

**Università degli Studi di Bergamo**

**Dipartimento di Ingegneria Gestionale, dell’Informazione e della trasmissione**

**VolText**

**User manual**

**Colombi Simone**

**Gambarara Alberto**

**Scarpellini Stefano**

**Indice**

[1. Introduzione 3](#_Toc71294394)

[2. Grammatica 4](#_Toc71294395)

[2.1. Descrizione generale 4](#_Toc71294396)

[2.2. Elementi di una pagina 7](#_Toc71294397)

[2.2.1. Immagini 7](#_Toc71294398)

[2.2.2. Testi 8](#_Toc71294399)

[2.2.3. Liste 9](#_Toc71294400)

[2.2.4. Div 10](#_Toc71294401)

[2.3. Scelte grammaticali 11](#_Toc71294402)

[2.4. Errori della grammatica 11](#_Toc71294403)

[3. Tool utilizzati 12](#_Toc71294404)

[4. Interfaccia utente 13](#_Toc71294405)

[4.1. Caso d’uso generale 13](#_Toc71294406)

[4.2. Reliability 15](#_Toc71294407)

[5. Possibili sviluppi futuri 17](#_Toc71294408)

# Introduction

VolText is a project developed in the context of the course of Formal Languages and Compilers.

The goal is the design and implementation of a Java library for generating a PDF document and a graphical interface that allows simplified use of it. The most inherent part of the course was the creation of a grammar that describes the structure of a PDF document using the typical constructs of HTML syntax, such as div, images, texts and lists, but using a syntax similar to JSON.

The library is particularly focused on developing documents that have simple graphics such as flyers, small announcements, notices, business cards, etc.

The tool doesn’t want and doesn’t act as a replacement for LaTeX but takes the essential elements, creating a different version both in terms of language and in terms of components, eliminating formulas, graphics, tables, etc. not useful for the development of graphic documents.

1. **Grammar**

## 2.1. General description

The grammar has been designed and implemented thinking about who works with web languages and are familiar with HTML/JSON.

Using the typical syntax of the JSON language for the construction of custom tags, the grammar has been structured as a PDF document composed of various elements that can be drawn inside in positions and sizes at will.

The grammar is reported and, for each rule, its function is described.

**grammar** VolText;

@header {

**package** antlr;

}

pdf : *A* pdfattr\* (stylesheet)? (page)+ *C*;

pdfattr: 'title:' *STRING* *ENDNLINE*

| 'author:' *STRING* *ENDNLINE*

| 'path:' *STRING* *ENDNLINE*;

stylesheet: 'stylesheet' *O* element\* *C*;

element: '@' *STRING* *O* attrStyle\* *C*;

attrStyle: 'cross-point:' *NVAL* *ENDNLINE*

| 'shape:"' ('RECTANGLE' | 'CIRCLE' | 'TRIANGLE') *ENDLINE*

| ('fit-x' | 'fit-y') ':' *TFVAL* *ENDNLINE*

| ('pos-x' | 'pos-y') ':' *NOTVAL*? *NVAL* (*UNIT*)? *ENDNLINE*

| 'angle-rotation' ':' *NOTVAL*? *NVAL* *ENDNLINE*

| ('height' | 'width') ':' *NVAL* (*UNIT*)? *ENDNLINE*

| ('p\_height' | 'p\_width') ':' *NVAL* *ENDNLINE*

| 'ordered:' *TFVAL* *ENDNLINE*

| 'bullet:' *STRING* *ENDNLINE*

| ('font-family:' | 'font-family-ttf:' | 'font-family-otf:') *STRING*

*ENDNLINE*

| 'font-size:' *NVAL* *ENDNLINE*

| ('bold:' | 'italics:' |'underline:') *TFVAL* *ENDNLINE*

| 'colorT-bullet:' *STRING* *ENDNLINE*

| 'color-bullet:' *COLORVAL* *ENDNLINE*

| 'colorT:' *STRING* *ENDNLINE*

| 'color:' *COLORVAL* *ENDNLINE*

| 'position:' *POSVAL* *ENDNLINE*

| 'alignment:' *ALIGNVAL* *ENDLINE*

| 'orientation:' *ORIENTATION* *ENDNLINE*

| 'oob:' *TFVAL* *ENDNLINE*

| 'format:' *FORMATVAL* *ENDNLINE*;

page: 'page' *O* pageattr\* pae\* *C*;

pae: elemd | div;

div: 'div' *O* (color | idval | imganumber | elemd | positionv | fitAttr |

figure | tvalue)\* *C*;

tvalue: 'cross-point:' *NVAL* *ENDNLINE*;

figure: 'shape:"' ('RECTANGLE' | 'CIRCLE' | 'TRIANGLE') *ENDLINE*;

elemd: text

| list

| img;

img: 'img' *O* imgattr\* imgElem imgattr\* *C*;

imgattr: (idval | imganumber | positionv | fitAttr);

imgElem: 'URL:' *STRING* *ENDNLINE*;

list: 'list' *O* (listattr | listElem)\* *C*;

listElem: 'item:' *STRING* *ENDNLINE*;

text: 'text' *O* txtattr\* txtElem\* txtattr\* txtElem\* txtattr\* *C*;

txtattr: (color | idval | imganumber | positionv | alignment | txtval |

fitAttr);

txtElem: 'string:' *STRING* *ENDNLINE*;

//ATTRIBUTES

fitAttr: ( 'fit-x'

| 'fit-y') ':' *TFVAL* *ENDNLINE*;

imganumber: ( 'pos-x'

| 'pos-y' ) ':' *NOTVAL*? *NVAL* (*UNIT*)? *ENDNLINE*

| 'angle-rotation' ':' *NOTVAL*? *NVAL* *ENDNLINE*

| ('height'

| 'width') ':' *NVAL* (*UNIT*)? *ENDNLINE*;

idval: 'id:' *STRING* *ENDNLINE*;

listattr: 'ordered:' *TFVAL* *ENDNLINE*

| 'bullet:' *STRING* *ENDNLINE*

| idval

| imganumber

| txtval

| positionv

| color

| colorBullet

| fitAttr;

txtval: ('font-family:'

| 'font-family-ttf:'

| 'font-family-otf:') *STRING* *ENDNLINE*

| 'font-size:' *NVAL* *ENDNLINE*

| ('bold:'

| 'italics:'

| 'underline:') *TFVAL* *ENDNLINE*;

colorBullet: 'colorT-bullet:' *STRING* *ENDNLINE*

| 'color-bullet:' *COLORVAL* *ENDNLINE*;

color: 'colorT:' *STRING* *ENDNLINE*

| 'color:' *COLORVAL* *ENDNLINE*;

positionv: 'position:' *POSVAL* *ENDNLINE*;

alignment: 'alignment:' *ALIGNVAL* *ENDLINE*;

pageattr: 'orientation:' *ORIENTATION* *ENDNLINE*

| 'oob:' *TFVAL* *ENDNLINE*

| 'p\_width' ':' *NVAL* *ENDNLINE*

| 'p\_height' ':' *NVAL* *ENDNLINE*

| 'format:' *FORMATVAL* *ENDNLINE*

| idval;

//TERMINALI

*NOTVAL*: '-';

*UNIT*: 'mm' | '%' | 'pt';

*FORMATVAL*: 'A0'

| 'A1'

| 'A2'

| 'A3'

| 'A4'

| 'A5'

| 'A6';

*ORIENTATION*: ('hor' | 'ver');

*COLORVAL*: '#' ([0-9a-fA-F][0-9a-fA-F][0-9a-fA-F][0-9a-fA-F][0-9a-fA-F]

[0-9a-fA-F][0-9a-fA-F][0-9a-fA-F]);

*ALIGNVAL*: 'left'

| 'center'

| 'right'

| 'justify';

*TFVAL*: 'true'

| 'false';

*POSVAL*: 'lt'

| 'ct'

| 'rt'

| 'lc'

| 'cc'

| 'rc'

| 'lb'

| 'cb'

| 'rb' ;

*NVAL*: [0-9]+('.'[0-9]+)?;

*ENDLINE*: '";';

*ENDNLINE*: ';';

*O* : ':{';

*A* : '{';

*C* : '}';

*STRING*: '"' (~[\t\r\n])+ '"';

*WS* : [ \t\r\n]+ **->** skip ; // skip spaces, tabs, newlines

The first rule, *pdf*, allows the opening of the document, the insertion of attributes such as title, author and saving path of the pdf, relating to the document, the possibility to insert a part of the document style, one or more pages and the closing of the document.

A summary structure of the *stylesheet* tag was made as

stylesheet:{

@IdElement1:{

<< attributes for that element >>

}

}

The page can contain various attributes such as orientation, oob (out-of-bound), width, height and format, as specified in the following table:

|  |  |
| --- | --- |
| **attributo** | **valori possibili** |
| orientation | hor, ver |
| oob | true,false |
| p\_width | decimal number |
| p\_height | decimal number |
| format | A0, A1, A2, A3, A4, A5, A6 |

Each page contains inside divs, therefore blocks that may also contain other elements, or elements themselves such as images, lists and texts. These elements and their attributes will be treated on *chapter 2.2*.

A page, as also reported in the ISO 32000-1 specification, section 11.4.7, must have a white background color. So, if you want to completely color a page, you can insert a div with dimensions fitted respect to the page or insert other elements (in this version, given the choice of not having internal containers into others, it is not possible to create different ones in a page with color of background different than white, precisely because it is already a colored container).

## 2.2. Page elements

Now we treat the various elements that can compose a page. We will treat the images, texts and lists that can be inserted referring to the size of the page. Finally, we will discuss the div, more complex elements and which may contain within them the other types of item, but with measuring references related to the div in which they are encapsulated. Each attribute of each element must always be closed by a ; (semicolon) while the closure of the elements is realized by a right brace.

### 2.2.1. Images

An image must be inserted through the "*img: {<<attributes>>}*" tag.

It may contain attributes placed in position even after the only mandatory attribute, namely the URL. The *url* is a string that specifies the relative or absolute path of that image inside the computer. If relative, the reference will start from the folder where the text document is located and, as we will see later, if a document is not opened but only the text is pasted into the GUI, we will have errors regarding the lack of the object at the specified address.

The various attributes available for images are now listed:

* *id*, the identifier of the element inside the document. This has been designed for use especially in the style sheet.
* *pos-x* and *pos-y*, indicate the position of the element on the horizontal and vertical axis. The value can be positive or negative and also contain the unit of measurement (millimeters (mm), percentage (%) or points (pt), by default mm).
* *angle-rotation*, indicates the rotation of the element respect to its center in degrees. If the specified value is positive, a counterclockwise rotation will be performed, on the contrary with a negative value a clockwise rotation will be performed.
* *height* and *width*, indicate the height and width of the image in case you need to resize the element contained on the page or div. They too can be expressed in millimeters, percentages or points.
* *position*, contains a pair of letters that allow you to insert the element in its container (page or div) in a predetermined position (imagining a subdivision into 9 squares of the container). They can therefore be *l*, *c* and *r* to indicate whether the element must be respectively on the left, centered or right in the container, and *t*, *c* or *b* to indicate whether the element must be at the top (*t*), central (*c*) or bottom (*b*).
* *fit-x* and *fit-y*, contains a boolean value (true, false) to indicate if the element must be resized over the entire size of the container (be it the page or the div) respect to the x or y axis.

### 2.2.2. Texts

A text can be entered via the tag “*text:{ <<attributes>> }”*.

It can contain attributes mixed with text strings without precedence constraints.

The strings/chapters are contained in "*string*" attributes and are written in the PDF in the order that is used in writing of the *text* element.

The attributes of the text element are the following:

* *id,* as already described for the images, identifies the object.
* *colorT* and *color*, *,* attributes that report the color of the text (unique for all strings). *color* specifies a color in hexadecimal #RGBA format, where R is the red color, G the green color, B the blue color and A the alpha, which is the transparency value of the element. In the texts and in the lists, the alpha is not considered, contrary to what happens for the div. *colorT* specifies a color in the form of text (black, blue, red, green, …). The last attribute that is declared will be used in the drafting of the document.
* *pos-x, pos-y, angle-rotation, height, width, position, fit-x* and *fit-y*: see the discussion in chapter 2.2.1 on images.
* *alignment*,used for aligning the text (*left*, *center*, *right* and *justify*).
* *font-family, font-family-ttf* and *font-family-otf*, *,* specify the font to be used in the drafting of the document. If you are using *font-family* you can use standard fonts such as helvetica, courier and times. If you are using *font-family-ttf*, then you can specify the path where the file with extension ttf (True Type Font) that describes the font to use is available. If the folder where the font resides also contains the relative font in bold, italic and / or bold italic then the path to specify will be that of the regular font (ending with Regular.ttf, and in which the other files will end with Bold.ttf, Italic.ttf and BoldItalic.ttf, respectively for bold, italic and bold italic): in this case any texts or parts of text in bold (written as *\bold text in bold \bold*), italic (written as *\italic text in italic \italic*) or bold italic (written as *\bold \italic text in bold italic \italic \bold*). The same rules work to the font-family-otf attribute, to which fonts with the extension otf (Open Type Font) apply.
* *font-size*, contains the font size to use in the texts.
* *bold, italic* and *underline*, are boolean attributes that allow you to specify whether the entire text should be written in bold, italic or underlined (the latter specification has been implemented but not completed due to difficulties in integrating the PDFBox-layout library with PDFBox).

### 2.2.3. List

Una lista può essere inserita attraverso il tag *list:{ <<attributi>> }*

Essa può contenere attributi misti a stringhe di testo (voci della lista, *item*) senza vincoli di precedenza.

Gli attributi di una lista possono essere:

* *id*, come i precedenti elementi
* *ordered*, che serve per segnalare se è una lista ordinata di elementi (lista numerica) oppure se è una lista non ordinata (lista a punti)
* *bullet*, specifica il tipo di “punto” che fa da inizio dell’elemento della lista. I possibili punti sono *odd* (pallino pieno, ), *even* (doppia freccia, ), oppure *new(Bullet)* (in cui il *Bullet* è una stringa decisa dall’utente e usabile come punto).
* tutti gli attributi relativi ai testi, tranne l’allineamento, sono implementati anche per le liste
* *colorT-bullet* e *color-bullet* , attributi che segnalano il colore del bullet (univoco per tutti i bullet). *color-bullet* specifica un colore in formato *#RGBA* esadecimale, dove R è il colore rosso, G il colore verde, B il colore blu e A l’alpha, ovvero il valore di trasparenza dell’elemento. Il valore di alpha non è considerato. *colorT-bullet* specifica un colore sotto forma di testo (black, blue, red, green, …). L’ultimo attributo che viene dichiarato verrà usato nella redazione del documento.

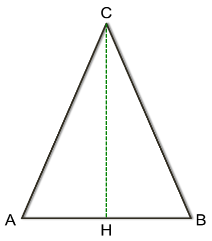
### 2.2.4. Div

Un div può essere inserito attraverso il tag *div:{ <<attributi e/o elementi>>}*.

Il div, come descritto anche in html, è un contenitore che permette di raggruppare elementi al suo interno. In questa versione del progetto un div può contenere solo immagini, testi e liste ma non altri contenitori. Il contenitore può essere personalizzato attraverso vari attributi e gli elementi che sono al suo interno prendono come riferimento dimensionale la dimensione del contenitore.

Esso può contenere attributi misti a elementi (*img*, *text* e *list*) senza vincoli di precedenza.

Si analizzano ora i vari attributi di un contenitore:

* *id,* come ogni elemento, specifica un identificativo che permette di riconoscere l’elemento.
* *color* e *colorT*, come già presentato precedentemente per quanto riguarda i testi e le liste, specifica il colore di riempimento del contenitore. Non è stata realizzata una suddivisione in colore di riempimento e colore del bordo, che potrà essere inserita in futuri sviluppi. *color* specifica il colore in formato *#RGBA* e in questo caso l’alpha (A) specifica la trasparenza del contenitore, per fornire la possibilità di vedere altri elementi sottostanti ad esso. *colorT* invece permette l’uso di colori attraverso l’uso di nomi di colori.
* *pos-x*, *pos-y*, *angle-rotation*, *height*, *width*, *fit-x* e *fit-y*, come già detto nelle precedenti sezioni, specificano la posizione, l’angolo di rotazione, le dimensioni e l’adattamento rispetto all’asse orizzontale e verticale.
* *position*, specifica la posizione del contenitore rispetto alla pagina nel quale viene inserito senza bisogno di specificare le coordinate spaziali
* *shape*, specifica la forma del contenitore, tra *RECTANGLE*, *CIRCLE* e *TRIANGLE*. Nel caso del rettangolo verranno usati i valori di posizione e dimensione per il disegno di esso. Nel caso del cerchio verranno utilizzati i valori di posizione e dimensione per disegnare un cerchio o una ellisse. Nel caso del triangolo oltre a posizione e dimensione sarà necessario l’inserimento di un attributo *cross-point* espresso come percentuale che specifica il punto H specificato in figura:
  1. **Scelte grammaticali**

La principale scelta è dovuta alla grammatica elaborata per implementare il foglio di stile. Si tratta di una scelta legata alla comodità di implementare gli attributi direttamente come figli di *attrStyle*, rispetto a implementarli come figli degli attributi degli elementi presenti nelle pagine.

Per come la grammatica legge il foglio, la posizione dello *stylesheet* come seconda parte del non terminale *pdf* non permette di avere una idea dell’elemento al quale le specifiche si riferiscono, dato che l’id presente nel foglio di stile fa riferimento a un oggetto generico, e quindi non verifica la correttezza dei singoli attributi. Non avendo controlli di alcun tipo sullo stylesheet, è possibile l’utilizzo di un id per più elementi di natura diversa e di poter inserire attributi anche se non appartenenti al tipo di oggetto che si vuole personalizzare.

* 1. **Errori della grammatica**

Trattiamo ora i vari errori e le varie situazioni che possono capitare nella redazione di un documento di testo e che il programma rileva e interpreta. In nessun caso il programma segnala un errore ma mostra solo avvisi per segnalare un errore di battitura o di tipo lessicale.

Il primo caso di errore (che verrà anche trattato come esempio di errore nel capitolo 4) è il caso di una dimenticanza di un punto e virgola (o una parentesi graffa). In questo caso viene riportato all’utente che in una specifica linea/posizione è mancante un simbolo ma il sistema riesce comunque a proseguire nella redazione. Questo è il tipico caso di un errore lessicale.

Lo stesso vale anche se un attributo non appartiene a un elemento (ad esempio un attributo *bullet* per il tag *img*). Questo è il tipico caso di errore sintattico.

Una segnalazione che il programma restituisce è anche riferita, identificandosi con l’*url* dell’immagine, all’errato percorso dove si trova l’immagine. Infatti, il sistema specifica che non è riuscito a leggere l’immagine in quello specifico path. Il sistema però evita il caricamento dell’immagine e prosegue con la continuazione dell’analisi del testo.

Lo stesso “errore” viene riportato se il path specificato non è relativo a un file per l’uso di font specifici. Questi ultimi due esempi son riferibili a errori semantici.

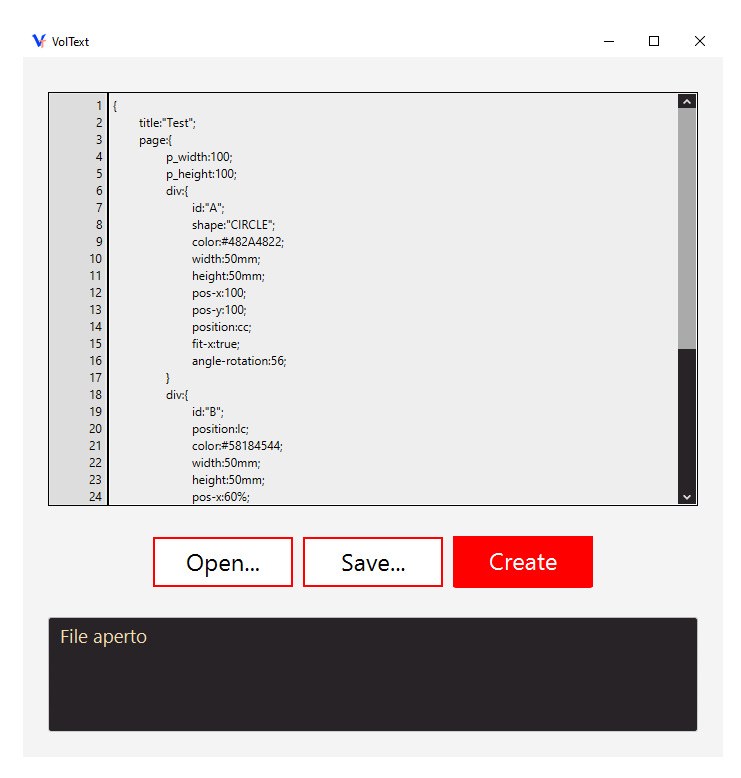
Altre segnalazioni vengono effettuate anche se la posizione inserita dell’elemento non è corretta o esterna alla pagina, se la pagina è specificata sia come formato che come dimensioni (in quel caso viene preso il formato della pagina) o se le dimensioni sono maggiori rispetto al contenitore che contiene l’elemento.

# Tool utilizzati

Per lo sviluppo sono stati utilizzati diversi strumenti:

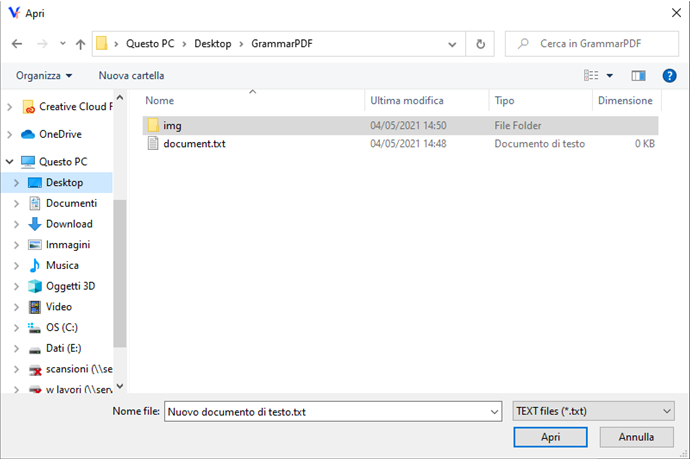
* GitHub: GitHub è un servizio di Version Control utilizzato nel nostro progetto per tenere traccia delle modifiche e dei mutamenti implementati nel nostro software. Ciò ha aiutato anche la condivisione quasi real-time di codice per un lavoro di gruppo migliore.
* GitHub Desktop: client che permette un utilizzo semplificato e rapido di GitHub, utile anche per tenere traccia dei cambiamenti e delle versioni del progetto.
* ANTLR 4: ANTLR (ANother Tool for Language Recognition) è un potente generatore di parser per leggere, elaborare, eseguire o tradurre testo strutturato o file binari. Da una grammatica, ANTLR genera un parser che può costruire e analizzare alberi di analisi. La versione 4 è stata utilizzata per stare al passo con le versioni più recenti e per una comodità con l’IDE Eclipse, vista la presenza di un plugin apposito.
* Eclipse: IDE per lo sviluppo della libreria e della GUI, che supporta anche vari plugin per ANTLR e JavaFX.
* PDFBox: libreria java per la creazione e modifica di documenti PDF.
* PDFBox-layout: libreria java aggiuntiva a PDF-box che permette una gestione (non del tutto compatibile) semplificata di testi e liste.
* JavaFX: libreria che permette l’implementazione di una interfaccia grafica per un programma java. In particolare, per Eclipse, è stato utilizzato il plugin e(fx)clipse che permette la creazione di progetti JavaFX e l’utilizzo delle librerie annesse.
* Launch4j: software per l’incapsulamento di eseguibili di tipo exe altamente personalizzabili a partire da eseguibili di tipo jar.

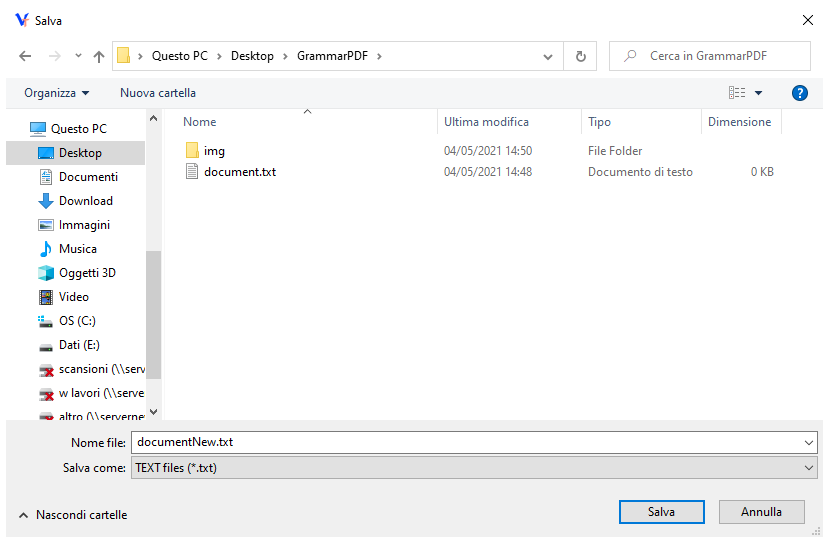
1. **User interface**

The application can be used through a graphical interface developed with JavaFX. At the execution it comes with a GUI consisting of a text area where the user can write and edit the grammar of a PDF document, two buttons similar to each other, respectively for opening and saving a text file, a red button for creating the document, and a non-editable text area that is used to communicate any messages and errors to the user.

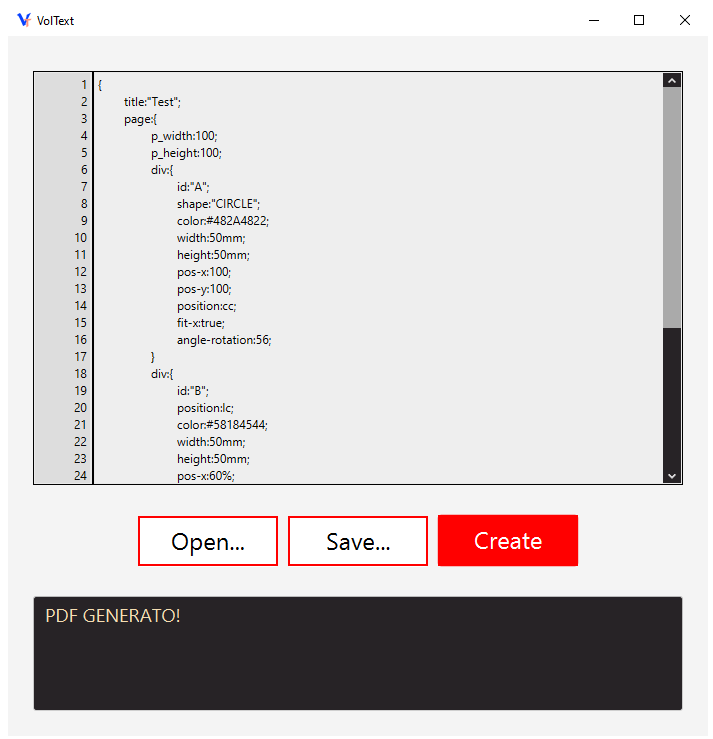
* 1. **General use case**

The user can write the grammar from scratch, but the most common use case is certainly the one in which the grammar, or a part of it, is already in an existing text file. In this case, with the "Open ..." button is possible to select the file to be loaded through a file system navigation interface, receiving a confirmation message.



 After opening the file, the user can make all the necessary changes to the grammar, and by pressing the "Save ..." button it can be saved to a file using an interface similar to the one used during the loading process.

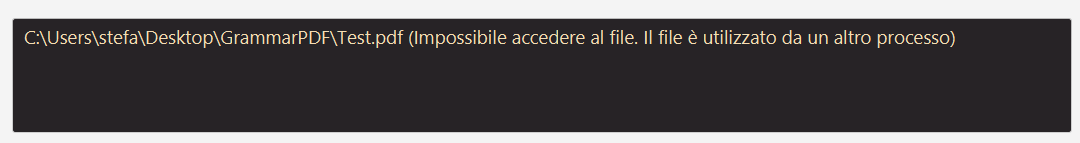
When the user is ready to start generating the document, it can be done by pressing the "Create" button and, at the end of the procedure, if it is successful, a confirmation message is shown, in addiction to any warnings that need to be reported to the user.



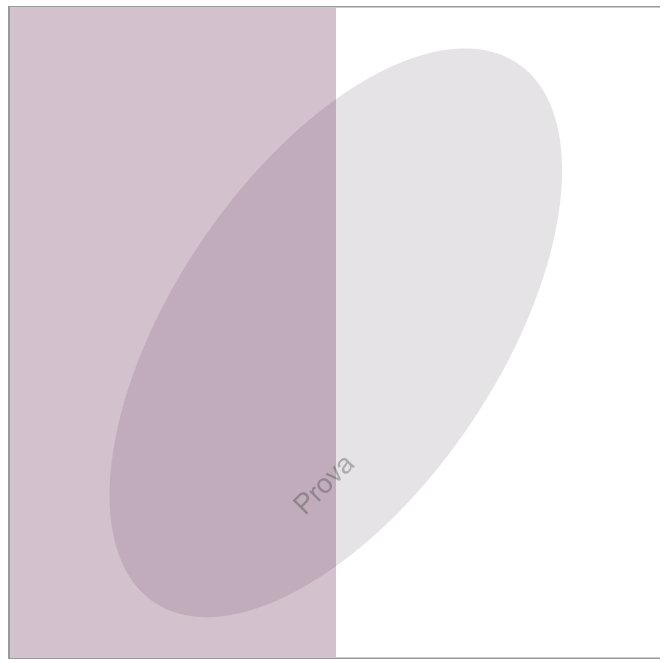
## Reliability

The application was developed to tolerate incorrect usage by the user without compromising the execution of the software, but instead showing clear error messages that can help the user to understand the cause.

For example, if the user, while opening a document, enters the name of a non-existent file, in the first version, he returned the message "The specified file cannot be found" in the textual error area, while currently the The system explorer helps to check the existence of the document. Another example is the situation where the destination path of the pdf file already exists and is used by another process; In this case, the message “Cannot access the file. The file is being used by another process”.

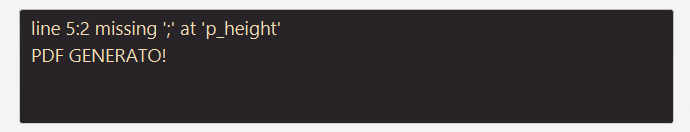


A possible example of grammar is the following one, which describes a simple PDF document called Test containing a single page of 100x100 size, two coloured containers and a rotated text:



It is interesting to note that the application generates a PDF document even with the presence of errors in the grammar, however, returning a warning message to the user.

For example, by eliminating the semicolon after 'p\_width: 100' in the previous grammar, the software produces a correct document, but reporting the error through the graphical interface.



# Possible future developments

During the development of the application, some interesting possible additions emerged for the expansion of the project. Here are exposed some ideas:

* The *underline* attribute, even if implemented, does not work correctly due to effective methods in the PDFBox library and incompatibility between PDFBox and PDFBox-layout.

The generation of a function that allows to underline text with a more simple mode in terms of programming and calculation is in future plans.

* The implementation of constructs such as tables, links, mathematical formulas and graphs.
* A possible improvement in grammar such as simplification of constructs, id without double quotes, colour management also through other formats.
* Code optimization, especially for the VolTextListener.java class.
* Management of containers within other containers, management of objects internal to non-rectangular containers, management inside the grammar of particular constructs to reduce the document drafted by the user.
* Implementation of auxiliary tools for previewing the document before the generation.