*PPTX* Reader Design Document

v1.0.1

Venketaram Ramachandran

Ben Chu

Table of Contents

[I. Introduction 6](#_Toc295825013)

[I.I Project Overview & Problem 6](#_Toc295825014)

[I.II Core Requirements for a Solution 6](#_Toc295825015)

[I.III Development Focus 6](#_Toc295825016)

[II. Technical Specifications 6](#_Toc295825017)

[II.I Languages & Formats 6](#_Toc295825018)

[II.II Library Dependency 7](#_Toc295825019)

[IV. Slide Generation 7](#_Toc295825020)

[III.I Overview 7](#_Toc295825021)

[III.II Model 7](#_Toc295825022)

[III.III Future Additions 8](#_Toc295825023)

[IV. XML Parsing 9](#_Toc295825024)

[IV.I Introduction 7](#_Toc295825021)

[IV.II 7](#_Toc295825022)

[IV.III How to Use 8](#_Toc295825023)

[A. Appendix 10](#_Toc295825025)

[A1. Figure 1 – Graphical Slide Model 10](#_Toc295825026)

[A2. Figure 2 – Graphic Slide Model (w/ descriptions) 11](#_Toc295825027)

[A3 – Slide Generator Implementation Model 12](#_Toc295825028)

[B. Version Updates 13](#_Toc295825029)

[B1. Version 1.0 13](#_Toc295825030)

[B2. Version 1.0.1 13](#_Toc295825031)

# I. Introduction

## I.I Project Overview & Problem

Current formats of Powerpoint files (.pptx) are unable to be viewed in Webex sharing modes. However, we can use the fact that these files are actually zipped XML files with information stored according to the guidelines of OpenXML’s PresentationML to add this feature. This application parses through the XML files representing the Powerpoint presentation of a given .pptx file and displays the presentation via HTML in an application frame. Currently, this application is designed for **WINDOWS ONLY**.

## I.II Core Requirements for a Solution

1. Take in an input .pptx file and unzip it to a temporary folder to view all of the XML files for the presentation.

1. Parse resulting unzipped XML files into a convenient format to translate into HTML5 or any other output format.
2. Build an architecture that allows for future expansion and scalability in OpenXML representation rather than a solution simply tailored for this specific problem.

## I.III Development Focus

In the initial versions (i.e. **version 1.0.1**), the focus of the development has been centered on the bottommost level of representation. In other words, the focus rests mainly on recreating a slide in HTML with basic text with fonts already temporarily preset. Once this basic representation is completed, the development can begin focusing on graphical and stylistic effects.

# II. Technical Specifications

## II.I Languages & Formats

The only supported input file format for this application is .pptx. Future versions of the applications may offer further support for different file formats.

The core of the application will be written in C++. This choice is influenced by the efficiency and control C++ provides in comparison to other Object Oriented Programming languages as well as the fact that the other components of WebEx is written in C++ for Windows and Mac platforms. This will help compatibility when eventually adding this feature to WebEx.

A PHP backend may be utilized to improve usability. However, **as of version 1.0.1** of the application, the central components of the application being focused upon do not require this.

## II.II Library Dependency

Aside from the Standard and I/O libraries, this application requires two main libraries from the Windows API:

1. A zipper library
   1. We require a basic zipper library which we can build upon or utilize in order to create a functional unzipping class that performs unzipping a .pptx file to exactly the location desired.
2. An XML Parser Library:
   1. In order to parse through XML files, we need a library to navigate through the tree structure. Random Access support in addition to Sequential Access support will be beneficial as long as performance is not significantly hampered. **As of version 1.0.1** of the application, the MSXML library is utilized.

# IV. Slide Generation

## III.I Overview

Since C++ does not natively offer any libraries for explicitly creating HTML pages, the main method will be to write strings to a file. However, this option is cumbersome and not ideal. As a result, an HTML Slide generator library will be written for the purposes of this project. In other words, this library does **NOT** offer direct C++ to HTML support. Rather, it presents a framework for drawing on a simple HTML rendered version of a blank .pptx slide. Refer to **Figure 1 in Section A1** in the Appendix for an example.

## III.II Model

We will use an Object Oriented model for the generator interface. For a graphical high-level explanation/representation of this current section, please refer to **Figure 3 in Section A3** in the Appendix.

Instead of the previously proposed HTML generation model, the current proposal creates a slide abstraction. Essentially, the interface allows addition of content on an HTML rendered slide, either in form of text, media, or drawings.

To utilize this library, each invocation needs to create a Slide object. In this version, each slide object will allow the user to:

* Obtain a Text object
  + Will allow the user to specify text properties, content, and location
* Obtain a Media object
  + Unimplemented
  + Plan:
    - Allow source, size specification
* Obtain a Drawing object
  + Unimplemented
  + Plan:
    - Allow custom shape selection (i.e. a circle of radius specified by argument)

After configuring the properties of each obtained object, the user can place the object in the Slide via a given method in the Slide class. Once all necessary objects are placed in the Slide object, the Slide class provides a writeOut() function that allows this information to be written to a script file, that will be used for the slide generation in HTML.

## III.III Future Additions

Currently, the Slide Generation model is fairly basic. It allows for placement of specific objects in a generated slide. Clearly, the features of the current model are not exhaustive for creation of PPTX slides. The initial version attempts to simply create architecture to generate objects on a slide and provide extensibility to add features. Cases to consider in the future include:

* Line-wrapping for texts
* Collisions (i.e. text and image)
* Slide transitioning
* Placement order

# IV. Parsing Framework

## IV.I Introduction

The parsing framework will use the Microsoft XML (MSXML) XML parsing library. We chose MSXML because it can be implemented easily in C++ (see code below). The MSXML dynamic-link library (msxml6.dll) is also included with Windows 7, which is helpful for our development on Visual Studio. The framework will abstract away the internal MSXML functions for the user. At the core are 3 main classes: Parser, Node and NodeList.

Parser Class:

* Represents the document and stores the XML file name.
* The root node of the XML tree can be extracted from the document.

Node Class:

* Represents an XML tag as a node in a tree.
* The tag name and tag text can be extracted from each node.
* To traverse the XML tree, there is access to the node’s children.

NodeList Class:

* Represents a list of nodes with an iterator interface.
* To iterate through the nodes, use the NextNode() method, which will output NULL after all the nodes have been traversed.

class Parser {

char fileName[255];

public:

Parser(char\*);

NodeList getChildren();

Node getRoot();

NodeList getElementsByTag(char\*);

void printFile(){std::cout << "file name: " << fileName << std::endl;}

};

class Node{

public:

MSXML2::IXMLDOMNodePtr NodePtr;

Node (); //default constructor

Node(MSXML2::IXMLDOMNodePtr np);

NodeList getChildren();

\_bstr\_t getName();

\_bstr\_t getText();

//getAttributes()

};

class NodeList{

private:

MSXML2::IXMLDOMNodeListPtr NodeListPtr;

public:

NodeList(MSXML2::IXMLDOMNodeListPtr nlp);

Node NextNode();

};

## IV.II

## IV.III How To Use

1. Initialize the Parser
   1. Construct a new parser document by supplying a file name.
   2. Then use the getRoot() method to extract the root node.
2. Traverse the Tree
   1. A node’s children can be extract by the getChildren() method (outputs a NodeList object).
   2. A NodeList can be iterated through with the NextNode() method (outputs a Node object) , which will output NULL after all the nodes have been traversed.
3. Extract a Node’s Data
   1. A node object’s data can be extracted using the getName() and getText() methods (both output a \_bstr\_t).

/\* Initializing the parser \*/

Parser pptParser("C:\\Users\\bechu\\Desktop\\slide1.xml"); //file name

//pptParser.printFile();

Node node = pptParser.getRoot();

std::cout << node.getName() << std::endl;

/\* Traversing the Tree \*/

NodeList nodeList(node.getChildren());

node = nodeList.NextNode();

while (node.NodePtr){

std::cout << node.getName() << std::endl;

node = nodeList.NextNode();

}

std::cout << std::endl;

/\* Tag Text \*/

NodeList titles(pptParser.getElementsByTag("a:t"));

node = titles.NextNode();

while (node.NodePtr){

std::cout << node.getText() << std::endl;

node = titles.NextNode();

}

std::cout << std::endl;

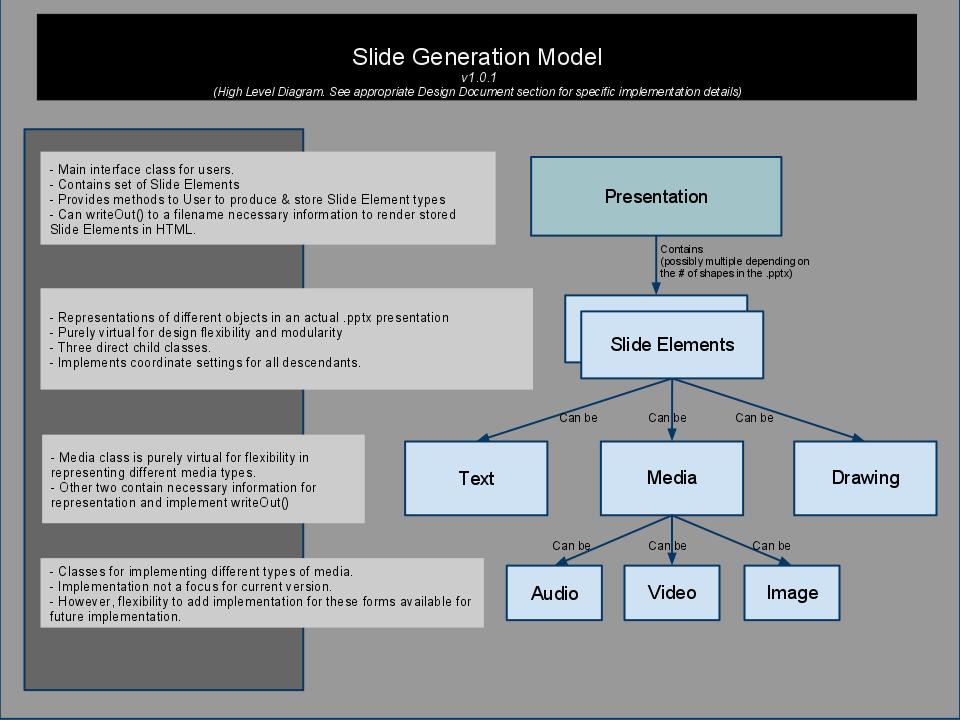
# A. Appendix

## A1. Figure 1 – Graphical Slide Model



## C:\Users\venkeram\Desktop\CSD.jpgA2. Figure 2 – Graphic Slide Model (w/ descriptions)

## A3 – Slide Generator Implementation Model



# B. Version Updates

## B1. Version 1.0

* Base version.
  + HTML Generation model
    - C++ interface to generate HTML code
    - Wide range of control in creating presentation slides.
  + XML Parsing design to be completed
* No changes.

## B2. Version 1.0.1

* Changed Generation Model from HTML Generation to Slide Generation.
  + Essentially, the interface for an XML parser does not have to specifically output HTML code.
  + A level of abstraction was added
    - I.E. the XML parser is outputting to a Slide interface, having the capability to define the styles of and add to the slide:
      * Media (such as images, video, & audio)
      * Text
      * Custom Drawings
  + This simplifies the work of the XML parser component & the complexity of the interface.
* Implementation focus remains on building architecture for generation of presentation