Okular

Simply a document viewer?

Pino Toscano, pino@kde.org

June 30th, 2007

Outline

1 Introduction

2 Expanding Okular

Outline

1 Introduction

2 Expanding Okular

What is okular?

The KDE 4 document viewer

Included with kdegraphics.
Based on the awarded technology called KPDF.

Different document types in a single application

No more need for different applications to read different types of documents.

Reading aids

Different ways to ease the reading of a document.

Features

What does okular bring to the user?

“Old“ KPDF features . . .

- Thumbnails
  Filtering by text search
- Table of Contents
- Links and hyperlinks
- Presentation mode
- Customizable view
  Single/facing, continuous
- Text extraction

Features (2)

What does okular bring to the user?

. . . and new features

- Multicolumns view
- Pages rotation
- Annotations
- Bookmarks
- Preliminary forms support
  no ActionScript, no possibility to submit
- Embedded files in documents
- Multimedia (sounds) support
- And much more . . .

!okular

What okular is not

- A document manipulator (a.l.a. pdftk)
- A document collector (a.l.a. digiKam)
- An image viewer

Formats

The supported document formats

PDF	PostScript
OpenDocument Text	DVI
TIFF	DjVu
CHM	XPS
ComickBook	FictionBook
Plucker	images

OpenUsability.org

The collaboration with OpenUsability.org

- Continuous collaboration with the OpenUsability.org project, thanks to the great help of Florian Grässle
- KPDF 0.5 got already many usability fixes
- Many of the new solutions were developed taking into account usability

Okular and the Summer Of Usability

- Okular took part in the recent SoU contest
- An Indian student, Sharad Baliyan, was selected
- Results so far:
  - surveys about user interface
  - a proposal about the toolbar

People

The people behind Okular

- Pino Toscano
- Albert Astals Cid
- Tobias König
- Brad Hards
- Luigi Toscano
- Jiri Klement
- Piotr Szymański
- Florian Grässle

Future

Planned work on Okular

- Better support of forms
- Better handling or annotations
- Possible integration with Nepomuk/Strigi
- OpenDocument generator using the KOffice/Flake library
- More and better document supports

Outline

1 Introduction

2 Expanding Okular

Okular API

Introduction

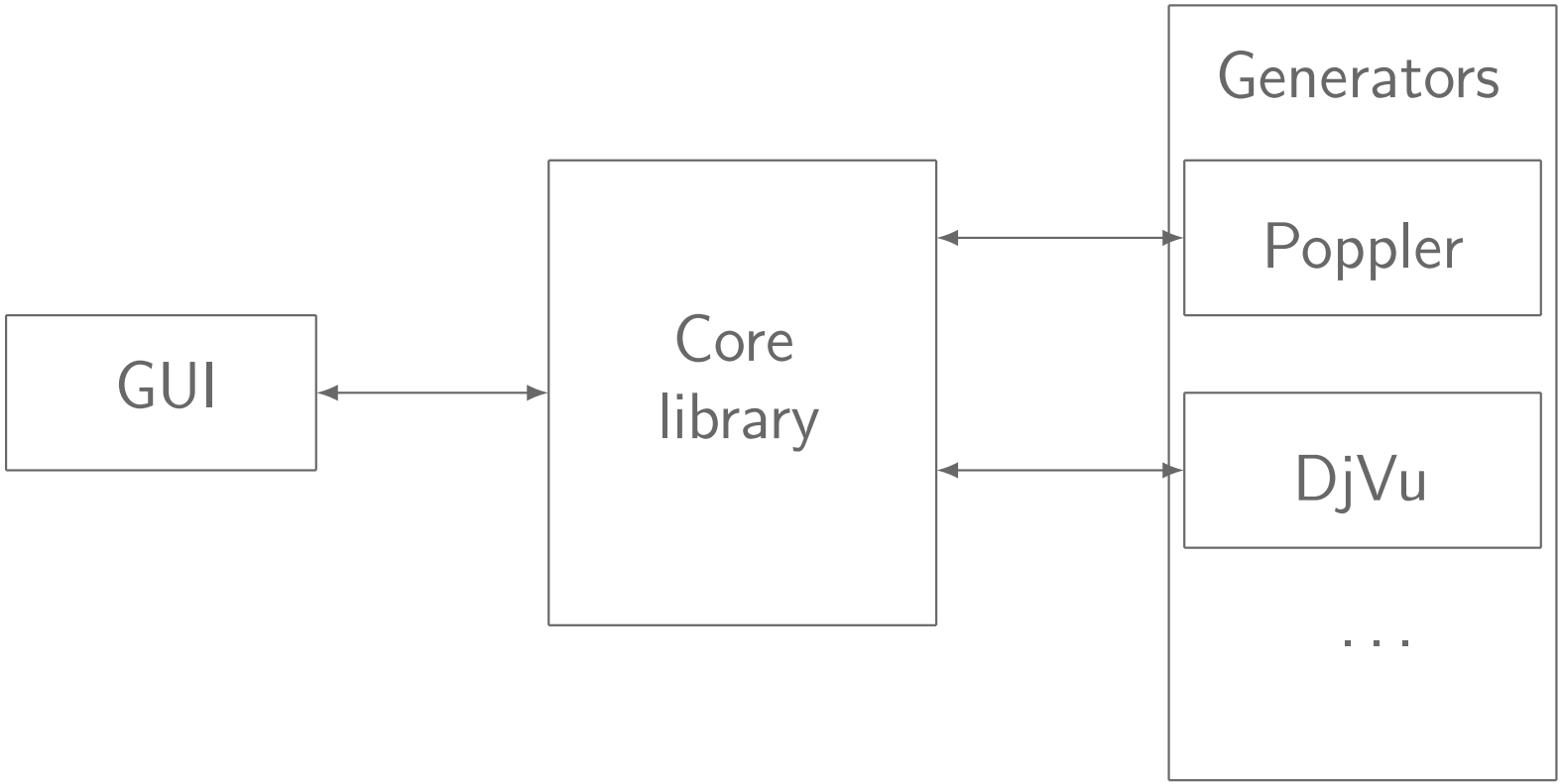
Okular provides a public API for writing generators for document types.

Why writing an Okular backend?

- No worries about the view of the document
  Think about writing a "model"...
- Common UI for reading the document
  Single place for usability enhancements and fixes
- New features and bugfixes become available for free for any generator

Okular API

General structure of the Okular API



Okular API

Core classes

All in the Okular namespace.

- Document
- Page
- Generator & TextDocumentGenerator
- PixmapRequest
- Action & its hierarchy
- TextPage
- Annotation & its hierarchy
- FormField & its hierarchy
- ...

Okular API

Base API

Okular::Generator

```
bool loadDocument(const QString &fileName,
QVector<Okular::Page*> &pagesVector);
bool closeDocument();
```

Easy job: opening a document and loading the pages from it.

Should also add all the page objects to the Page.

Okular API

Page rendering

```
protected:
  QImage image(PixmapRequest *page);
```

- synchronous
- activating the `Threaded` feature may help...

Own rendering strategy

```
bool canGeneratePixmap() const;
void generatePixmap(PixmapRequest *request);
```

`image()` is not needed in this case

Okular API

Text extraction

```
protected:
  TextPage* textPage(Page *page);
```

- sync/async depending on the type

Own text extraction strategy

```
bool canGenerateTextPage() const;
void generateTextPage(Page *page, GenerationType
type);
```

textPage() is not needed in this case

Okular API

Metadata

- general document information (`generateDocumentInfo()`)
- table of contents (`generateDocumentSynopsis()`)
- Fonts in the document (`generateDocumentFonts()`)
- Embedded files (`embeddedFiles()`)
- DRM (`isAllowed()`)
- . . . and `metaData()` for any other kind of metadata

Okular API

Give me paper!

Implementing the printing

```
bool print(KPrinter &printer);
```

- synchronous (no workaround at the moment)
- `printer` has all the needed stuff (size, margins, etc...)
- printing configuration via `PrintInterface`

Okular API

Basic generator example

```
class SimpleGenerator :  Okular::Generator() {
 public:
  SimpleGenerator();
  ~ SimpleGenerator();
  bool loadDocument(
   const QString &fileName,
   QVector<Okular::Page*> &pages) {
   // open the document, fill the pages vector
   return whether the loading succeeded;
  }
  bool closeDocument() {
   // cleanup your open document
   return true;
  } };
```

Okular API

Generator examples (2)

Example of more “complicated“ generators (TIFF & Poppler)

Questions

Anything to ask?

Questions ?

Pino Toscano and the Okular team

okular-devel@kde.org