**DOCUMENTATION**

**Summer Internship 2012**

**Porting Of proxyMITY**

**XMLEditorGUI**

Prepared by,

**Bhairavi.K**

Anna University, Madras Institute of Technology

Assisted by,

**Mr. Parag Tiwari**

**Ms. Sreelekha. S**

Under the Guidance of,

**Prof D.B.Phatak**



in the Department of Computer Science,

Indian Institute of Technology (Bombay)

during the period from May 2012 to July 2012.

**Acknowledgment**

I am over helmed in all humbleness and gratefulness to acknowledge to all those who have helped me to put my ideas and assigned work, well above the level of simplicity and into something concrete.

I thank Prof. Deepak Phatak whole heartedly for selecting me a part of his valuable project, constantly motivating for doing better and showing complete confidence in my work.

I am indebted to my Project Manager Mr. Parag Tiwari for his constant help and support in light of this project. I am very thankful to my mentor Ms.Sreelekha Nair for her valuable suggestions, guidance and encouragement. Also in the process, I learnt lot other technical and non-technical things from them.

I would like to thank Mr.Bikas Chhatri for making my stay here as summer interns comfortable and for all his administrative help.

Finally I also like to thank all other colleagues working in different projects under Prof.D.B Phatak for helping me at small problems as well as critical junctures.

**DECLARATION**

I declare that this written submission represents my own ideas and original sources have been adequately referenced and cited wherever required. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in the submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Date: 02-07-12 Bhairavi.K

**ABSTRACT**

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable

The design goals of XML emphasize simplicity, generality, and usability over the Internet. It is a textual data format with strong support via Unicode for the languages of the world. Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures, for example in web services.

XML can also be used for data-exchange between different computer .It is free and is easier to read especially when viewing relational data. It was really designed to be a platform-neutral way of sharing of structured data across different information systems, particularly via the internet.

XML is highly flexible as there is no fixed set of tags. New tags can be created as they are needed. Unlike traditional databases, data records does not require schemas. XML documents can be stored without any pre-definitions, because they contain meta data in the form of tags and attributes.

However, the creation and editing of XML files is not as simple as an interactive RDBMS. This hinders the usage of XML for storing large data. . In order to overcome this difficulty, **“XMLEditorGUI”** application has been designed.

This application is available in two versions namely the android and the desktop and it guarantees to simplify the creation, editing, deletion, parsing and manipulation of XML files. It eliminates the need for manual typing, and thus reduces the error rate and improves efficiency, hence saving time.

**Contents**

**Chapters**

**1 Introduction………………………………………………………………………….. 6**

* 1. Background ………………………………………………………………… 6
  2. Scope………………………………………………………………………... 7
  3. Design goals……………………………………………………………….... 7
  4. Abbreviations and Meanings………………………………………………... 7

**2 Project Plan ………………………………………………………………………….. 8**

2.1 Client Detail…………………………………………………………………. 9

2.2 Resource requirement……………………………………………………….. 9

* + 1. H/W requirement and S/W requirement………………………... 9

**3 Designs and Implementation………………………………………………………….9**

3.1 Project Description………………………………………………………….. 9

3.2 Functional Flow……………………………………………………………...21

3.2.1 Algorithm for Parsing………………………………………………21

3.2.1 Algorithm for Editing………………………………………………21

3.2.1 Algorithm for Update………………………………………………23

3.2.1 Algorithm for Deletion……………………………………………..23

3.3 High Level Design Document………………………………………………..24

3.3.1 Use Case Diagram………………………………………………….24

**4 Summaries……………………………………………..………………………………28**

4.1 Summary………………………………………………………………….. .29

* 1. Further Enhancements……………………………………………………29
     1. Improvement of GUI. ……………………………………………29
     2. Searching for values in XML files………………………………..29
     3. Enabling the creation of tags with multiple attributes……………29
     4. Validation of individual tags……………………………………..29
     5. Desktop Publication……………………………………………….29

4.3 References and Bibliography………………………………………………….29

**5 Conclusions…………………………………………………………………………......30**

**1. Introduction**

XML is a standard for describing how information is structured. In recent years it has gained a lot of popularity in the IT industry, mostly because of its ease of use. XML provides a standard method to access information, making it easier for applications and devices of all kinds to use, store, transmit, and display data.

XML has no dependencies on the programming language type, operating system platform, databases type, or even human languages and that makes it the perfect data format to allow programmers to build cross platform applications. This makes it much easier to move structured information from computers to computer or from one program to another.

However awesome XML is, there are some drawbacks which have hindered it from gaining widespread use since its inception. Its biggest drawback is the lack of adequate processing applications. The syntax is redundant or large relative to binary representations of similar data. This redundancy may eat up the time and effort in places where XML files are used heavily.

These days, parsing algorithms and tools continue to improve over time as more and more applications see the long-term benefits of migrating their data to XML. Efficient Parsing and manipulation will make the backend part of XML simpler and efficient.

The Generalised XML edit module of Porting of proxyMITY was developed with an aim to enable the user to create and edit an XML file in an easy and time-conscious way in both android as well as desktop platform. XMLEditorGUI removes the need for manual typing of the data in XML files, as given a structure and the data, the application creates the XML file for the user, which can further be edited both manually as well as through the application.

**1.1 Background**

The project Porting of proxyMITY extends the features of the new Multimedia Integration tool called proxyMITY that was developed in IIT-B in May 2011. proxyMITY is a software that helps to create dynamic, rich-media lectures which aids to better learning and profound understanding .

The software of proxyMITY enables us to share the lectures with others through the web, through institute intranet, or through a third party e-service provider. Thus, proxyMITY is a standalone application that lets users take the slides and synchronize them with video already recorded on the computer.

The data for this synchronisation, including the Details of the Lecture, Slide name, Professor Name, Duration, Start and End Time, etc..., are stored in XML files. In order to ease the process of creating such XML files, where the structure is repetitive, causing redundancy of work and consuming enormous time, and with an eye on the development of Aakash tablet, this application has been designed.

Further, XMLEditorGUI is ensured to work for any kind of XML file, required for any kind of application, which adds more value, with the charm of flexibility. Thus, this application guarantees the creation, reading, parsing, editing and writing of a majority of XML files, that are subject to constraints that are explained later in this document. The main goal of XMLEditorGUI tool is to edit the xml files for the corresponding video lectures prepared for delivering to the students both through Aakash tablet and through other media.

**1.2 Scope**

The android version of this XMLEditorGUI is compatible with Android 2.2 and above. The desktop version is compatible with all operating systems that support JVM. This is a standalone application that lets you to parse and manipulate the XML files that are already present. New XML files can also be created and can be manipulated both manually as well as from the application.

**1.3 Design Goals**

The final product is a rich application with two versions: Android and Desktop publishing. The product can be distributed to the people so that anyone who wants to manipulate XML files can use it.

The following goals are met by the XMLEditorGUI:

* Allow users to choose an existing file using a File Chooser.
* Permit users to create a blank XML file and save it in SD card.
* Parse the XML file using DOM Parser.
* Store the structure and content of an XML file in a set of array lists.
* Create a new XML file from the scratch.
* Add tags to the existing XML file, maintaining the same syntax and structure.
* View the contents and structure of an XML file.
* Delete existing tags and entries from an XML file.
* Edit the XML tags and entries and update the same to the file.
* Write an XML file using the data from the array list that contain the structure and data.
* Create a unique tag.
* Create new tags with options: Children/ Attribute and Children

**1.4 Abbreviations and Meanings:**

|  |  |
| --- | --- |
| XML | Extensible Mark-up Language |
| DOM | Document Object Model |
| NodeList | org.w3c.dom.NodeList |
| Element | org.w3c.dom.Element |
| Document | org.w3c.dom.Document |
| Data | Set of *ArrayList*s holding the content and structure of the parsed XML file |
| elementsList | *ArrayList* of Strings containing all the tag names that have either an attribute or a node value. |
| arrayList | *ArrayList* of *ArrayList* storing the values of the tags sequentially. The index of the tags’ values matches the index of the tag themselves in *elementsList.* |
| parentsList | *ArrayList* of Strings that contains all the tag names which have child nodes. |
| childTags | *ArrayList* of *Arraylist* that stores the names of all the child tags corresponding to the tag in the *parentsList* of same index. |
| attributeIndex | *ArrayList* of Integer that stores the index (within the *elementsList*) of the tags that have attributes. This is used to distinguish the tags that have node value and those that have attributes at the time of file writing. |
| Unique | The tag appears only once in the entire file. |
| MinimumIntegerValue | -2^31 |
| MaximumIntegerValue | 2^31-1. |

**2. Project Plan**

* 1. All the functionalities were jotted down.
  2. The DOM parsing of the file and the extraction of the content and structure from the XML file were done using java.
  3. The file viewing, editing, adding and file writing modules were implemented in java.
  4. The Android GUI was developed.
  5. The parse, extract and write java modules were included as classes.
  6. The edit, view and add modules were changed appropriate to the GUI used.
  7. Code was included for deleting values.
  8. Codes were added for creating a new XML file.
  9. Layouts and colours were improved.
  10. Code was added for choosing existing file and creating new file.

**2.1 Client Detail**

It is assumed that the client is well informed about the XML files and is expected not to make structural errors while entering the data, as validation cannot be performed for tags that are created during runtime.

**2.2 Resource requirement**

**2.2.1 H/W requirement and S/W requirement**

* Android mobile with the following specifications:

1. Operating System : 2.2 or above
2. File Chooser .
3. Memory for application: 60KB
4. DOM parser.

* Any desktop that has JVM (Platform Neutral)

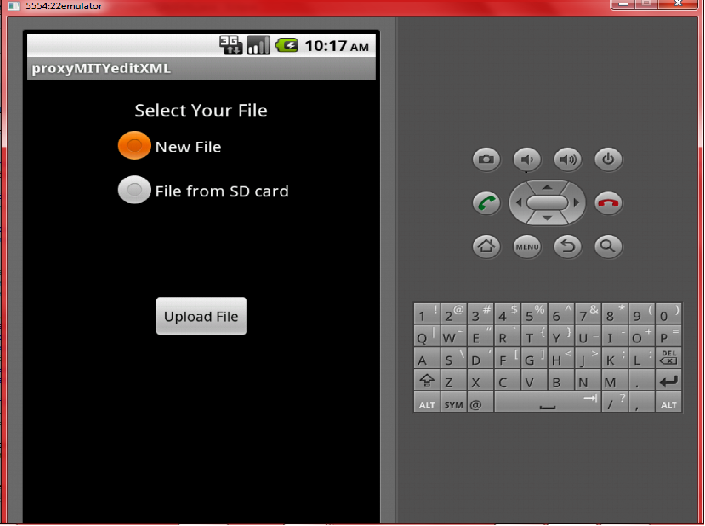
**3. Design and Implementation**

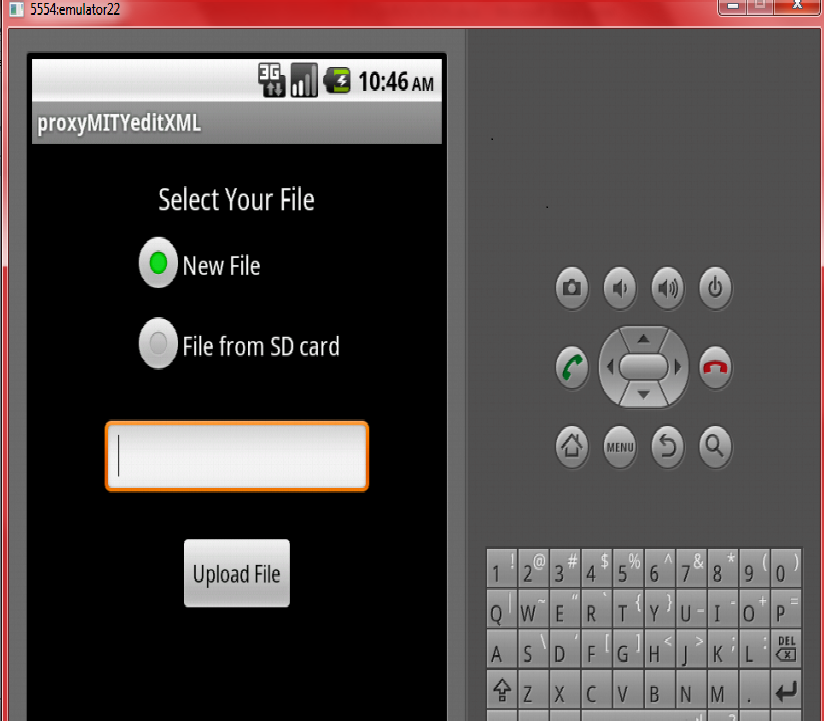
**3.1 Project Description**

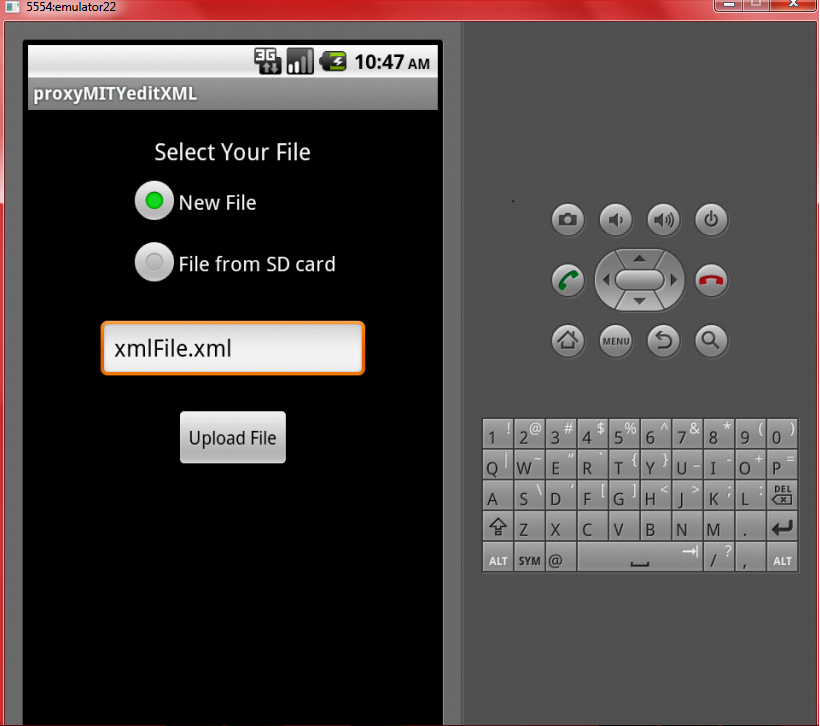
**Step 1: Choose File**

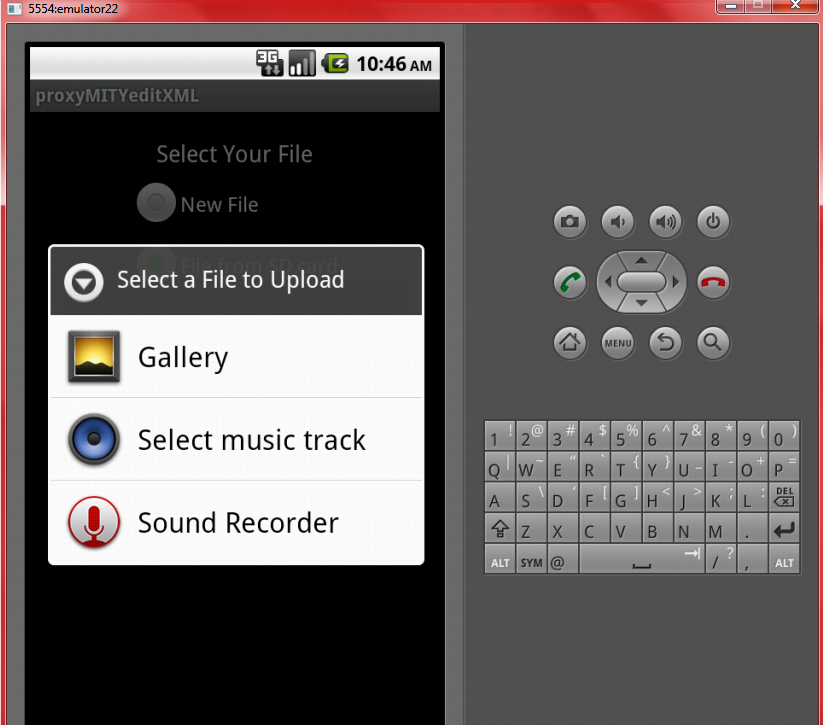
The first work done by XMLEditorGUI is to get the input file from the user. The input file may be an existing one or a blank file. If the user wants to open an existing file, a FileChooser is displayed which ensures the users to choose their XML file. If a new file is to be created, a textbox is displayed, wherein user can enter the name of the file to be created.

**Constraints and Assumptions**

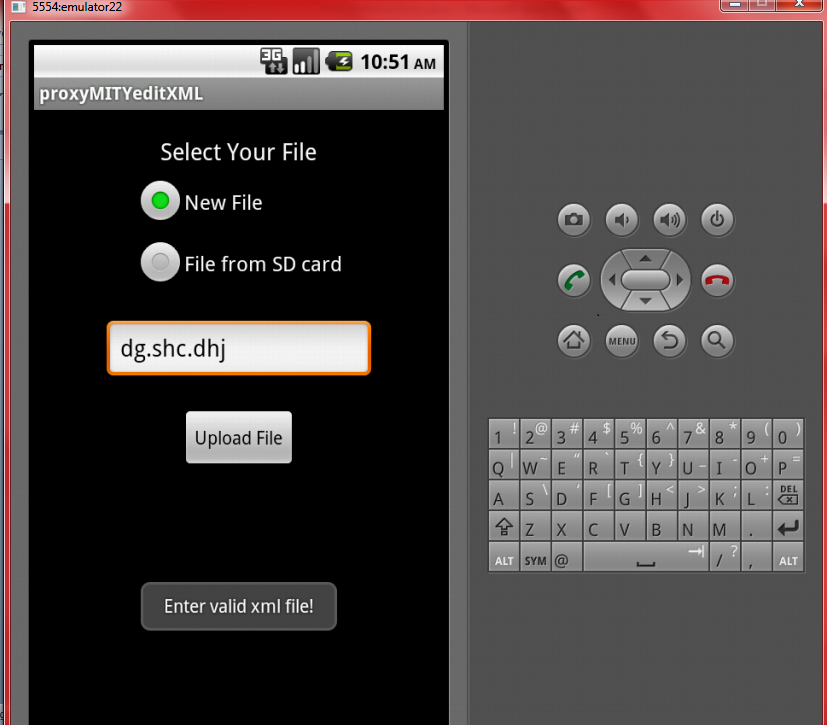
* The user can create/choose only XML files,i.e the files must have an extension “.xml”
* In the android version, the user can choose files only from SD card. i.e the files in the path /mnt/sdcard. In desktop, the file is chosen from the same folder as the application.
* ****The files created by the application are stored in the SD card in android version**.** In desktop version, the created files are stored in the same folder as the application

**Fig 2 : Create New File**

**Fig 3: Create New File**

****

**Fig 4 : Upload File**

****

**Fig 5 : Validation of File**

**Step 2 : Parse XML file**

If the user chooses an existing file, parsing is done. Dom Parser is used and all the nodes that occur in the file are stored in a *NodeList.* From this *NodeList, element* after *element* is inspected.

If the *element* is found to have childnodes and has not been previously added to the *parentsList,* it is added to the *parentsList.* All the children of this tag, except the text, are added to the *childTags.*

If the *element* has node value or attributes and has not been added in *elementsList,* it is added. If it has atrribute, the index of this tag is stored in the *attributeIndex.* All the node values/ attribute values are stored in *arrayList.*

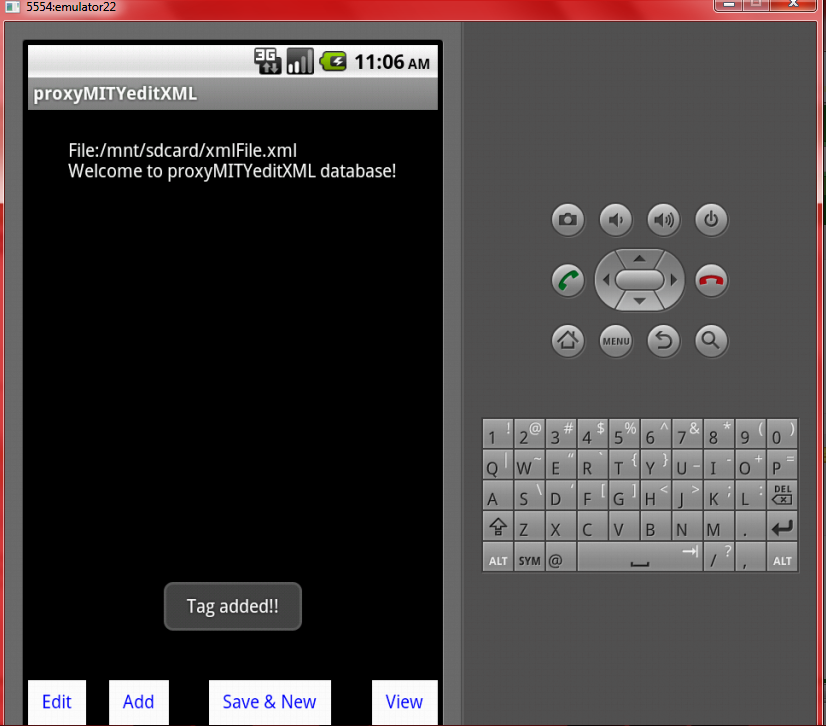
**Constraints and Assumptions:**

* An *element* cannot have both attributes as well as node value at the same time.
* Only up to one attribute can be added for any tag.
* The name of the attribute is always “name”.

**Step 3: Display options**

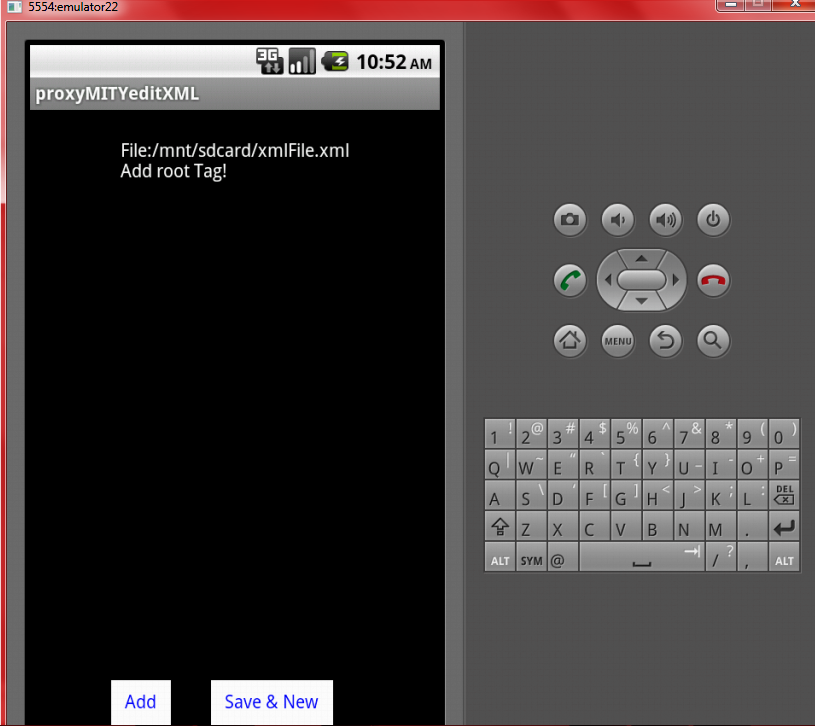
If the file exits, and has been parsed successfully, the options that are displayed are

* Edit Values.
* Add Values/ New Tag.
* View tags and values.
* Save and Open a new XML file

**Fig 6 : Display options for existing File**

If the file does not exist, the options that are displayed are,

* Add root tag.
* Save and Open a new XML file.

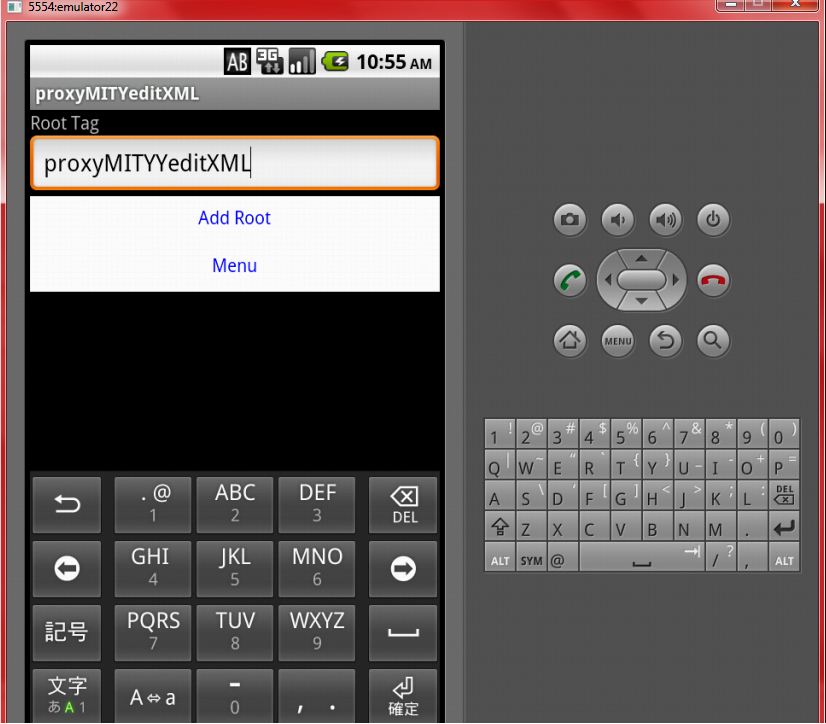
**Fig 7 : Display Options for empty File**

**Step 4: Add root tag**

If the file is created newly, the user is prompted to enter the root tag.

**Constraints and Assumptions:**

* Every XML file must have a root tag.

**Fig 8 : Add root tag**

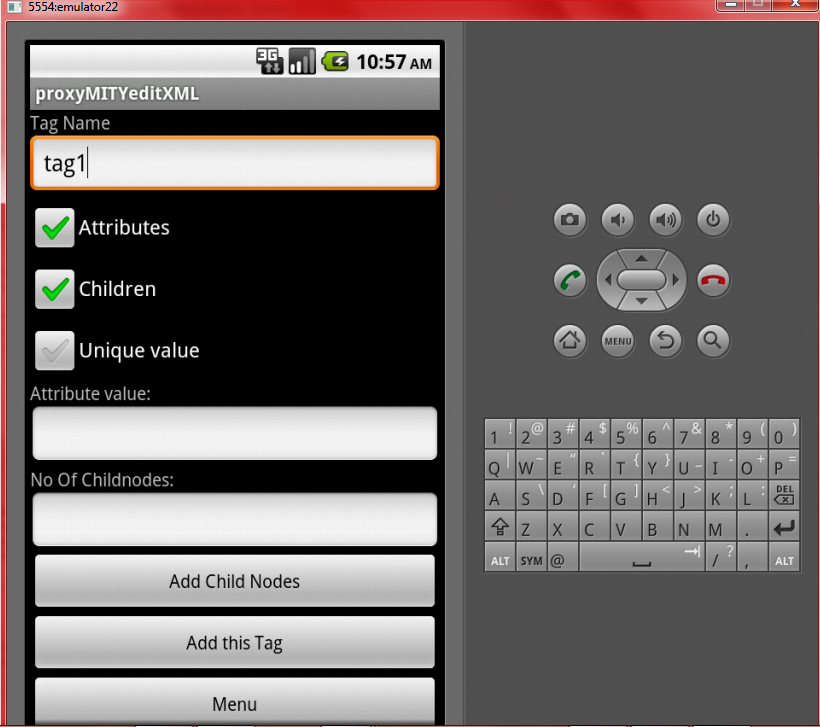
**Step 5: Add New Tag**

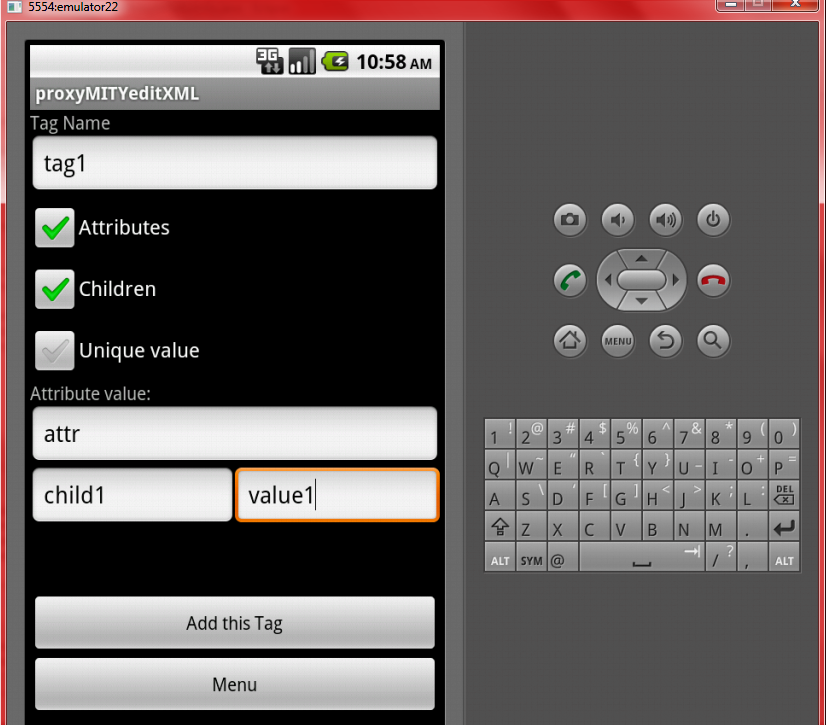
While adding a new tag, the tag name is got from the user. The Following options are available.

* Tag with 1 or more children
* Tag with 1 or more children and attributes
* Tag with 1 or more children (*Unique*)
* Tag with 1 or more children and attributes (*Unique*)

**Constraints and Assumptions:**

* If a tag has attribute, it must have atleast one child.
* Tags with node value alone cannot be added here. They must be added through the edit module.
* Only up to one attribute can be added for any tag.
* The name of the attribute is always “name”.
* The *Unique* tags are identified by the suffix “\_DONOTREPEAT”.

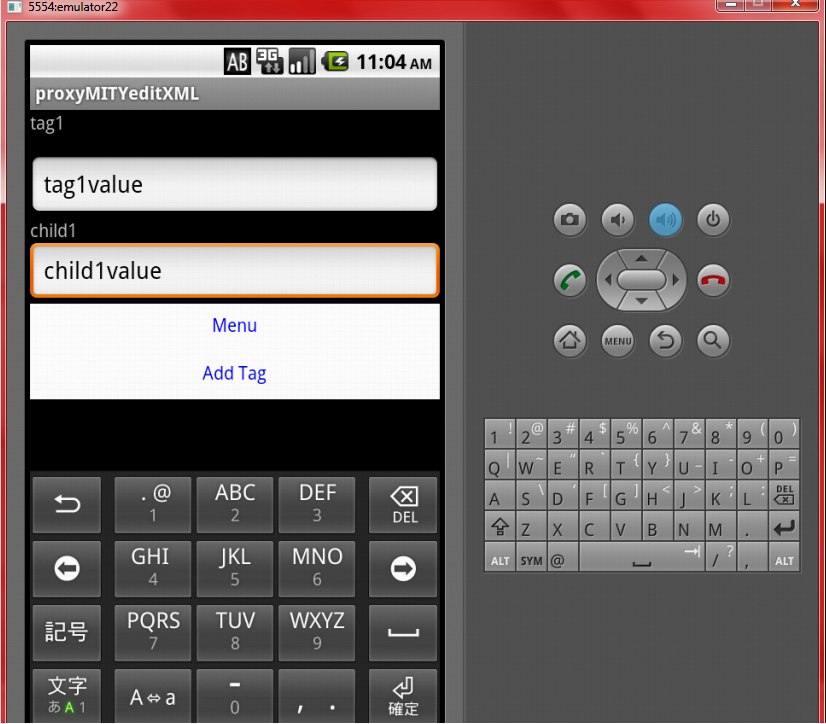
**Fig 9 : Add New Tag**

**Fig 10 : Add New Tag**

**Step 6: Add values to existing tag**

The user is presented with all tags that is not *unique* and the structure of which is available. As the user chooses the tag, the individual structure is displayed where the user can enter the node values/attributes/ child tags of the selected tag.

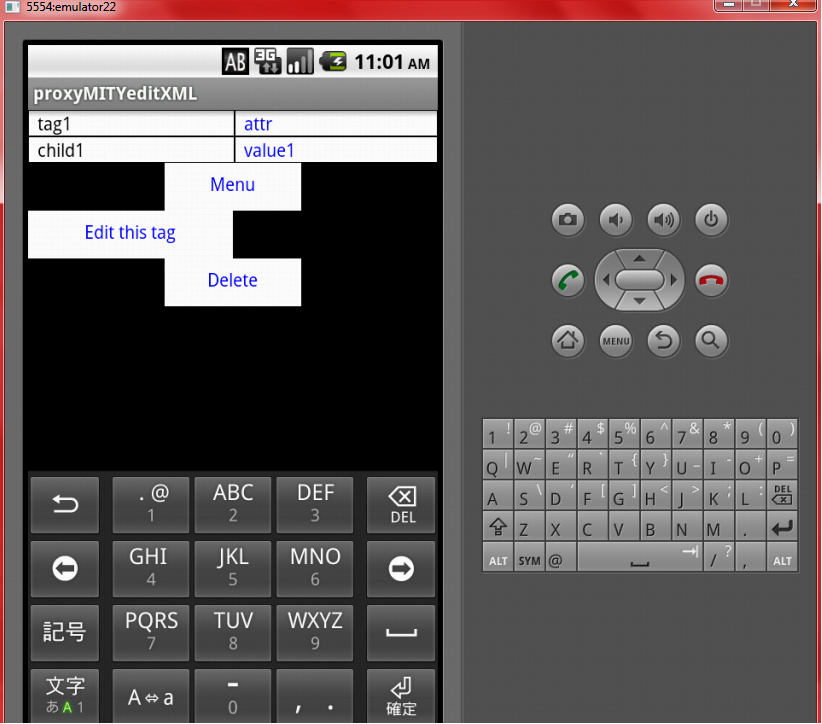
It is checked if the tag itself has attributes, if so, a textbox is displayed for the same. This is done by checking if the tag is present in *elementsList*. All the child tag names are fetched from the *childTags* andtextboxes are provided for each tag. The values that are entered are placed in a Queue, along with the tag names, and updatated to the *data.*

**Fig 11 : Add values to existing tag**

**Step 7: View values of existing tag**

The user is presented with all tags the structure of which is available. As the user chooses the tag, the individual structure is displayed where the user can view the node values/attributes/ child tags of the first entry of the selected tag. The user can move between tags using the next, previous, first, last buttons.

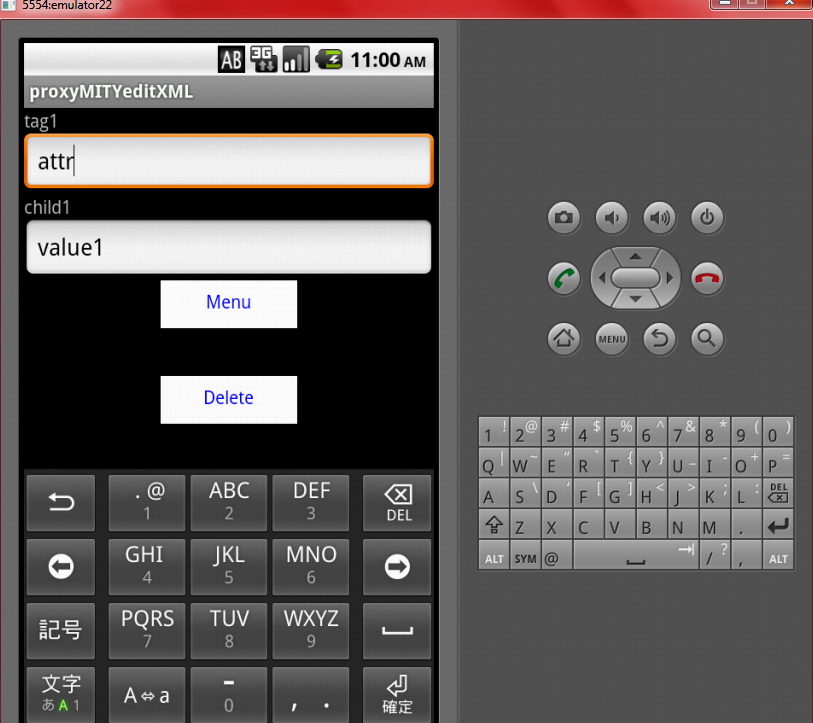
It is checked if the tag itself has attributes, if so, its value is displayed. This is done by checking if the tag is present in *elementsList*. All the child tag names are fetched from the *childTags* and are displayed intextboxes. Options are provided for editing the current entry in the current tag. Similarly, the user can delete the current entry of the selected tag.

**Fig 12 : View Values of existing tag**

**Step 8: Edit values of existing tag**

The user is presented with all tags the structure of which is available. As the user chooses the tag, the individual structure is displayed where the user can edit the node values/attributes/ child tags of first entry of the selected tag. The user can move between tags using the next, previous, first, last buttons.

It is checked if the tag itself has attributes, if so, its value is displayed in a edit box. This is done by checking if the tag is present in *elementsList*. All the child tag names are fetched from the *childTags* and are displayed inedit boxes. All the changes made by the user are placed in Queues, with the tag name and the tag value, and when the user is done, it is updated in the *data.*

****Fig 13 : Edit Values of Exiting Tag**

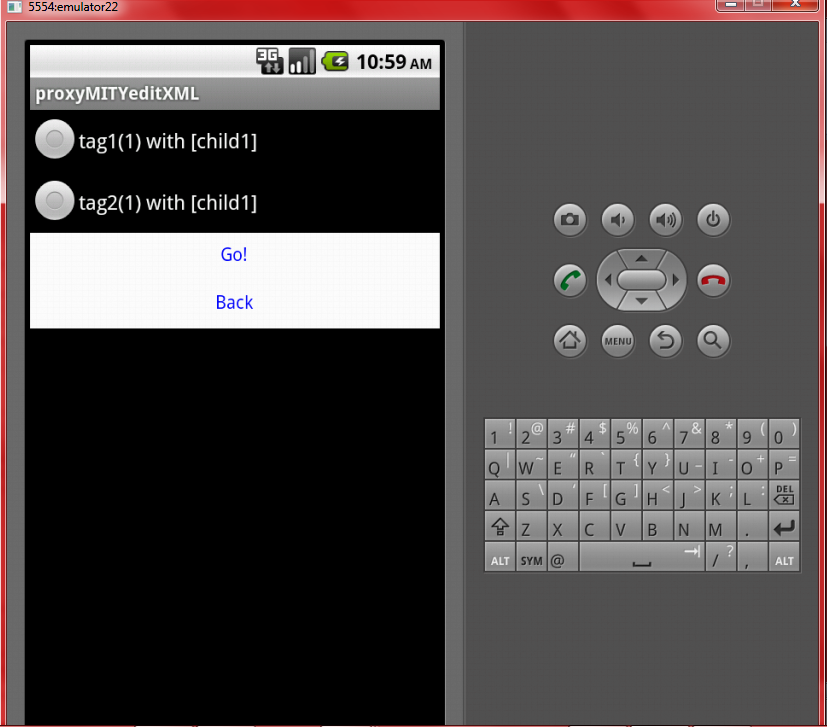
**Step 9: Delete values of existing tag**

Deletion can be done while viewing as well as editing the tag. As the user decides to delete a particular entry, all the corresponding values in the *data*  are removed. If next entry is available, the values of the next entry are displayed. If the user is deleting the last entry, it then displays the previous entry. If the user is deleting the only available entry, the tag is deleted and the main menu is displayed.

**Step 10: View Structure of the file.**

The structure of the file can be viewed while editing, viewing and adding the tags.

* The tags are displayed in radio buttons.
* The ID of the radio button is set to the index of the tag in *parentsList.*
* The ID of the “Add root tag” option is set to the *MinimumIntegerValue.*
* The ID of the “Add new tag” option is set to the *MaximumIntegerValue.*
* The tags that are mentioned as *unique* will not displayed while adding values to the existing tags.



**Fig 14 : View File Structure**

**Step 11: Update *data***

The values to be updated in the *data* are placed in a Queue, and the values are taken one after the other and written to *data.*

**Step 12: Write File**

The values in the *data* are written to the file recursively with the parent tag, child tags, attributes and node values, maintaining the same structure which was present while reading the file (if file existed)

**3.2 Functional Flow**

**3.2.1 Algorithm for parsing:**

0.Start

1.Get XML file.

2.Check if the file exists.

2.1 If File exists, Parse it using DOM Parser and store all the tags in a list.

2.2 For every node in the list, do

2.2.1 If the node has childnodes and the node is not yet added in the parentsList,then

2.2.1.1 Add node to parentsList.

2.2.1.2 Get the children of the node.

2.2.1.3 Add the child nodes to an array list, and then add this array list to childTags

2.2.2 If the node has nodevalue or attribute, then

2.2.2.1 Check if the node has attributes. If so, add this index to the attributeIndex.

2.2.2.2 Check if the node is present in elementsList. If not add it. Add the attribute value/ node value to an arraylist, and add this arraylist to arrayLists.

2.2.2.3 If node is already in parentsList, get the array list from the arrayList corresponding to this node, and add the attribute value/ node value to it

2.2 If file does not exist, throw error.

3.End

**3.2.2 Algorithm for editing:**

0.Start

1.Choose XML file.

2.If file exists, parse it and store the contents in data. If file is empty, data will be empty.

3.Display options for edit, view, add and pick new file.

3.1.If view Selected, display the available tags that have entries. Skip the tags that do not have entries.

3.1.1 Allow user to select the desired tag.

3.1.2 Display the first entry by default.

3.1.3 If the node has a value/attributes , display it.

3.1.4 Display all the child nodes with their node value.

3.1.5 Allow user to traverse between the entries

3.1.6 Allow user to delete/ edit the entries.

3.2 If edit is selected, display the available tags that have entries. Skip the tags that do not have entries.

3.2.1 Allow user to select the desired tag.

3.2.2 Display the first entry by default.

3.2.3 If the node has a value/attributes , display it.

3.2.4 Display all the child nodes with their node value.

3.2.5 Allow user to edit the tags and update the changes.

3.2.5 Allow user to traverse between the entries.

3.2.6 Allow user to delete the entries.

3.2.7 Write to file.

3.3 If the add is selected,

3.3.1 If the file is empty, display option for entering root tag.

3.3.2 If file exists then

3.3.2.1 Display all tags that are not unique.

3.3.2.2 Display all the unique tags that do not have any entries currently.

3.3.2.3 Display option for adding a new tag.

3.3.3 If user wants to add root tag then,

3.3.3.1 Get the tag name fom user.

3.3.3.2 Add this tag to the parentList.

3.3.3.2 Create an empty arrayList, and add it to the childTags.

3.3.4 If user wants to add new tag,

3.3.4.1 Provide checkboxes for Attributes, children and Unique.

3.3.4.1.1 If attribute is checked, get the attribute value and Check Children.

3.3.4.1.2 If children is checked, get the numver of children. Get the child tag name and child tag value.

3.3.4.1.3 If Unique is checked, set the flag for unique.

3.3.4.2 Get the new tag name. If unique flag is set, add "\_DONOTREPEAT" to the end of the tag name.

3.3.4.3 Check if the parentsList or elementsList already has this tag. If so throw error.

3.3.4.4 Add the tag to the parentsList.

3.3.4.5 If attribute is present, add the tag to the elementsList, its index to the attributeIndex and the attribute value to the arrayList.

3.3.4.6 Add the tag name to the childtags of the first tag (Root tag)

3.3.4.7 If children are present, create an array list and add the child tag names in it.

3.3.4.8 Add this array to childTags.

3.3.4.9 Add each child tag to elementsList.

3.3.4.10 For each child tag, create an array list, store the tag values of the child tag in i, and add this array list to arrayLists.

3.3.5 If the user wants to add values to an existing tag,

3.3.5.1 Display the structure of the tag with attributes, node value and child tags.

3.3.5.2 Allow user to enter the values. Store the entered value and the tag name in a queue.

3.3.5.3 Once user has entered all values, for each entry in queue

3.3.5.1 Check if it has entry in elementsList

3.3.5.2 Get the index of the tag within the elementsList.

3.3.5.3 Add the new value to the corresponding arrayList.

3.4. Write to file.

4. End

**3.2.3 Algorithm for update:**

0.Start

1. As the user edit a tag, store that index, tag name and tag value in a queue.

2. Wait for the user to finish editing. Add all changes to the queue.

3. Once user finishes editing, for each entry in the queue

3.1 Check if it has entry in elementsList

3.2 Get the index of the tag within the elementsList.

3.2 Find the arrayList that points to this tag.

3.3 Remove the old value from the index of the tag, and plae the new value.

4.Clear the queues.

5.End

**3.2.4 Algorithm for Delete:**

0.Start

1.If tag is present in elementsList then

1.1 Get the arrayList that holds the values of this tag.

1.2 Remove the desired index.

2. Get the children of the tag from childTags.

2.1 For each child,

2.1.1 Get the arrayList that holds the values of this child tag.

2.1.2 Remove the desired index.

3.If the deleted entry is the last entry, then

3.1 If it was the only entry, return to the menu.

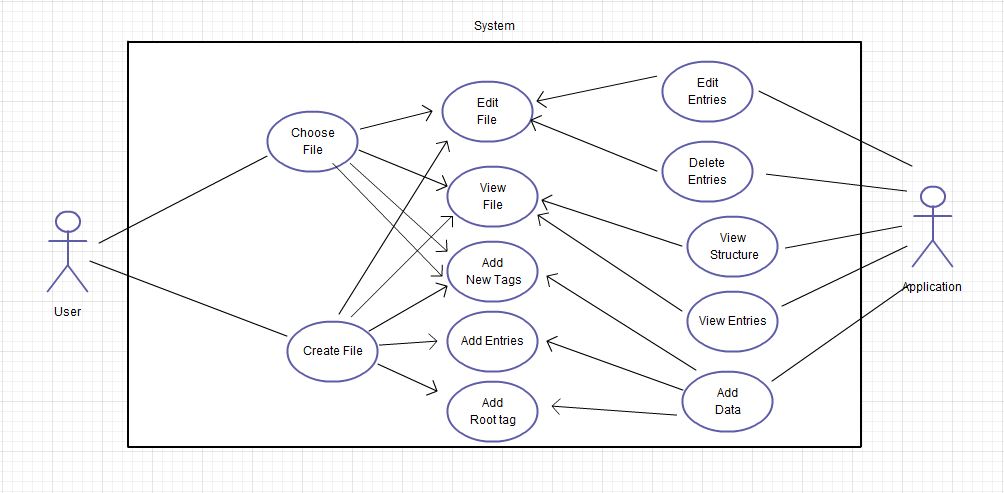
3.2 Display previous tag.

4.If there are more entries, display the next tag.

5.Write to file.

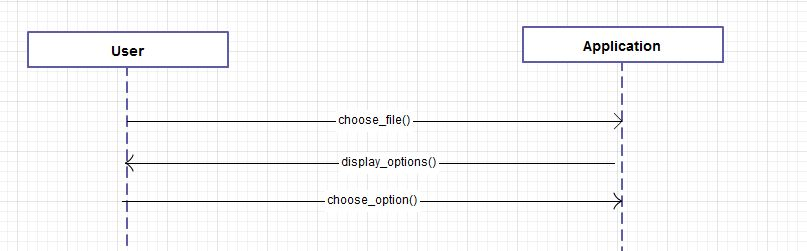
6.End

**3.3 High Level Design Document**

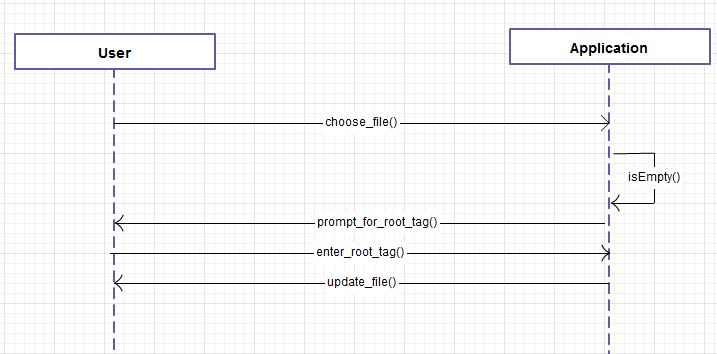
**3.3.1 Use Case Diagram**

**Fig 15 : Use case Diagram**

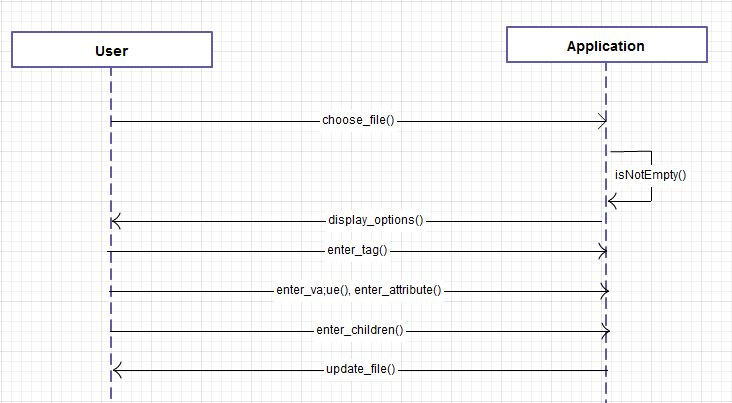
**3.3.2 Sequence Diagram**

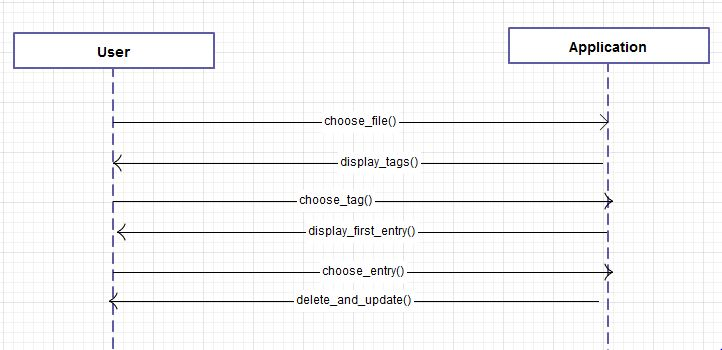
**Fig 16 : Display Options**

**Fig 17 : Add root tag**

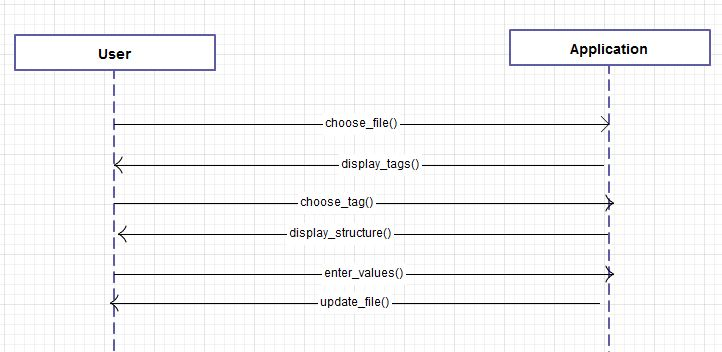
****

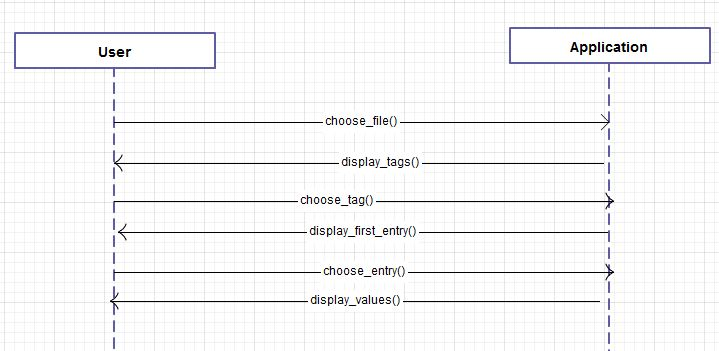
**Fig 18 : Add new Tag**

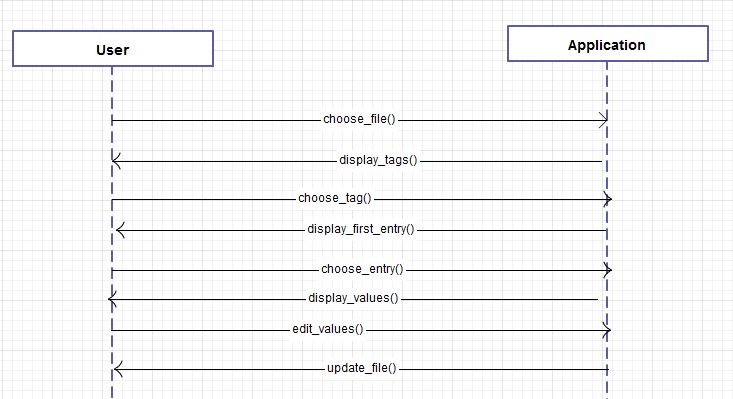
****

**Fig 19 : Delete Entry**

**Fig 20 : Add Values**

****

**Fig 21: View Entries**

**Fig 22 : Edit Values**

**4. Summary**

**4.1 Summary**

Thus the XMLEditorGUI is a flexible tool which gives options to create, view and edit XML files with any syntactical structure at any point of time while preparing the video lectures. The generality of this application empowers it to be used to create, view, edit, delete and parse XML files for other applications also.

**4.2 Uses in AAKASH**

The android version XMLEditorGUI will be very much useful to the AAKASH project where XML files are used as the data source. The desktop version can be used in any JVM installed machine.

The main goal of XMLEditorGUI tool is to edit the xml files for the corresponding video lectures prepared for delivering to the students both through Aakash tablet and through other media.

For instance, for the e-learning module the information about the lectures, slide names, professor name, duration, start and end Time, etc..., can be stored in XML files using this application. It is simple to use, and does not require much technical skills.

**4.3 Further Enhancements**

* **Improvement of GUI.**

The GUI can further be improved to ease the usage and improve the comfort level. The number of clicks required to perform any operation can be optimised in order to save time of the user, as this application primarily focuses to create XML files with large amount of data.

* **Searching for values in XML files.**

The search module can be included in the next version of XMLEditorGUI. Searching of both tags as well as values can be done. This can also include features like REPLACE, DELETE etc..

* **Enabling the creation of tags with multiple attributes.**

For now, any tag can have a maximum of only one attribute with the name “name”. Enhancements can be done to create tags with multiple attributes, where the name of the attribute can be specifies by the user.

* **Validation of individual tags.**

Due to the generality of this application and the creation of tags at runtime, the validation of individual tags has not been done yet. Ways can be sorted out to validate the individual tag values created by the user at runtime.

* **GUI for Desktop Publication**

The present desktop application works on console. Due to the boundless usability of this application, the development of GUI for desktop is to be done.

**4.3 References and Bibliography**

<http://androiddevelopmenttutorial.net/>

<http://developer.android.com/training/>

<http://stackoverflow.com/questions/tagged/android>

<http://www.theukwebdesigncompany.com/articles/xml-advantages-disadvantages.php>

**5. Conclusion**

XMLEditorGUI has been developed to turn the creation and manipulation of XML files an easy and on-the-go task. No more typological errors and wasting of time in manual typing. Saved time, improved efficiency and flexibility are all its advantages.