LatexEditor

Release Report

<Infinite Loop SLK>

<Foteini Karetsi, A.M. 2990,

Chaido-Effrosyni Louka, A.M. 3020>

VERSIONS HISTORY

Date	Version	Description	Author
04/19/2019	<1.0>	Release of 1.0 project's version	F.Karetsi, C.Louka

1 Introduction

This document provides information concerning the **<1.0>** release of the project.

1.1 Purpose

Latex is a well known high quality document preparation markup language. It provides a large variety of styles and commands that enable advanced document formatting. Typically, a Latex document is compiled with a tool like MikTex, Lyx, etc. to produce a respective formatted document in pdf, ps, etc. Formatting documents with Latex is a programming like process as it involves the proper usage of Latex commands which are embedded in the document contents. The objective of this project is to develop a simple Latex editor for inexperienced Latex users. The goal of the editor is to facilitate the usage of Latex commands for the preparation of Latex documents. One of the prominent features that distinguishes the LatexEditor from other similar applications is its multi-strategy version tracking functionalities that enable undo and redo actions.

1.2 Document Structure

The rest of this document is structured as follows. Section 2 specifies the acceptance tests that have been employed for this release of the project. Section 3 specifies the main design concepts for this release of the project.

2 Tests

<For the user stories included in this release specify below corresponding tests using a typical tabular form.>

2.1 Tests for User Story **<1>**

Test ID	*CreateCommandTestArticle*
User Story	*User Story 1*
Test Class	*CreateCommand*
Description	*We create a CreateCommand object with parameter "article", execute it and compare the contents of the returned document with those from the original template*

Test ID	*CreateCommandTestBook*
User Story	*User Story 1*
Test Class	*CreateCommand*
Description	*We create a CreateCommand object with parameter "book", execute it and compare the contents of the returned document with those from the original template*

Test ID	*CreateCommandTestLetter*
User Story	*User Story 1*
Test Class	*CreateCommand*
Description	*We create a CreateCommand object with parameter "letter", execute it and compare the contents of the returned document with those from the original template*

Test ID	*CreateCommandTestReport*
User Story	*User Story 1*
Test Class	*CreateCommand*
Description	*We create a CreateCommand object with parameter "report", execute it and compare the contents of the returned document with those from the original template*

Test ID	*CreateCommandTestOther*
User Story	*User Story 1*
Test Class	*CreateCommand*
Description	*We create a CreateCommand object with parameter "other", execute it and compare the contents of the returned document with those from the original template (empty-template)*

2.2 Tests for User Story **<2>**

Test ID	*EditCommandTestContent*
User Story	*User Story 2*
Test Class	*EditCommand*
Description	*We create a document with type "article" and an EditCommand object with parameter text, execute it and compare the contents of the returned document with those from the original template with the text inside.*

Test ID	*EditCommandTest*
User Story	*User Story 2*
Test Class	*EditCommand*
Description	*We create an empty document and an EditCommand object with parameter text, execute it and compare the contents of the returned document with those from the text .*

2.3 Tests for User Story **<3>**

Test ID	AddLatexCommandTestChapterA
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "article" without contents and an AddCommand object with parameter "Chapter", execute it and compare the contents of the returned document with those from the chapter's text.*

Test ID	AddLatexCommandTestChapterB
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "book" without contents and an AddCommand object with parameter "Chapter", execute it and compare the contents of the returned document with those from the chapter's text.*

Test ID	AddLatexCommandTestChapterL
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type “letter” without contents and an AddCommand object with parameter “Chapter”, execute it and compare the contents of the returned document with those from the chapter's text.*

Test ID	AddLatexCommandTestChapterR
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type “report” without contents and an AddCommand object with parameter “Chapter”, execute it and compare the contents of the returned document with those from the chapter's text.*

Test ID	AddLatexCommandTestChapterO
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type “other” and an AddCommand object with parameter “Chapter”, execute it and compare the contents of the returned document with those from the chapter's text.*

Test ID	AddLatexCommandTestSectionA
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type “article” without contents and an AddCommand object with parameter “Section”, execute it and compare the contents of the returned document with those from the sections's text.*

Test ID	AddLatexCommandTestSectionB
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type “book” without contents and an AddCommand object with parameter “Section”, execute it and compare the contents of the returned document with those from the sections's text.*

Test ID	AddLatexCommandTestSectionL
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "letter" without contents and an AddCommand object with parameter "Section", execute it and compare the contents of the returned document with those from the sections's text.*

Test ID	AddLatexCommandTestSectionR
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "report" without contents and an AddCommand object with parameter "Section", execute it and compare the contents of the returned document with those from the sections's text.*

Test ID	AddLatexCommandTestSectionO
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "other" and an AddCommand object with parameter "Section", execute it and compare the contents of the returned document with those from the sections's text.*

Test ID	AddLatexCommandTestSubsectionA
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "article" without contents and an AddCommand object with parameter "Subsection", execute it and compare the contents of the returned document with those from the subsections's text.*

Test ID	AddLatexCommandTestSubsectionB
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "book" without contents and an AddCommand object with parameter "Subsection", execute it and compare*

	the contents of the returned document with those from the subsections's text.

Test ID	AddLatexCommandTestSubsectionL
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "letter" without contents and an AddCommand object with parameter "Subsection", execute it and compare the contents of the returned document with those from the subsections's text.*

Test ID	AddLatexCommandTestSubsectionR
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "report" without contents and an AddCommand object with parameter "Subsection", execute it and compare the contents of the returned document with those from the subsections's text.*

Test ID	AddLatexCommandTestSubsectionO
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "other" and an AddCommand object with parameter "Subsection", execute it and compare the contents of the returned document with those from the subsections's text.*

Test ID	AddLatexCommandTestSubsubsectionA
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "article" without contents and an AddCommand object with parameter "Subsubsection", execute it and compare the contents of the returned document with those from the subsubsections's text.*

Test ID	AddLatexCommandTestSubsubsectionB
User Story	*User Story 3*
Test Class	*AddLatexCommand*

Description	*We create a document with type “book” without contents and an AddCommand object with parameter “Subsubsection”, execute it and compare the contents of the returned document with those from the subsubsections's text.*

Test ID	AddLatexCommandTestSubsubsectionL
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type “letter” without contents and an AddCommand object with parameter “Subsubsection”, execute it and compare the contents of the returned document with those from the subsubsections's text.*

Test ID	AddLatexCommandTestSubsubsectionR
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type “report” without contents and an AddCommand object with parameter “Subsubsection”, execute it and compare the contents of the returned document with those from the subsubsections's text.*

Test ID	AddLatexCommandTestSubsubsectionO
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type “other” and an AddCommand object with parameter “Subsubsection”, execute it and compare the contents of the returned document with those from the subsubsections's text.*

Test ID	AddLatexCommandTestBulletListA
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type “article” without contents and an AddCommand object with parameter “BulletList”, execute it and compare the contents of the returned document with those from the bulletlists's text.*

Test ID	AddLatexCommandTestBulletListB
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "book" without contents and an AddCommand object with parameter "BulletList", execute it and compare the contents of the returned document with those from the bulletlists's text.*

Test ID	AddLatexCommandTestBulletListL
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "letter" without contents and an AddCommand object with parameter "BulletList", execute it and compare the contents of the returned document with those from the bulletlists's text.*

Test ID	AddLatexCommandTestBulletListR
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "report" without contents and an AddCommand object with parameter "BulletList", execute it and compare the contents of the returned document with those from the bulletlists's text.*

Test ID	AddLatexCommandTestBulletListO
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "other" and an AddCommand object with parameter "BulletList", execute it and compare the contents of the returned document with those from the bulletlists's text.*

Test ID	AddLatexCommandTestEnumerationListA
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "article" without contents and an AddCommand object with parameter "EnumerationList", execute it and*

	compare the contents of the returned document with those from the enumerationlist's text.

Test ID	AddLatexCommandTestEnumerationListB
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "book" without contents and an AddCommand object with parameter "EnumerationList", execute it and compare the contents of the returned document with those from the enumerationlist's text.*

Test ID	AddLatexCommandTestEnumerationListL
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "letter" without contents and an AddCommand object with parameter "EnumerationList", execute it and compare the contents of the returned document with those from the enumerationlist's text.*

Test ID	AddLatexCommandTestEnumerationListR
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "report" without contents and an AddCommand object with parameter "EnumerationList", execute it and compare the contents of the returned document with those from the enumerationlist's text.*

Test ID	AddLatexCommandTestEnumerationListO
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "other" and an AddCommand object with parameter "EnumerationList", execute it and compare the contents of the returned document with those from the enumerationlist's text.*

Test ID	AddLatexCommandTestTableA

User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "article" without contents and an AddCommand object with parameter "Table", execute it and compare the contents of the returned document with those from the tables's text.*

Test ID	AddLatexCommandTestTableB
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "book" without contents and an AddCommand object with parameter "Table", execute it and compare the contents of the returned document with those from the tables's text.*

Test ID	AddLatexCommandTestTableL
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "letter" without contents and an AddCommand object with parameter "Table", execute it and compare the contents of the returned document with those from the tables's text.*

Test ID	AddLatexCommandTestTableR
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "report" without contents and an AddCommand object with parameter "Table", execute it and compare the contents of the returned document with those from the tables's text.*

Test ID	AddLatexCommandTestTableO
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "other" and an AddCommand object with parameter "Table", execute it and compare the contents of the returned document with those from the tables's text.*

Test ID	AddLatexCommandTestFigureA
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "article" without contents and an AddCommand object with parameter "Figure", execute it and compare the contents of the returned document with those from the figure's text.*

Test ID	AddLatexCommandTestFigureB
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "book" without contents and an AddCommand object with parameter "Figure", execute it and compare the contents of the returned document with those from the figure's text.*

Test ID	AddLatexCommandTestFigureL
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "letter" without contents and an AddCommand object with parameter "Figure", execute it and compare the contents of the returned document with those from the figure's text.*

Test ID	AddLatexCommandTestFigureR
User Story	*User Story 3*
Test Class	*AddLatexCommand*
Description	*We create a document with type "report" without contents and an AddCommand object with parameter "Figure", execute it and compare the contents of the returned document with those from the figure's text.*

2.4 Tests for User Story **<4>**

Test ID	EnableVersionManagementCommandTest
User Story	*User Story 4*
Test Class	*EnableVersionManagementCommand*
Description	*We create a document with type "article", an EnableVersionsManagementCommand, execute it, create an EditCommand, execute it and compare the contents of the document against the contents of the document before the edit action.*

Test ID	EnableVersionManagementCommandTestStable
User Story	*User Story 4*
Test Class	*EnableVersionManagementCommand*
Description	*We create a document with type "article", an EnableVersionsManagementCommand, execute it, create a VersionStrategy object with parameter "Stable", create an EditCommand, execute it and compare the contents of the document against the contents of the document before the edit action.*

2.5 Tests for User Story <5>

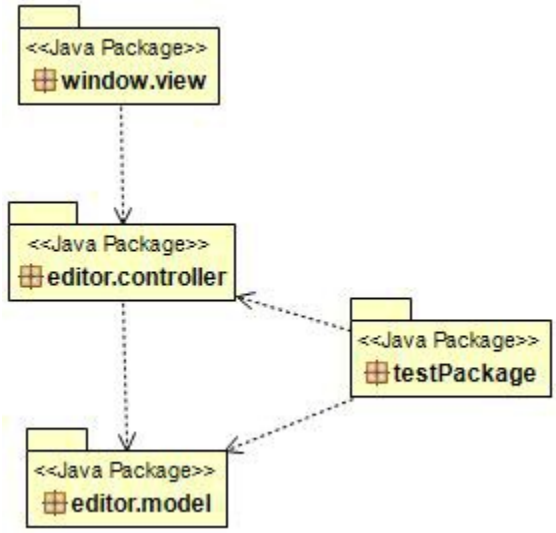
Test ID	ChangeVersionStrategyCommandTestVolatile
User Story	*User Story 5*
Test Class	*ChangeVersionStrategyCommand*
Description	*We create a VersionStrategy with type "Stable", a ChangeVersionsStrategy object with parameter "Volatile", execute it and compare both strategies if are the same types.*

Test ID	ChangeVersionStrategyCommandTestStable
User Story	*User Story 5*
Test Class	ChangeVersionStrategyCommand
Description	*We create a VersionStrategy with type "Volatile", a ChangeVersionsStrategy object with parameter "Stable", execute it and compare both strategies if are the same types.*

3 Design

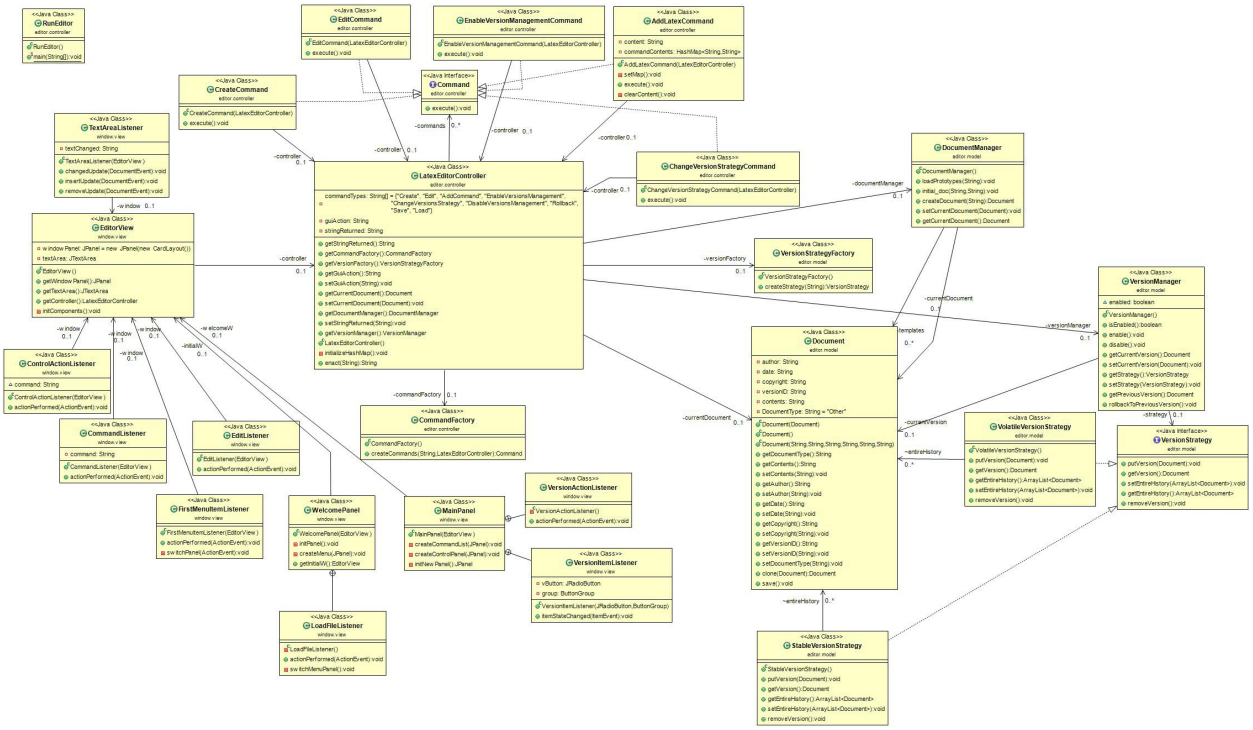
3.1 Architecture

<Specify the overall architecture for this release in terms of a **UML package diagram**.>



3.2 Design

<Specify the detailed design for this release in terms of **UML class diagrams**.>



<Document the classes that are included in this release in terms of CRC cards according to the template that is given below.>

Package *editor.controller:*

Class Name: LatexEditorController	
Responsibilities: ▪ Initialize commands and store them ▪ Retrieve and execute a command needed ▪ Keep track of current Document ▪ Keeps track of current GUI action ▪ Knows what text should be displayed on window ▪ Knows factories and managers	**Collaborations:** ▪ CommandFactory ▪ DocumentManager ▪ Command ▪ Document ▪ VersionManager ▪ VersionStrategyFactory

Class Name: AddLatexCommand	
Responsibilities: ▪ Inserts command contents in document's text when executed ▪ Initializes Map with Latex command contents	**Collaborations:** ▪ Document ▪ LatexEditorController

Class Name: ChangeVersionStrategyCommand	
Responsibilities: ▪ Changes VersionStrategy mechanism when executed	**Collaborations:** ▪ LatexEditorController ▪ VersionStrategy ▪ VersionManager ▪ VersionStrategyFactory

Class Name: CommandFactory	
Responsibilities: ▪ Creates and returns Command objects	**Collaborations:** ▪ LatexEditorController ▪ Command – CreateCommand, EditCommand, AddLatexCommand, EnableVersionManagementCommand, ChangeVersionStrategyCommand

Class Name: CreateCommand	
Responsibilities: ▪ Creates new Document when executed	**Collaborations:** ▪ LatexEditorController ▪ Document ▪ DocumentManager

Class Name: EditCommand	
Responsibilities: ▪ Updates document's contents when executed ▪ Save new version when mechanism is enabled	**Collaborations:** ▪ LatexEditorController ▪ Document ▪ VersionManager

Class Name: EnableVersionManagementCommand	
Responsibilities: ▪ Enables version tracking mechanism when executed ▪ Sets Volatile Strategy as default	**Collaborations:** ▪ LatexEditorController ▪ VersionStrategy ▪ VersionManager ▪ VersionStrategyFactory

Class Name: RunEditor	
Responsibilities: ▪ Creates main panel and controller	**Collaborations:** ▪ LatexEditorController, EditorView

Package *editor.model:*

Class Name: Document	
Responsibilities: ▪ Keeps track of document's contents, author, date, copyright, type and versionID ▪ Clones (deep) a given document	**Collaborations:**

Class Name: DocumentManager	
Responsibilities: ▪ Creates documents ▪ Updates current Document on editor ▪ Load and creates prototypes, save them in a map of templates	**Collaborations:** ▪ Document

Class Name: StableVersionStrategy	
Responsibilities: ▪ Stores entire version history as files ▪ Appends latest version to history ▪ Returns last stored version ▪ Sets and returns whole version history ▪ Deletes permanently a version from history	**Collaborations:** ▪ Document

Class Name: VersionManager	
Responsibilities: ▪ Keeps current version tracking strategy ▪ Enables and disables version tracking mechanism ▪ Rollbacks to previous version ▪ Keeps a reference to the current document version	**Collaborations:** ▪ Document ▪ VersionStrategy

Class Name: VersionStrategyFactory	
Responsibilities: ▪ Creates new VersionStrategy objects ▪	**Collaborations:** ▪ VersionStrategy – StableVersionStrategy, VolatileVersionStrategy

Class Name: VolatileVersionStrategy	
Responsibilities:	**Collaborations:**

▪ Stores entire version history in a list ▪ Appends latest version to history ▪ Returns last stored version ▪ Sets and returns whole version history ▪ Deletes permanently a version from history	▪ Document

Package *window.view:*

Class Name: CommandListener	
Responsibilities: ▪ Activates controller to add a Latex command when button clicked ▪ Displays pop-up window when trying to insert forbidden commands in document types	**Collaborations:** ▪ EditorView ▪ LatexEditorController

Class Name: ControlActionListener	
Responsibilities: ▪ Activates controller when *Save Version* button clicked ▪ Activates controller when *Load* button is pressed	**Collaborations:** ▪ EditorView ▪ LatexEditorController

Class Name: EditListener	
Responsibilities: ▪ Updates TextArea contents when JMenuItems *"Cut", "Copy", "Paste"* are pressed	**Collaborations:** ▪ EditorView

Class Name: EditorView	
Responsibilities: ▪ Sets frame settings	**Collaborations:** ▪ LatexEditorController

▪ Contains TextArea component ▪ Interacts with controller	▪ WelcomePanel ▪ MainPanel

Class Name: FirstMenuItemListener	
Responsibilities: ▪ Switches panels when *Create* button clicked	**Collaborations:** ▪ LatexEditorController ▪ EditorView

Class Name: MainPanel	
Responsibilities: ▪ Initializes main panel logic, contains buttons and menu items for every available action ▪ Listens to version tracking mechanism (de)activation ▪ Listens to version strategy changes	**Collaborations:** ▪ EditorView ▪ LatexEditorController

Class Name: TextAreaListener	
Responsibilities: ▪ Listens to all events that modify JTextArea's contents	**Collaborations:** ▪ EditorView

Class Name: WelcomePanel	
Responsibilities: ▪ Initializes welcome panel's structure ▪ Loads existing file into editor when *Load* menu item clicked	**Collaborations:** ▪ EditorView