innoConv Documentation

Release 0.0.3

Innocampus

Mar 27, 2019

Contents

CHAPTER 1

Table of contents

1.1 What is innoConv?

innoConv is a converter for educational content.

It transforms content into an intermediate format (JSON) that can be displayed with a *innodoc-compatible viewer*.

1.1.1 Course structure

TODO

- `toc.md`
- language directories - chapters and sections in sub-directories - `content.md`

1.1.2 Example course

There's an example course. It's a comprehensive demonstration of what is possible with innoConv.

It serves the following purposes:

- Showcase the capabilities and features
- Reference for course authors
- Case for automatic software tests

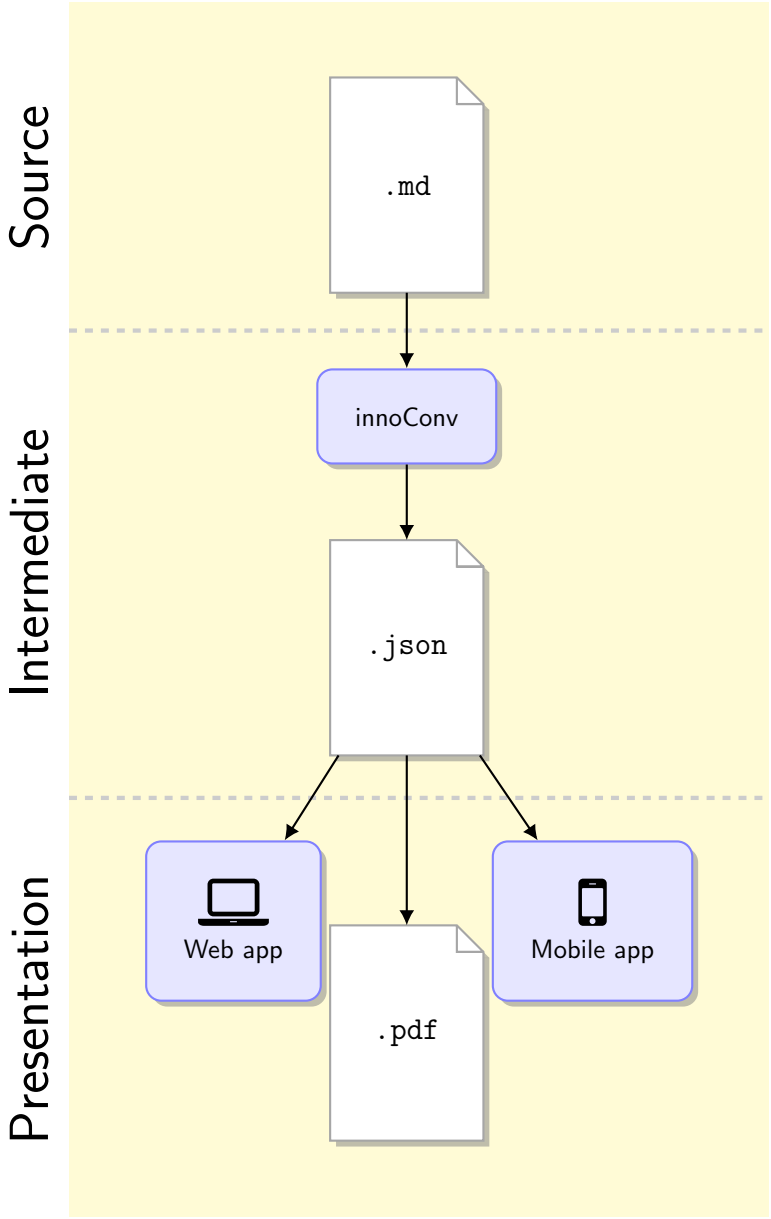
It can also serve as a starting point for trying out innoConv before writing your own content.

1.1.2.1 Links

- Demo course live version
- Content source repository

1.1.3 innoDoc

innoConv is one part in a software package called innoDoc.



1.1.3.1 Viewers

At the moment there are two viewers in development.

innodoc-webapp React-based HTML5 web application

innodoc-app React Native-based Smartphone App

1.2 Getting started

1.2.1 Prerequisites

innoConv is mainly used on Linux machines. It might work on Mac OS and Windows/Cygwin/WSL. You are invited to share your experiences.

1.2.2 Dependencies

The only dependencies you have to provide yourself is Pandoc and the Python interpreter.

1.2.2.1 Python 3

innoConv is being tested and developed with **Python 3.4-3.7**.

Python should be available on the majority of Linux machines nowadays. Usually it's being installed using a package manager.

1.2.2.2 Pandoc

You need to make sure to have a recent version of the **`pandoc`** binary available in `$PATH` (version 2.7 at the time of writing). There are several ways how to install Pandoc.

1.2.3 Installation

1.2.3.1 Using pip

The easiest way to install innoConv is to use **`pip`**.

Given you have a regular Python setup with **`pip`** available the following installs innoConv in your user directory (usually `~/.local` under Linux).

```
$ pip install --user innoconv
```

For the **`innoconv`** command to work, make sure you have `~/.local/bin` in your `$PATH`.

For a system-wide installation you can omit the `--user` argument.

```
$ pip install innoconv
```

1.2.3.2 In a virtual environment

It's possible to install innoConv into a virtual environment. Setup and activate a virtual environment in a location of your choice.

```
$ python3 -m venv /path/to/venv
$ source /path/to/venv/bin/activate
```

Install innoConv in your virtual environment using **`pip`**.

```
$ pip install innoconv
```

If everything went fine you should now have access to the **innoconv** command.

The next time you login to your shell make sure to activate your virtual environment before using **innoconv**.

1.3 How to use innoConv

The principle way of using innoConv is using its command-line interface (CLI) **innoconv**. Another option is to use innoConv in a *programmatic way* as a Python library.

Run the converter on your content directory.

```
$ innoconv /path/to/my/content
```

This will trigger the conversion and store the result in a folder `innoconv_output`. According to Unix philosophy you will not see any messages if the conversion was successful. Though you might pass the `--verbose` flag to change this behaviour.

1.3.1 Command line arguments

1.3.1.1 innoconv

Converter for interactive educational content.

```
innoconv [OPTIONS] SOURCE_DIR
```

Options

-o, --output-dir <output_dir>
: Set output directory. [default: ./innoconv_output]

-e, --extensions <extensions>
: Enable extensions (comma-separated). [default: join_strings,copy_static,generate_toc,write_manifest]

-f, --force
: Force overwriting of output.

-v, --verbose
: Print verbose messages.

--version
: Show the version and exit.

Arguments

SOURCE_DIR
: Required argument

1.3.2 Using innoConv as a library

Using innoConv within a Python program involves creating a *Manifest* object and using it with a *InnoconvRunner*.

```
>>> manifest = Manifest(manifest_data)
>>> runner = InnoconvRunner(source_dir, output_dir, manifest, extensions)
>>> runner.run()
```

Have a look at the source of *innoconv.cli* for a more detailed example.

1.4 Module overview

1.4.1 innoconv.cli

Command line interface for the innoConv document converter.

class innoconv.cli.**CustomEpilogCommand**(*name*, *context_settings=None*, *callback=None*, *params=None*, *help=None*, *epilog=None*, *short_help=None*, *options_metavar='[OPTIONS]'*, *add_help_option=True*, *hidden=False*, *deprecated=False*)

Format epilog in a custom way.

format_epilog(*_*, *formatter*)

Format epilog while preserving newlines.

1.4.2 innoconv.constants

Project constants.

innoconv.constants.**DEFAULT_OUTPUT_DIR_BASE**

Default innoconv output directory

innoconv.constants.**DEFAULT_EXTENSIONS**

Default enabled extensions

innoconv.constants.**ENCODING**

Encoding used in this project

innoconv.constants.**CONTENT_BASENAME**

Basename for the content file in a section

innoconv.constants.**MANIFEST_BASENAME**

Manifest filename

innoconv.constants.**STATIC_FOLDER**

Static folder name

1.4.3 innoconv.extensions

innoConv extensions.

Extensions are a way of separating concerns of the conversion process into independent modules. They can be enabled on a one-by-one basis as not all features are needed in all cases.

Extensions interface with *InnoconvRunner* through a set of methods defined in *AbstractExtension*.

innoconv.extensions.**EXTENSIONS = {'copy_static': <class 'innoconv.extensions.copy_static.C**
List of available extensions

1.4.4 innoconv.extensions.abstract

Base class for all other extensions.

The AbstractExtension is not instantiated directly but serves as super-class to all extensions.

class innoconv.extensions.abstract.**AbstractExtension**(*manifest*)
Abstract class for extensions.

The class all extensions inherit from. The to-be-implemented methods document the available events that are triggered during the conversion process.

Extension classes should have a `_helptext` attribute. It's used to display a brief summary.

classmethod helptext()
Return a brief summary of what the extension is doing.

start(*output_dir*, *source_dir*)
Conversion is about to start.

Parameters

- **output_dir** (*str*) – Base output directory
- **source_dir** (*str*) – Content source directory

pre_conversion(*language*)
Conversion of a single language folder is about to start.

Parameters **language** (*str*) – Language that is currently being converted.

pre_process_file(*path*)
Conversion of a single file is about to start.

Parameters **path** (*str*) – Output path

post_process_file(*ast*, *title*)
Conversion of a single file finished. The AST can be modified.

Parameters

- **ast** (*List of content nodes*) – File content as parsed by pandoc.
- **title** (*str*) – Section title (localized)

post_conversion(*language*)
Conversion of a single language folder finished.

Parameters **language** (*str*) – Language that is currently being converted.

finish()
Conversion finished.

1.4.5 innoconv.extensions.copy_static

Extension that copies static files.

Content can include figures, images and videos. Static files can be included in a special folder named `_static`. The files will be copied to the output directory automatically.

1.4.5.1 Translation

It's possible to have language-specific versions of a static file.

For that to work you need to have a `_static` folder beneath the language folder. Files in this folder will take precedence over the common `_static` folder for that language.

Example: `en/_static/example.png` takes precedence over `_static/example.png` in the English version.

1.4.5.2 Relative and absolute reference

Files can be referenced using relative or absolute paths.

Absolute paths are resolved to the root folder, either the common (`_static`) or language-specific (`en/_static`) folder.

Relative paths are resolved to the root folder but have the chapters path fragment appended.

Example

This example shows how a reference to an image is resolved. The references happen inside the section `chapter01` in the English language version.

Type	Resolution
Relative	`subdir/my_picture.png` → `en/_static/chapter01/subdir/my_picture.png`
Absolute	`/subdir/my_picture.png` → `en/_static/subdir/my_picture.png`

class innoconv.extensions.copy_static.**CopyStatic**(**args*, ***kwargs*)

Bases: *innoconv.extensions.abstract.AbstractExtension*

Copy static files to the output folder.

This extension copies checks the AST for references to static files and copies them from the content source directory to the output directory.

start(*output_dir*, *source_dir*)

Remember directories.

pre_conversion(*language*)

Remember current conversion language.

pre_process_file(*path*)

Remember file path.

post_process_file(*ast*, _)

Generate list of files to copy.

finish()

Copy static files to the output folder.

1.4.6 innoconv.extensions.generate_toc

Extension that generates a table of contents.

A table of contents is generated from the course sections and added to the *Manifest*.

class innoconv.extensions.generate_toc.**GenerateToc**(**args*, ***kwargs*)
Bases: *innoconv.extensions.abstract.AbstractExtension*

Generate a TOC from content sections.

start(*output_dir*, _)
Remember output directory.

pre_conversion(*language*)
Remember current conversion language.

pre_process_file(*path*)
Remember current path.

post_process_file(_, *title*)
Add this section file to the TOC.

1.4.7 innoconv.extensions.join_strings

Merge consecutive sequences of strings and spaces into a single string element.

The motivation behind this extension is to make the AST more readable and also to save space by compressing the representation. The actual appearance in a viewer remains identical.

This extension modifies the AST.

1.4.7.1 Example

```
{"t":"Str","c":"Foo"},{"t":"Space"},{"t":"Str","c":"bar"}]    →    {"t":"Str",
"c":"Foo bar"}]
```

class innoconv.extensions.join_strings.**JoinStrings**(**args*, ***kwargs*)
Bases: *innoconv.extensions.abstract.AbstractExtension*

Merge consecutive strings and spaces in the AST.

post_process_file(*ast*, _)
Process AST in-place.

1.4.8 innoconv.extensions.write_manifest

Extension that writes a `manifest.json` to the output folder.

Every course needs a *Manifest*. Additionally to the fields from the source manifest it can include a table of contents and a glossary.

class innoconv.extensions.write_manifest.**WriteManifest**(**args*, ***kwargs*)
Bases: *innoconv.extensions.abstract.AbstractExtension*

Write a manifest file when conversion is done.

start(*output_dir*, _)
Remember output directory.

finish()
Output course manifest.

1.4.9 innoconv.manifest

The manifest comprises course metadata.

A `manifest.yml` file needs to exist in every course content and resides at the content root directory.

There is also a representation in JSON format. It is generated automatically by the extension *WriteManifest* and copied to the output folder.

A `manifest.yml` is written by course authors while the `manifest.json` is generated by the converter and used by innoconv-compatible viewers. The included information for both versions differ.

1.4.9.1 Example

```
title:
  en: Example title
  de: Beispiel-Titel
languages: en,de
```

class innoconv.manifest.**Manifest**(*data*)
Represents course metadata.

classmethod from_directory(*dirpath*)
Read manifest from content directory.

Parameters **dirpath** (*str*) – Full path to content directory.

Return type *Manifest*

Returns Manifest object

classmethod from_yaml(*yaml_data*)
Create a manifest from YAML data.

Parameters **yaml_data** (*str*) – YAML representation of a manifest

Return type *Manifest*

Returns Manifest object

class innoconv.manifest.**ManifestEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)
JSON encoder that can handle Manifest objects.

default(*o*)
Return dict for Manifest objects.

1.4.10 innoconv.runner

The innoConv runner is the core of the conversion process.

It traverses the source directory recursively and finds all content files. These are converted one-by-one to JSON. Under the hood is uses Pandoc.

It receives a list of extensions that are instantiated and notified upon certain events. The events are documented in *AbstractExtension*.

class innoconv.runner.**InnoconvRunner**(*source_dir*, *output_dir*, *manifest*, *extensions*)
Convert content files in a directory tree.

run()
Start the conversion by iterating over language folders.

1.4.11 innoconv.utils

Utility module.

innoconv.utils.**to_ast**(*filepath*)
Convert a file to abstract syntax tree using pandoc.

Parameters **filepath** (*str*) – Path of file

Return type (list of dicts, str)

Returns (Pandoc AST, title)

Raises

- **RuntimeError** – if pandoc exits with an error
- **ValueError** – if no title was found

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index

Symbols

A

C

D

E

F

G

H

I

J

M

P

R

S

T

W