

# Leggere, modificare, e scrivere i PDF

Tutti hanno bisogno di realizzare o modificare documenti in formato PDF: è tipicamente una delle funzioni più richieste all'interno di una applicazione qualsiasi. Soprattutto per quanto riguarda il desktop, mercato in cui i clienti principali sono aziende e pubblica amministrazione, che ovviamente utilizzano i computer per produrre documenti digitali proprio in formato PDF. Questo perché il Portable Document Format inventato da Adobe nel 1993, e le cui specifiche sono open source e libere da qualsiasi royalty, è lo standard universale ormai accettato da qualsiasi sistema operativo e su qualsiasi dispositivo per la trasmissione di documenti. Proprio perché il formato è utilizzabile gratuitamente da chiunque in lettura e scrittura, è stato inserito in praticamente qualsiasi programma ed è così conosciuto dal grande pubblico. Ormai chiunque sa cosa sia un PDF, e qualsiasi utente vorrà poter archiviare informazioni in questo formato. È quindi fondamentale essere in grado di scrivere programmi che possano lavorare con i PDF, altrimenti si resterà sempre un passo indietro. Il problema è che il formato PDF è abbastanza complicato da gestire, ed è quindi decisamente poco pratico realizzare un proprio sistema per leggere e scrivere questi file. Bisogna basarsi su delle apposite librerie, e ne esistono varie, anche se purtroppo spesso non sono ben documentate come l'importanza dell'argomento richiederebbe, e chi si avvicina al tema rischia di non sapere da dove iniziare. Per questo motivo, abbiamo deciso di presentarvi un metodo per leggere e uno per creare PDF multipagina, con le principali caratteristiche dei PDF/A.

Come programma di esempio, abbiamo realizzato una interfaccia grafica per gli OCR Tesseract e Cuneiform, capace di

funzionare sia su Windows che su GNU/Linux e MacOSX. Per motivi di spazio e di pertinenza, non presenteremo tutto il codice del programma ma soltanto le parti relative alla manipolazione dei PDF. Trovate comunque il link all'intero codice sorgente alla fine dell'articolo.



Le librerie Qt, sulle quali si basa non soltanto l'interfaccia grafica multiplatforma del nostro programma di esempio, ma anche lo strumento di scrittura dei PDF, sono rilasciate con [due licenze libere e una commerciale](#). Le due licenze libere sono GNU GPL e GNU LGPL: in entrambe i casi sono completamente gratuite, la differenza è che la prima richiede la pubblicazione dei programmi basati sulle Qt con la stessa licenza (quindi si deve fornire il codice sorgente), mentre la LGPL permette di utilizzare le librerie pur non distribuendo il codice sorgente del proprio programma. L'opzione GPL è valida per tutte le applicazioni che verranno rilasciate come software libero da programmatori amatoriali, mentre la LGPL è più indicata per aziende che non vogliono rilasciare il programma come free software. La licenza commerciale serve solo nel caso si voglia modificare il codice sorgente delle librerie Qt stesse senza pubblicare il codice delle modifiche.

Il programma è scritto in C++ con le librerie multiplatforma Qt, delle quali ci serviremo per scrivere i PDF usando le funzioni della classe QPDFWriter. Per la lettura dei PDF, invece, utilizzeremo la libreria libera e open source Poppler, che si integra perfettamente con le librerie Qt.

## Table of Contents

- [Come funziona il formato PDF?](#)
- [Includere Poppler](#)

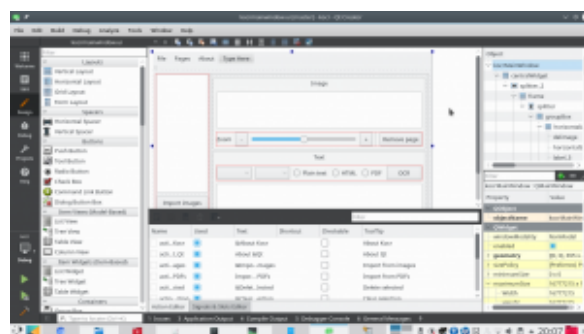
- [Aprire un PDF in lettura](#)
- [Fusione dei PDF](#)
- [Le varie TextBox](#)
- [Scrivere dell'HTML](#)
- [Il codice sorgente e il binario dell'esempio](#)

## Come funziona il formato PDF?

Il formato PDF è uno standard ufficiale dal 2007, declinato in una serie di sottoformati: A,X,E,H,UA, a seconda dei vari utilizzi che se ne vogliono fare. Quello che si segue solitamente è il PDF/A, progettato per l'archiviazione dei documenti anche a lungo termine: è pensato per integrare tutti i componenti necessari. Prima che si stabilisse questo standard, infatti, i PDF non erano davvero adatti a conservare e trasmettere documenti, perché mancavano spesso alcuni componenti fondamentali. Per esempio, se un PDF veniva visualizzato su un computer nel quale non erano installati i font con cui sul PC originale era stato scritto il testo, tutta l'impaginazione saltava. Ora, invece, i font possono essere integrati, assieme ad eventuali altri oggetti, così è possibile visualizzare correttamente un PDF/A su qualsiasi dispositivo, a prescindere dal suo sistema operativo. Questo significa che ogni PDF moderno è di fatto un po' più grande di quanto lo sarebbe stato un PDF degli anni '90, perché porta al suo interno i vari font, ma questo non è un problema considerando che il costo dello spazio dei dischi rigidi diminuisce continuamente e un paio di kilobyte in più in un file non si notano nemmeno.

Il formato PDF nasce da un formato precedente che è tutt'ora in uso e che si chiama PostScript. PostScript è di fatto un linguaggio di programmazione che permette di descrivere delle pagine: i file PS sono dei semplici file di testo che contengono una serie di istruzioni per il disegno di una pagina, con le sue immagini e il testo. Si tratta di un

linguaggio che va interpretato, quindi la sua elaborazione richiede una buona quantità di risorse e di tempo. Un file PDF, invece, è di fatto una sorta di PS già interpretato, il che permette di risparmiare tempo. Per fare un esempio, in un file PS si troveranno molte condizioni "if" e cicli "loop", e si tratta di istruzioni che consumano molte risorse quando vanno interpretate. Nei PDF, invece, viene direttamente inserito il risultato dei vari cicli, così da risparmiare tempo durante la visualizzazione. Quello che è importante capire è che il formato PDF è progettato per la stampa, è pensato per essere facilmente visualizzato e stampato allo stesso modo su qualsiasi dispositivo. Insomma, una funzione di sola lettura. Non è affatto progettato per permettere la continua modifica dei file. Ciò non significa che sia proibito, i file PDF possono ovviamente essere modificati come qualsiasi altro file, ma la modifica può essere molto complicata da fare in certi casi proprio perché le informazioni vengono memorizzate puntando a massimizzare l'efficienza della lettura, non della scrittura o della modifica. Per esempio, i testi vengono memorizzati una riga alla volta, e non in blocchi di paragrafi o colonne, come invece risulterebbe comodo per modificarli successivamente. Un'altra differenza importante è che nei PDF ogni pagina è un elemento a se stante, mentre nei PostScript le pagine sono legate e condividono alcune caratteristiche (come le dimensioni).



Utilizzando l'[IDE gratuito QtCreator](#) è molto facile anche disegnare

l'interfaccia grafica  
multipiattaforma

## Includere Poppler

Cominciamo subito col nostro programma di esempio. Le librerie necessarie possono essere incluse nell'intestazione del codice come da prassi del C++. Quelle che servono per la gestione dei PDF sono le seguenti:

Per scrivere i PDF utilizzeremo infatti la libreria `QpdfWriter`, che si trova nella stessa cartella di tutte le altre librerie Qt, e che quindi viene trovata in automatico dall'IDE. Per la lettura dei PDF, invece, useremo Poppler, che va installata a parte. Qui le cose cambiano un po', perché mentre in GNU/Linux esiste un percorso standard nel quale installare le librerie, e quindi si può facilmente trovare poppler nella cartella `poppler/qt5/`, su Windows questo non esiste. Quindi, sfruttando gli `ifdef` forniti dalle librerie Qt, possiamo distinguere la posizione dei file che contengono la libreria Poppler a seconda del fatto che il sistema sia Windows (`Q_OS_WIN`) o GNU/Linux (`Q_OS_LINUX`). La posizione delle librerie per Windows potrà essere stabilita nel file di progetto, che vedremo più avanti. Possiamo ora cominciare a vedere il codice: non lo vedremo tutto, solo le parti fondamentali per la gestione dei PDF.



Entro breve, le librerie Qt integreranno direttamente una classe per la lettura dei PDF, chiamata [QPDFDocument](#), senza quindi la necessità di usare Poppler. Al momento tale classe non è ancora considerata stabile, quindi abbiamo deciso di presentare questo articolo basandoci ancora su Poppler. Quando

il rilascio di QtPdf sarà ufficiale, la presenteremo in nuovo articolo.

La prima funzione che implementiamo dovrà permettere l'importazione dei PDF. Infatti, vogliamo permettere agli utenti di importare dei PDF scansionati, in modo da poter eseguire su di essi l'OCR e ricavare il testo.

Chiamando la funzione **getOpenFileNames** di **QFileDialog** si visualizza una finestra standard per consentire all'utente la selezione di più file, il cui percorso completo viene inserito in una lista di stringhe che chiamiamo **files**.

Possiamo anche creare una MessageBox per chiedere conferma all'utente, così se dovesse avere scelto i file per sbaglio potrà annullare il procedimento prima di cominciare a lavorare sui file (operazione che può richiedere del tempo).

Banalmente, se il pulsante premuto dall'utente è **Cancel**, allora interrompiamo la funzione. Altrimenti, con un semplice ciclo for scorriamo tutti gli elementi della lista di file, passandoli uno alla volta a una funzione che si occuperà di estrarre le pagine dal PDF e aggiungerle alla lista delle pagine su cui lavorare.

## Aprire un PDF in lettura

Abbiamo chiamato la funzione che opera effettivamente l'estrazione delle pagine da un PDF, **addpdftolist**.

Questa funzione comincia controllando che il file che ha ricevuto come argomento **pdfin** sia esistente (**QFileInfo.exists** controlla che il file esista e non sia vuoto).

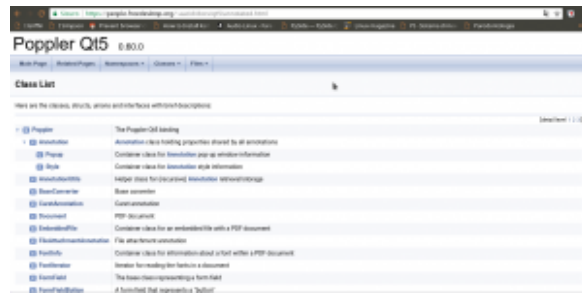
Ora abbiamo bisogno di una cartella temporanea, nella quale inserire tutte le immagini che estrarremo dalle pagine del PDF. La libreria **QTemporaryDir** si occupa proprio di creare una cartella temporanea a prescindere dal sistema operativo. Possiamo memorizzare il percorso di tale cartella in una stringa che chiamiamo **tmpdir**. Dobbiamo anche specificare che la cartella non va sottoposta all'auto rimozione, altrimenti il programma cancellerà la cartella automaticamente al termine di questa funzione, mentre noi ne avremo ancora bisogno in altre funzioni. La cancellazione di tale cartella potrà essere fatta manualmente alla chiusura definitiva del programma.

Siamo finalmente pronti per leggere il PDF. Basta creare un oggetto di tipo **Poppler::Document**, usando la funzione `load` che permette per l'appunto la lettura di un file PDF. Se il PDF non conteneva un documento valido, conviene terminare la funzione con l'istruzione `return` per evitare problemi.

Il documento potrebbe avere più pagine, quindi utilizziamo un ciclo `for` per leggerle tutte una alla volta.

Ogni pagina può essere estratta usando un oggetto **Poppler::Page**, e con l'apposita funzione `page` di un documento. Se la pagina è invalida, il ciclo si ferma.

La pagina può poi essere renderizzata in una immagine, rappresentata dall'oggetto **QImage**, secondo una carta risoluzione orizzontale e verticale (che di solito coincidono, ma non sempre).



Nel nostro programma, consideriamo tale risoluzione pari a 300 dpi, e l'abbiamo inserita in una apposita variabile all'inizio del programma chiamata per l'appunto **dpi**.

È bene ricordarsi di eliminare l'oggetto pagina e il documento, per liberare la memoria (lavorando ad alte risoluzioni è facile che venga richiesta molta RAM per svolgere queste operazioni).

Avremmo potuto inserire direttamente ogni immagine estratta nell'elenco delle immagini che vogliamo passare all'OCR, ma possiamo anche semplicemente leggere il contenuto della cartella temporanea cercando tutti i file che contiene e, scorrendoli uno ad uno, passare il loro percorso alla funzione **addimagestolist**. È infatti questa la funzione che si occuperà di inserire le singole immagini nella lista. Chiamando la



funzione soltanto dopo l'estrazione delle pagine, le immagini appariranno nell'interfaccia grafica tutte assieme e l'utente capirà che la procedura è terminata.

La funzione in questione è molto semplice: viene creato un nuovo elemento del `qlistwidget` (l'oggetto che nell'interfaccia grafica del nostro programma funge da elenco delle pagine). All'elemento viene assegnata una icona, che proviene dal file stesso e che quindi costituirà la sua anteprima. L'elemento viene infine aggiunto all'oggetto presente nell'interfaccia grafica (**ui**).



### **Il file di progetto**

Per consentire al compilatore di trovare la libreria Poppler basta inserire nel file di progetto (.pro) le seguenti righe:

In questo modo il compilatore saprà che su Windows i file .h si troveranno nella cartella **include/poppler-qt5** del codice sorgente, mentre la libreria compilata sarà nella cartella **lib**.

## **Fusione dei PDF**

Dopo avere eseguito l'OCR sulle varie pagine, si ottengono da Tesseract tanti PDF quante sono per l'appunto le pagine del documento. Ciò significa che dovremo riunirle manualmente, fondendo assieme tutti i vari file in un unico PDF. Per farlo, prima di tutto decidiamo il nome di un file temporaneo nel quale riunire tutti i PDF:

Lo facciamo sfruttando lo stesso meccanismo che abbiamo usato per la cartella temporanea, ma con la libreria **QTemporaryFile**. Ovviamente, il file dovrà avere estensione **pdf**, e il suo nome è contenuto nella variabile **tmpfilename**.

Per scrivere sul PDF temporaneo, basta creare un nuovo oggetto di tipo **QpdfWriter** associato al file e un oggetto **QPainter** associato al **pdfWriter**. Il **QPainter** è il disegnatore che si occuperà di, per l'appunto, disegnare il contenuto del PDF secondo le nostre indicazioni.

Le varie pagine, cioè i pdf da riunire, si trovano nella stringa **allpages** separati dal simbolo **|**. Con un semplice ciclo **for** possiamo prendere un pdf alla volta, inserendo il suo nome nella stringa **inp**.

Se quello su cui stiamo lavorando non è il primo dei file da unire (quindi il contatore delle pagine **i** è maggiore di 0), allora possiamo inserire una interruzione di pagina nel PDF finale con la funzione **newPage**. Questo ci permette di unire i vari file dedicando una nuova pagina a ciascuno.

Possiamo quindi aprire il pdf usando, come già visto, **Poppler::Document**. Con un ciclo **for** scorriamo le varie pagine: ciascuno dei file da unire dovrebbe contenere una sola pagina, ma è comunque più prudente usare un ciclo per non correre rischi.

## Le varie TextBox

Ora dobbiamo estrarre il testo della pagina, cioè il testo che Tesseract ha inserito grazie alla funzione di OCR.

Potremmo semplicemente prelevare il testo con la funzione

**text**, ma preferiamo usare **textList**. Infatti, la prima ci fornisce semplicemente tutto il testo della pagina, ma a noi questo non va bene: abbiamo bisogno di avere anche l'esatta posizione, nella pagina, di ogni parola. Per questo esiste **textList**, una lista di **Poppler::TextBox**, dei rettangoli che contengono il testo e hanno una precisa posizione e dimensione.

Con un ulteriore ciclo for possiamo scorrere tutte le **textBox** ottenendo il rettangolo (**QrectF** è un rettangolo con dimensioni float) che le rappresenta usando la funzione **boundingBox**.

Ora c'è un piccolo problema: le dimensioni e la posizione del rettangolo sono state indicate, da Poppler, con il sistema di riferimento della pagina che stiamo leggendo. Invece, il nostro pdfWriter avrà probabilmente un sistema di riferimento diverso, a causa della risoluzione. Possiamo calcolare il rapporto orizzontale e verticale semplicemente dividendo larghezza e altezza della pagina di pdfWriter per quelle della pagina di Poppler.

Adesso possiamo tranquillamente scrivere il testo usando il nuovo rettangolo, che abbiamo appena calcolato, come riferimento. Il testo (attributo **text** della **textBox** attuale) si aggiunge usando la funzione **drawText** del **painter**.

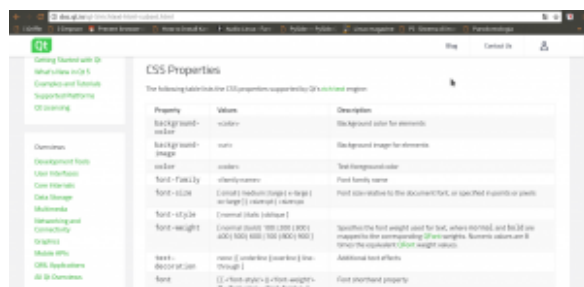
Soltanto dopo avere terminato questo ciclo for, e quindi avere scritto tutti i testi dove necessario, possiamo disegnare sulla pagina l'immagine di sfondo, con la funzione **drawPixmap** che si usa per inserire in un **painter** una immagine a mappa di pixel (una bitmap qualsiasi). La pixmap è ovviamente ottenuta dall'immagine che preleviamo tramite Poppler usando la già vista funzione **renderToImage**. Inserendo l'immagine dopo il testo, siamo sicuri che sarà visibile soltanto l'immagine, e il testo risulterà invisibile ma ovviamente selezionabile e

ricercabile. In alternativa avremmo anche potuto scegliere il colore “trasparente” per il testo.

Ovviamente, quando abbiamo finito di leggere un file, dobbiamo eliminare il suo oggetto **document** per non occupare troppo spazio. Per quanto riguarda il PDF che stiamo scrivendo, non c'è bisogno di chiudere il file: QpdfWriter lo farà automaticamente appena la funzione termina.

## Scrivere dell'HTML

C'è ancora un ultimo caso da considerare: se invece di Tesseract si vuole utilizzare l'OCR Cuneiform su Windows, purtroppo non si ottiene un PDF e nemmeno un file HOCR (cioè un HTML con la posizione delle varie parole). Si ottiene soltanto un semplice file HTML, che mantiene la formattazione ma non la posizione delle parole.



Un testo formattato può essere inserito in un PDF con QTextDocument usando la formattazione CSS delle pagine HTML (<http://doc.qt.io/qt-5/richtext-html-subset.html>)

Non è ottimale, ma può comunque essere utile avere un PDF che contenga il testo nella pagina, così lo si può ricercare facilmente. In questo caso, la prima cosa da fare è leggere il file html che si ottiene:

Leggendo il file come semplice testo grazie alle librerie `QFile` e `QTextStream`, possiamo inserire tutto il codice nella stringa **hocr**.

Ora, possiamo creare un nuovo documento di testo formattato, usando la libreria `QTextDocument`. Il contenuto del testo sarà indicato proprio dal codice **html** della stringa `hocr`, che quindi mantiene la formattazione. Impostiamo anche la larghezza massima del testo pari a quella della pagina di **pdfWriter**.

Come prima, dovremo calcolare la corretta dimensione con cui inserire il testo, per evitare che sia troppo piccolo o troppo grande. Siccome stavolta è solo testo, possiamo calcolare la dimensione del font con cui scriverlo usando una proporzione.

Dopo avere scelto la giusta dimensione del testo affinché riempia tutta la pagina, possiamo inserire il testo nel painter, e quindi nel PDF, usando la funzione `drawContents` del `QTextDocument`. Il vantaggio di questa funzione, rispetto a `drawText`, è che in questo modo si mantiene la formattazione e l'allineamento standard HTML.

Ovviamente, anche in questo caso si conclude la pagina inserendo sopra al testo l'immagine della pagina stessa, così il testo non sarà visibile, ma comunque ricercabile e selezionabile.

## **Il codice sorgente e il binario dell'esempio**

Per capire come venga organizzato il codice sorgente, vi conviene controllare quello del nostro programma di esempio. Banalmente, il programma è composto da un file di progetto, un

file **main.cpp** che costituisce la base dell'eseguibile, e due file (uno .h e uno .cpp) per la classe **mainwindow**, che rappresenta l'interfaccia principale del programma. Inoltre, abbiamo inserito due cartelle con il codice sorgente e il codice binario della libreria Poppler per Windows.

Trovate tutto il codice su GitHub assieme a dei pacchetti precompilati per Windows e GNU/Linux: <https://github.com/zorbaproject/kocr/releases>