# ABSTRACT

|  |
| --- |
| You must get the essence of what your report is about, usually in about 200 words. Most informative abstracts also have key parts in common. Each of these parts might consist of 1-2 sentences. The parts include: ♣ background ♣ aim or purpose of research ♣ method used ♣ findings/results ♣ conclusion |

The text of the abstract begins here. You may select this text and replace it with your own. The abstract counts as page iii, but as for all other pages with main headings (titles in all capital letters), a page number is not shown on it. The abstract is a required component of the thesis or dissertation, even in cases where individual chapters have their own abstracts. It should be a concise, carefully composed summary of the contents of the thesis or dissertation in which the problem is defined, the research method and design are described, and the results and conclusions are summarized.

There should be no more than 350 words in the abstract, which will normally make up about one and a half pages. No diagrams, citations, illustrations, or subheadings are included. If there are two pages in the abstract, the second page will be numbered iv, and this page number should appear centered at the bottom of the page.

(…)

# GLOSSARY

|  |  |
| --- | --- |
| SD | Succinct Data application |
| SMAC | Short Message Arithmetic Compressor |
| GUI | Graphical User Interface |
| XML | eXtensible Markup Language |
| XHTML | eXtensible HyperText Markup Language |
| IFRC | International Federation of the Red Cross |
| FRCS | Fiji Red Cross Society |
| NZ Red Cross | New Zealand Red Cross |
| iSMS | Interactive Short Message Service |
| UUID | Universal Unique IDentifier |
| SBD | Short Burst Data service (Iridium Constellation) |
| xsd | XML Schema Definition (used in the form tags) |
| MeshMS | Mesh Message Service |
| SM | State Machine |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Introduction

|  |
| --- |
| This is the introduction part, the aim of this part is to introduce the Serval project, its background and main components, as well as the Succinct Data application, its background, its scope, how it works in a high level understanding. |

## Report structure

This report will go through my personal work on the Succinct Data application within the context of the Serval project. The structure is made in the way that every reader can find his way, depending on his knowledge on the project. As a matter of fact, it goes for each part of the report from the overview, motivations and background to the specific details.

The aim is firstly, through the remainder of the introduction chapter, to understand the motivations and the background behind the Serval Project and Succinct Data and develop a good overview. Then, the second chapter presents the specifications of the current Succinct Data application and the background and specification of the needed improvements which make the core of my work. The third chapter presents the design of the suggested solutions. The fourth chapter introduces the materials and methods used to implement the solutions. The fifth chapter goes into the specific implementation of each functionnality. The sixth chapter presents the results. Finally, the seventh and eighth chapter conclude the work and present some potential further improvements. The appendices are essentially important examples of XML files used for compression.

## The Serval Project

|  |
| --- |
| At the end of the part, the reader should have an overview of the Serval Project, knows the main idea, the why, and the main components of the implementation of the project (the how). |

### Overview and background

|  |
| --- |
| Explain generally in what consists the Serval project: What is the main idea? What is the aim? How it is generally implemented? What problem it is solving ? |

The Serval Project [10] has been launched in 2010 and consists essentially on direct communications between terminals (D2D) through their Wi-Fi interface. This technology finds its applications in several fields, for example:

In some third-world countries or countries under development, the network infrastructure is partly or totally absent (or accessible only by a minority of urban/higher class people). Then the cost to set up a totally new infrastructure would be higher than relying on direct communications (and satellite).

Natural disasters happen often in places where the network infrastructure is poor (ocean, islands, high mountains, etc.) or destroys the existing network infrastructure.

Some NGO as Red-Cross New Zealand have often assignments in places where the network infrastructure is poor and the reliability on direct communications between Red-Cross staff members and also the communication to the rest of the world through satellite would be critical.

(…)

### Key elements on the Serval Project

|  |
| --- |
| What are the main components of the implantation? How does everything work? Keywords : Mobile phones, Serval Mesh, Serval Mesh Extender, InReach devices, Satellites |

The Serval Project relies on many components to fulfill its requirements.

The first flagship element of the project is the Serval Mesh [1], an android application that is able to use the phone as a root, activates the Wi-Fi interface and communicate directly through this interface with other devices. The current possible types of communication are mainly voice calls through the VoMP (Voice over Mesh Protocol), text messaging with MeshMS (A service comparable to SMS), and file sharing.



Figure : An overview on the Serval Mesh interface

(…)

## Succinct Data

|  |
| --- |
| In what consists the Succinct Data application? What is the context and the why behind its implantation? How does it work generally? What is the idea of the compression? Results of the compression? |

The work of the current project is focused on Succinct Data, a mobile application that operates right after Magpi/ODK phone application in order to minimize the data produced by this application.

Magpi is a phone application that allows people to create surveys and polls that are materialized into a form with questions that can be a simple text field to fill or a multiple-choices question.

Succinct Data takes the XML file representation of the Magpi record (Which is also refer to as an “XML record”) and “extremely” compresses it to as less bits as possible, getting close to the Shannon entropy (i.e. the minimal and necessary quantity of information contained in the data).

### Overview on the Magpi application and the forms

|  |
| --- |
| What is generally the Magpi application? What types of forms can we create? How are the forms filled in? What are the files generated by the Magpi application for each action (create form, fill in a record …) ? |



Magpi is an application available on Android smartphones and on browsers to create surveys, polls and more generally to share structured useful data in a one-to-one, many-to-one or one-to-many way. Magpi have proven its utility in many fields including Education, Microfinance, Agriculture, Health and Disaster Relief. For example, iSMS (A service of interactive SMS) allows a speaker to share a poll with the audience on their phone via SMS and displays the results live. As well, DatAgro is a service in Chile that uses Magpi iSMS for a two-way and broadcast communication between local farmers and the agricultural cooperative, for sharing crop prices, weather updates or any other useful information.

Besides, Magpi surveys, which are claimed to be easy-to-set-up and low cost by their provider, have been used for example, in collaboration with the International Federation of the Red Cross (IFRC) to determine the cash distribution in an earthquake-ravaged region in Nepal in function of the needs. Finally, and that’s in what we’re focusing, Magpi is used by the members of the Fiji Red Cross Society (FRCS) and of the New Zealand Red Cross for real time (or at least extremely responsive) disaster relief in remote areas of the Pacific region.

Mainly, it is used by the first aid team to inform about the nature of the disaster, the number of casualties or injured people, and all useful data in order for the association headquarter to send the right number/amount of goods and rescue teams. The results of the poll are summed up in statistical graphics for easy reading and understanding.

The forms that are interesting us are written in XML language and interpreted by the Magpi application GUI. To be interpreted by the Magpi application, the data from a survey record always comes with 2 files:

* The form specification (cf. Figure 2), an XHTML file for which the main role is to specify the form, the order of the elements, the type of each input and the associated constriaints (Min, Max) and the possibilities for a multiple-choice question input (radio question).
* The form record (cf. Figure 3), an XML file that contains *inter alia* the data content recorded by a user in the field, the timestamp of the record, and the record UUID.

**N.B.:** examples of a simple survey’s complete XHTML and XML files are provided in the appendices.

../pictures/magpi/magpi_01.png

Figure : The form design and its corresponding XHTML specification file

The XHTML file remains the same for the whole life of the form (unless the design is changed) and is divided in many sections. For simplicity purpose I presented the specification for a form that would only contain the 2nd question (Type your name).

Firstly, we have the children of the <meta> tag (highlighted in blue) which specify the meta fields (i.e. general information about the form). Then, in the instance tag, the children of the <data> tag specify the the survey/form questions data. The <bind> tags contains attributes that informs us on the field, including the type (In our case it is a “xsd:string”). Then, the <xf:input> tags specify the question phrase and also the possibilities (In the case of a multiple choice question).

The fields that are important to understand are:

* The form ID (“formid”) which is a unique identifier for this form design.
* The ID of each field, often built from the question phrase (In our case type\_your\_name)
* The type attribute of the <bind> tags

../pictures/magpi/magpi_02.png

Figure : The form record and its corresponding XML record file

The XML record is generated each time a form is filled with some data. Again for simplicity, I only present one field (but it’s the same for all fields). The XML record is much more simple than the XHTML because it only contains the actual recorded data while the XHTML specifies it. The record contains <meta> tag fields (in blue) which contain general information about the record (for example, the timestamp of the recording). The <data> tag fields contains the core information of the record.

Again, the most important fields to keep in mind are:

* The form ID which is the ID of the XHTML form specification linked to this file.
* The UUID which is the unique identifier of this record
* The ID of each question (the same that we can find in the form specification)

### Context and interests behind Succinct Data

|  |
| --- |
| In what consists the Succinct Data application? What is the context and the why behind its implantation? |

As seen in the previous Magpi overview, the Fiji FRCS and the New Zealand Red Cross first aid teams, and generally all disaster relief teams, use digital forms to report every important information in the disaster location (either by direct observation or polling the population). These forms are then sent to the NGO deciders so they can send as fast as possible the needed goods, materials and staff (this is sometimes a matter of life and death).

To communicate the important data to the deciders, field teams are using satellite communication devices such as *Delorme InReach*, or any other available mean (Cellular, SMS). There are two main reasons explaining the use of satellite communications and the need for a form extreme compressing application such as Succinct Data, one is practical and the other economic.

Practically, the reason is the same than the one behind the Serval Project. The area affected by the disaster is often hard to access, remote and has often a lack of telecommunications infrastructures or no infrastructures at all (destroyed or absent). The main constraint in disaster relief is the pace at which the teams, goods and materials are sent to heal injured people and save lives. Thus, we understand why the NGO such as the NZ Red Cross have no choice than using a time-guaranteed, low-latency satellite service, such as the SBD service, to send the data.

Economically, the Short Burst Data service is provided by Iridium and is “a simple and efficient satellite network transport capability to transmit short data messages between field equipment and a centralized host computing system” [4], with a relatively low latency: the burst message of less than 2 kB takes 6 to 22 seconds to arrive. [4] And the fact is that this service is extremely expensive: In addition to the relatively cheap service fee of US$40,00, the data charge is billed US$ 1,09 per kilobyte [5]. Considering that the average size of a Magpi form is around 700 Bytes, this means that it would cost US$ 700 for 1000 forms, which exceeds the price of the smartphone and the satellite device combined (The used *InReach* device costs US$ 300), and thus is really expensive.

That is why the NZ Red Cross, which already uses the Serval Mesh for the communication between the team members in the field, requested for an application such as Succinct Data. Succinct Data compresses the form in an effective way, halving the size of the file to as little as 8.2% of its initial size [6].

Figure 4 explains the architecture in which Succinct Data performs its work, and the whole process of compression, from the field polling by the NGO field teams, to the receipt of the form by the NGO deciders.

../pictures/other/succinct_data_overview%20(1).png

Figure : Architecture and whole process involving Succinct Data [14].

### Overview on Succinct Data application

|  |
| --- |
| How does Succinct Data work generally? What is the idea of the compression? Results of the compression? |

#### Succinct Data’s main purpose and processes.

Now that we know why Succinct Data was born and what is its main architecture, the network components involved and the used physical layer, we can focus on how it works at the Application layer.

As we know, SD takes XML records from Magpi (or any other forms provider such as ODK) and it compress it. But what is the global process besides the compression and what is exactly the nature of the compression? This is what this part will explain.

Actually, SD main objective is, as any other lossless compression system, to remove redundancy from the data to send (i.e. the XML record) at the sender side. The reversibility property of lossless compression will make the complete reconstruction of the data possible at the recipient side without any loss on the information. Assuming that both sides of the transmission have to keep this redundant information.

And it is the case, the only information actually unknown in the field is the value of each response in the form. All other information such as the order of the fields and the type of the fields can be completely implied. An analogy can be done with symmetric encryption: The compressed data is obtained by performing a series of reversible transformations that form the public key. Then, with the reverse operations, the other side reconstructs the data.

Figure 1 presents an overview on the operations made in the whole process.

../pictures/other/succinct_data_how_it_works.png

Figure : Overview of Succinct Data processes at the Application Layer

The following table explains the utility of each file involved in the Coding and Decoding Layer:

Table : Description of each file involved in the Coding and Decoding layer

|  |  |
| --- | --- |
| **File** | **Description** |
| Recipe | The recipe file is a shared file, obtained directly by extracting the information from the specification XHTML file, for which each line contains the following redundant information:  FieldName:Type:Minimum:Maximum:Precision,Select1,Select2,...,SelectN  “Select1, …, SelectN” is optional and corresponds to the N choices for multiple choices questions.  Combined with the information in the Stripped file, it allows the compression to a Succinct Data file. |
| Template | The template file is also a shared file and generated directly from the specification XHTML file.  It is a template for XML record files. It contains exactly the same information, except that the value of the field is replaced by “$field$”.  Thus, a field line looks like this: <field\_name>$field\_name$</field\_name>  It is used at the very end of the decoding layer to reconstruct the XML record file from the Stripped Data |
| Stripped Data | The Stripped Data file is extracted directly from the XML record. It’s an intermediary file before complete compression and contains only the salient information consisting in key value pairs. It has the following content in each line: field = value.  It is used with the recipes as an input for the compression. The compression only keeps the very salient information by applying range coding. |
| Succinct Data | The Succinct Data file is the binary file with the salient and lighter information to be sent via satellite or any other cheaper mean available (SMS, cellular, WiFi…). |

#### Elements on the compression

As we have seen in the previous part, the main objective of Succinct Data is to keep the salient information. This is done by implying all the fields redundant information and keeping that in both sides of the transmission while all that is transmitted is included in the very light Stripped file.

To go further and reduce the size again before transmission, we encode the fields in the stripped file with a range coder, which is very similar to an arithmetic coder [7]. This method is more effective than the very well-known Huffman coding. As a matter of fact, the compression has at least same results and often better results than the Huffman coding. One of the main characteristics of the range coding compared to the Huffman coding is that no context is involved and each message to be encrypted is treated as an independent unit, as the coder deducts the probability of the next characters to happen depending on the previous characters and on representative samples of the message. The Huffman coding is pre-conditioned as it affects a probability on each ASCII character to happen within the context of a language.

So, with the range coder, any multiple choice question is encoded to the least possible bits knowing the possible values. For example, a Boolean (“yes” or “no”) question is encoded on 1 bit because there are only 2 possible answers. A 3-way multiple choice question is encoded in 1,6 bits. Besides, any text string or unpredictable/unknown field is encoded as a text string and then compressed with the SMAC compressor [8] that also has an access to a range coder.

Finally, the results of the compression are quite good (c.f. Table 2) compared to what would do Gzip or Bzip2 on the same data. On 676 weed control ODK survey form records (Similar to Magpi forms), Succinct Data has the best performance with a compression rate of 7.2% (Size of the compressed data compared to the uncompressed form) [6].

Table : Comparative performance of Succinct Data, Gzip and Bzip2 for 676 weed control survey ODK form records

|  |  |  |  |
| --- | --- | --- | --- |
| Method | **Size (Bytes)** | **Reverse compression ratio** | **Price for the Iridium SBD in US$ (1.09 US$/kB)** |
| **Succinct Data** | 27254 | 7.20% | 29.7 |
| **gzip-9** | 54115 | 14.40% | 59.0 |
| **bzip-9** | 43605 | 11.60% | 47.5 |

# General specifications

|  |
| --- |
| The specifications part enters with a lower level of understanding in the Succinct Data application. It explains what is the main architecture of the application, what are the components and the links between them (the android application, the jni library, xhcreate, strip, rexml, compress, decompress). Then it goes into the reason for what we need improvements and the specifications of this need (What is expected). |

## The current Succinct Data application

|  |
| --- |
| Part description : The current Succinct Data application explained in low level |

### The android application

|  |
| --- |
| Explain how the android application generally works, Inputs, Outputs, and the link with the compression application. |

The SD application is already developed in Java (Android) with many packages including the reception of a Magpi form, the packetizing of the data and the transmission to the *InReach* satellite device. From the Magpi application, it is possible to export the data to SD when is is installed on the phone (c.f. Figure 6).

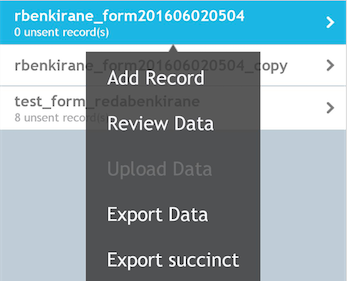


Figure : Exporting a form to SD from Magpi

A JNI (Java Native Interface) called jni.java is making the link with the C-written compression application SMAC library.

### The compression application

|  |
| --- |
| Explain how the compression application generally works for each part explained in the overview in the part I (xhcreate, strip, compress, decompress, rexml). For each component, try to explain in in a nice way (If possible include graphics). The reader should already have an understanding of the purpose of each part of the work and now we feed him with the detailed way it works for each part. |

#### Creation of recipes and templates out of a form specification

As we know, a recipe is a file where all specifications are written for each field of the form in a line that can be represented generally like that:

FieldName:Type:Minimum:Maximum:Precision,Select1,Select2,...,SelectN

A template is a file that has the same structure than an XML record but the values are replaced by a variable that will be replaced by the function that reconstructs the record from a template. Each field line of the template file has the following format:

<field\_name>$field\_name$</field\_name>

Please refer to the **appendices** to find a full example of a survey’s recipe and template files.

So, basically, to generate the template and recipe, we use an XML parser on the XHTML file that can be assimilated to finite a state machine and that calls appropriate functions for each state (cf. Figure 7). These functions are located in the xhtml2recipe.c in the SMAC library.

../pictures/figures/xhcreate_main_statemachine.png

Figure : main finite State Machine of the XML parser

To generate the template, we use a Boolean xhtml\_in\_instance that we set to true when we meet a new <data> tag (or data element) and to false when we meet the end </data> tag. Thus, when the Boolean is true, it means that we are between the data tags. We write all the fields between the data tag into the template file in this way:

<field >$field$</field>

To generate the recipe, we wait until we meet a <bind> tag and write the type and constraint (minimum, maximum) attribute. For multiple choices questions, we also write the choices that are in the <xf:input> tag.

#### Creation of the stripped file out of a record

The xml2stripped function located in the rexml.c file can be assimilated to a State Machine (I reverse engineered the existing function in the finite State Machine in Figure 8). The aim here is to create a stripped file out of an XML record and to

The main template for a transition is Trigger [Guard] / Effect. The trigger is the event that causes the transition, the guard is a prerequisite condition and the effect is an action done as a result of the transition.

../pictures/figures/draw2_statemachine_xml2stripped%20(4).png

Figure : detailed finite State Machine of the Stripped2Xml function

So, basically the path in the file is that we read the file character by character. Usually we start by reading the ‘<’ character (So we’re inside a tag) and thus we accumulate the name of the tag.

Once we read ‘>’ we go into the state “Tag finished” and set the interresting\_tag and state to 0. If it’s not an end tag, it means that what follows is information about the value of the tag so we set interesting\_tag to 1.

Then, we have state = 0 and interesting\_tag = 1 so we go into “Accumulate value” state and we keep reading the characters and registering it in a buffer for the value.

Once we meet again a ‘<’ character, if the tag before was not an end tag then the interesting\_tag variable has to be equal to 1 and we accumulated the value so value\_len > 0. We go consequently to the state where we write tag = value in the stripped file, then we go back and set state = 1 and interesting\_tag = 0.

If we meet a ‘\n’ or ‘\r’ character (i.e. back to line), we will then meet a new ‘<’ character and we just restart the same process again, writing “field = value” lines in the Stripped file buffer until we have reached the end of the file.

#### Regenerating the record from a stripped file

[ explain it in cleat steps and eventually draw a detailed state machine of it or a really simple state machine ]

#### Compressing a stripped file to a succinct data file

In the recipe\_compress\_file function, to compress a Stripped file to Succinct Data, we firstly read the stripped file and fine the form ID to use in it so we know which recipe to use (indeed, the name of the recipe file is formatted as formID.recipe).

Then, we load the right recipe and we call the recipe\_compress function and in this function we do the following:

* We write the formhash of the recipe (which corresponds basically to the memory address of the recipe) so the decompressing function can know which recipe was used in the first line of the SD file.
* We parse the Stripped file’s “key = value” lines into 2 arrays of keys (i.e. fields names) and values.
* We loop for each key in the keys array and when we find a corresponding field name in the recipe file, we encode it and copy the resulting bits in the output.

#### Decompressing a succinct data file to a stripped file

Decompressing is regenerating a Stripped file from a SD file and it’s done in the recipe\_decompress\_file function.

(…)

## Specification of the needs

|  |
| --- |
| There we go into the reasons for what we need improvements and the specifications of this need (What is expected). |

### Context and interests

|  |
| --- |
| The context behind the need |

Succinct Data is working nicely and provide support for simple Magpi forms. What we can call simple Magpi forms are those with as much as we want of questions of type: plain text input, multiple choice questions (radio button), check box, integers, date or GPS coordinates.

However, the field teams of the NGO may have the need to use another special type of question in the form, which is a Subform.

A Subform is basically a form nested in the original form. It may be needed if the team is for example polling a village containing 3 houses: A, B and C. To make every information fit into one form, …

SD users may have the need to use sub-forms in case of a survey that will be applied on an element that contains sub-elements. For example, in the case of a fire in a remote village containing 3 houses: A, B and C, the form will consist on general questions about the village (Village location, Village name, Number of households affected, …), then we can have sub-forms for each household in the village (House location, number of people damaged, list of materials damaged, …). The answers for a form will be the same for every sub-elements of the element (for example: The same answers for every house of the village). So, a solution for this issue that was implemented by Magpi is the sub-forms. That is why the support of sub-forms by the Succinct Data application is essential.

The SD application doesn’t manage the sub-forms. What we would like to do is to modify the code in the SD application so it is able to firstly recognize the tags and the elements of a sub-form and then process it correctly by transforming it into stripped data. Then, the code that transforms stripped data to binary data has to recognize the new stripped data structure and be able to transform it into binary data.

We want also that the method that creates a recipe from a form specification recognises the sub-form specification and calls itself recursively to create a recipe file for the sub-form.

(…)

### Functional specifications

|  |
| --- |
| Functional or technical specifications given for the solution : What are the functionalities to implement, the expected results |

# Design of the solution

|  |
| --- |
| In this part, we enter into the design explanation of the solution. After an introduction on the Serval project and on Succinct Data and the specifications of the needs, this part will explain the approach to implement the solution, the main algorithms, if possible state diagrams, any good graphical way to represent the design … |

## Overview

|  |
| --- |
| Overview of the solution : How will the solution be designed generally knowing that we have a legacy code and knowing how the legacy code works ? |

## Forms to recipes and templates

|  |
| --- |
| Part description : Design of the creation of recipes and templates from forms with subforms |

## Records to stripped data conversion

|  |
| --- |
| Part description : Design of the creation of stripped data from records with subforms and regeneration of xml file (reverse process). |

### XML record to stripped data

|  |
| --- |
| Part description : Design of the creation of strippeed data from records with subforms |

### Stripped data to XML record

|  |
| --- |
| Part description : Design of the regeneration of the original xml file from stripped data with subforms |

## Stripped data to succinct data conversion

|  |
| --- |
| Part description : Design of the compression of stripped data with subforms and reverse process (decompression) |

### Stripped data to SD

|  |
| --- |
| Part description: Compression |

### SD to stripped data

|  |
| --- |
| Part description: Decompression |

# Materials and methods

|  |
| --- |
| Description of the environment, the tools, and the methods for understanding the code, coding, testing. Also, it’s possible to give a planning of what was done from the beginning to the end |

Keywords: Eclipse IDE, Shell, Script, Tests, “Structural” Analysis (Compression app, JAVA…), Test Cases, Testing each function independently then the whole, Other tools: State machine, Layers, Architecture with draw.io …

# Implementation details

|  |
| --- |
| Part description: After presenting the design of the solution, we present here the implementation details for each part: What are the key variables and structures of data? How are they manipulated? What are the states and what is done for each state? … For each implementation part we can talk about some difficulties and how did we solve that. |

## Forms to recipes and templates

|  |
| --- |
| Part description : Recipe xhcreate |

## Records to stripped data conversion

|  |
| --- |
| Part description : Recipe strip and rexml |

### XML record to stripped data

|  |
| --- |
| Part description : Recipe strip |

### Stripped data to XML record

|  |
| --- |
| Part description : Recipe rexml |

## Stripped data to succinct data conversion

|  |
| --- |
| Part description : Recipe compress and decompress |

### Stripped data to SD

|  |
| --- |
| Part description : Recipe compress |

### SD to stripped data

|  |
| --- |
| Part description : Recipe decompress |

# Results

## Overview

|  |
| --- |
| Part description: Give an overview on the results and on what is achieved. What were the expectations, what did I achieve, what remains to do… |

## Test cases

|  |
| --- |
| Part description: Here give the test cases ran against the code to check its validity (either manual or automatic tests…). |

## Impacts / Discussion

|  |
| --- |
| Part description: What are the impacts of what I achieved |

# Conclusion

|  |
| --- |
| Part description: Talk about personal experience, path, general difficulties, achievements, sum up all the process and give a personal reflection about the work… |

# Further work

|  |
| --- |
| Part description: Further improvements |

# Appendices

## Appendix 1: Survey example’s XML record file

<?xml version="1.0" encoding="UTF-8"?>

<form>

<meta>

<email>kartoffel</email>

<password>potato</password>

<formid>27251</formid>

<lastsubmittime>2015-04-14 12:45:38</lastsubmittime>

<endrecordtime>2015-04-14 12:45:36</endrecordtime>

<startrecordtime>2015-04-14 12:40:35</startrecordtime>

<version>2.5</version>

<uuid>606044c5ae181046-1429004435022</uuid>

<geotag>

<longitude>0</longitude>

<latitude>0</latitude>

</geotag>

</meta>

<data>

<interviewer>briss</interviewer>

<clusterid>2</clusterid>

<hhid>7</hhid>

<gps>~</gps>

<hhinterviewed\_yn>yes</hhinterviewed\_yn>

<consent>yes</consent>

<interview\_start\_time>1</interview\_start\_time>

<name\_hhhead>omiondi</name\_hhhead>

<head\_hh\_school\_yn>no</head\_hh\_school\_yn>

<head\_hh\_educ\_highest>Do not know</head\_hh\_educ\_highest>

<num\_km\_to\_facility>2</num\_km\_to\_facility>

</data>

</form>

## Appendix 2: Survey example’s XHTML form specification file

<html xmlns=http://www.w3.org/1999/xhtml (…) >

<head>

<meta>

<userid/>

<deviceid/>

<timestartrecorded/>

<timeendrecorded/>

</meta>

<title>1\_KEN2surv\_house\_v15\_Copy\_2013\_01\_30\_18\_58\_05</title>

<xf:model dd:formid="27251" dd:geotag="on" id="1\_KEN2surv">

<xf:instance>

<data>

<interviewer/>

<clusterid/>

***(…)***

</data>

</xf:instance>

<bind id="interviewer" nodeset="/data/interviewer"

required="true()" type="xsd:string"/>

<bind

constraint="not( /data/clusterid &lt; 1 or /data/clusterid > 30 )"

id="clusterid"

jr:constraintMsg="Your answer must be between 1 and 30"

nodeset="/data/clusterid" required="true()" type="xsd:integer"/>

***(…)***

</xf:model>

</head>

<body>

<xf:input bind="interviewer">

<xf:label>Interviewer name</xf:label>

<hint/>

</xf:input>

<xf:input bind="clusterid">

<xf:label>Cluster number. REMEMBER TO CHANGE THE CLUSTER NUMBER AT THE BEGINNING OF EACH DAY. </xf:label>

<hint/>

</xf:input>

***(…)***

</body>

</html>

## Appendix 3: Survey example’s Stripped file

meta=

email=kartoffel

password=potato

formid=27251

lastsubmittime=2015-04-14 12:45:38

endrecordtime=2015-04-14 12:45:36

startrecordtime=2015-04-14 12:40:35

version=2.5

uuid=606044c5ae181046-1429004435022

geotag=

longitude=0

latitude=0

data=

interviewer=briss

clusterid=2

hhid=7

hhinterviewed\_yn=yes

consent=yes

interview\_start\_time=1

name\_hhhead=omiondi

head\_hh\_school\_yn=no

head\_hh\_educ\_highest=Do not know

num\_km\_to\_facility=2

## Appendix 4: Survey example’s Template file

<?xml version="1.0" encoding="UTF-8"?>

<form>

<meta>

<userid>$userid$</userid>

<accesstoken>$accesstoken$</accesstoken>

<formid>$formid$</formid>

<lastsubmittime>$lastsubmittime$</lastsubmittime>

<endrecordtime>$endrecordtime$</endrecordtime>

<startrecordtime>$startrecordtime$</startrecordtime>

<version>$version$</version>

<uuid>$uuid$</uuid>

<geotag>

<longitude>$longitude$</longitude>

<latitude>$latitude$</latitude>

</geotag>

</meta>

<data>

<interviewer>$interviewer$</interviewer>

<clusterid>$clusterid$</clusterid>

<hhid>$hhid$</hhid>

<gps>$gps$</gps>

<hhinterviewed\_yn>$hhinterviewed\_yn$</hhinterviewed\_yn>

<consent>$consent$</consent>

<interview\_start\_time>

$interview\_start\_time$ </interview\_start\_time>

<name\_hhhead>$name\_hhhead$</name\_hhhead>

<head\_hh\_school\_yn>

$head\_hh\_school\_yn$

</head\_hh\_school\_yn>

<head\_hh\_educ\_highest>

$head\_hh\_educ\_highest$

</head\_hh\_educ\_highest>

<num\_km\_to\_facility>

$num\_km\_to\_facility$

</num\_km\_to\_facility>

</data>

</form>

## Appendix 5: Survey example’s Recipe file

userid:string:0:0:0

accesstoken:string:0:0:0

formid:string:0:0:0

lastsubmittime:magpitimestamp:0:0:0

endrecordtime:magpitimestamp:0:0:0

startrecordtime:magpitimestamp:0:0:0

version:string:0:0:0

uuid:magpiuuid:0:0:0

latitude:float:-90:90:0

longitude:float:-200:200:0

interviewer:string:0:0:0

clusterid:integer:1:30:0

hhid:integer:1:15:0

gps:geopoint:0:0:0

interview\_start\_time:integer:0:2400:0

name\_hhhead:string:0:0:0

num\_km\_to\_facility:integer:0:999:0

hhinterviewed\_yn:enum:0:0:0:yes,no

consent:enum:0:0:0:yes,no

head\_hh\_school\_yn:enum:0:0:0:yes,no,do not know

head\_hh\_educ\_highest:enum:0:0:0:Primary,Secondary,Higher,Do not know

# Bibliography

[1] P. Gardner-Stephen, A. Bettison, R. Challans, J. Hampton, J. Lakeman and C. Wallis. (2013) "Improving Compression of Short Messages", International Journal of Communications, Network and System Sciences, Vol. 6 No. 12, pp. 497-504.

[2] P. Gardner-Stephen, J. Lakeman, R. Challans, C. Wallis, A. Stulman and Y. Haddad. (2012) "MeshMS: Ad Hoc Data Transfer within Mesh Networks", International Journal of Communications, Network and System Sciences, vol. 5, no. 8, pp. 496-504.

[3] P. Gardner-Stephen et al. (2014) “Succinct Data: Extreme Compression for ODK Forms”, Global Humanitarian Technology Conference, pp. 70-77.

[4] P. Gardner-Stephen. (2011) “The Serval Project: Practical Wireless Ad-Hoc Mobile Telecommunications”

[5] P. Gardner-Stephen et al. (2013) “The serval mesh: A platform for resilient communications in disaster & crisis”, Global Humanitarian Technology Conference, pp. 162-166.

[6] P. Gardner-Stephen, R. Challans, A. Bettison., C. Wallis, J. Hampton and J. Lakeman. (2014) “Survey acquisition management tools for Android.” Internet: <https://github.com/servalproject/survey-acquisition->management [May 2016].

[7] P. Gardner-Stephen, R. Challans, A. Bettison., C. Wallis, J. Hampton and J. Lakeman (2014) “Short strings compression library” Internet: <https://github.com/servalproject/smac/tree/succinctdata>[May 2016].

[8] Magpi. “Advanced Mobile Data, Messaging, and Visualization anywhere on any device” Internet: <http://home.magpi.com/>[May 2016]

[9] P. Gardner-Stephen et al. “The Serval Project Wiki” Internet: <http://developer.servalproject.org/> [May 2016]

[10] P. Gardner-Stephen et al. “The Serval Project website” Internet: <http://www.servalproject.org/> [May 2016]

Other links/interesting :

<http://www.geekinsider.com/serval-mesh-chat-to-other-android-phones-without-network/>