Writing Hydra configuration files

Martijn J. Schuemie

2023-01-18

Contents

1 Introduction

This vignette describes how developers of package skeletons can write Hydra configuration files. Hydra configuration files tell Hydra how to hydrate the skeleton according to a study specifications object. The Hydra configuration file should be embedded in the skeleton zip file.

1.1 Study specifications

The study specifications are generated by some external editor, such as ATLAS. Study specifications will be in JSON format. The specifications will define every aspect of the study, such as the cohorts to use as exposures and outcomes, and what covariates to include. Here are the first few lines of an example study specification:

```
{
    "id": 1,
    "version": "v0.9.0",
    "name": "Study of some cohorts of interest",
    "packageName": "ASimpleStudy",
    "skeletonType": "SimpleExampleStudy",
    "skeletonVersion": "v0.0.1",
    "createdBy": "schuemie@ohdsi.org",
    "createdDate": "2018-03-09T18:25:43.511Z",
    "modifiedBy": "schuemie@ohdsi.org",
    "modifiedDate": "2018-04-13T18:25:43.511Z",
    "cohortDefinitions": [{
            "id": 1,
```


1.2 Package skeleton

A package skeleton is an R package for fully executing a specific type of study, such as a new-user cohort study or a predictive modeling study. The skeleton has placeholders for study elements such as those specified in the study specifications. Package skeletons are provided as zip files, and are embedded inside Hydra (see the `inst/skeletons` folder) .

1.3 Hydration

By hydration we mean the process by which the package skeleton is configured according to the study specification to become a fully executable study package. A study package will perform all tasks necessary to execute the study at a site, including creation of the cohorts needed in the study.

2 Hydration configuration file example

The hydration configuration is a JSON file. Below is an example configuration:

```
{
    "skeletonType": "SimpleExampleStudy",
    "skeletonVersion": "v1.0.0",
    "requiredHydraVersion": "v0.0.1",
    "actions":[{
        "type": "fileNameFindAndReplace",
        "input": "packageName",
        "find": "SimpleExampleStudy"
    },{
        "type": "stringFindAndReplace",
        "input": "packageName",
        "find": "SimpleExampleStudy"
    },{
        "type": "jsonArrayToCsv",
        "input": "cohortDefinitions",
        "mapping": [{"source": "id", "target": "cohortId"},
                    {"source": "id", "target": "atlasId"},
                    {"source": "name", "target": "name", "modifiers": ["convertToFileName"]}],
        "output": "inst/settings/CohortsToCreate.csv"
    },{
```

```
        "type": "jsonArrayToJson",
        "input": "cohortDefinitions",
        "fileName": "name",
        "payload": "expression",
        "output": "inst/cohorts"
    },{
        "type": "jsonArrayToSql",
        "input": "cohortDefinitions",
        "fileName": "name",
        "payload": "expression",
        "output": "inst/sql/sql_server"
    }]
}
```

The configuration consists of some meta data such as "skeletonType", followed by an array of actions. In this example, there are 5 actions that will be executed by Hydra in sequence after unzipping the skeleton:

1. Find all files named `SimpleExampleStudy.*`, and rename them to the name specified in the "packageName" attribute in the study specifications (while keeping the same extention).
2. Find all mentions of the string `SimpleExampleStudy` in all files in the package folder, and replace it with the string specified in the "packageName" attribute in the study specifications.
3. Create a CSV file called `inst/settings/CohortsToCreate.csv`, and populate it with the elements from the `cohortDefinitions` array in the study specifications.
4. Create JSON files in the `inst/cohorts` folder, one for each element in the `cohortDefinitions` array in the study specifications.
5. Create SQL files in the `inst/sql/sql_server` folder, one for each element in the `cohortDefinitions` array in the study specifications. The SQL is generated by Hydra using Circe.

3 Conditions

Each action can be made conditional by adding a *condition* field. For example, in this configuration the `jsonToArgs` action is only executed if the `doPositiveControlSynthesis` field in the study specifications resolves to the value 'true' (or '1).

```
{
        "type": "jsonToRargs",
        "input": "positiveControlSynthesisArgs",
        "condition": "doPositiveControlSynthesis",
        "file": "R/SynthesizePositiveControls.R",
        "startTag": "# Start positiveControlSynthesisArgs",
        "endTag": "# End positiveControlSynthesisArgs"
}
```

Boolean logic is also permitted in condition fields. Here are some more examples of conditions:

```
"(mainSettings.subsetting.type == 'foo') & bar != 3"
"fooOption IN ('bar', 'pie', 'sky')"
"(foo == 1) | (bar == 2)"
```

4 Action types

Below all available action types and their arguments are described. Whenever an element in the study specification is referenced, nested elements can be accessed by using a dot ('.'). For example, "estimationAnalysisSettings.analysisSpecification" returns the `analysisSpecification` element of the `estimationAnalysisSettings` root element.

4.1 fileNameFindAndReplace

Finds all files with the given name, and renames them to a target name (while maintaining the file extension).

Arguments:

- *input*: The path to the element in the study specification containing the target name.
- *find*: The name to look for. Extensions should not be included.
- *condition*: Optional: a condition that must be met to execute the action (see section on conditions).

4.2 stringFindAndReplace

Finds all mentioned of a string in all files in the package, and replaces them with a target string.

- *input*: The path to the element in the study specification containing the target string
- *find*: The string to look for.
- *condition*: Optional: a condition that must be met to execute the action (see section on conditions).
- *exclude*: Optional: file mask to exclude conforming filenames from action execution, it could a single string or an array of strings. For example, `*.jar` excludes all jar files.

4.3 jsonArrayToCsv

Convert an array in the study specifications to a CSV file in the package, where each element of the array will generate one row in the CSV file.

- *input*: The path to the array in the study specifications.
- *mapping*: An array of objects with the following elements:
  - *source*: The name of the element to pull from the array element in the study specification
  - *target*: The name of the column in the CSV file.
  - *separator*: Optional: if the source element is itself an array, what separator should be used to collapse that array into a single string?
  - *modifiers*: Optional: modifiers to be applied to the source value before entering in the CSV. This should be an array of strings, with these possible values:
    - *convertToFileName*: Make the source string suitable for creating file names.
- *output*: The name of the CSV file to create.
- *condition*: Optional: a condition that must be met to execute the action (see section on conditions).

4.4 jsonArrayToJson

Convert an array in the study specifications to a set of JSON files in the package, where each element of the array will generate one JSON file.

- *input*: The path to the array in the study specifications.
- *fileName*: The element inside the array element that will be used to generate the JSON file name. The source string will be modified to be made suitable for file names. The ".json" extention will automatically be added.
- *payload*: The element inside the array element that will be the content of the JSON file.
- *output*: The folder in the package where the JSON files will be written.
- *condition*: Optional: a condition that must be met to execute the action (see section on conditions).

4.5 jsonArrayToSql

Convert an array in the study specifications to a set of SQL files in the package, where each element of the array will generate one SQL file. SQL files are automatically generated using Circe.

- *input*: The path to the array in the study specifications.
- *fileName*: The element inside the array element that will be used to generate the SQL file name. The source string will be modified to be made suitable for file names. The ".sql" extention will automatically be added.
- *payload*: The element inside the array element that will be converted to SQL.
- *output*: The folder in the package where the SQL files will be written.
- *generateStats*: Optional: Add statements to the SQL that will compute inclusion rule statistics? ('true' or 'false')
- *condition*: Optional: a condition that must be met to execute the action (see section on conditions).

4.6 jsonToSql

Convert a single element in the study specifications to a SQL file in the package using Circe.

- *input*: The path to the element in the study specifications.
- *expressionType*: Optional: The type of expression that is to be converted. Currently supports 'cohort' for generic cohort expressions, and 'outcome' for (negative control) outcome cohorts. If not specified, will default to 'cohort'.
- *output*: The file in the package where the SQL will be written.
- *generateStats*: Optional: Add statements to the SQL that will compute inclusion rule statistics? ('true' or 'false')
- *condition*: Optional: a condition that must be met to execute the action (see section on conditions).

4.7 jsonToJson

Convert a single element in the study specifications to a JSON file in the package.

- *input*: The path to the element in the study specification containing the JSON to write to file.
- *output*: The name of the JSON file to create.
- *condition*: Optional: a condition that must be met to execute the action (see section on conditions).

4.8 jsonToRargs

Convert an element in the study specifications to R arguments.

- *input*: The path to the element in the study specification that should be converted.
- *file*: The name of the existing R file in the package where the arguments should be inserted.

- *startTag*: The string in the R file that denotes the start location for inserting the arguments.
- *endTag*: The string in the R file that denotes the end location for inserting the arguments.
- *argumentFunctions*: An array of objects that defines what R functions to use to create nested arguments. These objects should have these elements:
  - *source*: The name of the element that contains nested arguments
  - *function*: The function to use in R to combine the nested arguments into a single argument.
- *condition*: Optional: a condition that must be met to execute the action (see section on conditions).

For example, imagine the study specifications contains this element:

```
"args" : {
  "foo": "Hello",
  "bar": {
    "x": 123,
    "y": 456
  }
}
```

And the Hydra configuration specifies:

```
{
    "type": "jsonToRargs",
    "input": "args",
    "file": "R/fooBar.R",
    "startTag": "# Start fooBar",
    "endTag": "# End fooBar",
    "argumentFunctions": [{"source": "bar", "function": "createBar"}]
}
```

If the original file fooBar.R in the skeleton contains:

```
doFooBar(
  # Start fooBar
  foo = "Bye",
  bar = NULL
  # End fooBar
)
```

The hydrated fooBar.R will become:

```
doFooBar(
  foo = "Hello",
  bar = createBar(x = 123,
                  y = 456)
)
```