. Note the double-quote characters are required.

**An RCI client**

It is the originator of an RCI request. A client sends request to a device and a device responds to a client.
1 Introduction

Nowadays, mobile computing is becoming a powerful technology and virtually the unhindered connectivity for higher education students and faculty, making it possible to access learning materials, advanced learners and teachers anytime, anywhere, thus inducing the concept of Internet of Things (IoT) and Machine-to-Machine (M2M) communication. Many different mobile devices and machines communicate to each other over the internet without having to have any human intervention in between.

This master’s thesis is part of the concept of the Internet of Things and Machine-to-Machine (M2M) project conducted at the Helsinki Metropolia University of Applied Sciences, department of Information and Communication Technology, PIHA LAB. The goal was to provide a user friendly and customized interface for the control system of TEKLAB multipurpose workstations. And in fact we are not to develop a remote application for TEKLAB GC software, but instead the objective is to develop a mobile application in a more user friendly way than the original TEKLAB GC software to fulfil the need for a mobile graphical user interface operating in parallel with the TEKLAB GC software to control and monitor the TEKLAB multipurpose workstations remotely. With the Android mobile control application, the teachers at Helsinki Metropolia University of Applied Sciences can open the laboratory by unlocking workstations and enabling electricity as well as closing the laboratory by switching power off and driving motorised instruments panel down or up. Workstations can also be locked.

However, the development of such a mobile application will demand to address certain aspects. This thesis addresses the aspects into three parts. The first part presents the framework of the TEKLAB Multipurpose workstations as well about platform and security issues. The second part outlines and describes the general specification for the system and the last parts describe the implementation and draws conclusions for the future.
2 Theoretical framework

As earlier mentioned in the introduction, the goal is to develop a mobile control system that provides a user friendly and customized interface for the control system of TEKLAB multipurpose workstations. The theoretical framework and overview of TEKLAB multipurpose workstations and its embedded module as well as platform and security issues will be discussed or resolved before the actual development phase of the mobile control system.

2.1 TEKLAB ELP100NET Laboratory tables

The TEKLAB ELP100NET laboratory tables are multipurpose workstations designed by TEKLAB, a Finnish company, for educational purposes. They have integrated instruments to control these TEKLAB ELP100NET laboratory tables manually. More so, these tables can be controlled or monitored over the Ethernet via an embedded communication module known as the Digi Embedded Module on which all the controlling and monitoring of the TEKLAB ELP100NET laboratory tables is done as shown in figure 1. [1.]

![Multipurpose workstation for education](https://example.com/multipurpose_workstation.png)  

Figure 1. TEKLAB Multipurpose workstations. Copied from TEKLAB. (2013) [1]
Each of these tables has the following functions: motorised panel, integrated instrument panel, supply unit, integrated training kits, Ethernet connection and locked storing space to help teachers and students as shown below.

Figure 2. TEKLAB ELP100NET Laboratory table. Copied from TEKLAB. (2013)[1.]

For example, if students do not need to do any practical exercises during a teacher’s lectures, he can use the motorised panel to move all the tables in the down position or up position and shut down the power supply so that the laboratory looks like an ordinary class.
The down and up position can be seen in figure 3 and figure 4 respectively above.

2.1.1 Digi Embedded Module

The actual control of the ELP100NET laboratory tables over the Ethernet is performed on the Digi Connect Embedded Module which is a secured wired device server developed by Digi International in 2003. The Digi Connect Embedded Module has nine GPIO pins that interface between the Digi Connect Embedded Module and the Teklab laboratory tables. These GPIO pins once configured as input pins read digital signals from the laboratory tables or configured as output sends data to the tables.
The Digi Connect EM device has been configured by TEKLAB Company during the manufacturing of the ELP100NET laboratory table and all specification for Digi Connect embedded Module can be seen in appendix 1. [1; 2.]

2.2 Platform

Firstly, we were to figure out on what platform would this system be best developed as well as providing a secured system for communication to happen between the TEKLAB ELP100NET table and the mobile device. This was the most difficult part of our project because TEKLAB Company does not supply any information about the platform and security issues but we knew that communication can happen via the HTTP protocol.

There are handfuls of mobile OS such as Window mobile 7, Apple iPhone, Palm Pre just to mention a few designed to enhance mobile hardware. These mobile platforms offer rich and simple development for mobile application but are built on proprietary operating systems that restrict the development and deployment of third-party applications. Referring to the pros and cons of these mobile technologies we realized that the Android Mobile phone OS offers a free, no licensing, and an open development environment that will easily connect to iDigi Device Cloud platform will provide the best deal for the development of such an application. iDigi Device Cloud platform was chosen to develop the Mobile control system for TEKLAB Multipurpose Workstations because of its abilities to handle secured and reliable connections. Related information to the iDigi platform is briefly introduced below.

2.2.1 iDigi Device Cloud platform

The iDigi Device Cloud platform is actually a network server or operating platform that integrates many different embedded devices for management. It is based on cloud computing principles and includes a variety of API. The platform embraces the interaction of application to device data, application and device data storage, and remote management of devices. Communication between the different devices, server, and applications on the platform is via Internet or other wide area network connection. The platform uses standard http API to connect applications to iDigi Device Cloud for data accessing.
Data on the iDigi Device Cloud can be assessed or retrieved in the following ways: a standard browser by typing in the appropriate URL, Google Gadget, Java application, Python Application running on a PC, Python Application running on a Device or anything that supports standard HTTP methods.

Some fundamental concepts such as subscription, device ID, data services, Device Information Caching would describe the principle involved in iDigi Device Cloud platform. [3, 2.]

**Subscription**

In iDigi platform subscription is required to gain access to devices integrated on the platform. Services in iDigi platform are controlled via subscriptions. [3, 2.]

**Device ID**

Devices are identified on iDigi platform by their ID. iDigi platform requires devices to have a Device ID which a unique 16octet number to the device that does not change over its lifetime. An example of Device ID is: 6YT7-6XA7-780U-7489. [3, 2.]

**Embedded Device Development**

All Digi international devices contain a firmware that is enabled for iDigi Cloud Device platform but we could not connect the Digi Embedded Module implemented in TEKLAB ELP100NET laboratory tables to the iDigi Device Cloud platform due to the fact the firmware of the embedded module was an old version that could not support iDigi Cloud Device integration or connection. To get around this, a Digi ConnectPort x2 gateway module that would routes data from iDigi Device Cloud to the Digi Embedded Module implemented in TEKLAB ELP100NET laboratory tables was considered as the iDigi enabled device used in this project. [3, 3.]
Data Services

iDigi Device Cloud platform has a data service that enables the collection of data from remote devices. Data files can be uploaded from remote devices to the server cached temporarily in a database on the iDigi Device Cloud. [3, 3.]

Device Information Caching

Data pushed from remote devices to iDigi Device Cloud need to be organised or stored properly in a relatively fast response time and reduced network bandwidth. The iDigi Device Cloud server implements numerous caching methods to control the amount of data sent and received between the remote devices and the server.

Posting an SCI request may require to set the attribute cache value on the send_message command.

- If the cache value is set to “false” then iDigi Device Cloud server will ignore the cache and always forward the request on to the device.
- If the cache value is set to “Only” then iDigi Device Cloud server will provide responses from the cache and never send them on to the device.

The amount of data returned by the SCI request can be control by setting the attribute reply value of send_message command.

- To get all reply data, the reply value is set to “all”.
- To get only error replies, the reply value is set to “only”.
- To get none of the replies, the reply value is set to “none”.

We made use of cache="false" and reply=" all" to establish communication between Digi connectPort X2 and iDigi Device Cloud. [3, 3.]
2.3 Communication protocols

iDigi server provides a REST-style API over HTTP(https) that allows client applications to access data from the iDigi Device Cloud. Communication between the Teklab laboratory tables and the mobile device over iDigi Device Cloud can be done via RCI and HTTP protocols.

2.3.1 RCI protocol

Remote Command Interface (RCI) is a programmable interface designed to be used by a program to configure and control Digi devices. It uses an XML-based request/response method to query and modify device configurations, access statistics, reboot the device, and reset the device to factory as well as sending requests and retrieve replies from dynamically registered agents such as python programs. RCI requests can be sent to devices via iDigi server and are wrapped in a server request called SCI. RCI is made up of two parts:

- XML documents containing requests and replies
- The Transport layer exchanges content between the server and device. [4,4-5.]

RCI request and reply XML documents

In RCI, data can be exchanged between clients and devices using XML documents. All RCI XML documents are well-composed XML. To reduce the complexity of XML parsing on devices with limited capabilities, a limited set of XML parsing requirements are enforced in an RCI-capable device: such as the XML version, ISO-8858-1 encoding, and some predefined entities such as &amp; &lt; &gt; &quot; &apos; [4, 6.]

Transport layer

The Transport layer of RCI handles communication between a server and a device. The initialization process, the sending and responding mechanism, the closing mechanism, any error recovery mechanism needed, and security are specified by the transport layer. The Transport layer for iDigi is EDP but the underlying transport layer
used by RCI is HTTP, through the embedded web server. The Web server will provide
the initialization, receiving and sending, and security. [4, 6.]

RCI requests are sent to the device using an URI of UE/rci. For example, if the Digi
Device’s IP address is 192.168.1.1, then RCI requests are sent to
http://192.168.1.1/UE/rci . [4, 6.]

RCI document structure

An RCI XML document is identified by the root XML elements <rci_request> or
<rci_reply>. An RCI request specifies the XML element tag <rci_request> with an op-
tional version number. An example of request element is <rci_request version="1.1">
whereas, an RCI reply specifies the element tag <rci_reply> along with the version
number as an attribute. For example: <rci_reply version="1.1">. [4, 7.]

RCI commands

The command section of the protocol indicates the action requested or action per-
formed in replies and the commands are specified as sub-elements to <rci_request>
and <rci_reply>.

```
<rci_request version="1.1">
<query_setting/> <!-- request config of device -->
</rci_request>
```

Listing 1. An example of RCI requests for all configuration settings

This is an example for requesting configuration information for boot settings and serial
settings.

```
<rci_request version="1.1">
<query_setting>
<boot/>
<serial/>
</query_setting>
</rci_request>
```

Listing 2. Configuration information for just boot settings and serial settings.
Table 1.  RCI commands. Copied from Digi (2013). [4, 8.]

<table>
<thead>
<tr>
<th>Command</th>
<th>Request description</th>
<th>Reply description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query_descriptor</td>
<td>Request device capabilities.</td>
<td>RCI descriptor.</td>
</tr>
<tr>
<td>query_setting</td>
<td>Request for device settings.</td>
<td>Returns requested settings.</td>
</tr>
<tr>
<td>set_setting</td>
<td>Set settings specified in setting</td>
<td>Empty setting groups returned as confirmation of set.</td>
</tr>
<tr>
<td></td>
<td>element.</td>
<td>Errors are returned as specified in RCI errors and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>warnings on page 22.</td>
</tr>
<tr>
<td>set_state</td>
<td>Set the device state.</td>
<td>Same semantics as set_setting</td>
</tr>
<tr>
<td>query_state</td>
<td>Request current device state,</td>
<td>Returns requested state.</td>
</tr>
<tr>
<td></td>
<td>such as statistics and status.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sub-element may be supplied to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>subset results.</td>
<td></td>
</tr>
<tr>
<td>set_factory_default</td>
<td>Sets device settings to factory</td>
<td>Same semantics as set_setting</td>
</tr>
<tr>
<td></td>
<td>defaults.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Same semantics as set_setting.</td>
<td></td>
</tr>
<tr>
<td>reboot</td>
<td>Reboots device immediately.</td>
<td>Confirm reboot command.</td>
</tr>
<tr>
<td>do_command</td>
<td>Send a request to a subsystem</td>
<td>Response from subsystem.</td>
</tr>
<tr>
<td></td>
<td>specified by the target element.</td>
<td></td>
</tr>
</tbody>
</table>

The table above summarizes and describes all the required commands in RCI. [4, 8.]

2.3.2 HTTP protocol

The Hypertext Transfer Protocol is a protocol that provides or establishes communication between an HTTP client and an HTTP server. This protocol uses the request-response methods for communication between the client and server.

![HTTP data transaction](image)

Figure 6. HTTP data transaction.

An HTTP session is established between sequences of network request-response transactions. The client always initiates the transaction by sending a request message...
via the Transmission Control Protocol on port 80. The HTTP server listens on port 80 to receive the request and returns an appropriate response message back to the client. Figure 13 shows a graphical representation of HTTP request and response message architecture. Both messages consist of four fields and they are almost similar to each other. The difference between a request and a response message is in the first field.

![HTTP architecture](image)

Figure 7. HTTP architecture.

**Request line**

An HTTP communication is always invoked with a request message. This message made of a request line which is responsible for sending the actual HTTP request type to the server. A request line is made up of three different elements separated by a space.

![Request line](image)

Figure 8. Request line.

The request line is the request method, also known as request type. This element defines the request method that must be sent to the server. Nowadays, HTTP/1.1 sup-
ports nine different methods which are all listed in appendix 2. The Uniform Resource Locator (URL) defines the location of the server where the request must be sent to. The HTTP version tells the server the type of HTTP version is used by the client. [5.]

**Status line**

When a server receives a request message from a client, it sends an appropriate response message back with the requested data and a status code. This status code is contained in the status line of the response message. The status line is made up the same way as the request line. It consists of three elements separated by a space.

![Status line diagram](image)

The first element of the status line defines the HTTP version used by the server. The next two elements are the status code and status phrase. These two elements give information back to the client so that he can monitor the HTTP transaction. Each status code is made up of three digits. The first digit defines the class of response and the last two digits indicate the response code in the selected class. There are five different classes and each class has its own specific status codes. For a list of the most common status codes and phrases, see appendix 3. [5.]

**Headers**

The headers exchange additional information between the client and the server. There are four different categories of headers, namely general, request, response, and entity headers.

- The general header gives general information about the message and can be present in a request message or a response message.
- The request header specifies the client configuration and the preferred document format. This type of header can only be present in a request message.
- The response header can only be present in a response message and specifies the server configuration.
- The entity header gives some information about the body of the document. This type of header is mostly present in a response message, but request messages with PUT or POST methods can also have an entity header. See appendix 4 for a list of the most common headers. [5.]

Blank line

The blank line separates the headers from the body. [5.]

Body

The body is always present in a request or response message. Usually, the body contains the requested data that must be sent to the server or the client. [5.]

HTTP demonstration

An Http transaction starts with a request message to retrieve an image from a server. The request line in the message shows the method, the URL, and the HTTP version. The header is made of two lines, which tells the server that the client can accept images in the GIF or JPEG format.

The response message contains the status line and four lines of header. The header lines define the date, server, MIME version, and length of the document. The body of the document, which is separated by a blank line from the headers, contains the requested image. [5.]
An HTTP demonstration. This example retrieves an image from a server.

All HTTP sessions and connections undergo the HTTP demonstration shown in figure 10.

2.4 Security

Security is at the heart of every engineering product nowadays. The threats and challenges exposed to this engineering product are of vital importance. Developing the mobile control system for TEKLAB multipurpose workstation will need to address the security issue since it involves connectivity of the Android mobile application to iDigi Device Cloud as well as to the Digi connectPort X2 gateway over the Internet. The iDigi Device Cloud depends on SSL security protocol to provide protection of data during transmission to connecting devices that support standard HTTP methods. The mobile application that will be developed on Android platform supports standard HTTP methods. So, to provide a secured connection for the Android mobile application to control the TEKLAB multipurpose workstations via iDigi Server will be based on HTTP basic authentication over SSL protocol.

HTTP basic authentication

HTTP basic authentication is an authentication mechanism based on user name and password defined in the HTTP/1.0 specification to control accesses to client and server resources. The user names and passwords are sent over the Internet as unencrypted base64 encoded text. However, the authentication mechanism used in HTTP basic authentication is not very secure unless all connections are over SSL. Figure 3 below describes the HTTP basic authentication process. [6.]
SSL security protocol

Secure Sockets Layer (SSL) is a security protocol designed by Netscape to ensure a secured communication across the Internet by using a public key encryption method to authenticate the client (HTTP, LDAP or POP3) and server, ensuring data integrity and securing data privacy. It was also designed to provide a secured, reliable and authenticated connection between an HTTP server and client applications over a network using TCP as a communication layer. [7,376; 8.]

Two layers of protocol provided by SSL are found within the TCP framework namely; SSL Record protocol and three protocols (i.e. Handshake Protocol, the SSL Cipher Change protocol, and the SSL Alert Protocol) are designed to establish an SSL connection. [7,376; 8.]
SSL Record Protocol role is to ensure data security and integrity as well as to fragment, compress, encrypt the application data, and embody it with headers prior for transmission under the TCP protocol. The header fields to be used to embody the data are Content type identifies the kind of data transmitted, Major version and Minor version. [7,376; 8.]

Secure Sockets Layer (SSL) protocol operation

SSL protocol uses two concepts to operate namely; SSL connection implies a suitable type of service bind to a logical client/server link and SSL session is a session invoked
by an SSL handshake protocol allowing parameters to be shared among SSL connections established between the server and the client.

The SSL protocol operates as follows: After an HTTP session is established between a client and server, the client tries to connect to a portion of the website secured with SSL, a message is delivered to the client by the server requesting for a secured connection to be established. The client, in turn, sends its public key and security parameters to the server for verification. Once the public key and security parameters are matched, the server issues a digital certificate to the client for authentication. The client verifies if the issued certificate received can be trustworthy and valid. If so, a message is sent to the server which issues a digital acknowledgement to establish an encrypted SSL session. This encrypted data is shared between the client and server as long as the session remains active. [7,376; 8.]
3 System description

The following components Digi ConnectPort X2 gateway and Application, which are the cornerstone of the system, will be described.

3.1 Digi ConnectPort X2 Overview

The Digi ConnectPort X2 according to Digi international can be described as:

ConnectPort X2 is a small ZigBee to Ethernet/Wi-Fi gateway that provides low-cost IP networking of RF devices and sensor networks. Featuring an easy development environment, ConnectPort X2 enables custom applications to run locally while interfacing across existing Ethernet/Wi-Fi networks for WAN connectivity to a centralized server. ConnectPort X2 products feature an end-to-end development environment based on local customization via the iDigi Dia framework, allowing for rapid M2M-specific application development on the industry standard Python scripting engine. [9.]

Simply put, it is a Python programmable gateway that can route or transfer data from devices to servers over the Ethernet or Internet and can be integrated on iDigi Device Cloud platform to ease communication to connecting devices. [9.]

![ConnectPort X2](image)

Figure 15. ConnectPort X2. Copied from Digi (2013). [9.]

This Digi ConnectPort X2 module as earlier mentioned will be used as the iDigi enabled device in the project.
3.1.1 Specifications

See appendix 1 for detailed specification of Digi ConnectPort X2.

3.1.2 ConnectPort X2 gateway configuration

As earlier mentioned, we could not connect the Digi Embedded Module implemented in TEKLAB ELP100NET laboratory tables to the iDigi Device Cloud platform due to the fact that the firmware of the embedded module was an old version that could not support iDigi Device Cloud integration or connection. To get around this, a Digi ConnectPort x2 gateway module that would routes data from iDigi Device Cloud to the Digi Embedded Module implemented in TEKLAB ELP100NET laboratory tables was considered as the iDigi enabled device used in this project.

The ConnectPort x2 was configured and connected to the Piha lab Ethernet network by Antti Toura. He implemented a list of XML RPC INTER-FACE remote commands in Python script that executes on connectPort x2 gateway over iDigi Device Cloud to control the TEKLAB multipurpose workstations. Below is the list of the supported remote commands implemented.

Table 2. supported remote commands

<table>
<thead>
<tr>
<th>Commands</th>
<th>Request/Reply Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;status&gt;X&lt;/status&gt;</td>
<td>Requests the status of a table, where X is a table number in the range [1..10]</td>
</tr>
<tr>
<td></td>
<td>Response: return GPIO pins status of table X.</td>
</tr>
<tr>
<td>&lt;table_down&gt;X&lt;/table_down&gt;</td>
<td>Set a table to down position, where X is a table number in the range [1..10]</td>
</tr>
<tr>
<td></td>
<td>Response: Ok!</td>
</tr>
<tr>
<td>&lt;table_up&gt;X&lt;/table_up&gt;</td>
<td>Set a table to an up position, where X is a table number in the range [1..10]</td>
</tr>
<tr>
<td></td>
<td>Response: Ok!</td>
</tr>
<tr>
<td>&lt;unlock&gt;X&lt;/unlock&gt;</td>
<td>Unlock a table, where X is a table number in the range [1..10]</td>
</tr>
<tr>
<td></td>
<td>Response: Ok!</td>
</tr>
<tr>
<td>&lt;allow_power&gt;X&lt;/allow_power&gt;</td>
<td>Allows power for a table, where X is a table number in the range [1..10]</td>
</tr>
<tr>
<td></td>
<td>Response:</td>
</tr>
<tr>
<td>&lt;/disable_power&gt;X&lt;/disable_power&gt;</td>
<td>Disable power supply for a table, where X is a table number in the range [1..10]</td>
</tr>
<tr>
<td></td>
<td>Response: Ok!</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&lt;unlock_all&gt;1&lt;/unlock_all&gt;</td>
<td>Unlocks all tables, i.e 10 tables</td>
</tr>
<tr>
<td>&lt;status_all&gt;1&lt;/status_all&gt;</td>
<td>Requests the status of all tables i.e 10 tables</td>
</tr>
<tr>
<td>&lt;/disable_power&gt;1&lt;/disable_power&gt;</td>
<td>Disable power supply for all tables i.e 10 tables</td>
</tr>
<tr>
<td>&lt;allow_power&gt;1&lt;/allow_power&gt;</td>
<td>Allow power supply of all tables i.e 10 tables</td>
</tr>
</tbody>
</table>

These supported remote commands can be executed manually on the web console of iDigi Device Cloud. In order to retrieve the status of each GPIO pin that would enable us to monitor the status of all the TEKLAB tables in the laboratory, we supplied the password and username for iDigi Device Cloud because it supports basic HTTP authentication and only valid users can access the database and execute the SCI request on the web service console of iDigi Device Cloud.

```xml
<sci_request version="1.0">
  <send_message reply="all">
    <targets>
      <device id="00000000-00000000-00409DFF-FF52FB61"/>
    </targets>
    <rci_request version="1.1">
      <do_command target="at_">
        <status>1</status> {Could be replaced with the remote commands mentioned above}
      </do_command>
    </rci_request>
  </send_message>
</sci_request>
```

Listing 3. Request data string on the web service console of iDigi Device Cloud

It will return when the operation has completed (or timed out) as body response:

```xml
<sci_reply version="1.0">
  <send_message>
    <device id="00000000-00000000-00409DFF-FF52FB61"> {Connect Port X2 id}
    <rci_reply version="1.1">
      <do_command target="at_">
        <status>Table1:
          GPIO States:
          pin#  state
          1   off
          2   off
      </do_command>
    </rci_reply>
  </send_message>
</sci_reply>
```
After using iDigi Device Cloud web service console to manually execute the remote commands mentioned above, it enabled us to understand that it is possible to establish communication between the TEKLAB ELP100NET tables and the mobile device via HTTP protocol over iDigi Device Cloud. The server supports basic HTTP authentication and only valid users can access the database. The followings were taken into consideration.

The data to be sent is an SCI request composed of XML. The URL: " /my.idigi.co.uk:80 /ws/sci" defines where the SCI data string is posted to.

```xml
Data =" <sci_request version="1.0"> <send_message reply="all"> <targets>
    <device id="00000000-00000000-00409DFF-FF52FB61"/>
</targets>
    <rci_request version="1.1"> <do_command target="at_">COMMANDS
    </do_command> </rci_request> </send_message> <sci_request>
```


3.2 Application

The programming environment used for developing the mobile application to control and monitor the Teklab laboratory tables over iDigi Device Cloud is Android, although several iDigi devices support additional configurable applications.
3.2.1 Android

Android is an operating system developed on Linux kernel for mobile devices such as tablets and mobile phones. It respects the Open handset Alliance principle as described below:

A commitment to openness, a shared vision for the future, and concrete plans to make the vision a reality. To accelerate innovation in mobile and offer consumers a richer, less expensive, and better mobile experience. [10]

Android version 1.0 as a free open OS developed by OHA was firstly used in HTC dream handsets released in October 2008. Since then, there have been many platforms and SDK releases. [11, 4]

A lot of features are supported in Android platform to develop mobile application. These features such as camera, GPS, accelerometer, Google maps, geocoding, location based services, background services, SQLITE database for storing retrieving data, shared data and inter-application communication, memory and process management, extensive media support and 2D or 3D graphics as well as widgets, live folders, and live wallpaper to enhance home screen using Android APIs provide the opportunity to create applications. [11, 5-7]

**Android development framework**

Android has many different types of applications that can be developed namely: native, widgets, services, web applications, native applications which are applications specifically designed to be used on a particular device or operating system and just to mention a few. This thesis concentrates on a native application approach. All these applications in Android platform can be written in either Java using the Android Software Development Kit (SDK) or in C or C++ using the Native Development Kit (NDK) provided by Google but the applications are executed by means of a custom virtual machine called Dalvik rather than a traditional Java VM.

The Android software development kit (SDK) is required for developing, testing and debugging Android applications. The SDK includes the following packages, Android APIs, Development tools, the Android Virtual Device Manager and Emulator, Full documentation, Sample code and an online support. Furthermore, the most official rec-
ommended integrated development environment (IDE) used in this project was Eclipse IDE which includes an Android Development Tool (ADT) plug-in used to simplify project creation and tightly integrates Eclipse with the Android Emulator and debugging tools. Figure 16 shows the Eclipse development environment. [11, 12.]

Figure 16. Eclipse development environment

Android Software Stack

Android software stack is made of a kernel based on Linux kernel version 2.6 and, from Android 4.0 Ice Cream Sandwich onwards, version 3.x, with middleware, libraries and APIs written in C, and application software running on an application framework which includes Java-compatible libraries based on Apache Harmony. Android uses the Dalvik virtual machine with just-in-time compilation to run Dalvik 'dex-code' (Dalvik Executable), which is usually translated from Java byte code. Figure 17 shows the components of the Android software stack. [11, 13.]
Figure 17. Android software stack diagram. Copied from Wikipedia (2013). [12.]

Android Application Architecture

The development of any Android application lies on the following architectural components

- Activity Manager controls the life cycle of Activities as well as the management of the Activity stack.
- Views are used to build the user interfaces for Activities
- Notification Manager provides a consistent and nonintrusive mechanism for signalling.
- Content Providers used for applications to share data.
- Resource Manager used to create application resources like strings and graphics. [11,15.]
3.2.2 Testing tool

3.2.2.1 Android Emulator

Android Emulator is a virtual mobile device included in the Android SDK designed as a Dalvik virtual machine to test, and debug Android applications without using a real Android device. All the hardware and software features of a real mobile device are found on the emulator. Tested and running Android applications make use of these features such as services of the Android platform to invoke other applications, access the network, play audio and video, store and retrieve data, notify the user, and render graphical transitions and themes to be displayed on the screen provided by the Emulator.

More so, the Android emulator is configured using the AVD manager to easily model or test Android applications. The emulator cannot be used without creating one or more AVD configurations. Hardware aspects on the emulated phone are defined by AVDs which allows the creation of many configurations to test many Android platforms and hardware permutations. [13.]
4 Implementation

4.1 Development and Design

The development of the application was based on a client-server design. iDigi Device Cloud acted as the server while the Android application is the client as shown below.

![Development environment diagram](image)

Figure 18. Development environment.

The application was developed in two phases. In phase one, a Python application developed by Antti Toura was running on the Digi connectPort X2 gateway interconnected to the iDigi server via the internet and TEKLAB multipurpose workstations via Ethernet so that communication works properly. In the second phase, an Android application developed by me was running on a real Android device that connects to iDigi Device Cloud via HTTP calls over the internet.

4.1.1 Python program

As earlier mentioned, a Python program was developed running on Digi ConnectPort X2 which routes commands sent by the Android application via iDigi server to control and monitor TEKLAB ELP100NET tables in the laboratory. Commands towards the Python application are sent by using HTTP. The Python program running on Digi ConnectPort x2 supports HTTP POST messages via iDigi Device Cloud server. This pro-
gram sends XML-RPC data string in an HTTP request message and returns the requested information in an XML format depending on the HTTP message.

4.1.2 Android program

I developed a simple Android monitor and control program called Piha Mobile Control that runs on a real Android device which talks to the Python program over the iDigi Device Cloud to control and monitor all the TEKLAB ELP100NET tables in the laboratory. This simple program posts an RCI request composed of XML data string wrapped in an SCI over http request message to ConnectPort X2 via iDigi Device Cloud. The program also gets a reply from the Digi ConnectPort X2 over iDigi Device Cloud when the RCI request is posted. The purpose of this program is that teachers can monitor the TEKLAB ELP100NET tables worldwide, anywhere, anytime and need not to be at the laboratory. The Android application can be run on devices having Android 2.3 version or later Android version.

4.1.2.1 The Piha Mobile Control program

This program is divided into two parts the login and control screen. A full control of the TEKLAB ELP100NET tables in the laboratory is done with these parts.

4.1.2.2 Piha Mobile Login screen

The login screen is the first screen of the Piha mobile control program. In this screen, the username and password are required to gain access to the control screen.
Figure 19. Login Screen of Piha Mobile Control

The login screen above is shown when the application is started.

4.1.2.3 Piha Mobile Control screen

This program is accessible if valid username and password has been provided to the login screen. On the Control screen, table buttons can either be seen in a locked or an unlocked state as shown below.
More so, all the TEKLAB laboratory tables are controlled by control_monitor program. See appendix 6 for the Android code of control_monitor program. When any table button is clicked, a command window is popup and the following command buttons, Table-Up, Table-Down and Table-Unlock can be seen in figure 21.
The Table-Down button command when applied on a table moves the table in the down and lock position.

The Table-UnLock button command when applied on a table sets the table to an unlock state.

The Table-Up button command when applied on a table moves the table in the Up and lock position.

A click on Unlock/power button pops up a command window as shown in figure 22.
The Unlock-all button when applied sets all the tables in an unlock state.

The Power-OFF button when applied disables the power supply for all tables.

The Power-ON button when applied allows the power supply for all tables.

4.1.3 HTTP request data string

HttpDataString function was firstly developed. This function constructs and returns an SCI data string corresponding to a table number and table remote command button. See the construction of the SCI data in appendix 5.

4.1.4 HTTP response data string

The second major function developed was the HttpConnection function as seen in appendix 7, posts RCI request wrapped over SCI XML constructed by HttpDataString function to iDigi Device Cloud, and then receives reply from Digi ConnectPort X2
via iDigi Device Cloud as HTTP response composed of SCI XML. To carry out this, the following Android public interfaces and classes were used; BasicHttpParams, HttpConnectionParams, DefaultHttpClient,HttpPost,HttpResponse, StringEntity, and HttpEntity for the transmission of the Http data string.

Furthermore, ReadpinStatus function was developed to analyse the HTTP response data pushed by ConnectPort X2 via iDigi Device Cloud whenever an HttpConnection function is invoked or instantiated. A text file is used for storing the response data string. The content of the text file is then transferred to a vector which is then used by two implemented sub functions namely, showtablestatus function to display the status of all the TEKLAB laboratory tables if they are unlocked or locked and Updatetablestatus function to automatically update the status of all the Teklab tables if they are unlocked or locked. See appendix 5 for the whole Android codes that controls all the TEKLAB laboratory tables.

4.2 Testing

Using Eclipse IDE, while the Python application was running on the Digi ConnectPort X2 connected to iDigi Device Cloud via internet and TEKLAB Multipurpose Workstations via Ethernet, the Android application package was built automatically under test and the test package by ADT then installed on the emulator for execution ensuring that the application worked. The look and feel on Android emulator was the same in the device used. During the application development, Samsung Galaxy S2 mobile device was used while the application was tested. The Android application runs on version 2.3 and later versions.
5 Discussion

Conflicts or Bottlenecks were encountered during the realisation of this project. This section will discuss or outline the immediate solutions used to handle these conflicts or bottlenecks.

5.1 HTTP Server timeout

Server timeout was the first conflict encountered during the establishment of an HTTP connection in Android to the iDigi Device Cloud. Server timeout always occurred and no response was ever received because of little information on how to set the appropriate time for the server to response. To handle this bottleneck, it was found on the website of Stack Overflow [14.] on how to set the timeout for the server by using classes from the website of Apache [15.] and the settings used were:

```java
HttpParams httpParams = new BasicHttpParams();
int some_reasonable_timeout = (int) (20 * DateU-tils.SECOND_IN_MILLIS);
HttpConnectionParams.setConnectionTimeout(httpParams, some_reasonable_timeout);
HttpConnectionParams.setSoTimeout(httpParams, some_reasonable_timeout);
DefaultHttpClient httpClient=new DefaultHttpClient(httpPa-
```

Listing 6. Setting the timeout for the server.

These classes were implemented in the application and it resolved the server timeout.

5.2 Android UI thread

The second bottleneck encountered was on Android user interface. Every time an http connection was invoked, it caused the entire application to appear to lock up until the task is completed. This typically resulted in the operating system displaying an “Application is unresponsive” warning to the GUI. We found this difficult to resolve because of little knowledge about multitasking in Android. To handle this bottleneck, we found and learnt from Techotopia website that “in such situations, this can be avoided simply
by launching the task to be performed in a separate thread, allowing the main thread to continue unhindered with other tasks". [16.]

This was achieved with the AsyncTask class in Android. AsyncTask helps in the proper management of UI thread as well as performing background operations to publish results on the UI thread without having to manipulate threads and/or handlers. The following methods were used to implement AsyncTask class; doInBackground(Params…) , onProgressUpdate() and onPostExecute(Result) . See appendix 7 on how Asyntask class was implemented in this project. [17.]
6 Conclusion

The development of mobile applications on different mobile platforms or OS presents enormous challenges as this might not only be time consuming but also cost effective. According to the words borrowed from the website of Accenture, it presents the challenges as:

The plight of a mobile application developer these days is a challenging one. On the one hand, development in this space is vibrant and full of opportunities; a spectrum of new devices, from smartphones to tablets, is redrawning the boundaries of what users can do. On the other hand, this new landscape also brings new development questions – including, what devices to target, how to create simple yet effective applications, and how to secure the data that is uploaded and downloaded. In particular, the trend of the consumerization of IT weighs heavily on enterprise mobile application developers. This trend encompasses many facets. Increasingly, corporate users are accessing enterprise data from mobile devices which may be their own or may be deployed by their internal IT department. That means developers may not know what the target platform is, requiring either a cross-platform or multi-platform development effort. [18]

The aim of the thesis was to develop a mobile application that provides a friendly graphical user interface for monitoring and controlling TEKLAB multipurpose workstations. The application was specifically developed to run on Android smartphones that would provide teachers at Metropolia University of Applied Sciences a convenient and suitable way to monitor and control TEKLAB multipurpose workstations remotely instead of teachers to control the TEKLAB multipurpose workstations from a single computer at the laboratory.

During the design and development phase, several conflicts occurred. These conflicts were successfully resolved but the future will still demand to address a cross platform mobile application development as well as enhancing the GUI design and security for the mobile application.
References

1. TEKLAB multipurpose workstations. [online] 

2. Digi Connect Embedded Module. [online], URL: 

3. iDigi Device Cloud. [online], URL: 

4. Digi. Remote Command Interface Protocol. [online], URL: 

5. Hyper Text Transfer Protocol. [online], URL: 

6. Oracle. HTTP basic authentication. [online] 


8. Window. Secure Socket Layer. [online] 

   Assessed April 6, 2013

10. Open Handset Alliance. [online], URL: http://www.openhandsetalliance.com/ 
    Accessed April 6, 2013


12. Android-System-Architecture. [online], URL: 

13. Google. Android Emulator. [online], URL: 
Accessed April 6, 2013


Appendix 1: Digi Specifications

Digi Connect Embedded module specification Copied from Digi (2011) [2.]

<table>
<thead>
<tr>
<th>Specifications</th>
<th>Digi Connect EM®</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware</strong></td>
<td></td>
</tr>
<tr>
<td>Processor Type</td>
<td>32-bit Digi N57520 processor</td>
</tr>
<tr>
<td>ARM Core</td>
<td>ARM7TDMI</td>
</tr>
<tr>
<td>Processor Speed</td>
<td>55 MHz</td>
</tr>
<tr>
<td>Memory Base</td>
<td></td>
</tr>
<tr>
<td>Population</td>
<td>4 MB NOR flash</td>
</tr>
<tr>
<td></td>
<td>8 MB SDRAM</td>
</tr>
<tr>
<td>Population Options</td>
<td>LED array, LED pin header, Ethernet (RJ-45) connector, Ethernet pin header, Antenna connectors</td>
</tr>
<tr>
<td>Pins/Form Factor</td>
<td>PCB-style module with 12-pin main pin header</td>
</tr>
<tr>
<td>High-Speed TTL Serial Interface</td>
<td>2 with up to 230 Kbps data rate TXD, RXD, RTS, CTS, DTR, DSR and DCD, including hardware/software flow control Second port RXD/TXD only</td>
</tr>
<tr>
<td>GPIO</td>
<td>9 shared</td>
</tr>
<tr>
<td>SPI</td>
<td>Master mode</td>
</tr>
<tr>
<td>(Available through NET+OS development kit only)</td>
<td></td>
</tr>
<tr>
<td>JTAG Interface</td>
<td>NET+OS development modules only (P/N DC-EM-02T-JT, DC-WEM-02T-JT)</td>
</tr>
<tr>
<td>Dimensions (L x W x H)</td>
<td>1.935 in (49.149 mm) x 1.575 in (40.005 mm) x 0.670 in (17.018 mm)</td>
</tr>
<tr>
<td>Height and width (overhang) varies depending on connector population. See hardware reference manual for additional information.</td>
<td></td>
</tr>
<tr>
<td>Network Interface - Wired</td>
<td></td>
</tr>
<tr>
<td>Standard</td>
<td>IEEE 802.3</td>
</tr>
<tr>
<td>Physical Layer</td>
<td>10/100Base-T</td>
</tr>
<tr>
<td>Data Rate</td>
<td>10/100 Mbps (auto-sensing)</td>
</tr>
<tr>
<td>Mode</td>
<td>Full- or half-duplex (auto-sensing)</td>
</tr>
<tr>
<td>Connector</td>
<td>RJ-45 or Pin Header</td>
</tr>
<tr>
<td>Environmental</td>
<td></td>
</tr>
<tr>
<td>Operating Temperature</td>
<td>-40° C to +85° C (-40° F to +185° F)</td>
</tr>
<tr>
<td>Storage Temperature</td>
<td>-50° C to +125° C (-58° F to +257° F)</td>
</tr>
<tr>
<td>Relative Humidity</td>
<td>5% to 90% (non-condensing)</td>
</tr>
<tr>
<td>Altitude</td>
<td>12,000 feet (3,658 meters)</td>
</tr>
<tr>
<td>Power Requirements @3.3VDC</td>
<td></td>
</tr>
<tr>
<td>Maximum</td>
<td>270 (891 mW)</td>
</tr>
</tbody>
</table>
### Specifications

<table>
<thead>
<tr>
<th></th>
<th>ConnectPort® X2 Industrial</th>
<th>ConnectPort® X2 Commercial</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>General</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Management</td>
<td>HTTP/HTTPS web interface,</td>
<td>KConnect™ Device Cloud™</td>
</tr>
<tr>
<td></td>
<td>Password access control,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IP service port control,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Optional secure enterprise</td>
<td></td>
</tr>
<tr>
<td></td>
<td>management via KConnect™</td>
<td></td>
</tr>
<tr>
<td>Protocols</td>
<td>LES/TE, DCE, SNNPc</td>
<td>EDP/TE, DCE, LDCF</td>
</tr>
<tr>
<td>Total Memory</td>
<td>8 MB flash, 16 MB RAM</td>
<td>8 MB RAM, 1 MB flash</td>
</tr>
<tr>
<td></td>
<td>(Max available memory varies firmware and OS version)</td>
<td></td>
</tr>
<tr>
<td>LEDs</td>
<td>Ethernet status, Power, ZigBee link/activity</td>
<td>Ethernet status, Power, ZigBee link/activity</td>
</tr>
<tr>
<td>Security</td>
<td>SSL tunnels</td>
<td>SSL tunnels</td>
</tr>
<tr>
<td>Dimensional (L x W x H)</td>
<td>5.50 in x 2.75 in x 1.33 in (13.5 cm x 7.0 cm x 3.2 cm)</td>
<td>2.75 in x 3.75 in x 1.25 in (7.0 cm x 9.5 cm x 3.2 cm)</td>
</tr>
<tr>
<td></td>
<td>0.44 lb (2.0 kg)</td>
<td>0.20 lb (0.90 kg)</td>
</tr>
<tr>
<td><strong>Antenna</strong></td>
<td>XBi2 Antenna: 4&quot; dipole, RP-SMA (Center pin – on device: male; on antenna: female)</td>
<td>Internal</td>
</tr>
<tr>
<td><strong>RF (optional)</strong></td>
<td>4&quot; dipole with 2’ cable, table top mountable; RP-SMA (Center pin – on device: male; on antenna: female)</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Interface</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td>ZigBee, 802.15.4</td>
<td>Smart Energy 1.0 certified ZigBee-PRI</td>
</tr>
<tr>
<td>Ethernet</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ports</td>
<td>1 RJ-45 port</td>
<td></td>
</tr>
<tr>
<td>Physical Layer</td>
<td>10/100Base-T</td>
<td></td>
</tr>
<tr>
<td>Data Rate and Modus</td>
<td>10/100 Mbps (auto-negotiate); full or half duplex (auto-negotiate)</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>WiFi (802.11ab/g)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>2.4 GHz</td>
<td>N/A</td>
</tr>
<tr>
<td>Data Rate</td>
<td>Up to 15 Mbps with fallback</td>
<td>N/A</td>
</tr>
<tr>
<td>Modulation</td>
<td>OOKPSK (1 Mbps), QPSK (2 Mbps), BPSK (11.5 Mbps)</td>
<td>N/A</td>
</tr>
<tr>
<td>Transmit Power</td>
<td>16 dBm typical</td>
<td>N/A</td>
</tr>
<tr>
<td>Receiver Sensitivity</td>
<td>-48 dBm at 11 Mbps</td>
<td>N/A</td>
</tr>
<tr>
<td>Modes</td>
<td>Ad-hoc, AP Client, Monitor modes only; Access Point Mode not supported</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Power Requirements</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Input</td>
<td>9-30 VDC</td>
<td>1 VDC</td>
</tr>
<tr>
<td>Power Supply</td>
<td>11 VDC power supply for 4°C to 60°C (-4° F to 140° F) with locking hard connector included; Extended temperature power supply available separately</td>
<td>5 VDC power supply included</td>
</tr>
<tr>
<td></td>
<td>5 VDC power supply included</td>
<td></td>
</tr>
<tr>
<td>Power Consumption</td>
<td>1.2 W Max. 3.4 W</td>
<td>1.2 W Max. 3.4 W</td>
</tr>
<tr>
<td>Surge Protection</td>
<td>4 W burst (FFT) per-4+, 2 KV surge per EN61000-4</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Environmental</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operating Temperature</td>
<td>-20°C to +70°C (-4°C to +160° F)</td>
<td>0°C to +40°C (-4° F to 104° F)</td>
</tr>
<tr>
<td>Relative Humidity</td>
<td>5% to 95% (non-condensing)</td>
<td></td>
</tr>
<tr>
<td>Ethernet Isolation</td>
<td>1500 VAC/min per IEEE802.3V (ANSI X12.5)</td>
<td></td>
</tr>
<tr>
<td>Regulatory Approvals</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safety</td>
<td>EN60950</td>
<td>N/A</td>
</tr>
<tr>
<td>Emissions/Immunity</td>
<td>CE, FCC Part 15 (E-Class A)</td>
<td>N/A</td>
</tr>
</tbody>
</table>

---

Digi ConnectPort X2 Specification. Copied from Digi (2013) [9.]
Appendix 2: HTTP requests methods. Copied from Wikipedia (2013) [5.]

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Method</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GET</td>
<td>Requests a document from the server.</td>
</tr>
<tr>
<td>2</td>
<td>HEAD</td>
<td>Requests information (headers) about the document but not the document itself.</td>
</tr>
<tr>
<td>3</td>
<td>POST</td>
<td>Sends data from client to server.</td>
</tr>
<tr>
<td>4</td>
<td>PUT</td>
<td>Sends data from server to client.</td>
</tr>
<tr>
<td>5</td>
<td>DELETE</td>
<td>Removes the requested recourse from the webserver.</td>
</tr>
<tr>
<td>6</td>
<td>TRACE</td>
<td>Echoes the incoming request back, so that the client can see what servers are adding or changing in the request.</td>
</tr>
<tr>
<td>7</td>
<td>OPTION</td>
<td>Returns the HTTP methods that the server supports.</td>
</tr>
<tr>
<td>8</td>
<td>CONNECT</td>
<td>Converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communications (HTTPS).</td>
</tr>
<tr>
<td>9</td>
<td>PATCH</td>
<td>Is used to apply partial modifications to a resource.</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Code</th>
<th>Phrase</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Continue</td>
<td>The initial part of the request has been received, and the client may continue with his request.</td>
</tr>
<tr>
<td>101</td>
<td>Switching</td>
<td>The server is complying with a client request to switch protocols defined in the upgrade header.</td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
<td>The request is successful.</td>
</tr>
<tr>
<td>201</td>
<td>Created</td>
<td>A new URL is created.</td>
</tr>
<tr>
<td>202</td>
<td>Accepted</td>
<td>The request is accepted.</td>
</tr>
<tr>
<td>204</td>
<td>No content</td>
<td>There is no content in the body.</td>
</tr>
<tr>
<td>301</td>
<td>Moved permanently</td>
<td>The requested URL is no longer used by the server.</td>
</tr>
<tr>
<td>302</td>
<td>Moved temporarily</td>
<td>The requested URL is moved temporarily.</td>
</tr>
<tr>
<td>304</td>
<td>Not modified</td>
<td>The document has not been modified.</td>
</tr>
<tr>
<td>400</td>
<td>Bad request</td>
<td>There is a syntax error in the request.</td>
</tr>
<tr>
<td>401</td>
<td>Unauthorized</td>
<td>The request lacks proper authorization.</td>
</tr>
<tr>
<td>403</td>
<td>Forbidden</td>
<td>Service is denied.</td>
</tr>
<tr>
<td>404</td>
<td>Not found</td>
<td>The document is not found.</td>
</tr>
<tr>
<td>405</td>
<td>Method not allowed</td>
<td>The method is not supported in this URL.</td>
</tr>
<tr>
<td>406</td>
<td>Not acceptable</td>
<td>The format requested is not acceptable.</td>
</tr>
<tr>
<td>500</td>
<td>Internal server error</td>
<td>There is an error at the server side.</td>
</tr>
<tr>
<td>501</td>
<td>Not implemented</td>
<td>The requested action cannot be performed.</td>
</tr>
<tr>
<td>503</td>
<td>Service unavailable</td>
<td>The service is temporarily unavailable.</td>
</tr>
</tbody>
</table>
### HTTP: General Headers

<table>
<thead>
<tr>
<th>Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache-control</td>
<td>Specifies information about caching.</td>
</tr>
<tr>
<td>Connection</td>
<td>Shows whether the connection should be closed or not.</td>
</tr>
<tr>
<td>Date</td>
<td>Shows current date.</td>
</tr>
<tr>
<td>MIME-version</td>
<td>Shows the MIME version used</td>
</tr>
<tr>
<td>Upgrade</td>
<td>Specifies the preferred communication protocol.</td>
</tr>
</tbody>
</table>

### HTTP: Request Headers

<table>
<thead>
<tr>
<th>Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Shows the medium format that the client can accept.</td>
</tr>
<tr>
<td>Accept-charset</td>
<td>Shows the character set that the client can accept.</td>
</tr>
<tr>
<td>Accept-encoding</td>
<td>Shows the encoding scheme that the client can handle.</td>
</tr>
<tr>
<td>Accept-language</td>
<td>Shows the language that the client accepts.</td>
</tr>
<tr>
<td>Authorization</td>
<td>Shows what permissions the client has.</td>
</tr>
<tr>
<td>Cookie</td>
<td>Specifies the cookies name.</td>
</tr>
<tr>
<td>From</td>
<td>Shows the e-mail address of the user.</td>
</tr>
<tr>
<td>Host</td>
<td>Shows the host and port number of the server.</td>
</tr>
<tr>
<td>If-modified_since</td>
<td>Sends the document if newer than the specified date.</td>
</tr>
<tr>
<td>If-match</td>
<td>Sends the document only if it matches the given tag.</td>
</tr>
<tr>
<td>If-non-match</td>
<td>Sends the document only if it does not match the given tag.</td>
</tr>
<tr>
<td>If-range</td>
<td>Sends only the portion of the document that is missing.</td>
</tr>
<tr>
<td>If-unmodified_since</td>
<td>Sends the document if not changed since the specified date.</td>
</tr>
<tr>
<td>Referer</td>
<td>Specifies the URL of the linked document.</td>
</tr>
<tr>
<td>User-agent</td>
<td>Identifies the client program.</td>
</tr>
</tbody>
</table>

### HTTP: Response Headers

<table>
<thead>
<tr>
<th>Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept-range</td>
<td>Shows if server accepts the range requested by client.</td>
</tr>
<tr>
<td>Age</td>
<td>Shows the age of the document.</td>
</tr>
<tr>
<td>Public</td>
<td>Shows the supported list of methods.</td>
</tr>
<tr>
<td>Retry-after</td>
<td>Specifies the date after which the server is available.</td>
</tr>
<tr>
<td>Server</td>
<td>Shows the server name and version number.</td>
</tr>
</tbody>
</table>

### HTTP: Entity Headers

<table>
<thead>
<tr>
<th>Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td>Lists valid methods that can be used with a URL.</td>
</tr>
<tr>
<td>Content-encoding</td>
<td>Specifies the encoding scheme.</td>
</tr>
<tr>
<td>Content-language</td>
<td>Specifies the language.</td>
</tr>
<tr>
<td>Content-length</td>
<td>Shows the length of the document.</td>
</tr>
<tr>
<td>Content-range</td>
<td>Specifies the range of the document.</td>
</tr>
<tr>
<td>Content-type</td>
<td>Specifies the medium type.</td>
</tr>
<tr>
<td>ETag</td>
<td>Gives an entity tag.</td>
</tr>
<tr>
<td>Expires</td>
<td>Gives the date and time when contents may change.</td>
</tr>
<tr>
<td>Last-modified</td>
<td>Gives the date and time of the last change.</td>
</tr>
<tr>
<td>Location</td>
<td>Specifies the location of the created or moved document.</td>
</tr>
</tbody>
</table>
Appendix 5: SCI data string

//This method builds up and returns an SCI http request data based on the table remote commands

```java
public String HttpDataString(String tablecommand, int tablenumber) {
    StringBuffer mes = new StringBuffer();
    String data = "";
    if (tablecommand.equals("Table" + tablenumber + "-Up"))
        data = "<table_up>" + tablenumber + "</table_up>");
```
```java
if (tablecommand.equals("Table" + tablenumber + "-Down"))
    data = "<table_down>" + tablenumber + "</table_down>");
```
```java
if (tablecommand.equals("Table" + tablenumber + "-Unlock"))
    data = "<unlock>" + tablenumber + "</unlock>");
```
```java
if (tablecommand.equals("Table" + tablenumber + "-Disable"))
    data = "<disable_power>" + tablenumber + "</disable_power>");
```
```java
if (tablecommand.equals("Table" + tablenumber + "-Power"))
    data = "<allow_power>" + tablenumber + "</allow_power>");
```
```java
if (tablecommand.equals("Table" + tablenumber + "-Unlock_all"))
    data = "<unlock_all>" + (tablenumber - 10) + "</unlock_all>");
```
```java
if (tablecommand.equals("Table" + tablenumber + "-Power_all"))
    data = "<allow_power_all>" + (tablenumber - 10) + "</allow_power_all>");
```
```java
if (tablecommand.equals("Table" + tablenumber + "-Disable_all"))
    data = "<disable_power_all>" + (tablenumber - 10) + "</disable_power_all>");
```
```java
if (tablecommand.equals("Table" + tablenumber + "-Status_all"))
    data = "<status_all>" + (tablenumber - 10) + "</status_all>");
```
```java
mes.append("<sci_request version="1.0">
" + data + "
" + (tablenumber - 10) + "</rci_request>
" + (tablenumber - 10) + "</send_message>
" + (tablenumber - 10) + "</sci_request>";
```
```java
return mes.toString();
```
package com.mobilePackage;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.util.*;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.protocol.HTTP;
import com.mobilePackage.ConnectionDetector;
import com.comm.R;
import android.app.*;
import android.os.*;
import android.widget.*;
import android.util.Log;
import android.view.KeyEvent;
import android.view.View;
import android.text.format.DateUtils;
import android.content.DialogInterface;
import android.view.View.OnClickListener;
import android.graphics.drawable.Drawable;
public class MobileActivity extends Activity implements OnClickListener {
    private int Default=0;
    private HttpAsyncTask T=new HttpAsyncTask();
    private Boolean HttpPostExecuted=false;
    private Boolean isInternetPresent = false;
    private Boolean updateTablestatus = false;
    ConnectionDetector cd,cd2;
    private Button tab_1,tab_2,tab_3,tab_4,tab_5,tab_6,tab_7,tab_8,tab_9,tab_10,lockup,stop,login;
    private AlertDialog Alert,Alert1,Alert2;
    private TableLayout loginLayout,buttonlayout;
    private ImageView img;
    private ProgressBar pb,pb2,pb3;
    private Drawable k,Color;
    private int TableNumber,butnum;
    private String nam, passwd,HttpRequest,command;
    private EditText name,password;
    private final static String filename="tableStatus.txt";
    private String Uname;
    private String pword;
    @Override
}
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.auto);  
    Alert1 = new AlertDialog.Builder(this).create();  
    Alert2 = new AlertDialog.Builder(this).create();  
    loginLayout=(TableLayout)findViewById(R.id.Loginpanel);  
    buttonlayout=(TableLayout)findViewById(R.id.buttonpanel);  
    img=(ImageView)findViewById(R.id.imageView2);  
    login = (Button) findViewById(R.id.login);  
    name=(EditText)findViewById(R.id.username);  
    password=(EditText)findViewById(R.id.password);  
    Uname = getResources().getString(R.string.Uname);  
    pword = getResources().getString(R.string.Pword);  
    name.setText(Uname);  
    password.setText(pword);  
    lockup = (Button) findViewById(R.id.lockup);  
    stop = (Button) findViewById(R.id.stop);  
    tab_1 = (Button) findViewById(R.id.table1);  
    tab_2 = (Button) findViewById(R.id.table2);  
    tab_3 = (Button) findViewById(R.id.table3);  
    tab_4 = (Button) findViewById(R.id.table4);  
    tab_5 = (Button) findViewById(R.id.table5);  
    tab_6 = (Button) findViewById(R.id.table6);  
    tab_7 = (Button) findViewById(R.id.table7);  
    tab_8 = (Button) findViewById(R.id.table8);  
    tab_9 = (Button) findViewById(R.id.table9);  
    tab_10 = (Button) findViewById(R.id.table10);  
    pb=(ProgressBar)findViewById(R.id.progressbar);  
    pb2=(ProgressBar)findViewById(R.id.progressbar2);  
    pb3=(ProgressBar)findViewById(R.id.progressbar3);  
    tab_1.setOnClickListener(this);  
    tab_2.setOnClickListener(this);  
    tab_3.setOnClickListener(this);  
    tab_4.setOnClickListener(this);  
    tab_5.setOnClickListener(this);  
    tab_6.setOnClickListener(this);  
    tab_7.setOnClickListener(this);  
    tab_8.setOnClickListener(this);  
    tab_9.setOnClickListener(this);  
    tab_10.setOnClickListener(this);  
    login.setOnClickListener(this);  
    lockup.setOnClickListener(this);  
    stop.setOnClickListener(this);
}

public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_BACK) {
        LogOutButton("Yes","No ","Piha-Mobile-Remote-Control");
    }
    return super.onKeyDown(keyCode, event);
}

@Override
public void onClick(View view) {
    // TODO Auto-generated method stub
    if (view==login){
        TableNumber=0;
        nam = getResources().getString(R.string.Uname);  
        passwd = getResources().getString(R.string.Pword);  
}
loginScreen(nam, passwd);
if(view==tab_1){
    TableNumber=1;
    ControlScreen(1, "Table-Up", "Table-Down", "Table-Unlock");
}if(view==tab_2){
    TableNumber=2;
    ControlScreen(2, "Table-Up", "Table-Down", "Table-Unlock");
}if(view==tab_3){
    TableNumber=3;
    ControlScreen(3, "Table-Up", "Table-Down", "Table-Unlock");
}if(view==tab_4){
    TableNumber=4;
    ControlScreen(4, "Table-Up", "Table-Down", "Table-Unlock");
}if(view==tab_5){
    TableNumber=5;
    ControlScreen(5, "Table-Up", "Table-Down", "Table-Unlock");
}if(view==tab_6){
    TableNumber=6;
    ControlScreen(6, "Table-Up", "Table-Down", "Table-Unlock");
}if(view==tab_7){
    TableNumber=7;
    ControlScreen(7, "Table-Up", "Table-Down", "Table-Unlock");
}if(view==tab_8){
    TableNumber=8;
    ControlScreen(8, "Table-Up", "Table-Down", "Table-Unlock");
}if(view==tab_9){
    TableNumber=9;
    ControlScreen(9, "Table-Up", "Table-Down", "Table-Unlock");
}if(view==tab_10){
    TableNumber=10;
    ControlScreen(10, "Table-Up", "Table-Down", "Table-Unlock");
}if(view==stop){
    TableNumber=11;
    UnlockAndPowerButton("Unlock-all", "Power-ON", "Power-OFF");
}if(view==lockup){
    LogOutButton("Yes", "No", "Table status");
}

//This class allows to perform background operations and publish the status of all laboratory tables

private class HttpAsyncTask extends AsyncTask<String, Integer, Double>{
    String HttpResponse;

    // Execute http post in background
    @Override
    protected Double doInBackground(String... params) {
        HttpResponse = HttpConnection(params[0]);
        return null;
    }

    // updates the table status when an action is performed.
    protected void onPostExecute(Double Result) {
        HttpAsyncTask T3 = new HttpAsyncTask();
    }
}
if (HttpPostExecuted){
    if (updateTablestatus)
        pb3.setVisibility(View.GONE);
    Toast.makeText(getApplicationContext(), "Table Status Updated", Toast.LENGTH_LONG).show();
    if (!ReadPinStatus(HttpResponse).equals(null))
        Updatetablestatus(ReadPinStatus(HttpResponse));
}if (TableNumber==0){
    loginLayout.setVisibility(View.GONE);
    login.setVisibility(View.GONE);
    buttonlayout.setVisibility(View.VISIBLE);
    img.setVisibility(View.VISIBLE);
    Toast.makeText(getApplicationContext(), "Connected", Toast.LENGTH_LONG).show();
    if (!ReadPinStatus(HttpResponse).equals(null))
        showtablestatus(ReadPinStatus(HttpResponse));
}if ((TableNumber>0) && (TableNumber<11)) {
    buttonlayout.setVisibility(View.VISIBLE);
    UpdateStatusOnbutton(TableNumber, command, Color);
    Toast.makeText(getApplicationContext(), "Command executed", Toast.LENGTH_LONG).show();
    HttpRequest=HttpDataString("Status_all", Default);
    T3.execute("http://my.idigi.co.uk:80/ws/sci");
    updateTablestatus=true;
    TableNumber=-2;
}if (TableNumber==11){
    buttonlayout.setVisibility(View.VISIBLE);
    setAllbuttonstatus(command);
    HttpRequest=HttpDataString("Status_all", Default);
    T3.execute("http://my.idigi.co.uk:80/ws/sci");
    updateTablestatus=true;
    TableNumber=-2;
    Toast.makeText(getApplicationContext(), "Command executed", Toast.LENGTH_LONG).show();
} else {
    pb.setVisibility(View.GONE);
    Toast.makeText(getApplicationContext(), "Service Not Available ", Toast.LENGTH_LONG).show();
    finish();
}

protected void onProgressUpdate(Integer... progress){
    if (TableNumber==0)
        pb.setProgress(progress[0]);
    else if (TableNumber==-1)
        pb2.setProgress(progress[0]);
    else
        pb3.setProgress(progress[0]);
}

//This method initiate http connection with iDigi Cloud Service for posting http request
// data to control the laboratory tables.

public String HttpConnection(String url){
    String Hresponse=null;
try{
    HttpParams httpParams = new BasicHttpParams();
    int some_reasonable_timeout = (int) (20 * DateUtils.SECOND_IN_MILLIS);
    HttpConnectionParams.setConnectionTimeout(httpParams, some_reasonable_timeout);
    HttpConnectionParams.setSoTimeout(httpParams, some_reasonable_timeout);
    DefaultHttpClient httpClient=
    new DefaultHttpClient(httpParams);
    HttpPost post=
    new HttpPost(url);
    HttpClient.getCredentialsProvider().setCredentials( new
    AuthScope("my.idigi.co.uk", 80), new UsernamePasswordCredentials(" ", "1" ));
    StringEntity request =
    new StringEntity(HttpRequest, HTTP.UTF_8);
    request.setContentType("text/xml");
    post.setEntity(request);
    HttpResponse response = httpClient.execute(post);
    HttpEntity responseEntity= response.getEntity();
    InputStream is = responseEntity.getContent();
    Scanner isscanner=
    new Scanner(is);
    StringBuffer buf=
    new StringBuffer();
    while (is scanner.hasNextLine()){
        buf.append( is scanner.nextLine() + "\n");
    }
    HttpResponse=buf.toString().replaceAll("<>", "<\n>");
    if(response.getStatusLine().getStatusCode()==200)
        HttpPostExecuted=true;
} catch (ClientProtocolException e) {
    Log.d("Error", "Failed to do something: " + e.getMessage());
} catch (IOException e) {
    return HttpResponse;
}

// This method extracts and returns the GPIO pins status of all
// the tables from the http response data string into a vector array

public Vector<String> ReadPinStatus(String data) {
    Vector<String> buf=null;
    try{
        OutputStreamWriter out=
        new OutputStreamWriter(openFileOutput( filename, 0));
        out.write(data);
        out.close();
        InputStream in=openFileInput( filename);
        if (in!=null) {
            InputStreamReader tmp=
            new InputStreamReader(in);
            BufferedReader reader=
            new BufferedReader(tmp);
            String str;
            buf=new Vector<String>();
            while ((str = reader.readLine()) != null) {
                buf.add(str+"\n");
            }
            in.close();
        }
    } catch (java.io.FileNotFoundException e) {
    } catch (Throwable t) {
        Log.d("Error", "Failed to do something: " + t.toString());
    }
public String HttpDataString(String tablecommand, int tablenumber) {
    StringBuffer mes = new StringBuffer();
    String data = "";
    if (tablecommand.equals("Table" + tablenumber + "-Up"))
        data = "<table_up>" + tablenumber + "</table_up>";
    if (tablecommand.equals("Table" + tablenumber + "-Down"))
        data = "<table_down>" + tablenumber + "</table_down>";
    if (tablecommand.equals("Table" + tablenumber + "-Unlock"))
        data = "<unlock>" + tablenumber + "</unlock>";
    if (tablecommand.equals("Table" + tablenumber + "-Disable"))
        data = "<disable_power>" + tablenumber + "</disable_power>";
    if (tablecommand.equals("Table" + tablenumber + "allow_power"))
        data = "<allow_power>" + tablenumber + "</allow_power>";
    if (tablecommand.equals("unlock_all"))
        data = "<unlock_all>" + (tablenumber - 10) + "</unlock_all>";
    if (tablecommand.equals("allow_power_all"))
        data = "<allow_power_all>" + (tablenumber - 10) + "</allow_power_all>";
    if (tablecommand.equals("disable_power_all"))
        data = "<disable_power_all>" + (tablenumber - 10) + "</disable_power_all>";
    if (tablecommand.equals("Status_all"))
        data = "<status_all>" + (tablenumber - 10) + "</status_all>";
    mes.append("<sci_request version="1.0">\r\n");
    mes.append("<send_message reply="all">\r\n");
    mes.append("<targets>\r\n");
    mes.append("<device id="00000000-00000000-00409DFF-FF52FB61"/>\r\n");
    mes.append("</targets>\r\n");
    mes.append("<rci_request version="1.1">\r\n");
    mes.append("<do_command target="at_">\r\n");
    mes.append(data);
    mes.append("</do_command>\r\n");
    mes.append("</rci_request>\r\n");
    mes.append("</send_message>\r\n");
    return mes.toString();
}

// Updates each control screen table button status
public void UpdateStatusOnButton(int button, String state, Drawable colour) {
    String On = "PD";
    switch (button) {
    case 1:
        if (state.equals("disable_power_all")) {
            tab_1.setText("Table " + button + On);
            tab_1.setTextSize(10);
        }
        else {
            tab_1.setText("Table " + button);
            tab_1.setTextSize(10);
        }
        if (!state.equals("allow_power_all"))
            tab_1.setBackgroundDrawable(colour);
    // This method builds up and returns an SCI http request data based on the table remote commands
case 2:
if (state.equals("disable_power_all")) {
    tab_2.setText("Table " + button + "on");
    tab_2.setTextSize(10);
} else {
    tab_2.setText("Table " + button);
    tab_2.setTextSize(10);
    if (!state.equals("allow_power_all"))
        tab_2.setBackgroundDrawable(colour);
}
break;

case 3:
if (state.equals("disable_power_all")) {
    tab_3.setText("Table " + button + "on");
    tab_3.setTextSize(10);
} else {
    tab_3.setText("Table " + button);
    tab_3.setTextSize(10);
    if (!state.equals("allow_power_all"))
        tab_3.setBackgroundDrawable(colour);
}
break;

case 4:
if (state.equals("disable_power_all")) {
    tab_4.setText("Table " + button + "on");
    tab_4.setTextSize(10);
} else {
    tab_4.setText("Table " + button);
    tab_4.setTextSize(10);
    if (!state.equals("allow_power_all"))
        tab_4.setBackgroundDrawable(colour);
}
break;

case 5:
if (state.equals("disable_power_all")) {
    tab_5.setText("Table " + button + "on");
    tab_5.setTextSize(10);
} else {
    tab_5.setText("Table " + button);
    tab_5.setTextSize(10);
    if (!state.equals("allow_power_all"))
        tab_5.setBackgroundDrawable(colour);
}
break;

case 6:
if (state.equals("disable_power_all")) {
    tab_6.setText("Table " + button + "on");
    tab_6.setTextSize(10);
} else {
    tab_6.setText("Table " + button);
    tab_6.setTextSize(10);
    if (!state.equals("allow_power_all"))
        tab_6.setBackgroundDrawable(colour);
}
break;

case 7:
if(state.equals("disable_power_all")){
tab_7.setText("Table "+button+On);
tab_7.setTextSize(10);
}else{
tab_7.setText("Table "+button);
tab_7.setTextSize(10);
if(!state.equals("allow_power_all"))
tab_7.setBackgroundDrawable(colour);
}
break;
case 8:
if(state.equals("disable_power_all")){
tab_8.setText("Table "+button+On);
tab_8.setTextSize(10);
}else{
tab_8.setText("Table "+button);
tab_8.setTextSize(10);
if(!state.equals("allow_power_all"))
tab_8.setBackgroundDrawable(colour);
}
break;
case 9:
if(state.equals("disable_power_all")){
tab_9.setText("Table "+button+On);
tab_9.setTextSize(10);
}else{
tab_9.setText("Table "+button);
tab_9.setTextSize(10);
if(!state.equals("allow_power_all"))
tab_9.setBackgroundDrawable(colour);
}
break;
case 10:
if(state.equals("disable_power_all")){
tab_10.setText("Table "+button+On);
tab_10.setTextSize(10);
}else{
tab_10.setText("Table "+button);
tab_10.setTextSize(10);
if(!state.equals("allow_power_all"))
tab_10.setBackgroundDrawable(colour);
}
break;

//Updates and sets all the control screen table buttons
public void setAllbuttonstatus(String command){
for(int i=1;i<11;i++)
UpdateStatusOnbutton(i,command,Color);
}

//This method quit the software
public void LogOutButton(String a, String b,String tittle){
Alert1.setTitle(tittle);
Alert1.setMessage("Do you want to exit?");
Alert1.setButton(a, new DialogInterface.OnClickListener(){
@Override
public void onClick(DialogInterface dialog, int which){
finish();
}});
public void UnlockAndPowerButton( String a, String b, String c){
    Alert2.setTitle("Table Status");
    Alert2.setMessage("Do you want to Power on or off ALL tables?");
    Alert2.setButton(a, new DialogInterface.OnClickListener(){
        @Override
        public void onClick(DialogInterface dialog, int which){
            command="unlock_all";
            Color=getResources().getDrawable(R.drawable.tablebuttonunlocked);
            pb3.setVisibility(View.VISIBLE);
            HttpAsyncTask T=new HttpAsyncTask();
            HttpRequest=HttpDataString(command, TableNumber);
            T.execute("http://my.idigi.co.uk:80/ws/sci");
            dialog.dismiss();
        }
    });
    Alert2.setButton2(b, new DialogInterface.OnClickListener(){
        @Override
        public void onClick(DialogInterface dialog, int which){
            command="allow_power_all";
            Color=getResources().getDrawable(R.drawable.tablebuttonunlocked);
            pb3.setVisibility(View.VISIBLE);
            HttpAsyncTask T=new HttpAsyncTask();
            HttpRequest=HttpDataString(command, TableNumber);
            T.execute("http://my.idigi.co.uk:80/ws/sci");
            dialog.dismiss();
        }
    });
    Alert2.setButton3(c, new DialogInterface.OnClickListener(){
        @Override
        public void onClick(DialogInterface dialog, int which){
            command="disable_power_all";
            Color=getResources().getDrawable(R.drawable.powerbackground);
            pb3.setVisibility(View.VISIBLE);
            HttpAsyncTask T=new HttpAsyncTask();
            HttpRequest=HttpDataString(command, TableNumber);
            T.execute("http://my.idigi.co.uk:80/ws/sci");
            dialog.dismiss();
        }
    });
    Alert2.show();
}

//This method post a command to unlock a table or sets a table in an upward or downward position.
public void ControlScreen(int i, String a, String b, String c){
    butnum=i;
    Alert.setTitle("Table "+ i + " status");
    Alert.setMessage("Do you want to put table "+i+" at Up, or Down or Unlock level"+"?");
**Alert**.setButton(a, new DialogInterface.OnClickListener(){
    @Override
    public void onClick(DialogInterface dialog, int which){
        Color = getResources().getDrawable(R.drawable.up_arrow);
        pb3.setVisibility(View.VISIBLE);
        HttpAsyncTask T = new HttpAsyncTask();
        HttpRequest = HttpDataString(command, TableNumber);
        T.execute("http://my.idigi.co.uk:80/ws/sci");
        dialog.dismiss();
    }});
    Alert.setButton2(b, new DialogInterface.OnClickListener(){
        @Override
        public void onClick(DialogInterface dialog, int which){
            Color = getResources().getDrawable(R.drawable.down_arrow);
            pb3.setVisibility(View.VISIBLE);
            HttpAsyncTask T = new HttpAsyncTask();
            HttpRequest = HttpDataString(command, TableNumber);
            T.execute("http://my.idigi.co.uk:80/ws/sci");
            dialog.dismiss();
        }});
    Alert.setButton3(c, new DialogInterface.OnClickListener(){
        @Override
        public void onClick(final DialogInterface dialog, final int which){
            Color = getResources().getDrawable(R.drawable.tablebuttonunlocked);
            pb3.setVisibility(View.VISIBLE);
            HttpAsyncTask T = new HttpAsyncTask();
            HttpRequest = HttpDataString(command, TableNumber);
            T.execute("http://my.idigi.co.uk:80/ws/sci");
            dialog.dismiss();
        }});
    Alert.show();
    }}
    public void GetAlltablestatus(String request,String Url){
        HttpRequest = request;
        T.execute(Url);
    }
    public void loginScreen(String nam,String passwd){
        cd = new ConnectionDetector(getApplicationContext());
        isInternetPresent = cd.isConnectingToInternet();
        Uname = getResources().getString(R.string.Uname);
        pword = getResources().getString(R.string.Pword);
        if(nam.equals(Uname) & passwd.equals(pword)){
            if(isInternetPresent) {
                pb.setVisibility(View.VISIBLE);
                GetAlltablestatus(HttpDataString("Status_all",Default),
"http://my.idigi.co.uk:80/ws/sci");
            } else cd.showAlertDialog(this, "No Internet Connection","You don't have
internet connection.", false);
        else Toast.makeText(getBaseContext(), "Invalid Username or Pass-
word.", Toast.LENGTH_LONG).show();
        name.setText(Uname);
        password.setText(pword);
    }
// sets the unlock or lock color for laboratory table.
public Drawable Setcolor(String pin6,String pin7){
    if(pin6.equals("6 on") & pin7.equals("7 off"))
    k=getResources().getDrawable(R.drawable.tablebuttonlocked);
    if(pin7.equals("7 on") & pin6.equals("6 off"))
    k=getResources().getDrawable(R.drawable.tablebuttonlocked);
    if(pin7.equals("7 off") & pin6.equals("6 off"))
    k=getResources().getDrawable(R.drawable.tablebuttonunlocked);
    if(pin6.equals("6 on") & pin7.equals("7 on"))
    k=getResources().getDrawable(R.drawable.tablebuttonlocked);
    return k;
}

//sets the power off or on signal for laboratory tables
public String SetPowerState(String pin){
    if(pin.equals("1 off"))
    return " PD";
    else
    return "";
}

//sets the signal for a table button.
public void setTablecolors(int tablenumber,Drawable color,String pin){
    switch (tablenumber){
    case 1:{
    tab_1.setText("Table "+tablenumber+SetPowerState(pin));
    tab_1.setTextSize(10);
    tab_1.setBackgroundDrawable(color);
    }break;
    case 2:{
    tab_2.setText("Table "+tablenumber+SetPowerState(pin));
    tab_2.setTextSize(10);
    tab_2.setBackgroundDrawable(color);
    }break;
    case 3:{
    tab_3.setText("Table "+tablenumber+SetPowerState(pin));
    tab_3.setTextSize(10);
    tab_3.setBackgroundDrawable(color);
    }break;
    case 4:{
    tab_4.setText("Table "+tablenumber+SetPowerState(pin));
    tab_4.setTextSize(10);
    tab_4.setBackgroundDrawable(color);
    }break;
    case 5:{
    tab_5.setText("Table "+tablenumber+SetPowerState(pin));
    tab_5.setTextSize(10);
    tab_5.setBackgroundDrawable(color);
    }break;
    case 6:{
    tab_6.setText("Table "+tablenumber+SetPowerState(pin));
    tab_6.setTextSize(10);
    tab_6.setBackgroundDrawable(color);
    }break;
    case 7:{
    tab_7.setText("Table "+tablenumber+SetPowerState(pin));
    tab_7.setTextSize(10);
    tab_7.setBackgroundDrawable(color);
    }break;
    case 8:{
    tab_8.setText("Table "+tablenumber+SetPowerState(pin));
    tab_8.setTextSize(10);
    tab_8.setBackgroundDrawable(color);
    }break;
    case 9:{
    tab_9.setText("Table "+tablenumber+SetPowerState(pin));
    tab_9.setTextSize(10);
    tab_9.setBackgroundDrawable(color);
    }break;
    }
case 10:
{
    tab_10.setText("Table " + tablenumber + SetPowerState(pin));
    tab_10.setBackgroundDrawable(color);
    tab_10.setTextSize(10);
    break;
}

// Automatically updates all the tables if they are unlocked or locked.
public void UpdateTablestatus(Vector<String> v) {
    int i, index1, index2, index3;
    String pin1, pin6, pin7;
    for (i = 0; i < 10; i++) {
        index1 = 5 + 14 * (i);
        index2 = index1 + 5;
        index3 = index2 + 1;
        pin1 = v.elementAt(index1).trim();
        pin6 = v.elementAt(index2).trim();
        pin7 = v.elementAt(index3).trim();
        setTablecolors(i + 1, Setcolor(pin6, pin7), pin1);
    }
    UpdateTablestatus = false;
    HttpPostExecuted = false;
}

// Shows the status all the tables if they are unlocked or locked.
public void showtablestatus(Vector<String> v) {
    int i, index1, index2, index3;
    String pin1, pin6, pin7;
    for (i = 0; i < 10; i++) {
        index1 = 5 + 14 * (i);
        index2 = index1 + 5;
        index3 = index2 + 1;
        pin1 = v.elementAt(index1).trim();
        pin6 = v.elementAt(index2).trim();
        pin7 = v.elementAt(index3).trim();
        setTablecolors(i + 1, Setcolor(pin6, pin7), pin1);
    }
}

Appendix 7: AsyncTask class

// This class allows to perform background operations and publish the status of all laboratory tables

private class HttpAsyncTask extends AsyncTask<String, Integer, Double> {
    String HttpResponse;

    // Execute http post in background
@Override

protected Double doInBackground(String... params) {
    HttpResponse = HttpConnection(params[0]);
    return null;
}

// updates the table status when an action is performed.

protected void onPostExecute(Double Result){
    HttpAsyncTask T3 = new HttpAsyncTask();
    if (HttpPostExecuted)
        if (updateTablestatus)
            pb3.setVisibility(View.GONE);
            Toast.makeText(getApplicationContext(), "Table Status Updated", Toast.LENGTH_LONG).show();
            if (!ReadPinStatus(HttpResponse).equals(null))
                Updatetablestatus(ReadPinStatus(HttpResponse));
    if (TableNumber==0){
        loginLayout.setVisibility(View.GONE);
        login.setVisibility(View.GONE);
        pb.setVisibility(View.GONE);
        buttonlayout.setVisibility(View.VISIBLE);
        img.setVisibility(View.VISIBLE);
        Toast.makeText(getApplicationContext(), "Connected", Toast.LENGTH_LONG).show();
        if (!ReadPinStatus(HttpResponse).equals(null))
            showtablestatus(ReadPinStatus(HttpResponse));
    }if ((TableNumber>0) & (TableNumber<11)) {
        buttonlayout.setVisibility(View.VISIBLE);
        UpdateStatusOnbutton(TableNumber, command, Color);
        Toast.makeText(getApplicationContext(), "Command executed", Toast.LENGTH_LONG).show();
        HttpRequest=HttpDataString("Status_all", Default);
        T3.execute("http://my.idigi.co.uk:80/ws/sci");
        updateTablestatus=true;
        TableNumber=-2;
    }if (TableNumber==11){
buttonLayout.setVisibility(View.VISIBLE);
setAllButtonStatus(command);
HttpRequest=HttpDataString("Status_all", Default);
T3.execute("http://my.idigi.co.uk:80/ws/sci");
updateTableStatus=true;
TableNumber=-2;
Toast.makeText(getApplicationContext(), "Command executed", Toast.LENGTH_LONG).show();
}
}
else{
pb.setVisibility(View.GONE);
Toast.makeText(getApplicationContext(), "Service Not Available ", Toast.LENGTH_LONG).show();
finish();
}
}
protected void onProgressUpdate(Integer... progress){
    if(TableNumber==0)
pb.setProgress(progress[0]);
else if(TableNumber==-1)
pb2.setProgress(progress[0]);
else
    pb3.setProgress(progress[0]);
}
//This method initiate http connection with iDigi Cloud Service for posting http request
// data to control the laboratory tables.

public String HttpConnection(String url){
String Hresponse=null;
try{
    HttpParams httpParams = new BasicHttpParams();
    int some_reasonable_timeout = (int) (20 * DateUtils.SECOND_IN_MILLIS);
    HttpConnectionParams.setConnectionTimeout(httpParams, some_reasonable_timeout);
    HttpConnectionParams.setSoTimeout(httpParams, some_reasonable_timeout);
    DefaultHttpClient httpClient=new DefaultHttpClient( httpParams);
    } catch (Exception e) {
        e.printStackTrace();
    return null;
}
HttpPost post = new HttpPost(url);
Httpclient.getCredentialsProvider().setCredentials(new AuthScope("my.idigi.co.uk", 80),
new UsernamePasswordCredentials(nam, passwd));
StringEntity request = new StringEntity(HttpRequest, HTTP.UTF_8);
request.setContentType("text/xml");
post.setEntity(request);
HttpResponse response = Httpclient.execute(post);
HttpEntity responseEntity= response.getEntity();
Scanner iscanner=new Scanner(is);
StringBuffer buf=new StringBuffer();
while (iscanner.hasNextLine()){
  buf.append( iscanner.nextLine() + 
    "\n" );
}
Hresponse=buf.toString().replaceAll("<>", "<\n>");
if(response.getStatusLine().getStatusCode()==200)
HttpPostExecuted=true;
} catch (ClientProtocolException e) {
  Log.d("Error", "Failed to do something: " + e.getMessage());
} catch (IOException e) {
  // TODO Auto-generated catch block
}
return Hresponse;

// This method extracts and returns the GPIO pins status of all
// the tables from the http response data string into a vector array

public Vector<String> ReadPinStatus(String data) {
  Vector<String> buf=null;
  try{
    OutputStreamWriter out=new OutputStreamWriter(openFileOutput(filename,
      0));
    out.write(data);
    out.close();
    InputStream in=openFileInput(filename);
    if (in!=null) {

InputStreamReader tmp = new InputStreamReader(in);
BufferedReader reader = new BufferedReader(tmp);
String str;
buf = new Vector<String>();
while ((str = reader.readLine()) != null) {
    buf.add(str + "\n");
}
in.close();
}
}
catch (java.io.FileNotFoundException e) {
}
catch (Throwable t) {
    Log.d("Error", "Failed to do something: " + t.toString());
}
return buf;
}