Using Input Manifests

Contents

1 Introduction

Manifests are files the track inputs that are needed for a study. Ulysses provides a series of helper functions to initialize and populate manifest files to keep inputs to a study organized and tracked. The idea of the manifest is to track input file and its progression over the study life cycle. There are two kinds of manifests: concept set and cohort. Additionally there are two file types: manifest and manifestLog. The manifestLog file adds in checks to ensure if an asset has been deprecated. Users only need to look at the manifest file. The log is used to resolve updates over time.

1.1 Cohort Manifest

The cohort manifest tracks assets that define study populations and cohorts of interest in a study. With OHDSI tools, we track a "circe-be" object that lists the logic of a cohort definition in a json file which can be serialized into a standard sql script that can be consistently executed to enumerate the population of interest. Ulysses is optimized to track the json representations of cohorts, but one can easily alter this to tracking a custom sql script.

1.2 Concept Set Manifest

The concept set manifest tracks assets that capture clinical terminology used in a study. More commonly these are thought of as code lists. The json file stored in Ulysses is optimized for OHDSI tools, however it can be easily altered to use code list csvs for raw database analyses.

1.2.1 A tangent on Concept Sets

A concept set describes a clinical term plus its related terms, including descendancy and mapped terms. With the OMOP CDM we use standard vocabularies to represent clinical terms and improve roll-up logic to additional terms. Each standard term has parents and children nodes that form the heirarchy of the clinical term. For example diabetes includes: type 1 and type 2 diabetes. Instead of searching for each descendant term, I could identify the parent term and include its descendant concepts; a more efficient way to represent code lists programmatically. On a related note, clinical terms have different coding in different databases. For

example CPRD uses READ codes while a typical US claims database uses ICD10CM. Standard vocabularies allow us to efficiently identify clinical terms across different databases following a standard representation. Concept sets in OHDSI allow us to do this!

1.3 Manifest Structure

A manifest file is a csv that tracks important fields in a tabular fashion. They include the following fields:

- id: this is the manifest Id, it should not be touched
- atlasId: the id of the asset as stored in WebApi
- label: the common name for asset
- category: a category tag that gives the asset a identification group
- subCategory: a sub category tag that gives a second identification group
- name: a file name of the asset
- path: the relative path of the asset file

The manifestLog adds a field called isDeprecated which tracks whether the asset still exists in the file structure.

2 Workflow

Below we provide an example of how to use the manifest functions from Ulysses to track cohorts and concept sets over the study life-cycle.

2.1 Step 1: Initialize the Manifest

To start we initialize the manifest, creating an empty table for each manifest type.

```
# initialize cohort manifest
initializeManifest(manifestType = "cohort")

# initialize concept set manifest
initializeManifest(manifestType = "conceptSet")
```

Once the manifest is initialized for the first time, we can then manually added content into the file. Open the raw version of the file and add details. See example below:

```
atlasId,label,category,subCategory,id,name,path
123,Type 2 Diabetes,target,,,,,
456,Heart Failure,outcome,condition,,,
```

Do not fill in the id, name and path columns. These are filled out by the populateManifest function. Notice we add in the labelling information that we know such as the atlasId, the label of the asset and the grouping categories.

It is also possible to load this information prior to initializing the manifest using the function `defineLoadTable()`, see example below.

```
# set cohorts to use
tb <- defineLoadTable(
  atlasId = c(123,456),
  label = c("Type 2 Diabetes", "Heart Failure"),
  category = c("target", "outcome"),
  subCategory = c(NA_character_, "condition")
)

# init cohort Manifest
```

```
initializeManifest(
  manifestType = "cohort",
  loadTable = tb
)
```

By importing the prespecified table, we do not need to manually add this to the manifest csv file.

The `initializeManifest` function has one more option called overwrite. This will remove any existing manifest with a newly initialized manifest.

2.2 Step 2: Populate the Manifest

Once the manifest has been initialized, it is time to populate the information. The populate step takes cohorts or concept sets saved as files and stores them as entries in the manifest using the meta information applied, such as the label.

```
populateManifest(manifestType = "cohort", importFromAtlas = TRUE)
```

Populate manifest will only fill out information for files that exist in the cohorts or concept set folders. You can do this manually or use the importFromAtlas utility. This option uses your webApi credentials to connect and scrap circe json to place in Ulysses. In order to use the importFromAtlas feature, users need to configure a connection to WebApi. See that section for details.

3 Set up WebApi Connection

3.1 Set Credentials

Before a use can import atlas assets, they need to set their WebApi credentials as system variables to pass through authentication. Ulysses stores WebApi credentials in .Renviron file. To access the `.Renviron` file use the function: `usethis::edit_r_environ()`. Ulysses offers a template function to provide guidance of the proper credential set up:

```
templateAtlasCredentials()
```

To find the baseUrl you can look in the configuration tab of Atlas. The user is usually an email address if the authentication method is ad and the password is the password to the corresponding user. Please contact someone in your organization for details. Input these items into the `.Renviron()` and restart the session to invoke the changes.

3.2 Atlas Connection Object

Once credentials have been set, users can create a WebApiConnection R6 class that manages credentials, authorization and scraping of assets. Note that this is all handled implicitly in the `populateManifest` function. This section only provides additional details.

```
atlasCon <- setAtlasConnection() # create object
atlasCon$checkAtlasCredentials() # check credentials. note password is hidden
atlasCon$getCohortDefinition(cohortId = 123) # grab cohort def from WebApi
```