PyReportJasper

Release 2.1.0

Dec 20, 2020

Contents

This package aims to be a solution to compile and process JasperReports (.jrxml & .jasper files).

Did you ever had to create a good looking Invoice with a lot of fields for your great web app or desktop?

I had to, and the solutions out there were not perfect. Generating *HTML + CSS* to make a *PDF*? WTF? That doesn't make any sense! :)

Then I found **JasperReports** the best open source solution for reporting.

From their website:

> The JasperReports Library is the world's most popular open source reporting engine. It is entirely written in Java and it is able to use data coming from any kind of data source and produce pixel-perfect documents that can be viewed, printed or exported in a variety of document formats including HTML, PDF, Excel, OpenOffice and Word.

CHAPTER 1

Installation

PyReportJasper is available either as a pre-compiled binary for Anaconda and PYPI, or may be built from source though various methods.

1.1 Binary Install

PyReportJasper can be installed as pre-compiled binary if you are using the Anaconda Python stack. Binaries are available for Linux, OSX, and windows on conda-forge.

1. Ensure you have installed Anaconda/Miniconda. Instructions can be found here.
2. Install from the conda-forge software channel:

```
conda install -c conda-forge pyreportjasper
```

1.2 Source Install

Installing from source requires:

Python PyReportJasper works CPython 3.5 or later. Both the runtime and the development package are required.

Java Either the Sun/Oracle JDK/JRE Variant or OpenJDK. PyReportJasper has been tested with Java versions from Java 1.9 to Java 15.

Jpype1 JPype is a Python module to provide full access to Java from within Python.

Once these requirements have been met, one can use pip to build from either the source distribution or directly from the repository. Specific requirements from different achitectures are listed below.

1.2.1 Build using pip

PyReportJasper may be built and installed with one step using pip.

To install the latest PyReportJasper, use:

```
pip install pyreportjasper
```

This will install PyReportJasper either from source or binary distribution, depending on your operating system and pip version.

To install from the current github master use:

```
pip install git@github.com:acesseonline/pyreportjasper.git
```

More details on installing from git can be found at Pip install. The git version does not include a prebuilt jar the JDK is required.

1.2.2 Build and install manually

PyReportJasper can be built entirely from source.

1. Get the PyReportJasper source

The PyReportJasper source may be acquired from either github or from PyPi.

2. Build the source with desired options

Compile PyReportJasper using the included `setup.py` script:

```
python setup.py build
```

3. Test PyReportJasper with (optional):

```
python -m unittest discover ./test -p '*.py'
```

4. Install PyReportJasper with:

```
python setup.py install
```

1.2.3 If it fails...

Most failures happen when setup.py is unable to find the JDK home directory which shouble be set in the enviroment variable `JAVA_HOME`. If this happens, preform the following steps:

1. Identify the location of your JDK systems installation and set the environment variable.

   ```
   export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64/
   ```

2. If that setup.py still fails please create an Issue on github and post the relevant logs.

1.2.4 Platform Specific requirements

PyReportJasper is known to work on Linux, OSX, and Windows. To make it easier to those who have not built CPython modules before here are some helpful tips for different machines.

Debian/Ubuntu

Debian/Ubuntu users will have to install `g++` and `python-dev`. Use:

```
sudo apt-get install g++ python-dev python3-dev
```

Windows

CPython modules must be built with the same C++ compiler used to build Python. The tools listed below work for Python 3.5 to 3.9. Check with Python dev guide for the latest instructions.

1. Install your desired version of Python (3.5 or higher), e.g., Miniconda is a good choice for users not yet familiar with the language
2. For Python 3 series, Install either 2017 or 2019 Visual Studio. Microsoft Visual Studio 2019 Community Edition is known to work.

From the Python developer page:

> When installing Visual Studio 2019, select the Python development workload and the optional Python native development tools component to obtain all of the necessary build tools. If you do not already have git installed, you can find git for Windows on the Individual components tab of the installer.

When building for windows you must use the Visual Studio developer command prompt.

CHAPTER 2

PyReportJasper User Guide

2.1 Introduction

This package aims to be a solution to compile and process JasperReports (.jrxml & .jasper files).

2.2 What can I do with this?

Well, everything. JasperReports is a powerful tool for **reporting** and **BI**.

From their website:

> The JasperReports Library is the world's most popular open source reporting engine. It is entirely written in Java and it is able to use data coming from any kind of data source and produce pixel-perfect documents that can be viewed, printed or exported in a variety of document formats including HTML, PDF, Excel, OpenOffice and Word.

It is recommended using Jaspersoft Studio to build your reports, connect it to your datasource (ex:JSON, XML, MySQL, POSTGRES, SQL Server), loop thru the results and output it to PDF, XLS, DOC, RTF, ODF, etc.

Some examples of what you can do:

- Invoices
- Reports
- Listings

2.2.1 No support

Warning: For now we do not support MongoDB but we are working to make it happen

2.3 The Hello World example.

See also:

We provide a repository with several reports that you can use to do your tests. Just clone your machine to open the `run.py` file and start coding.

Repository link with examples here

2.3.1 Compiling

First we need to compile our `JRXML` file into a `JASPER` binary file. We just have to do this one time.

Note 1: You don't need to do this step if you are using *Jaspersoft Studio*. You can compile directly within the program.

```
# -*- coding: utf-8 -*-
import os
from pyreportjasper import PyReportJasper

def compiling():
    REPORTS_DIR = os.path.join(os.path.abspath(os.path.dirname(__file__)), 'reports')
    input_file = os.path.join(REPORTS_DIR, 'csv.jrxml')
    output_file = os.path.join(REPORTS_DIR, 'csv')
    pyreportjasper = PyReportJasper()
    pyreportjasper.compile(write_jasper=True)
```

This commando will compile the `csv.jrxml` source file to a `csv.jasper` file.

2.3.2 Processing

Now lets process the report that we compile before:

```
# -*- coding: utf-8 -*-
import os
from pyreportjasper import PyReportJasper

def processing():
   REPORTS_DIR = os.path.join(os.path.abspath(os.path.dirname(__file__)), 'reports')
   input_file = os.path.join(REPORTS_DIR, 'csv.jrxml')
   output_file = os.path.join(REPORTS_DIR, 'csv')
   pyreportjasper = PyReportJasper()
   pyreportjasper.config(
     input_file,
     output_file,
     output_formats=["pdf", "rtf"]
   )
   pyreportjasper.process_report()
```

Now check the reports folder! :) Great right? You now have 2 files, `csv.pdf` and `csv.rtf`.

2.4 Advanced example - using a database

We can also specify parameters for connecting to database:

```
# -*- coding: utf-8 -*-
import os
from platform import python_version
from pyreportjasper import PyReportJasper

def advanced_example_using_database():
    REPORTS_DIR = os.path.join(os.path.abspath(os.path.dirname(__file__)), 'reports')
    input_file = os.path.join(REPORTS_DIR, 'hello_world.jrxml')
    output_file = os.path.join(REPORTS_DIR, 'hello_world')
    conn = {
      'driver': 'postgres',
      'username': 'DB_USERNAME',
      'password': 'DB_PASSWORD',
      'host': 'DB_HOST',
      'database': 'DB_DATABASE',
      'schema': 'DB_SCHEMA',
      'port': '5432'
      'jdbc_dir': '<path>/postgres.jar'
    }
    pyreportjasper = PyReportJasper()
    pyreportjasper.config(
      input_file,
      output_file,
      db_connection=conn,
      output_formats=["pdf", "rtf"],
      parameters={'python_version': python_version()},
      locale='en_US'
    )
    pyreportjasper.process_report()
```

Note 2:

For a complete list of locales see Supported Locales

2.5 Reports from a XML

See how easy it is to generate a report with a source an XML file:

```
# -*- coding: utf-8 -*-
import os
from pyreportjasper import PyReportJasper

def xml_to_pdf():
    RESOURCES_DIR = os.path.join(os.path.abspath(os.path.dirname(__file__)), 'resources
↪')
    REPORTS_DIR = os.path.join(os.path.abspath(os.path.dirname(__file__)), 'reports')
    input_file = os.path.join(REPORTS_DIR, 'CancelAck.jrxml')
    output_file = os.path.join(REPORTS_DIR, 'cancel_ack2')
    data_file = os.path.join(RESOURCES_DIR, 'CancelAck.xml')
    pyreportjasper = PyReportJasper()
    self.pyreportjasper.config(
        input_file,
        output_file,
        output_formats=["pdf"],
        db_connection={
```

(continues on next page)

(continued from previous page)

```
            'driver': 'xml',
            'data_file': data_file,
            'xml_xpath': '/CancelResponse/CancelResult/ID',
        }
    )
    self.pyreportjasper.process_report()
    print('Result is the file below.')
    print(output_file + '.pdf')
```

2.6 Reports from a CSV File

See how easy it is to generate a report with a source an CSV file:

```
# -*- coding: utf-8 -*-
import os
from pyreportjasper import PyReportJasper

def csv_to_pdf():
    RESOURCES_DIR = os.path.join(os.path.abspath(os.path.dirname(__file__)), 'resources
↪')
    REPORTS_DIR = os.path.join(os.path.abspath(os.path.dirname(__file__)), 'reports')
    input_file = os.path.join(REPORTS_DIR, 'csv.jrxml')
    output_file = os.path.join(REPORTS_DIR, 'csv')
    conn = {
        'driver': 'csv',
        'data_file': os.path.join(self.RESOURCES_DIR, 'csvExampleHeaders.csv'),
        'csv_charset': 'utf-8',
        'csv_out_charset': 'utf-8',
        'csv_field_del': '|',
        'csv_out_field_del': '|',
        'csv_record_del': "\r\n",
        'csv_first_row': True,
        'csv_columns': "Name,Street,City,Phone".split(",")
    }
    pyreportjasper = PyReportJasper()
    self.pyreportjasper.config(
        input_file,
        output_file,
        output_formats=["pdf"],
        db_connection=conn
    )
    self.pyreportjasper.process_report()
    print('Result is the file below.')
    print(output_file + '.pdf')
```

2.7 Reports from a JSON File

See how easy it is to generate a report with a source an JSON file:

```
# -*- coding: utf-8 -*-
import os
```

(continues on next page)

(continued from previous page)

```
from pyreportjasper import PyReportJasper

def json_to_pdf():
    RESOURCES_DIR = os.path.join(os.path.abspath(os.path.dirname(__file__)), 'resources
↪')
    REPORTS_DIR = os.path.join(os.path.abspath(os.path.dirname(__file__)), 'reports')
    input_file = os.path.join(REPORTS_DIR, 'json.jrxml')
    output_file = os.path.join(REPORTS_DIR, 'json')
    conn = {
        'driver': 'json',
        'data_file': os.path.join(self.RESOURCES_DIR, 'contacts.json'),
        'json_query': 'contacts.person'
    }
    pyreportjasper = PyReportJasper()
    self.pyreportjasper.config(
        input_file,
        output_file,
        output_formats=["pdf"],
        db_connection=conn
    )
    self.pyreportjasper.process_report()
    print('Result is the file below.')
    print(output_file + '.pdf')
```

2.8 Reports from a JSONQL

```
# -*- coding: utf-8 -*-
import os
from pyreportjasper import PyReportJasper

def jsonql_to_pdf():
    RESOURCES_DIR = os.path.join(os.path.abspath(os.path.dirname(__file__)), 'resources
↪')
    REPORTS_DIR = os.path.join(os.path.abspath(os.path.dirname(__file__)), 'reports')
    input_file = os.path.join(REPORTS_DIR, 'jsonql.jrxml')
    output_file = os.path.join(REPORTS_DIR, 'jsonql')
    conn = {
        'driver': 'jsonql',
        'data_file': os.path.join(self.RESOURCES_DIR, 'contacts.json'),
        'json_query': 'contacts.person'
    }
    pyreportjasper = PyReportJasper()
    self.pyreportjasper.config(
        input_file,
        output_file,
        output_formats=["pdf"],
        db_connection=conn
    )
    self.pyreportjasper.process_report()
    print('Result is the file below.')
    print(output_file + '.pdf')
```

CHAPTER 3

API Reference

3.1 Class PyReportJasper

class PyReportJasper(**args*, ***kwargs*)

Class responsible for facilitating the management and export of reports

classmethod config(*input_file, output_file=False, output_formats=['pdf'], parameters={}, db_connection={},locale='pt_BR', resource=None*)

Method responsible for preparing and validating the settings and parameters

Parameters

- **input_file** (*str*) – Input file (.jrxml|.jasper|.jrprint)
- **output_file** (*str*) – Output file or directory
- **output_formats** (*list*) – List with the formats you want to export. Options: `pdf`, `rtf`, `docx`, `odt`, `xml`, `xls`, `xlsx`, `csv`, `csv_meta`, `ods`, `pptx`, `jrprint`
- **parameters** (*dict*) – Dictionary with the parameters your report expects
- **db_connection** (*dict*) – Dictionary with the necessary information to generate the report. Options: `driver`, `username`, `password`, `host`, `database`, `port`, `jdbc_driver`, `jdbc_url`, `jdbc_dir`, `db_sid`, `xml_xpath`, `data_file`, `json_query`, `jsonql_query`, `csv_first_row`, `csv_columns`, `csv_record_del`, `csv_field_del`, `csv_out_field_del`, `csv_charset`, `csv_out_charset`.
- **locale** (*str*) – Set locale with two-letter ISO-639 code or a combination of ISO-639 and ISO-3166 like de_DE. For a complete list of locales see Supported Locales
- **resource** (*str*) – Directory the files with resources that the report needs for correct generation and export.

classmethod compile(*write_jasper=False*)

Compiles the report or all reports into a directory if the input file parameter is a directory

Parameters **write_jasper** (*bool*) – Sets whether to write the `.jasper` output or not.

classmethod process_report()
Process the report and export to the formats specified in the config method

classmethod list_report_params()
Lists the parameters defined in the report

Returns Returns a parameter list

Return type `list(str)`

classmethod process(*input_file, output_file=False, format_list=['pdf'], parameters={}, db_connection={}, locale='pt_BR', resource=""*)

> **Warning:** This method still works more in the next versions will be removed.

Deprecated since version 2.1.0: Use *PyReportJasper.process_report()* instead.

3.2 Class Config

class Config(**args*, ***kwargs*)
Class for defining the settings to generate the reports.

jvm_maxmem
Maximum memory used by the JVM

Type str

Value '512M'

jvm_classpath
The class path is the path that the Java runtime environment searches for classes and other resource files.

Type str

dbType
Data source type

Type str

Options `None`, `csv`, `xml`, `json`, `jsonql`, `mysql`, `postgres`, `oracle`, `generic`

dbDriver
Jdbc driver class name for use with dbType: generic

Type str

dbHost
Database host

Type str

dbName
Database name

Type str

dbPasswd
Database password

Type str

dbPort
Database port

Type int

dbSid
Oracle sid

Type str

dbUrl
Jdbc url without user, passwd with dbType: generic

Type str

dbUser
Database user

Type str

jdbcDir
Directory where jdbc driver jars are located.

Type str

input
Input file (.jrxml|.jasper|.jrprint)

Type str

dataFile
Input file for file based datasource

Type str

csvFirstRow
First row contains column headers

Type bool

csvColumns
Comma separated list of column names

Type list(str)

csvRecordDel
CSV Record Delimiter - defaults to line.separator

Type str

csvFieldDel
CSV Field Delimiter - defaults to ","

Type str

csvCharset
CSV charset - defaults to "utf-8"

Type str

xmlXpath
XPath for XML Datasource

Type str

jsonQuery
JSON query string for JSON Datasource

Type str

jsonQLQuery
JSONQL query string for JSONQL Datasource

Type str

locale
Set locale with two-letter ISO-639 code or a combination of ISO-639 and ISO-3166 like en_US.

For a complete list of locales see Supported Locales

Type str

output
Directory or basename of outputfile(s)

Type str

outputFormats
A list of output formats

Type list(str)

Options `pdf`, `rtf`, `docx`, `odt`, `xml`, `xls`, `xlsx`, `csv`, `csv_meta`, `ods`, `pptx`, `jrprint`

params
Dictionary with the names of the parameters and their respective values.

Exemple: `{'NAME_PARAM_1': 'value param 1', 'NAME_PARAM_2': 'value param 2'}`

Type dict

printerName
Name of printer

Type str

reportName
Set internal report/document name when printing

Type str

resource
Path to report resource dir or jar file. If <resource> is not given the input directory is used.

Type str

writeJasper
Write .jasper file to imput dir if jrxml is processed

Type bool

Value False

outFieldDel
Export CSV (Metadata) Field Delimiter - defaults to ","

Type str

outCharset
Export CSV (Metadata) Charset - defaults to "utf-8"

Type str

askFilter

Type str

Options `a` - all (user and system definded) prarms

`ae` - all empty params

`u` - user params

`ue` - empty user params

`p` - user params marked for prompting

`pe` - empty user params markted for prompting

classmethod has_output()

Valid if there is a path or file for the output

Returns Returns true if there is a defined path or file for output otherwise false.

Return type bool

classmethod is_write_jasper()

Valid if it is to generate a .jasper

Returns Returns `true` if the .jasper is to be generated, otherwise it is `false`.

Return type bool

classmethod has_jdbc_dir()

Validates if there is a path or file for jdbc .jar

Returns Returns `true` if it exists, otherwise `false`.

Return type bool

classmethod has_resource()

Validates if there is a .jar or path with several .jar to add to the class path

Returns Returns `true` if it exists, otherwise `false`.

Return type bool

3.3 Class Report

class Report(*config: Config*, *input_file*)

Class responsible for instantiating the JVM and loading the necessary Java objects for manipulating the files to compile, generate and export the reports.

Parameters

- **config** (Config) – Config class instance
- **input_file** (*str*) – Input file (.jrxml|.jasper|.jrprint)

classmethod compile()

Compile the report

classmethod compile_to_file()

Emit a .jasper compiled version of the report definition .jrxml file.

classmethod fill()

Executes the `fill_internal()` method

classmethod fill_internal()

Method responsible for filling the report

classmethod get_output_stream(*suffix*)

Return a file-based output stream with the given suffix

Parameters **suffix** (*str*) – File suffix

Returns Returns an output stream from the input file.

Return type OutputStream (java)

classmethod export_pdf()

Export the report in `pdf` format

classmethod export_rtf()

Export the report in `rtf` format

classmethod export_docx()

Export the report in `docx` format

classmethod export_odt()
Export the report in `odt` format

classmethod export_xml()
Export the report in `xml` format

classmethod export_xls()
Export the report in `xls` format

classmethod export_xls_meta()
Export the report in `xls` Metadata Exporter format

classmethod export_xlsx()
Export the report in `xlsx` format

classmethod export_csv()
Export the report in `csv` format

classmethod export_csv_meta()
Export the report in `csv` Metadata Exporter format

classmethod export_ods()
Export the report in `ods` format

classmethod export_pptx()
Export the report in `pptx` format

classmethod export_jrprint()
Export the report in `jrprint` format

classmethod get_report_parameters()
Returns a list of all report parameters

Returns Returns a list of parameters

Return type list(str)

classmethod get_main_dataset_query()
For JSON, JSONQL and any other data types that need a query to be provided, an obvious default is to use the one written into the report, since that is likely what the report designer debugged/intended to be used. This provides access to the value so it can be used as needed.

Returns Return a string of main dataset query.

Return type str

classmethod add_jar_class_path(*dir_or_jar*)
Method responsible for adding a `.jar` to `class_path` or a list of `.jar` files in an informed directory

Parameters **dir_or_jar** (*str*) – A `.jar` file or directory containing one or more `.jar`

3.4 Class Db

class Db
Class responsible for managing the report data source

classmethod get_csv_datasource(*config: Config*)
Method responsible for creating a data source from an informed csv file

Parameters **config** (Config) – Config class instance

Returns Returns a data source of type csv

Return type net.sf.jasperreports.engine.data.JRCsvDataSource (java)

classmethod get_xml_datasource(*config: Config*)
Method responsible for creating a data source from an informed xml file

Parameters **config** (Config) – Config class instance

Returns Returns a data source of type xml

Return type net.sf.jasperreports.engine.data.JRXmlDataSource (java)

classmethod get_json_datasource(*config: Config*)
Method responsible for creating a data source from an informed json file

Parameters **config** (Config) – Config class instance

Returns Returns a data source of type json

Return type net.sf.jasperreports.engine.data.JsonDataSource (java)

classmethod get_jsonql_datasource(*config: Config*)
Method responsible for creating a data source from an informed json file

Parameters **config** (Config) – Config class instance

Returns Returns a data source of type jsonql

Return type net.sf.jasperreports.engine.data.JsonQLDataSource (java)

classmethod get_data_file_input_stream(*config: Config*)
Get InputStream corresponding to the configured dataFile.

Parameters **config** (Config) – Config class instance

Returns Returns a InputStream

Return type java.io.InputStream (java)

classmethod get_connection(*config: Config*)
Method responsible for obtaining a connection to a database

Returns Returns database connection

Return type java.sql.Connection (java)

CHAPTER 4

Changelog

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index

L

O

P

R

W

X