Template Development Kit

The Tutorial

Marek Suchánek
marek.suchanek@ds-wizard.org

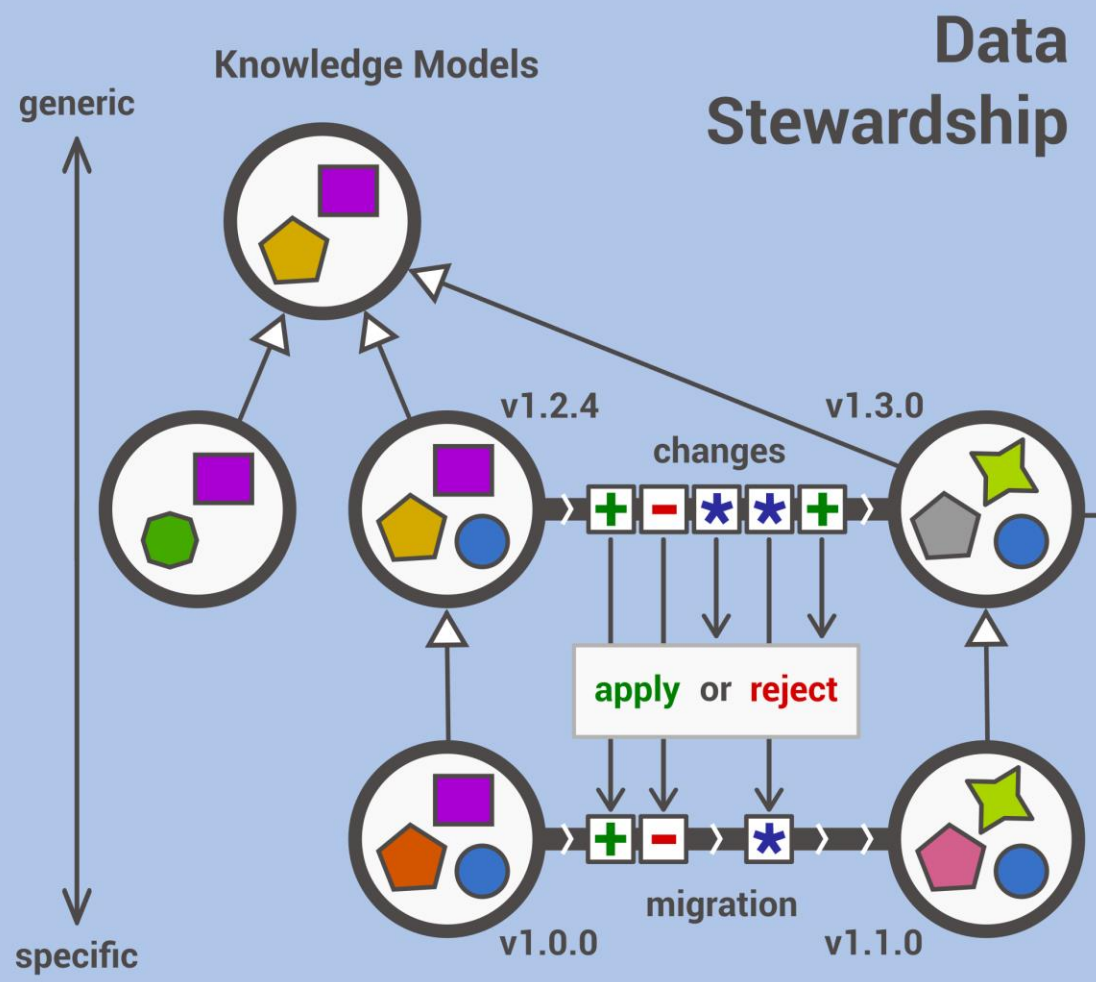
Welcome!

- This is the very first tutorial on DSW TDK released on 4th November
- All information regarding this tutorial is in the document

shorturl.at/gluDT

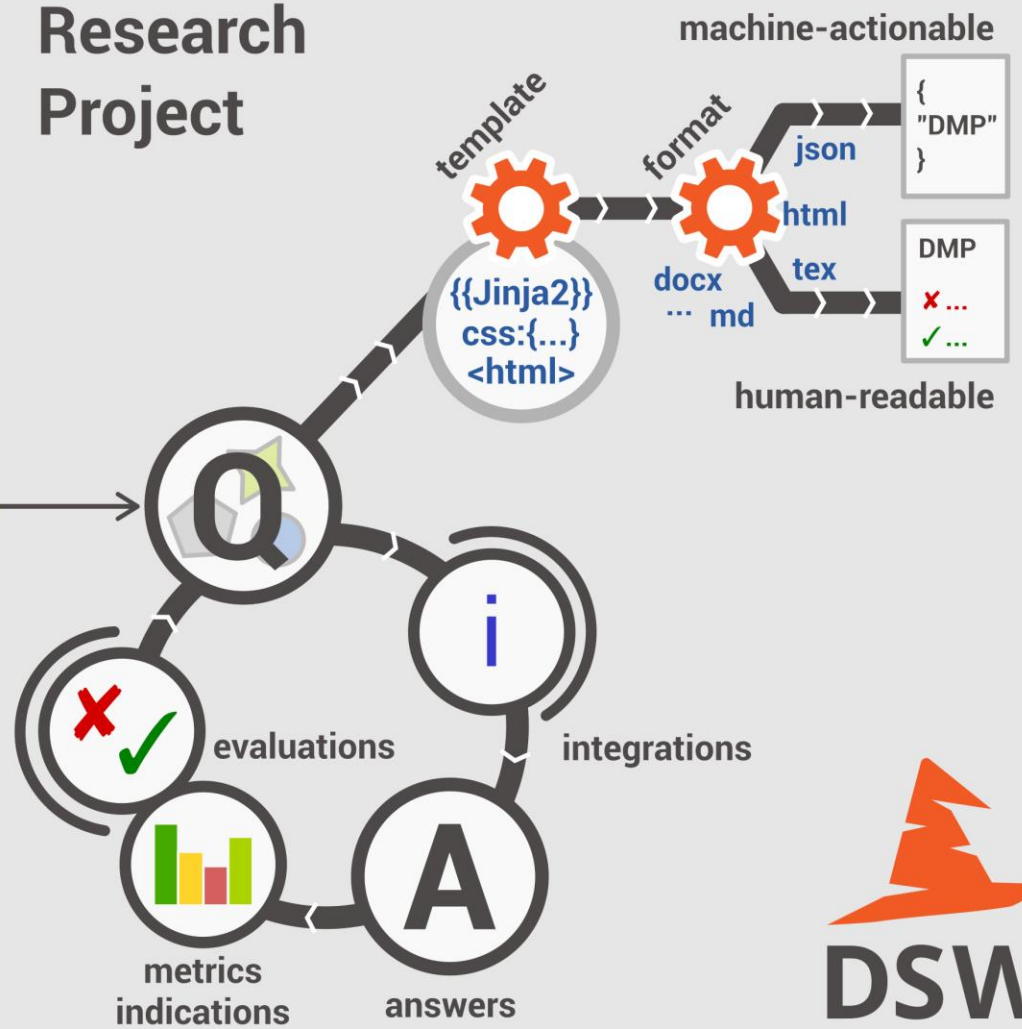
- Please fill your information in the attendance list
- We will be grateful for any feedback and remarks

Templates in DSW

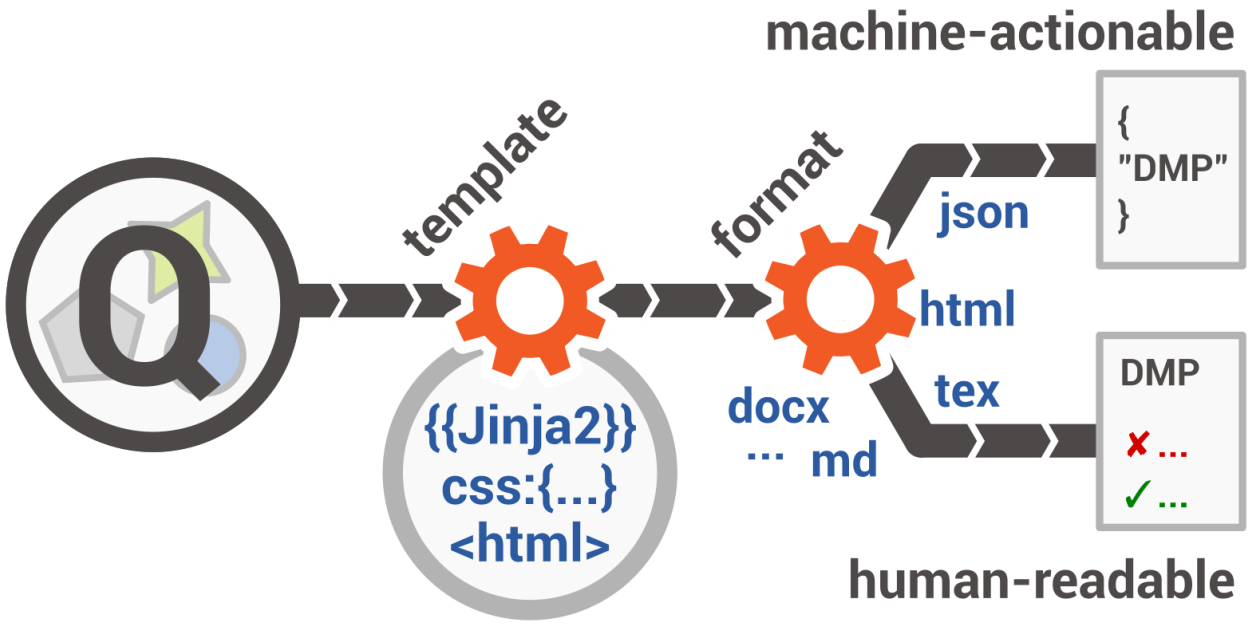


Data Stewardship

Research Project



Templates in DSW



Templates in DSW

New document

Name

HVSC: Hypothetical Vascular Study by a Chemist

Answered (current phase): 4/28

Answered: 11/62

Template

Q **Questionnaire Report** 1.1.0
Exported questions and answers from a questionnaire

Format

- ◉ JSON Data
- ○ HTML Document
- ○ PDF Document
- ○ LaTeX Document
- ○ MS Word Document
- ○ OpenDocument Text
- ○ Markdown Document

Cancel Create

New document

Name

HVSC: Hypothetical Vascular Study by a Chemist

Answered (current phase): 4/28

Answered: 11/62

Template

M **Machine-Actionable DMP** 1.1.0
Machine-Actionable DMP according to the RDA Common Standard

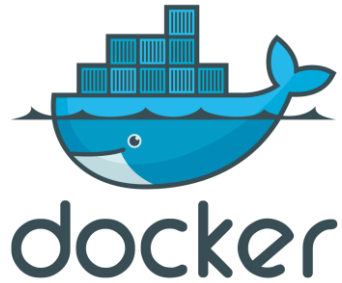
Format

- ○ JSON
- ○ Turtle
- ○ N3
- ○ RDF/XML
- ○ JSON-LD
- ○ N-Triples
- ○ Trig

Cancel Create

Template Development: Prerequisites

- Python (3.6 or higher) or Docker
- DSW (2.8.0 or higher)
- DSW-TDK (**matching** version)



- Technical Knowledge
  - Basic IT knowledge of CLI tools
  - Jinja2 Template Development
  - (optional) Experience with `git`

DSW TDK: Installation

- dsw-tdk is standard Python package on public PyPI

```
# Simply install DSW TDK
$ pip install dsw-tdk

# … or (better) use Python virtual environment
$ python3.9 -m venv env
$ . env/bin/activate
(env) $ pip install dsw-tdk

# … or on Windows
$ python -m venv env
$ . env/Scripts/activate
(env) $ pip install dsw-tdk
```

DSW TDK: Installation

- dsw-tdk is open-source on GitHub

```
# Clone and install directly
$ git clone https://github.com/ds-wizard/dsw-tdk.git
$ git checkout <branch or tag>
$
$ python3.9 -m venv env
$ . env/bin/activate
(env) $ pip install -e .
```

DSW TDK: Basic Usage


```
# Check your TDK version
$ dsw-tdk --version

# … and what next?
$ dsw-tdk --help

# … OK and what „list“ can do?
$ dsw-tdk list --help
```

DSW TDK: Use with Docker

- Optionally you can use Dockerized TDK
- Docker image: [datastewarshipwizard/dsw-tdk]

```
# Simply run with Docker
$ docker run datastewardshipwizard/dsw-tdk --help


# … or some specific version (tag)
$ docker run datastewardshipwizard/dsw-tdk:2.8.0 --help
```

DSW TDK: Connect to DSW Instance


```
# Check your TDK version
$ dsw-tdk list --api-server http://localhost:3000
Username: albert.einstein@example.com
Password: [hidden-input]

# … or use environment variables
$ export DSW_API=http://localhost:3000
$ export DSW_USERNAME=albert.einstein@example.com
$ export DSW_PASSWORD=password
$ dsw-tdk list
```

DSW TDK: Use .env file


```
# Create such .env file
$ cat /path/to/my/.env
DSW_API=http://localhost:3000 Password:
DSW_USERNAME=albert.einstein@example.com
DSW_PASSWORD=password

# Use the .env file
$ dsw-tdk --dot-env /path/to/my/.env list
$ dsw-tdk -e /path/to/my/.env list

# By default it uses the local one
$ cd /path/to/my
$ dsw-tdk list
```

DSW TDK: Verbosity


```
# Normal verbosity (INFO and higher)
$ dsw-tdk list

# Quiet verbosity (WARNING and higher)
$ dsw-tdk --quiet list
$ dsw-tdk -q list

# Debug verbosity (also DEBUG messages)
$ dsw-tdk --debug list
```

DSW TDK: new template


```
# Launch wizard for initiating a template project
$ dsw-tdk new
Template name: My first template
Organization ID: marek
Template ID [my-first-template]: first
Version [0.1.0]:
…

# Investigate the new template project
$ ls -la marek_first_0.1.0
```

Anatomy of template project

- There is always the **template.json** file with:
  - Basic metadata: identifiers, license, description, version, ...
  - Formats: what document formats are supported and how (steps)
  - TDK Config: configuration for local template project
- Then there is **README.md** with details and changelog
- And finally any other files for the documents: **templates and assets**
- For more details, visit our documentation

DSW TDK: `put` local template to DSW


```
# Upload template project from a specific directory
$ dsw-tdk put marek_first_0.1.0

# Or just from the local one
$ cd marek_first_0.1.0
$ dsw-tdk put

# Clean up first (in case that the template already exists)
$ cd marek_first_0.1.0
$ dsw-tdk put --force
$ dsw-tdk put -f

# Watch the template files and re-upload on change
$ dsw-tdk put --watch
$ dsw-tdk put -w
```

Investigating document context

- DSW passes document context to Jinja2 template as `ctx`

- It contains:
  - Metadata about the questionnaire and document
  - Used knowledge model (KM) for the questionnaire with all entities
  - Replies to the questions in the questionnaire
  - … and other including metric evaluations etc.

- It can be easily represented as JSON object using the default template or by using `json` step in your template

Document Context: Metadata


```
{
  "config": {
    "clientUrl": "https://demo.ds-wizard.org",
    "levelsEnabled": true
  },
  "createdAt": "2020-11-09T09:17:42.279013944Z",
  "createdBy": {…},
  …
  "package": {…},
  "questionnaireName": "My own questionnaire",
  "questionnaireUuid": "1587c855-ef55-4658-91c5-21342fb70e38",
  "updatedAt": "2020-11-09T09:17:42.279013944Z",
  "uuid": "cfc7cb73-076a-4137-8356-f5cdc137a728"
}
```

Document Context: KM Entities

- `knowledgeModel` contains:
  - `entities` = typed maps of KM entities indexed by UUIDs
  - `…Uuids` = „root“ lists of UUIDs, e.g., chapters in KM

```
{
  …
  "knowledgeModel": {
     "chapterUuids": […],
     "entities": {…},
     …
  },
  …
}
```

Document Context: Replies

- `questionnaireReplies` is a map of replies with paths as keys
- path is unique dot-concenated list of UUIDs (chapter, ..., question)
- value is object with `value` and `type` attributes

```
{
  …
  "questionnaireReplies": {
     "<reply-uuid-path>": { type: "…", value: "…"},
     …
  },
  …
}
```

Accessing specific replies

- You can access the replies when knowing UUID path from KM
- There are prepared Jinja2 filters for this purpose

```
{%- set repliesMap = ctx.questionnaireReplies -%}
{%- set path = [<chapterUuid>,…,<questionUuid>]|replyPath -%}
{%- set stringReply = repliesMap[path]|reply_str_value -%}
{#- similarly with reply_items or reply_int_value #-}
{#- then you can work with the reply… e.g. print it out -#}
And you reply is: {{ stringReply }}
```

Iterating throught KM

- Sometimes your need go through all chapters and questions…
- … or just access details of a single question.

```
{%- set km = ctx.knowledgeModel -%}
{%- for chapterUuid in km.chapterUuids -%}
    {%- set chapter = km.entities.chapters[chapterUuid] -%}
    {#- then you can work with chapter object easily… -#}
    Chapter: {{ chapter.title }}
{%- endfor -%}
```

What else Jinja2 offers

- Conditions: `if`, `elif`, `endif`
- Macros (for re-usability): `macro`, `endmacro`
- Composability of template: `include`, `extends`, `import`
- Many built-in filters
- … and may more, see their documentation

Template Development Suggestions

- Use human-friendly naming and structure of the template
- Use `git` repository for your template
- Maintain **changelog** in your **README.md** file
- Make corresponding `git tag`s in repository upon release
- Test the template carefully before deploying to production instance

DSW TDK: `get`, `verify`, and `package`


```
# Retrieve template from DSW instance as a local project
$ dsw-tdk get <template-id>

# Check the contents of the current template project
$ dsw-tdk verify

# Create a ZIP package for manual import to DSW
$ dsw-tdk package
```

DSW

DATA STEWARDSHIP WIZARD

Questions and Discussion

We are always grateful for any feedback!

This work was supported by ELIXIR CZ research infrastructure project (MEYS Grant No: LM2015047)