**s o f i a**

**Software Design Specification**

*Teach Easy & Learn Easy Software Design Specification*

**Document Control**

|  |  |  |
| --- | --- | --- |
| **Editor** | **Date** | **Update** |
| **Alistair Jewers** | **28/02/2015** | **0.0 Document Created, very rough draft** |
| **Alistair Jewers** | **11/03/2015** | **0.1 Updated to reflect project developments** |
| **Alistair Jewers** | **12/03/2015** | **1.0 First Draft finished for inclusion in the release 1 documentation.** |
| **Alistair Jewers** | **02/06/2015** | **2.0 Document Updated to cover the full software suite.** |

**Table of Contents**

[1. Overview 4](#_Toc421127355)

[1.1 Teach Easy 4](#_Toc421127356)

[1.2 Learn Easy 4](#_Toc421127357)

[2. Requirements 5](#_Toc421127358)

[2.1 User Stories 5](#_Toc421127359)

[3. Design 6](#_Toc421127360)

[3.1 XML 6](#_Toc421127361)

[3.1.1 XML Format 6](#_Toc421127362)

[3.1.2 XML Parser 6](#_Toc421127363)

[3.1.3 XML Writer 8](#_Toc421127364)

[3.1.4 XML Handler 8](#_Toc421127365)

[3.2 Data 9](#_Toc421127366)

[3.2.1 Lesson 9](#_Toc421127367)

[3.2.2 Page 10](#_Toc421127368)

[3.2.3 Page Objects 10](#_Toc421127369)

[3.2.4 Lesson Metadata Objects 10](#_Toc421127370)

[3.3 Application Backend 10](#_Toc421127371)

[3.3.1 Runtime Data 10](#_Toc421127372)

[3.3.2 Editor Runtime Data 11](#_Toc421127373)

[3.4 User Interface 12](#_Toc421127374)

[3.4.1 Learn Easy 12](#_Toc421127375)

[3.4.2 Teach Easy 12](#_Toc421127376)

[3.5 Rendering 12](#_Toc421127377)

[3.5.1 Renderer 12](#_Toc421127378)

[3.5.2 Instantaneous Media Handlers 12](#_Toc421127379)

[3.5.3 Non-Instantaneous Media Handlers 13](#_Toc421127380)

[4.0 Data Flow 13](#_Toc421127381)

[Appendix 1. Packages: 14](#_Toc421127382)

**Software Design Specification**

# 1. Overview

The *Teach Easy* software suite is a pair of applications designed to allow the creation and completion of interactive, multimedia rich, digital lessons. The suite includes two applications, *Teach Easy* and *Learn Easy*. This document outlines the design and structure of the software suite in terms of the constituent code elements. This document also describes the data path for each program. The programs both revolve around the

## 1.1 Teach Easy

The *Teach Easy* program provides the ability to create a digital lesson by adding multimedia elements to a number of sequential ‘*pages*’ contained within a single, overarching lesson.

## 1.2 Learn Easy

The *Learn Easy* program provides the ability to open the digital lesson files created in *Teach Easy* and complete the lesson by interacting with the various media elements added by the lesson creator.

# 2. Requirements

## 2.1 User Stories

The following user stories are taken from the Functional Specification document; please refer back to that document for more information.

TeachEasy User Stories:

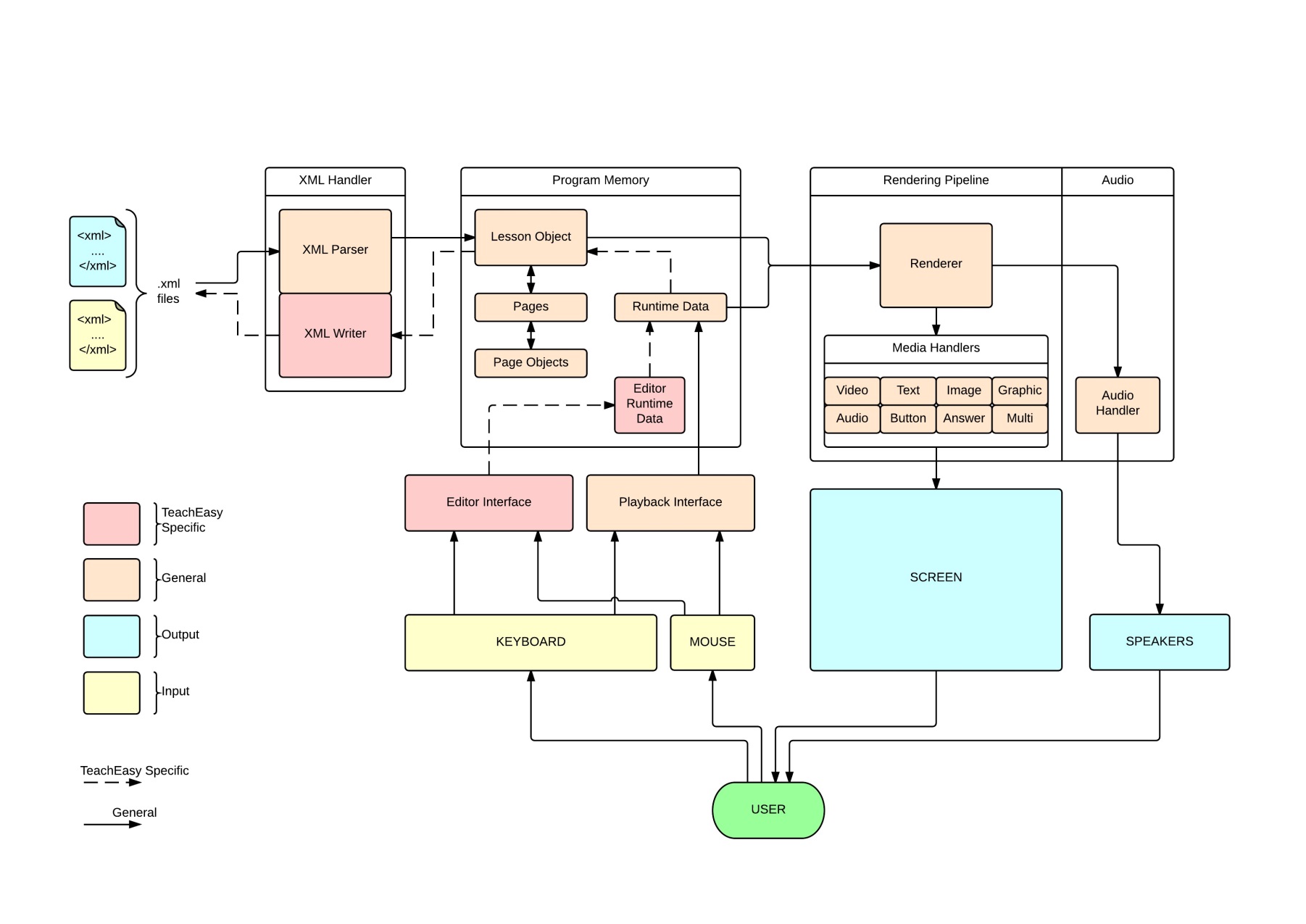
1. “As a teacher, I can create an interactive lesson for a student of mine to complete independently”
2. “As a teacher, I can provide any number of my students with access to a lesson I created previously”
3. “As a teacher, I can create a lesson comprised of a number of discrete pages, each of which I can customise”
4. “As a teacher, I can pick a page category from a number of pre-defined templates (e.g. video, quiz, etc.)”
5. “As a teacher, I can include multimedia objects in the lessons I create”
6. “As a teacher, I can save lessons I am working on then access them again and edit them at a later date”

LearnEasy User Stories:

1. “As a student, I can complete an interactive lesson created for me by my teacher, in my own time”
2. “As a student, I can view a lesson created by my teacher”
3. “As a student, I cannot edit a lesson”
4. “As a student, I can choose the lesson I want to work on from a selection of lessons provided to me by my teacher(s)”
5. “As a student, I can pause and resume lessons”
6. “As a student, I can view all forms of multimedia that my teacher has included in the lesson”
7. “As a student, I can interact with all suitable forms of multimedia (e.g. pause a video)”
8. “As a student, I am provided with a record detailing my lesson completion and level of achievement, which I can choose to print out”

Throughout this document these user stories are referenced to contextualise code elements and relate their purpose back to the customer requirements.

# 3. Design

  
The software has been developed in the Java programming language for PCs running the Windows operating system. The two applications in the *Teach Easy* suite share a lot of common functionality, and can therefore also share a large amount of code. Figure 3-1, the Software Design Diagram, shows how the two programs are structured and how the different code elements fit together. Elements specific to the Teach Easy application are highlighted. Figure 3-1 also shows the system inputs and outputs. A full size version of Figure 3-1 is available in the Software Design Diagram document.

## 3.1 XML

Figure 3‑1 - Software Design Diagram

### 3.1.1 XML Format

Digital lessons are stored as XML files using a custom set of XML elements and attributes. This format is an extension of the Software Engineering Project Wide Standard format, which can be found in the Project Wide Standards document.

### 3.1.2 XML Parser

User stories 2, 7, 8 and 10 require that the applications can open the Digital Lesson XML files. Hence the software requires the ability to parse the data within an XML file and store it in memory. This functionality is provided by the XML Parser class (XMLParser.java) and several content handlers found in the package *teacheasy.xml.contenthandlers*. These classes utilise the Java implementation of the Simple API for XML (SAX) parser.

The code works by iterating through the contents of the XML file and calling methods based on the elements, attributes and text it finds. For the majority of the objects in the digital lesson format there is a distinct XML content handler. When one of these elements is found the currently handler is swapped for the appropriate handler for the found element. In general the handlers create a new data object of the type identified by the element name. Data is added to the object under construction based on the values of the attributes associated with the element, the text between the elements start and end tags, and any internal elements. When the end tag for the element is reached the object under construction is inserted into the overall data structure and control is returned to the previous content handler. When a file has been parsed completely the parser then stores this data structure, ready to be retrieved by another part of the code. Note that if a data structure is stored in the parser and a request is made to parse another xml file the first data structure is erased and overwritten by the new data. For this reason the parser can only hold the data for one digital lesson at any one time, although if the data was retrieved from the parser before the second parse request was made, both sets of data could be available.

The following table describes the API for the XML Parser class.

|  |  |  |  |
| --- | --- | --- | --- |
| Class: **XMLParser.java**  Function: Reads the digital lesson XML files and stores the information in data. Provides access to this data. | | | |
| **Method** | **Arguments** | **Returns** | **Description** |
| XMLParser() *constructor* | None | Void | Instantiates the parser before use. |
| parse() | Location of the file to parse as a String. | An Array List of any errors that occurred during parsing. | Tells the parser to parse an xml file identified in the arguments. The parser will attempt to parse the file and store the data. When finished an Array List of XMLNotification objects will be returned with one entry for each error or warning encountered during parsing. A clean parse is indicated by returning an empty Array List. |
| getLesson() | Void | A Lesson object, see section 3.2 | Retrieves from the parser the data object created from the most recent parse. This function will succeed regardless of whether or not errors were found, so error checking is the responsibility of the calling class. Note that this function may return an empty or partially constructed Lesson object. This will not however include any null pointers. |

### 3.1.3 XML Writer

User stories 1, 3 and 6 require that the applications can save files in the Digital Lesson XML format. Hence the software requires the ability to convert the data stored in memory into an XML file in the Teach Easy Digital Lesson format and save that XML file in the computers file system. The XML Writer class (XMLWriter.java) provides this functionality. The code works by iterating through all the data present in a given lesson data structure and constructing a Document Object Model (DOM) of the XML file to be created by adding the elements and attributes to describe each of the objects found in the data structure. These elements and attributes are created as defined in the Teach Easy Digital Lesson File Format document. Once all the data has been added to the DOM it is saved as an XML file with a specified name and location.

The following table describes the API for the XML Writer class.

|  |  |  |  |
| --- | --- | --- | --- |
| Class: **XMLWriter.java**  Function: Create an XML file from a given lesson data structure at a given location in the file system and with a given file name. | | | |
| **Method** | **Arguments** | **Returns** | **Description** |
| XMLWriter() *constructor* | Void | Void | Initialise the XML Writer before use |
| writeXML() | The data object of the lesson to export.  A string of the file name and location to save the XML with. | An array list of strings with one entry for each error encountered whilst creating the file. | Called to create an XML file based on a Lesson data object. Returns any errors encountered as Strings in an Array List. A clean write is indicated by returning an empty array list. Note that a clean write does not mean the file is valid, simply that the data provided was written to an XML file successfully. |

### 3.1.4 XML Handler

The functionality related to reading and writing XML files is encapsulated into the XML Handler class (XMLHandler.java) for convenience. This class provides the ability to parse a specified XML into memory using an instance of the XML Parser class as well as the ability to create an XML file based on a specified Lesson data structure using an instance of the XML Writer class. Further to this it also maintains a record of the location of the most recently opened XML file and the location of the most recently written XML file. This allows the ‘Open File’ and ‘Save File’ dialogs to default to a sensible location.

The following table describes the API of the XML Handler.

|  |  |  |  |
| --- | --- | --- | --- |
| Class: **XMLHandler.java**  Function: Encapsulate functionality relating to XML reading and writing. | | | |
| **Method** | **Arguments** | **Returns** | **Description** |
| XMLHandler() *constructor* | Void | Void | Initialise the XML hanlder before use. Instantiates an XMLParser object and an XMLWriter object. |
| parseXML() | See XMLParser API in section 3.1.2 | | |
| getLesson() | See XMLParser API in section 3.1.2 | | |
| writeXML() | See XMLWriter API in section 3.1.3 | | |
| getRecentReadPath() | Void | The location of the most recently parsed file as a String | Used to obtain the location of the most recently parsed file to inform the file browser. |
| getRecentWritePath() | Void | The location of the most recently written file as a String | Used to obtain the location of the most recently written file to inform the file browser. |
| setRecentReadPath() | The location of the most recently parsed file as a String | Void | Used to save the location of the most recently parsed file. |
| setRecentWritePath() | The location of the most recently written file as a String | Void | Used to save the location of the most recently written file. |

## 3.2 Data

A lesson inside the *Teach Easy* environment is made up of a nested set of data objects. For more information on this see the General Design Specification. In the code this structure is created through a combination of several different classes, each storing the data for an individual part of the lesson. In the following sections that data structure is broken down into its constituent parts.

### 3.2.1 Lesson

The top level class of the data structure is the Lesson class (Lesson.java). An instance of this class fully describes a Teach Easy Digital Lesson, and can contain many instances of the smaller classes which describe individual lesson elements. Each Lesson object includes one instance of each of the lesson metadata objects described in section 3.2.4. Beyond these the Lesson object also contains an Array List of the Page objects described in section 3.2.2. These are indexed in the sequence in which they appear in the lesson, from 0 to PAGECOUNT - 1.

### 3.2.2 Page

The Page object (Page.java) encapsulates the data for a single page within a Teach Easy Digital Lesson. It contains two class level fields; the number identifying the page’s position within the lesson and the background colour of the page, described as a String in 8 digit ARGB hexadecimal colour format. The Page object also contains an Array List of objects which extend the abstract Page Object class described in section 3.2.3. This list encapsulates all of the objects present on this page. They are indexed in layer order from back to front.

### 3.2.3 Page Objects

For each of the media objects that can be present on a page within a Teach Easy Digital Lesson there is a corresponding class which extends the abstract Page Object class (PageObject.java). These objects are Text (TextObject.java), Image (ImageObject.java), Graphic (GraphicObject.java), Audio (AudioObject.java), Video (VideoObject.java), Answer Box (AnswerBoxObject.java), Multiple Choice Question (MultipleChoiceObject.java). Each one contains fields to store the relevant data for that type of object as well as methods for retrieving and changing the data. The objects do not contain data relating to their state during runtime, as this data structure simply describes a lesson’s components and should remain unchanged by a lesson being opened and used.

### 3.2.4 Lesson Metadata Objects

Each lesson also includes some metadata. This is broken down into Lesson Info (LessonInfo.java) Default Settings (LessonDefaultSettings.java) and Grade Settings (LessonGradeSettings.java). The Lesson Info object stores information relating to the lesson such as the title, the author, a short description etc. The Default Settings object stores values for a lesson’s default visual settings like font, font size, font colour etc.

## 3.3 Application Backend

Besides the data that describes the Lesson currently being created or completed, the applications will require some variables to keep track of the current state of the program, for example the page of the lesson that is currently open. This data is likely to change whilst the program is running but is not part of the lesson itself.

### 3.3.1 Runtime Data

The Runtime Data class (RuntimeData.java) maintains data related to the current state of the LearnEasy application while it is running. This includes class level fields to keep track of the current page and whether or not a lesson has been opened. Further to this the Runtime Data class acts as a bridge between the user interface and all the other parts of program, controlling the data flow and containing methods which are called based on the user inputs. It can therefore be seen as the core of the program. The Runtime Data class therefore maintains a reference to the lesson currently open, an instance of the XML Handler class described in section 3.1.4, and an instance of the Renderer class described in section 3.5.1.

The following table describes the API of the Runtime Data class.

|  |  |  |  |
| --- | --- | --- | --- |
| Class: **RuntimeData.java**  Function: Maintains the state of the LearnEasy application whilst it is running. Includes methods to manipulate the data in different ways to provide functionality. | | | |
| **Method** | **Arguments** | **Returns** | **Description** |
| RuntimeData()  *Constructor* | Void | Void | Initialises the runtime data before use. Instantiates an XML Handler object. |
| nextPage() | Void | Void | Attempts to increment the current page if a next page exists. |
| prevPage() | Void | Void | Attempts to decrement the current page if a previous page exists. |
| isNextPage() | Void | True if there is a page after the current one, false otherwise. | Used to determine if a next page exists. |
| isPrevPage() | Void | True if there is a page before the current one, false otherwise. | Used to determine if a previous page exists. |
| isLessonOpen() | Void | True if a lesson is already open. False otherwise. | Used to determine if a lesson is currently open. |
| openLesson() | Void | Void | Called to open a lesson. |
| getLesson() | Void | The current Lesson object | Used to obtain the currently open Lesson object. |
| closeLesson() | Void | Void | Called to close the current lesson. |
| redraw() | Void | Void | Called to redraw the current page. |

### 3.3.2 Editor Runtime Data

The Editor Runtime Data class (EditorRuntimeData.java) maintains data specific to the editor application. It has all of the same functionality as the standard run time data, as well as many other editor specific functionalities. This includes instances of a variety of subclasses that allow different editor functionality. These are listed below.

|  |  |
| --- | --- |
| **Sub classes of EditorRuntimeData.java** | |
| **Class Name** | **Purpose** |
| *MouseController.java* | Contains methods related to interpreting mouse input. |
| *PropertiesPane.java* | Maintains data related to which object is selected and methods for manipulating the objects. |
| *NewObjectController.java* | Contains methods for adding new objects to pages and copying and pasting objects. |
| *TemplateController.java* | Contains methods relating to applying page templates. |
| *(\*)PropertiesController.java* | Individual controllers for each object type. Displays their properties in the properties pane and allows for modifications. |

Beyond the standard runtime methods the editor runtime data class features several other methods specific to editor functionality. These are summarised in the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| Class: **EditorRuntimeData.java**  Function: Maintains the state of the TeachEasy application whilst it is running. Includes methods to manipulate the data structure in different ways to provide editing functionality. | | | |
| **Method** | **Arguments** | **Returns** | **Description** |
| *saveLesson()* | none | void | Saves the currently open lesson structure by writing it to an XML file using the XML handler. |
| newPage() | Template Type | void | Adds a new page to the lesson after the currently open page. If the template type parameter is a valid template type that template is applied to the new page. |
| removePage() | none | void | Deletes the current page. |
| newObject() | Page Object Type | void | Adds a new object of the type indicated by the page object type parameter to the current page. |
| removeObject() | none | void | Removes the currently selected object, if one is selected. |
| copyObject() | none | void | Copy the selected object to the clipboard. |
| pasteObject() | none | void | Paste the object currently on the clip board to the current page. |
| mousePressed() | Position information | void | Called when the mouse is pressed, passes information about the position of the press to the mouse controller |
| mouseReleased() | Position information | void | Called when the mouse is released, passes information about the position of the release to the mouse controller |
| mouseMoved() | Position information | void | Called when the mouse is moved, passes position info to the mouse controller. |

## 3.4 User Interface

### 3.4.1 Learn Easy

The Learn Easy GUI utilises the Java FX 2.2 platform to provide the user with the ability to interact with and complete a Teach Easy Digital Lesson. This functionality is required to fulfil user stories 7, 8, 12 and 13. The GUI is contained in the Learn Easy Client class (LEClient.java). This class is the entry point for the Learn Easy application and maintains an instance of the Runtime Data class described in section 3.3.2. The Learn Easy Client class therefore provides the interface by which a user can activate methods within the Runtime Data class to perform the various Learn Easy functionalities whilst simultaneously providing a framework inside which the renderer, described in section 3.5.1 can display the lesson.

### 3.4.2 Teach Easy

The Teach Easy GUI utilises the Java FX 2.2 platform to provide the user with the ability to create and preview a Teach Easy Digital Lesson by combining and editing different media elements. This functionality is required to fulfil user stories 1, 3, 4, and 5. The GUI is contained in the Teach Easy Client Class (TEClient.java). This class is the entry point for the Teach Easy application and maintains instances of the Runtime Data (RuntimeData.java) and Editor Runtime Data (EditorRuntimeData.java) classes which contain all the data and functionality of the program. The Teach Easy Client class therefore provides the interface by which a user can activate methods within the Runtime and Editor Runtime Data classes to perform the various Teach Easy functionalities.

## 3.5 Rendering

### 3.5.1 Renderer

The Renderer class (Renderer.java) provides the functionality for combining the Lesson data structure and the runtime data to produce output for the user. This includes both visual output on the screen and audible output. This is achieved through the use of a number of media handlers which each deal with a different type of media object. The Renderer class maintains a reference to the Java FX object on which to draw as well as an instance of each media handler.

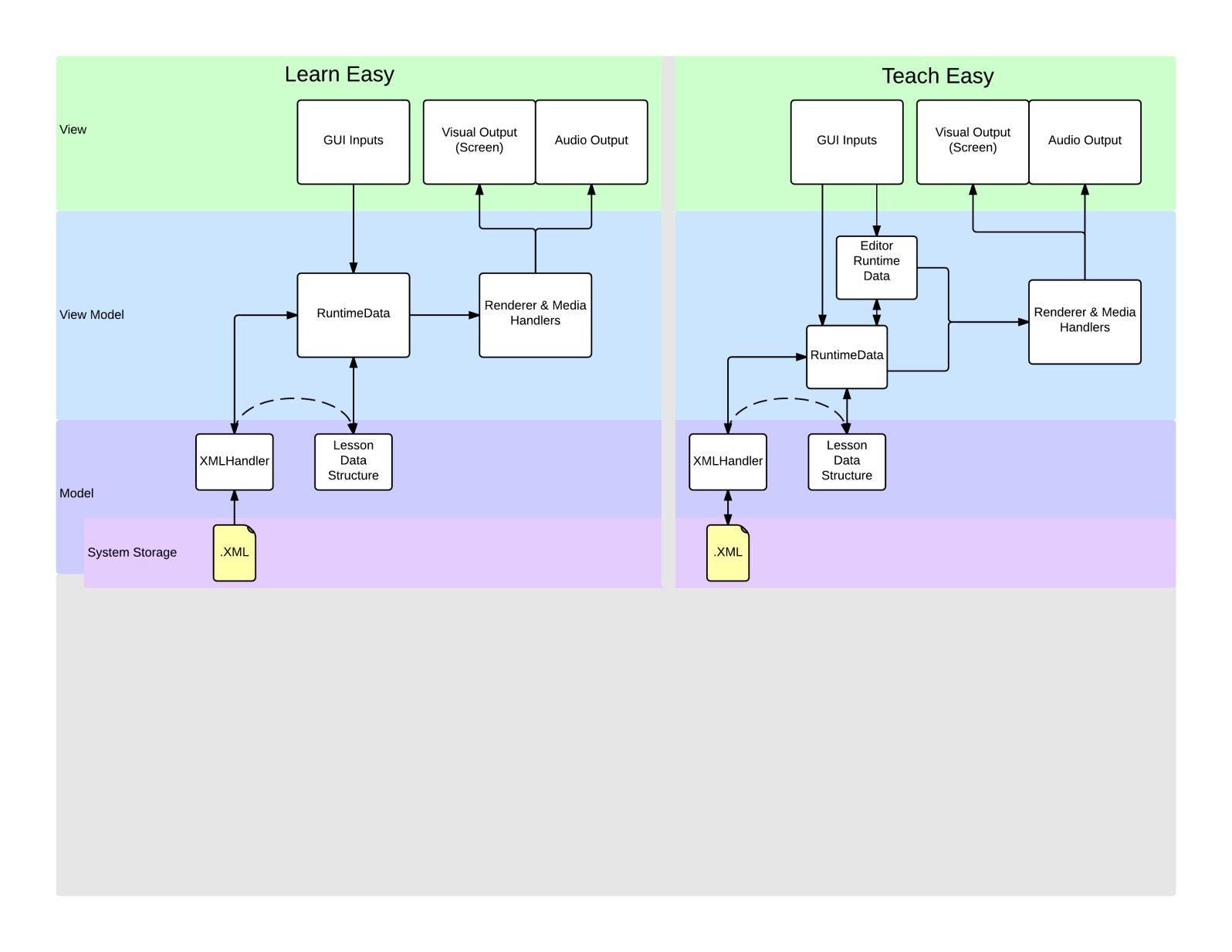
### 3.5.2 Media Handlers

For each of the possible types of object that can be present on a page there exists a media handler class which is used by the renderer to render those objects on screen. The media handlers are instantiated in the renderer and given enough information to perform their function. When rendering a page the renderer iterates through each object in the data structure for that page, calling the method in the appropriate handler with the data found to render the object on the page. The majority of the media handlers maintain lists of the objects that are being handled so that they can be referenced later, for example to obtain their size in the editor.

# 4.0 Software Architecture

The code that makes up the *Learn Easy* and *Teach Easy* applications is structured based on a model view view-model (MVVM) system. The top level is the view, which includes the inputs and outputs of the system that relate to the user as well as the GUI, where the user provides inputs and receives visual and audible outputs produced by the various media. The user inputs send information to the layer below, the view-model, where the core of the program is. This layer uses the inputs it receives to determine what function to perform and what data to use. In the case of *Teach Easy* this is split into two sections, the standard run time data that is common to *Learn Easy* as well as some run time data specific to the ‘editor’ functionality of *Teach Easy.* The renderer and its associated media handlers are found in this layer and are used to control what outputs are produced for the user. The lowest layer is the model layer where the data structure describing the lesson being used is held, as well as the code for converting XML files to program data. The XML files are stored in the computers file system which can be accessed through the model layer. The code in the view-model layer accesses the model layer as needed and controls the conversion of data from one form to another. Figure 4-1 shows this MVVM architecture as a block diagram. A full size version of Figure 4-1 is available in the Software Architecture Diagram document.

Figure 4‑1 - Software Architecture Diagram



# Appendix 1. Packages:

This table details the packages within the code base and their purpose.

|  |  |
| --- | --- |
| **Package** | **Description** |
| *teacheasy.main* | Contains the main executable classes for running the applications. |
| *teacheasy.data* | Contains the data objects the describe a lesson and its components. |
| *teacheasy.data.lessondata* | Contains the data objects that describe general lesson information. |
| *teacheasy.data.multichoice* | Contains the data objects related to multiple choice objects. |
| *teacheasy.runtime* | Contains the classes that control the applications at runtime. |
| *teacheasy.runtime.editor* | Contains the classes used by the editor runtime to manipulate lessons and objects. |
| *teacheasy.mediahandler* | Contains the media handler classes. |
| *teacheasy.mediahandler.video* | Contains classes specific to the video handler. |
| *teacheasy.mediahandler.audio* | Contains classes specific to the audio handler. |
| *teacheasy.mediahandler.multichoice* | Contains classes specific to the multiple choice handler. |
| *teacheasy.mediahandler.answerbox* | Contains classes specific to the answer box handler. |
| *teacheasy.debug* | Contains classes to debug and test the applications. |
| *teacheasy.render* | Contains classes relevant to rendering the pages. |
| *teacheasy.test* | Contains automated test classes. |
| *teacheasy.certificate* | Contains the class related to rendering the certificate. |
| *teacheasy.icons* | Contains all the icon files to compile into the |
| *teacheasy.xml* | Contains the classes the work on and are related to XML. |
| *teacheasy.xml.contenthandlers* | Contains the content handler classes for parsing different XML elements. |
| *teacheasy.xml.util* | Contains the XML utility classes |
| *learneasy.homepage* | Contains the classes to control displaying the home page when learn easy is loaded. |
| *learneasy.trackprogress* | Contains the classes related to tracking a students progress in learn easy. |
| *wavemedia.graphic* | Contains graphic handler code produced by wave media. |
| *wavemedia.text* | Contains text handler code produced by wave media. |