Yayasan Jasa Aviasi Indonesia
And
LeTourneau University, Computer Science Department

Jungle Jepps

Design Specification

Version: 0.1.0   Date: 10/10/2013

Revision History

Revision #	Revision Date	Change	Author
0.1.0	October 10, 2013	Initial document	Joel Jeske

Distribution

Date	Revision #	Recipient	Sender	Method
October 10, 2013	0.1.0	Development Team	Joel Jeske	GIT
October 10, 2013	0.1.0	Zach Osterloo	Joel Jeske	Email
October 10, 2013	0.1.0	Brent Baas	Joel Jeske	BlackBoard

Table of Contents

1. Scope

This document contains the design specification of Jungle Jepps Desktop application. Note: a separate document will contain the design specification of Jungle Jepps Mobile. This document is preliminary and will change as development progresses. This document should conform to the requirements in the SRS and should reflect a valid implementation of Jungle Jepps Desktop.

1.1. System Objectives

As stated in the SRS Section 2 initially given by YAJASI, the major objectives of this application suite will be to manage a simple relational database of various types by using a GUI front end. The data in the database will be easily rendered with an image into a PDF document for use in the Jungle Jepps mobile application for in-aircraft use.

1.2. Hardware, software, and human interfaces]

Jungle Jepps Desktop will run in the Java runtime on a Microsoft Windows machine. It shall rely on components in the standard Java Virtual Machine as well as libraries included in the release. The software shall interact with the user via a keyboard, mouse, and display as managed by the operating system.

1.3. Major software functions

The software shall be able to accomplish the following tasks

- Take input from a user in a form-like interface that spans multiple screens
- Save this data in a database that is chosen by the system administrator
- Make changes to this data and record the changes made
- Compile the information given into a PDF document and filed in a repository
- Synchronize with Jungle Jepps Mobile, the iOS counterpart of the Jungle Jepps suite

1.4. Externally defined database

JJ Desktop will allow for read-only use of externally defined databases through the use of an open and generic database protocol; either ODBC or JDBC. Further specification shall be added when further requirements are given (i.e. the permissibility of JDBC over ODBC).

2. Reference Documents

2.1. Sources for existing software documents

Proper end-user documentation will be provided at a time nearer to the end of version 1.0 development. The Software Requirements Specification as provided by YAJASI stands as the current requirements document.

2.2. System documentation

Documentation of development and system internals will consist of; this document, diagram documents, and a technical users guide. The technical users guide will be provided nearer to the end the initial development phase.

2.3. Sources for technical reference documents

A complete JavaDoc web archive will be provided for ease of future development. Future developers will be aided through this document and attached diagrams.

3. Design Description

JJ Desktop will essentially be a multi-faceted form used to create new runway-aircraft combinations. A user will complete the form, including various text fields and number fields and will choose an image of the topographical runway layout. The user will then have an option to publish this information into a PDF. At any time over the local network, JJ Mobile will be able to contact JJ Desktop and request synchronization. During this synchronization, JJ Desktop will transmit any out-of-date or new published PDF documents and miscellaneous documents to JJ Mobile. JJ Mobile will then be able to recall these documents for in-aircraft use. In a future release of JJ Mobile, the user will be able to annotate these PDF documents and email any annotations to the JJ Desktop administrator. JJ Desktop will make use of various forms of local data.

3.1. Data description

3.1.1. Review of data flows

The data will originate from the JJ Desktop user and/or a third party ODBC capable source. (The IT administrator will choose from which source each field will originate.) This data will be published into a PDF and then transmitted to the repository and be accessible on the local network. These documents will be synchronized with JJ Mobile and they will reside on the iOS file system until a newer PDF replaces it. Diagram located in Appendix A.

3.1.2. Review of data structures

- There will be one or more database files that will be used by a Java implementation of SQLite. Each instance of JJ Desktop will make use of a SQLite database whether or not the runway data will be stored locally. It will be used to store the configuration preferences that can be edited within JJ Desktop.
- There will be a file repository in each local network of JJ Desktop. The file repository's root folder shall be shared location on the local network. The file repository will be structured as a simple folder hierarchy (5.1.1) containing published runway PDF documents, miscellaneous PDF documents, and the topographical images used in the creation of the published PDF documents. The published PDF files organized and named by the runway and aircraft and will contain the data fields that exist in the runway creation/edit form. The miscellaneous PDF documents will be any document that the user adds to the document section of JJ Desktop. The image files will be located in the same location of the published PDF document and will be used in future versions of that PDF document.
- There will be a configuration file used for editing the names of the fields of JJ Desktop and other options that would only need to be changed by an IT Professional. This configuration file will be loaded exactly once during the life of the program. It will be loaded on startup.
- There will be XML files that represent the markup of the various interfaces. These XML documents should not be changed, except by a developer. JJ Desktop will load these on startup to create each interface.

3.2. Data structure

3.2.1. Runtime Data Structures

The internal data structures will be implemented using object-oriented principals. The data structures will be specified in UML Class diagrams in Appendix A (UML).

3.2.2. Persistent Data Structures

The external data structures will be comprised of a one or more SQLite database files, a text based configuration files, multiple PDF documents, multiple image files (.jpg and .png).

3.3. Program Structure

The JJ Desktop application will be structured by responding mostly to user-initiated events, having little system initiated processes. On startup, the application will load any graphical interfaces and load a list of runway-aircraft selections from the database. The application will respond to clicks and keyboard events and allow the user to fill out the form. The application will switch interfaces upon clicks on different tabs along the top of the main window. When editing a previous form, all fields will be loaded with the past information from the database. When publishing a PDF form, the fields and imaged will be compiled into a template that will form the PDF. The PDF will be saved in the file repository.

3.4. Interfaces within the structure

There will be two different types of interfaces in the JJ Desktop application: application interfaces and object oriented interfaces.

Application interfaces

- SQLite [Using external SQLite connectivity library]
- MySQL [Using external JDBC MySQL connectivity library]
- ODBC [Consider using JDBC instead of ODBC. JDBC is analogous to ODBC but built for Java]
- TCP/IP [Using built-in support in JDK]

Object-oriented interfaces

- Database connection abstraction (including JDBC abstraction)
- HTTP handler (built in JDK)

4. Packages, Classes and Modules

Appendix-A diagrams the format of the packages, classes and modules that will be used. Diagrams adhere to the UML standard.

4.1. Code organization: Packages

Standard Java packages will be used for consistent code organization. The base package will be named org.yajasi.JungleJepps. Inner packages; db, ui, pdf and jjtp will be used to compartmentalize different sections of the code. The UML diagram references dependencies between packages.

4.1.1. Database

Package org.yajasi.JungleJepps.db will contain necessary classes and interfaces to manage the databases and data sources. The class DatabaseManager will be a static class from which other classes will request a handle to the current DatabaseConnection Interface. This interface abstracts all the database functions so the program can interface with the different types of databases needed: web server database, local database, database across LAN. The interface designs the all database connections as asynchronous. The SQLite connection may implement synchronous methods as well for ease of retrieving settings. This database connection will merge selected fields from a third party source into the runtime data structure, Strip, but will not save the information in the primary database. In the event the user requested a server-hosted database, a local database will still be created to hold configuration values for the program as a whole. This package will contain a PropertiesManager class that reads from a Java .properties file current values that the IT professional user has entered (i.e. label name changes, ODBC connection values, data sources, etc…).

4.1.2. User-Interface

Package org.yajasi.JungleJepps.ui will contain the necessary classes and interfaces to manage the user interface.

4.1.3. Network

Package org.yajasi.JungleJepps.jjtp will contain the necessary classes and interfaces to manage the local network connection. There will be a class, Server, used to serve client JJDesktop requests over the LAN in addition to JJMobile synchronization requests. The server will act as an implementation of the HTTP standard using the GET, POST, and PATCH methods. Further detail of HTTP method specification and syntax will be available in a future release of this document. There will be a class, Client, which will be available to make requests to the Server. This class will be an implementation of the DatabaseConnection interface, making it behave as if it were a database, when in fact, it processes the database connection requests across the LAN. There will also be a class, JJmDNS, which specifies a multicast DNS service type and makes methods available to broadcast a service as well as locate services being broadcasted. This aids in little to none network configuration within JJDesktop.

4.1.4. PDF / Repository

Package org.yajasi.JungleJepps.pdf will contain the necessary classes and interfaces to prepare an XHTML template with the specific information about a runway-aircraft and to output a PDF binary into the repository. It does the preparing using a custom XHTML tag <hook></hook> which specifies that text must be injected into its text node. Each hook tag should have an attribute, “id”, which specifies which field to inject. The <img/> tag to hold the topographical map has a specified “id” attribute which alerts the preparer of where to inject the URL for the topographical image. The template will be stored in a way that an IT professional or future developer would be able to rearrange the PDF output document easily. Note should be taken that improper changes to this document may result in undesired outcomes.

4.2. Processing narrative with cross reference to SRS

4.3. Interface description

The graphical interface will be designed to specification and appear similar to the legacy interface as shown in the SRS Section 2.1. The interface shall consist primarily of a two-column form with labels and text fields.

The labels will be customizable by the system administrator. The upper portion of the form will contain a tab bar for navigating through the form as specified in SRS 2.2.1.

4.4. Design language

UML is the chosen design language of choice using Microsoft Visio and the open-source solution Dia. Output documents from these programs are attached in Appendix A. Simple functional prototyping will be done in Java JDK 1.6. Simple visual prototyping will be done in Microsoft Power Point as well as Java.

4.5. Modules

Additional open source libraries are being used and will be listed as a package with a simple name on the diagrams. Libraries currently in use:

- Flying Saucer: flying-saucer-core-renderer.jar, provides a simple XHTML+CSS to PDF rendering.
- iText: iText-2.0.8.jar, provides PDF rendering used by Flying Saucer
- JmDNS: jmdns.jar, provides mDNS environment for broadcasting and discovering services.
- SQLite JDBC: sqlite-jdbc-3.7.15-M1.jar, provides front-end for creating and managing and interfacing with SQLite database files.
- GSON: gson-2.2.4.jar, provides simple front-end for serializing Java objects in standard JSON format used in network transfer.

5. File structure and global data

5.1. External file structures

There shall be an external repository to contain published and archived PDF documents, as well as the image files for publishing the PDF documents.

5.1.1. Logical structure

The repository shall be structured in the following way

//aircraft_type/runway_identifier/diagram.pdf

//aircraft_type/runway_identifier/archive/*.pdf

// aircraft_type/runway_identifier/photos/*.jpg

Photo and archive files shall be named in the following format: ZZZZ-RunwayName_20130521-01

- ZZZZ is the 3-4 alphanumeric character unique identifier.
- RunwayName is the complete name of the runway in CamelCase with no spaces.
- 20130521 is the date the file was created in the YYYYMMDD format.
- 01 is a sequential number for the iteration of that file, should more than one file with the same name be created on the same date.

5.1.2. Logical record description

The runway directories shall contain the active diagram. The archive subdirectories shall contain inactive diagram files. The photos subdirectories shall contain photos of their corresponding runways.

5.1.3. Access method

The repository shall be hosted using Windows folder sharing.

5.2. Global data

5.3. File and data cross reference

The photos are used in creating the PDF files. The PDF files are viewed using external software, or the JJ Mobile app.

6. Test provisions

There will be two types of testing performed during the development process: unit testing and code audits.

6.1. Unit testing

Unit testing will be run on each class. A test for each class will be determined before construction of that class is begun. The test will be a function within the class that calls each of the other functions in the class feeding them preselected data sets and checking to make sure the functions return the correct data.

6.2. Code Audits

Four code audits will be conducted. In essence, these audits will be larger unit test in which a specific use case is tested and then peer reviewed by one of the other members of the team.

6.2.1. Audit 1

Audit 1 will occur when the form part of the server is completed: purpose is to ensure that all data gets into the database correctly. Testing new data and data updates.

6.2.2. Audit 2

Audit 3 will occur when the PDF generation function is completed: purpose is to ensure that the output PDF meats all requirements.

6.2.3. Audit 3

Audit 3 will occur when the client code is done: purpose is to ensure that all data is being pass correctly

6.2.4. Audit 4

Audit 4 will occur at the completion of the JJ Mobile: purpose is to ensure that the entire software suit is functioning as intended.

7. Packaging

7.1. Special program overlay provisions

The transferred and installed program must provide easy access by the system administrator to a configuration file to setup and configure Jungle Jepps Desktop. It must provide easy access by the system administrator to the plugins folder where the installation of additional JDBC drivers may occur.

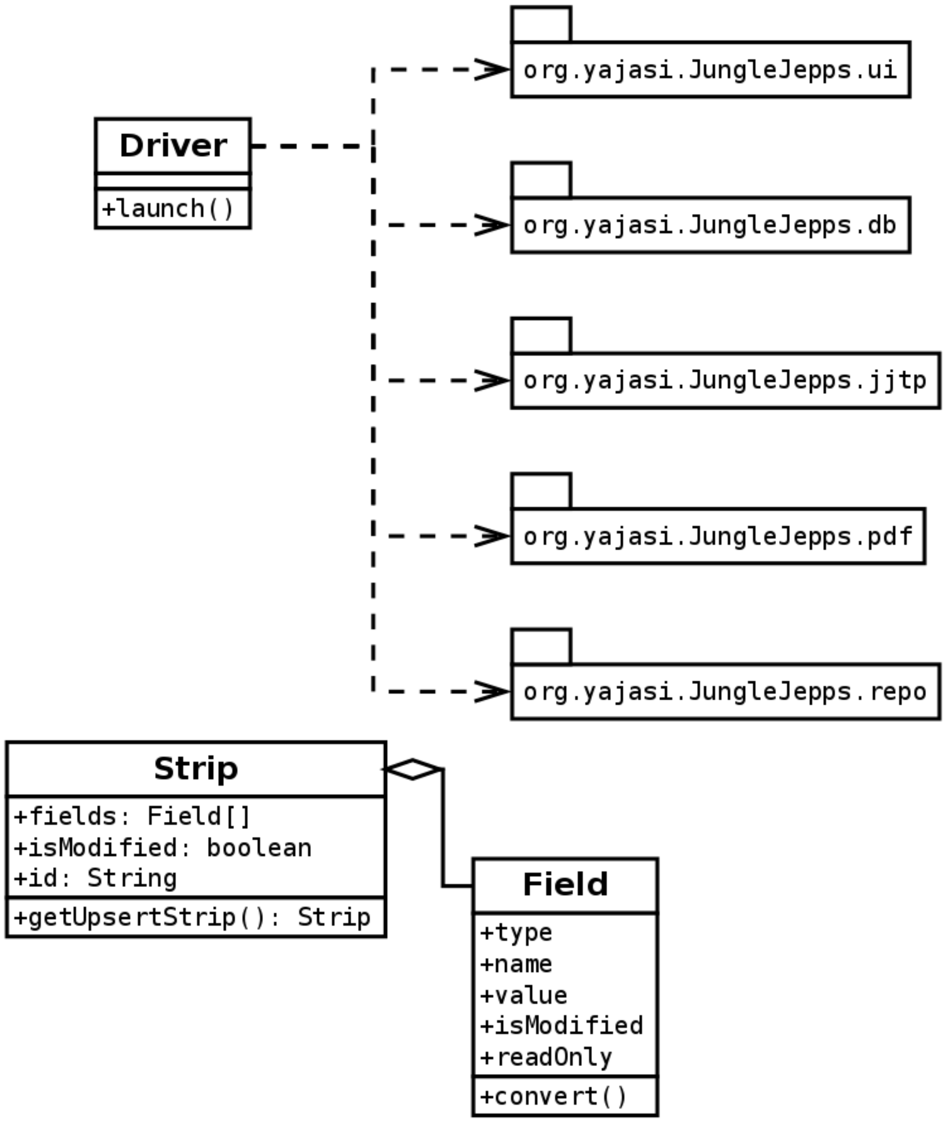
7.2. Transfer considerations

The transmission of the initial release of Jungle Jepps desktop will occur via an email attachment of a Windows executable (.exe), which will lead the system administrator through an installation and configuration process. The transmission of the source code of Jungle Jepps Desktop will occur via the allowed fork of our version control GIT repository, currently hosted on BitBucket.org.

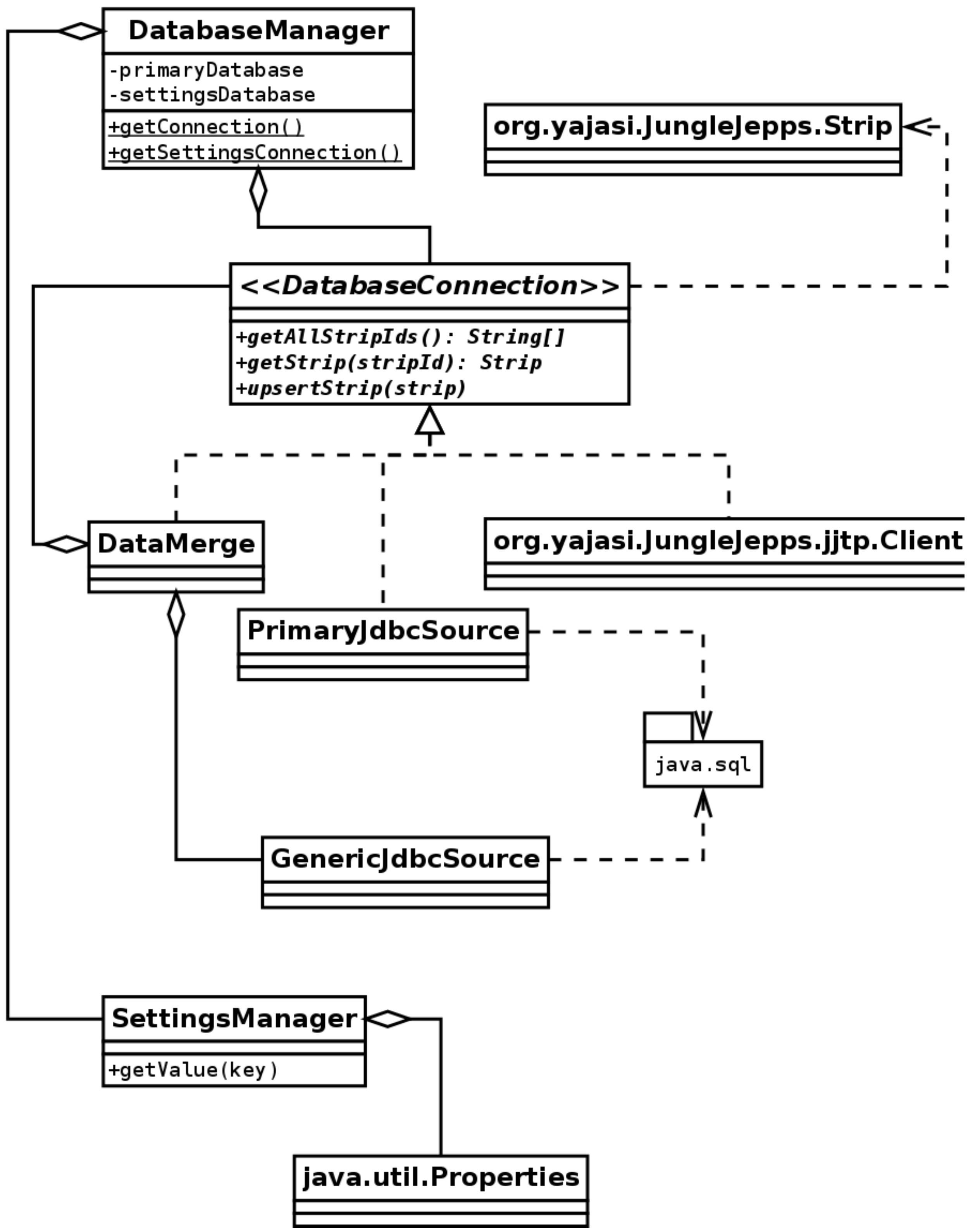
8. Appendices

8.1. Appendix A – UML Class Diagrams

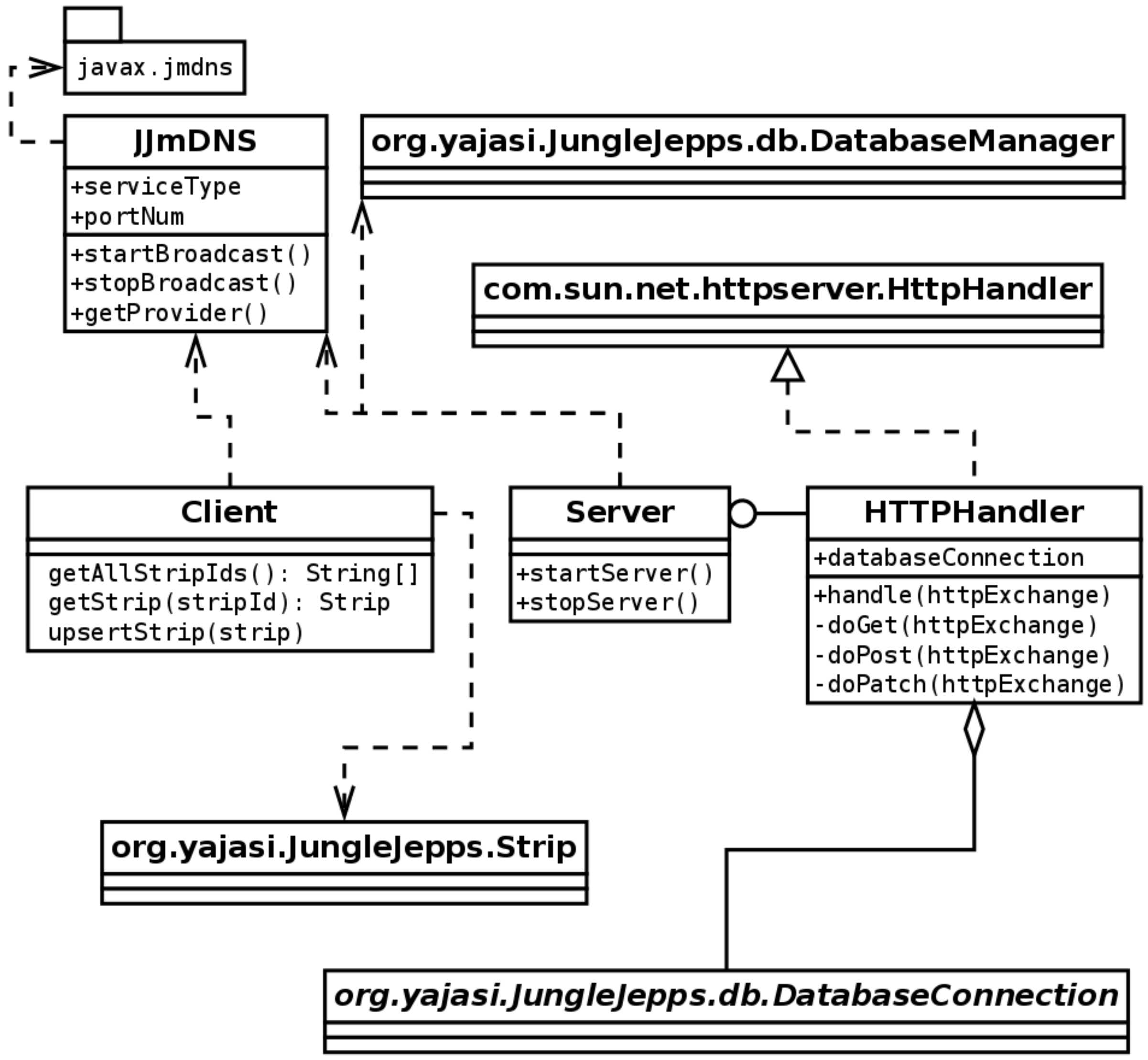
Class Diagram org.yajasi.JungleJepps



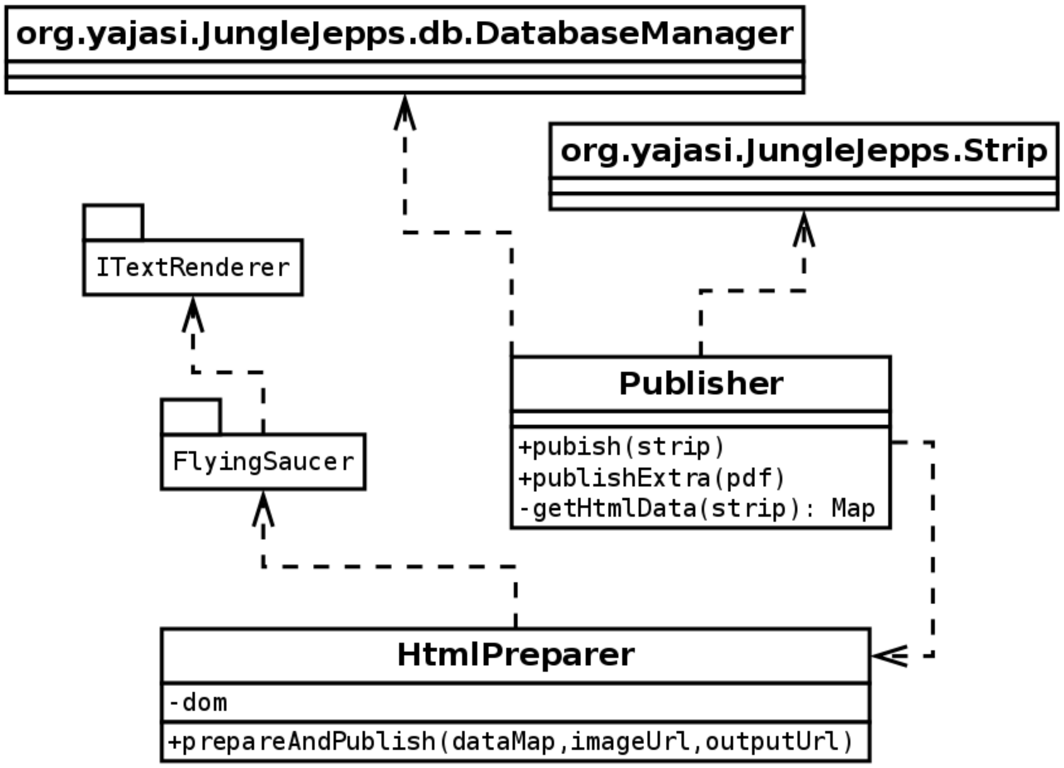
Class Diagram org.yajasi.JungleJepps.db

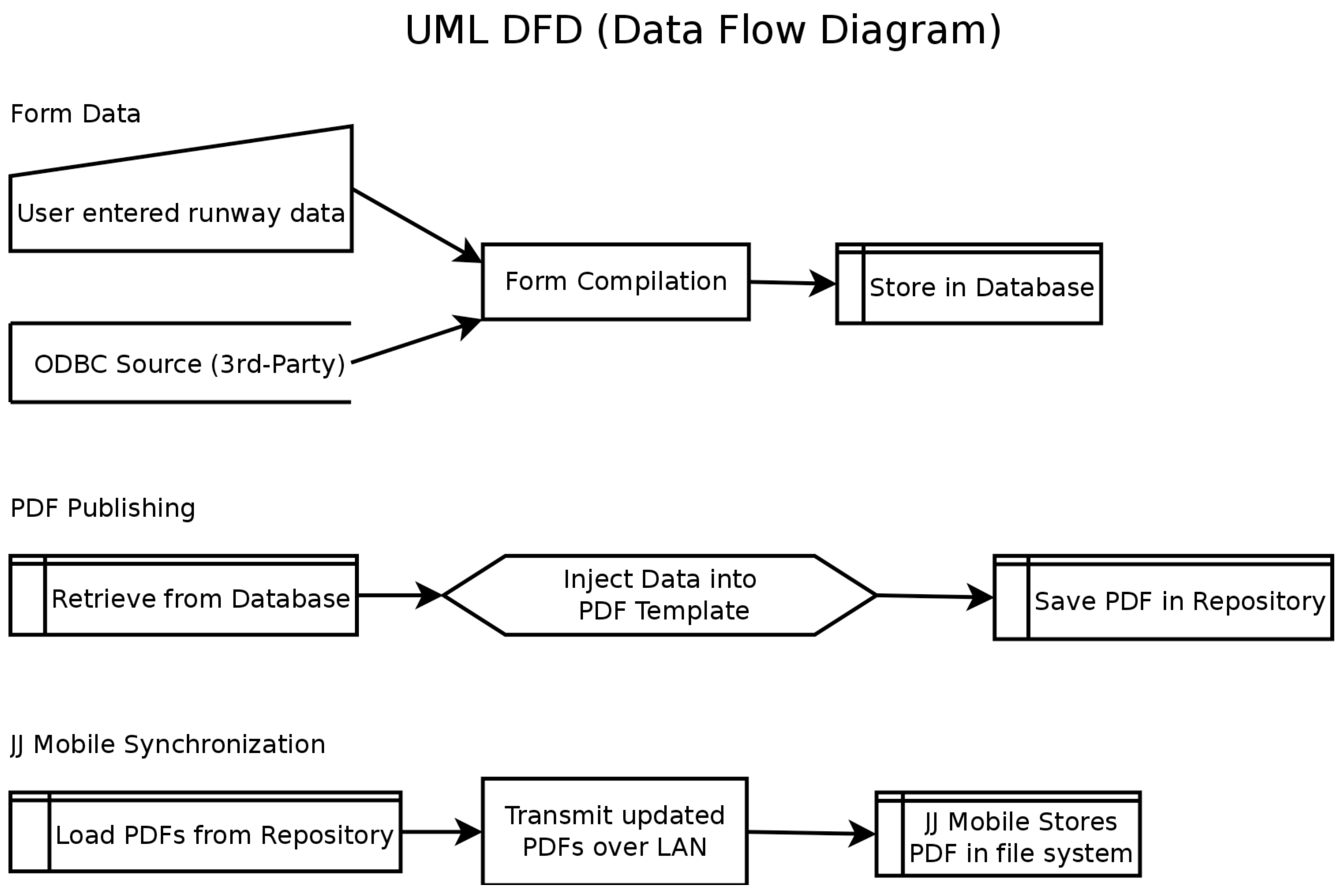


Class Diagram org.yajasi.JungleJepps.jjtp

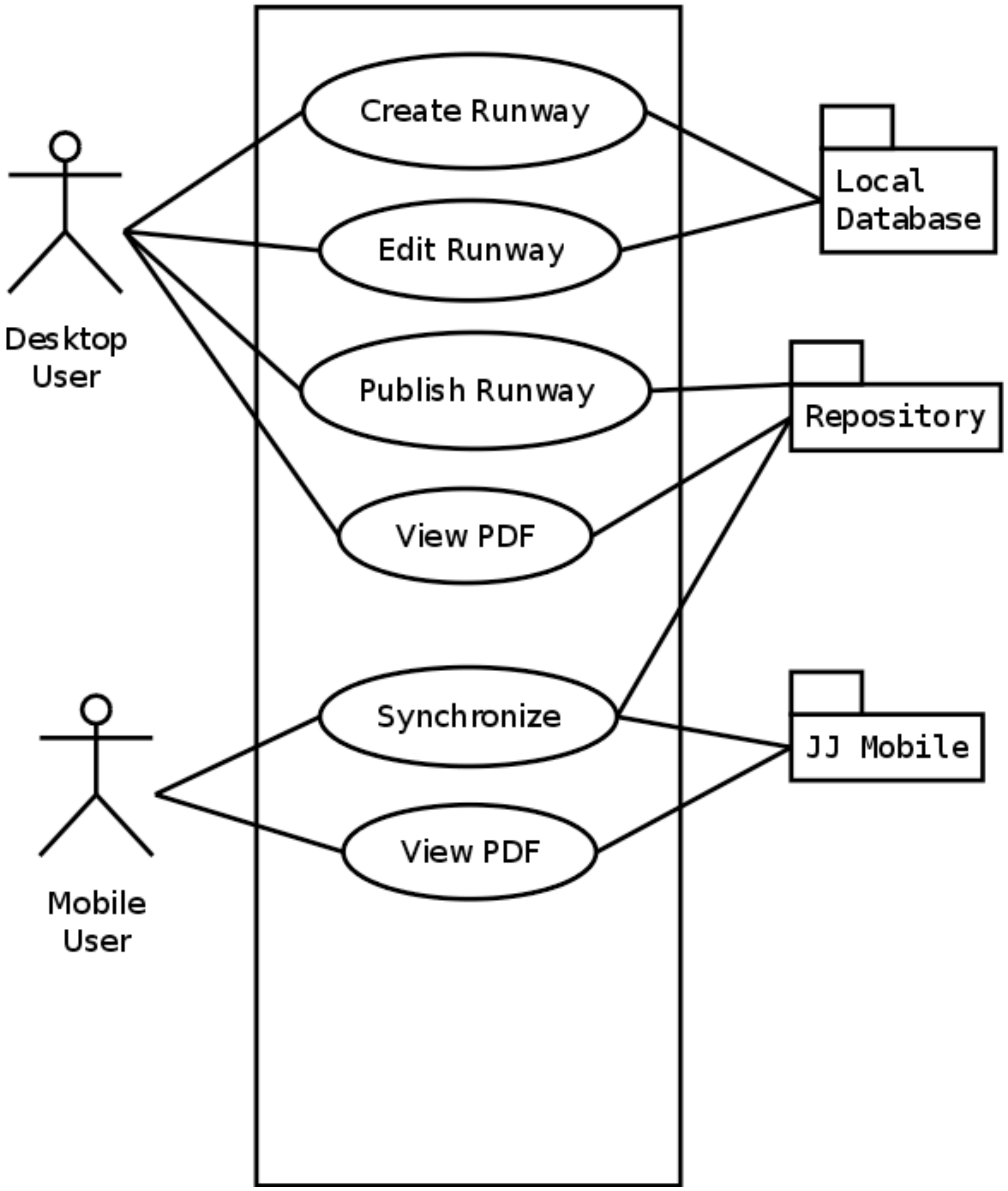


Class Diagram org.yajasi.JungleJepps.pdf

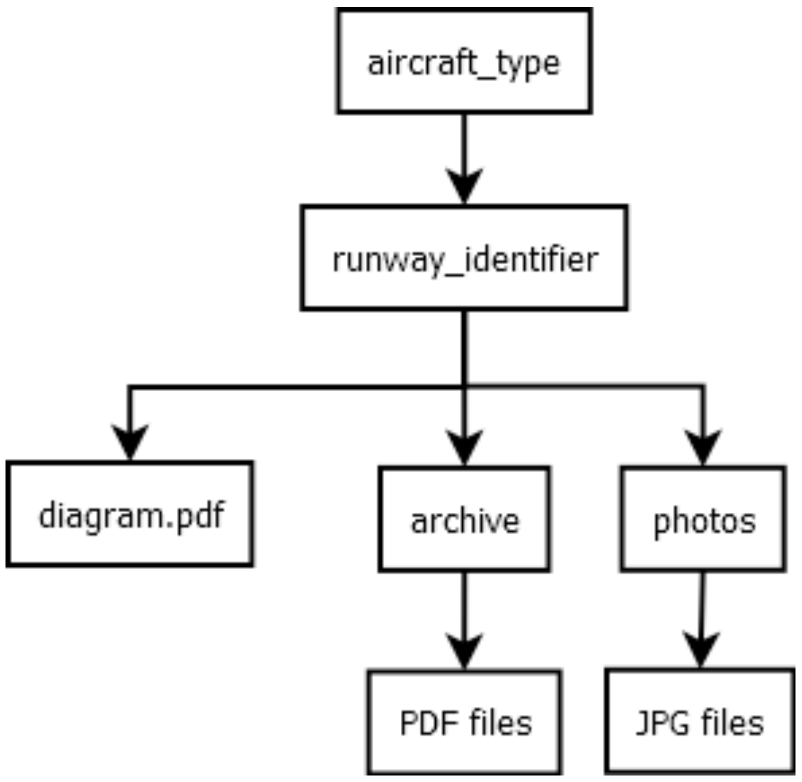

org.yajasi.JungleJepps.db.DatabaseManager
org.yajasi.JungleJepps.Strip
ITextRenderer
FlyingSaucer
Publisher
+pubish(strip)
+publishExtra(pdf)
-getHtmlData(strip): Map
HtmlPreparer
-dom
+prepareAndPublish(dataMap,imageUrl,outputUrl)


UML DFD (Data Flow Diagram)
Form Data
User entered runway data
ODBC Source (3rd-Party)
Form Compilation
Store in Database
PDF Publishing
Retrieve from Database
Inject Data into PDF Template
Save PDF in Repository
JJ Mobile Synchronization
Load PDFs from Repository
Transmit updated PDFs over LAN
JJ Mobile Stores PDF in file system

Use Case Diagram



Repository File Structure


aircraft_type
runway_identifier
diagram.pdf
archive
photos
PDF files
JPG files