Mighty Metadata: YAML-Powered ADaM Specifications

Aksel Thomsen

CDISC Europe Interchange, 21 May 2026

Disclaimer

- All presenters are employees of Novo Nordisk A/S
- Views and opinions expressed are those of the presenters and not necessarily Novo Nordisk A/S
- Neither Novo Nordisk A/S nor the presenters have any financial interests in any of the software mentioned during the presentation

Speaker

Aksel Thomsen

Clinical Data Science Specialist @ Novo Nordisk A/S

Economist turned R developer and clinical data scientist with a goal to make trial programming easier and more efficient.

Developer on several Open-Source packages including {whirl} and {connector}.

akselthomsen

akselthomsen

novo nordisk®

Agenda

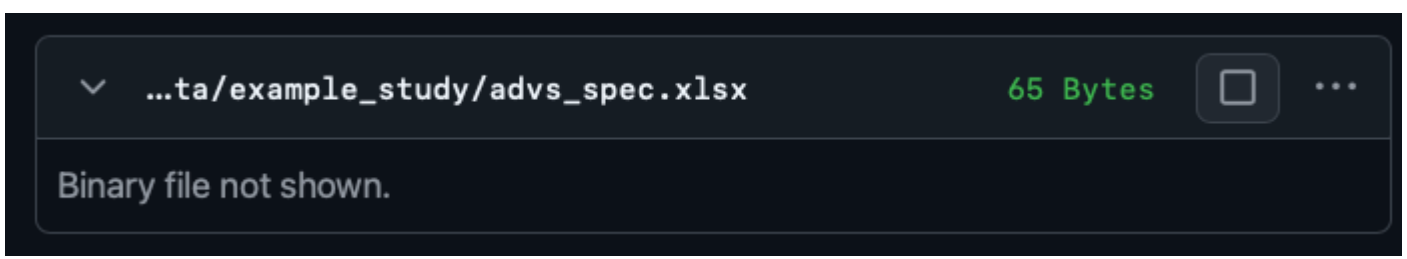
1. **Motivation** — Why replace current Excel solutions?
2. **YAML Specifications** — Study ADaM structure in YAML
3. **{mighty.metadata}** — Inspect, modify, validate metadata with R
4. **Ecosystem Sneak Peek** — Automation and downstream use

Motivation

Current challenges

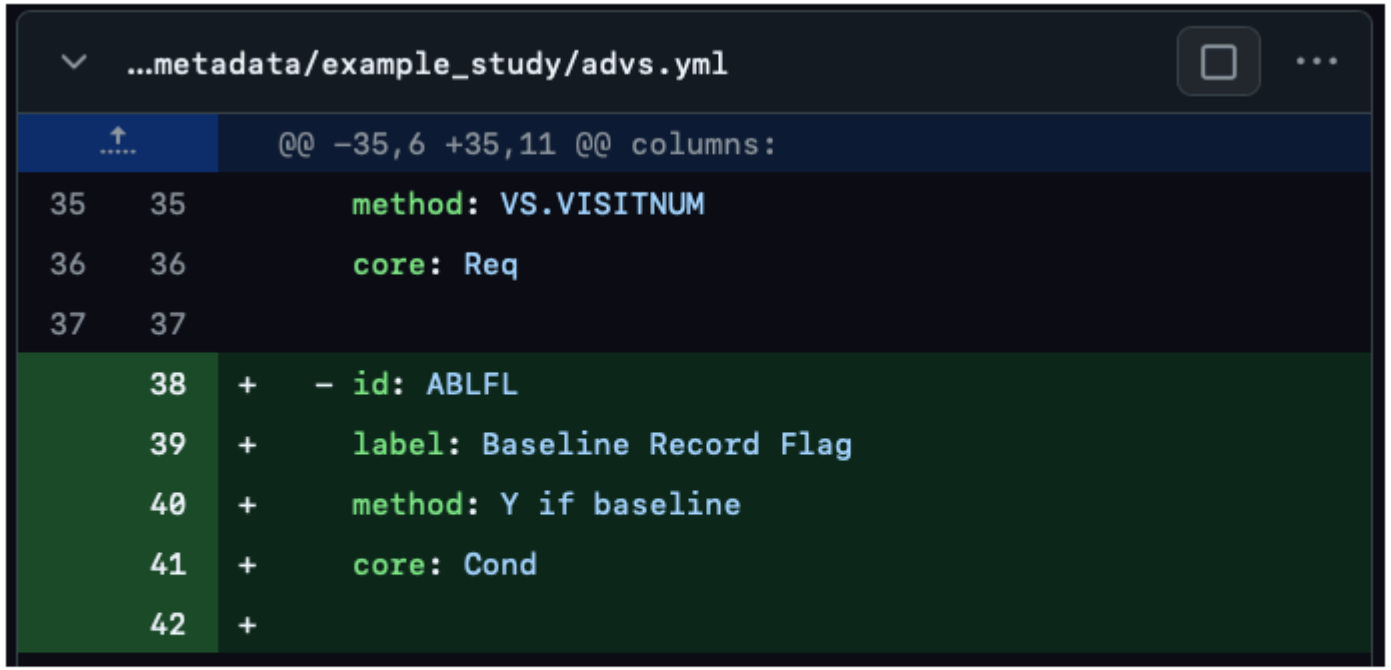
- ADaM specifications live in Excel
- Collaboration and same-file editing is challenging
- Not directly machine-readable
- Scripts are programmed separately
- Not suitable for version control — no diffs, no history, no blame
- Validation on differences next to impossible

	A	B	C	D	E	F	G	H
1	table_id	table_label	order	id	label	origin	core	method
2	ADVS	Vital Signs A	1	STUDYID	Study Identifier		Req	VS.STUDYID
3	ADVS	Vital Signs A	2	USUBJID	Unique Subject Identifi		Req	VS.USUBJID
4	ADVS	Vital Signs A	3	PARAMCD	Parameter Code		Req	VS.VSTESTCD
5	ADVS	Vital Signs A	4	PARAM	Parameter		Req	VS.VSTEST
6	ADVS	Vital Signs A	5	AVAL	Analysis Value		Req	VS.VSSTRESN
7	ADVS	Vital Signs A	6	AVISITN	Analysis Visit (N)		Req	VS.VISITNUM
8	ADVS	Vital Signs A	7	ABLFL	Baseline Record Flag		Cond	Y if baseline



Advantages of our new framework

- ADaM specification now in YAML
- Easy collaboration - same as for scripts
- Both human and machine readable - enables tooling and AI Assistants
- Enables standard version control with Git
- Easy to validate differences



YAML Specifications

Directory structure

- Organise all ADaM metadata in one directory
- Underscore-prefixed files are study or project configurations
- **One YAML** file per **ADaM domain**

```
example_study/
  _study.yml       # Study-level metadata
  _mighty.yml      # Mighty framework configuration
  adsl.yml         # Subject-Level Analysis Dataset
  adae.yml         # Adverse Events Analysis Dataset
  advs.yml         # Vital Signs Analysis Dataset
  ...
```

Study-level metadata

- Describe all study level information
- Key-value pairs, but no other restrictions
- Enables downstream automations

`_study.yml`:

```yaml
study_id: example_study
study_description: This is an example study for the CDISC EU Interchange 2026
```

Domain-level metadata

- Describe all domain level information
- Fixed structure
- Enforcement of CDISC compliance

`advs.yml (selected entries)`:

```yaml
id: ADVS
label: Vital Signs Analysis Dataset
class: BASIC DATA STRUCTURE
structure: One record per vital sign parameter, per visit, per subject
keys: [USUBJID, PARAMCD, AVISITN]
columns:
  - id: STUDYID
    label: Study Identifier
    method: VS.STUDYID
    core: Req
  - id: AVAL
    label: Analysis Value
    method: VS.VSSTRESN
    core: Req
parameters:
  - id: BMI
    label: Body Mass Index (kg/m^2)
    columns:
      - id: AVAL
        method: Derived from height and weight
```

{mighty.metadata}

Study-level Specifications

Load study:

```
1 study <- mighty_study("example_study")
```

Inspect study object:

```
1 print(study)
```

```
<mighty.metadata::mighty_study/list/S7_object>
@ mighty: `external_data`
@ study: `study_id` and `study_description`
$ ADAE: <mighty.metadata::mighty_domain>
$ ADSL: <mighty.metadata::mighty_domain>
$ ADVS: <mighty.metadata::mighty_domain>
```

List study-level metadata:

```
1 print(study@study)
```

```
List of 2
 $ study_id         : chr "example_study"
 $ study_description: chr "This is example study for the CDISC EU Interchange 2026"
```

Inspect domain-level metadata:

```
1 print(study$ADVS)
```

```
<mighty.metadata::mighty_domain>

ADVS: Vital Signs Analysis Dataset

Class: BASIC DATA STRUCTURE

Keys: USUBJID, PARAMCD, and AVISITN
```

Domain-level specifications

List current content of ADVS:

```
1 list_columns(study$ADVS)
```

```
[1] "STUDYID" "USUBJID" "PARAMCD" "PARAM"   "AVAL"    "AVISITN"
```

```
1 list_parameters(study$ADVS)
```

```
[1] "BMI"
```

How is BMI calculated?

```
1 select_column(study$ADVS, "PARAMCD")
```

```
List of 4
 $ id    : chr "PARAMCD"
 $ label : chr "Parameter Code"
 $ method: chr "VS.VSTESTCD"
 $ core  : chr "Req"
```

```
1 select_column(study$ADVS, "AVAL")
```

```
List of 4
 $ id    : chr "AVAL"
 $ label : chr "Analysis Value"
 $ method: chr "VS.VSSTRESN"
 $ core  : chr "Req"
```

```
1 select_parameter(study$ADVS, "BMI")
```

```
List of 3
 $ id     : chr "BMI"
 $ label  : chr "Body Mass Index (kg/m^2)"
 $ columns:List of 1
  ..$ :List of 2
  .. ..$ id    : chr "AVAL"
  .. ..$ method: chr "Derived from height and weight"
```

Add new parameter

We want to add a BMI grouping variable:

1. New AVALC column
2. New BMIGRP parameter with AVAL and AVALC defined

```
study$ADVS <- study$ADVS |>
  add_column(id = "AVALC", method = "See parameter") |>
  add_parameter(
    id = "BMIGRP",
    label = "BMI grouping",
    columns = list(
      list(id = "AVAL", method = "Sorting variable for AVALC"),
      list(id = "AVALC", method = "BMI grouped into <25, >=25")
      )
    )
```

Check result

```
1 select_parameter(study$ADVS, "BMIGRP")
```

```
List of 3
 $ id     : chr "BMIGRP"
 $ label  : chr "BMI grouping"
 $ columns:List of 2
  ..$ :List of 2
  .. ..$ id    : chr "AVAL"
  .. ..$ method: chr "Sorting variable for AVALC"
  ..$ :List of 2
  .. ..$ id    : chr "AVALC"
  .. ..$ method: chr "BMI grouped into <25, >=25"
```

Consistent interface

Action	Columns	Parameters	Rows
List	`list_columns()`	`list_parameters()`	`list_rows()`
Select	`select_column()`	`select_parameter()`	`select_row()`
Add	`add_column()`	`add_parameter()`	`add_row()`
Update	`update_column()`	`update_parameter()`	`update_row()`
Move	`move_column()`	`move_parameter()`	`move_row()`
Remove	`remove_columns()`	`remove_parameters()`	`remove_rows()`

Complete the workflow

Save new configuration

```
# Entire study
write_config(study, "path/to/my_study")

# Individual domain
write_config(study$ADVS, "path/to/my_study/advs.yml")
```

New `advs.yml`:

```yaml
id: ADVS
label: Vital Signs Analysis Dataset
class: BASIC DATA STRUCTURE
structure: One record per vital sign parameter, per visit, per subject
keys:
- USUBJID
- PARAMCD
- AVISITN
columns:
- id: STUDYID
  label: Study Identifier
  method: VS.STUDYID
  core: Req
- id: USUBJID
  label: Unique Subject Identifier
  method: VS.USUBJID
  core: Req
- id: PARAMCD
  label: Parameter Code
  method: VS.VSTESTCD
```

Validation

- How do we ensure correct metadata format?
- Easy validation
- Safe downstream use and automation
- Solution: JSON schema
- Defined for both study- and domain-level metadata
- How does it look in practice?

Validate entire objects:

```
validate(study)
validate(study$ADVS)
```

Validates when updating:

```
study$ADVS |>
  add_parameter(id = "NEW", label = "Param with missing columns", columns = list())
```

```
Error in `validate_S7schema()`:
! /parameters/3/columns must NOT have fewer than 1 items
✖ limit: 1
```

Automation and downstream use

Core variables

- Core variables repeat across all ADaM domains
- Manually adding them to every domain is tedious and error-prone
- This has to be automated!

```r
list_columns(study$ADAE)
```

```
[1] "STUDYID"  "USUBJID"  "AESEQ"    "AETERM"   "AEDECOD"
"AEBODSYS" "ASTDT"
[8] "AENDT"    "TRTEMFL"
```

```r
study <- populate_core(study)
```

```r
list_columns(study$ADAE)
```

```
 [1] "STUDYID"  "USUBJID"  "AESEQ"    "AETERM"   "AEDECOD"
"AEBODSYS"
 [7] "ASTDT"    "AENDT"    "TRTEMFL"  "SEX"      "RACE"
```

`adsl.yml` (excerpt):

```yaml
columns:
  - id: SEX
    label: Sex
    is_core: true
    core: Req

  - id: RACE
    label: Race
    is_core: true
    core: Req
```

`adae.yml` (excerpt):

```yaml
id: ADAE
label: Adverse Events Analysis Data
class: OCCURRENCE DATA STRUCTURE
subclass: ADVERSE EVENT
structure: One record per adverse ev
keys: [USUBJID, AESEQ]
usecore: true
```

Sparse information

- Core variables were propagated
- Metadata is still incomplete
- *origin* tells us it is a **Predecessor**
- *method* tells us the source is **ADSL.SEX**
- *populate_sparse()* fills in missing metadata automatically
- **label** and **core** copied from source

```
1 select_column(study$ADAE, "SEX")
```

```
List of 3
 $ id    : chr "SEX"
 $ method: chr "ADSL.SEX"
 $ origin: chr "Predecessor"
```

```
1 study <- populate_sparse(study)
```

```
1 select_column(study$ADAE, "SEX")
```

```
List of 5
 $ id    : chr "SEX"
 $ method: chr "ADSL.SEX"
 $ origin: chr "Predecessor"
 $ label : chr "Sex"
 $ core  : chr "Req"
```

Metadata datasets

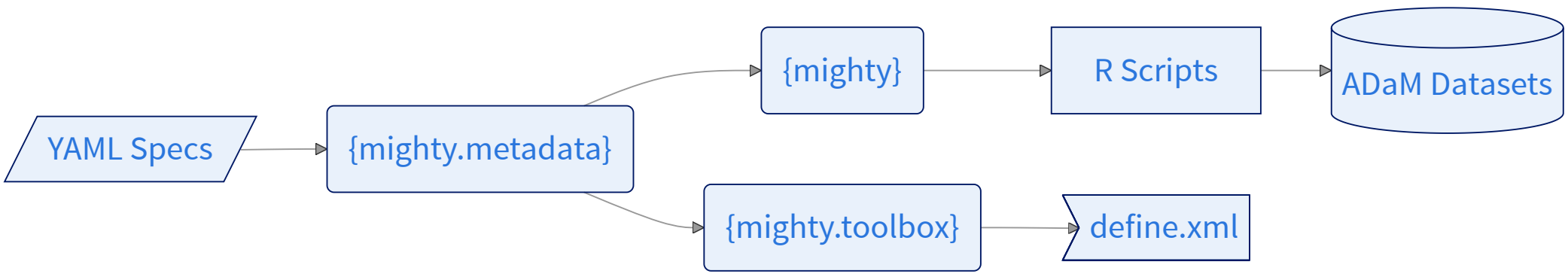
- YAML is good, but tables can give better overview
- Integration with existing tooling
- Create datasets for tables, columns, and parameters

```
create_md_col(study)
```

```
# A tibble: 26 × 15
   table_id table_label      order id    label origin key   is_core core  method
   <chr>    <chr>            <int> <chr> <chr> <chr>  <lgl> <lgl>   <chr> <chr>
 1 ADAE     Adverse Events …     1 STUD… Stud… Prede… FALSE NA      Req   AE.ST…
 2 ADAE     Adverse Events …     2 USUB… Uniq… Prede… TRUE  NA      Req   AE.US…
 3 ADAE     Adverse Events …     3 AESEQ Sequ… Prede… TRUE  NA      Cond  AE.AE…
 4 ADAE     Adverse Events …     4 AETE… Repo… Prede… FALSE NA      Req   AE.AE…
 5 ADAE     Adverse Events …     5 AEDE… Dict… Prede… FALSE NA      Cond  AE.AE…
 6 ADAE     Adverse Events …     6 AEBO… Body… Prede… FALSE NA      Cond  AE.AE…
 7 ADAE     Adverse Events …     7 ASTDT Anal… <NA>   FALSE NA      Cond  Deriv…
 8 ADAE     Adverse Events …     8 AENDT Anal… <NA>   FALSE NA      Cond  Deriv…
 9 ADAE     Adverse Events …     9 TRTE… Trea… <NA>   FALSE NA      Cond  Deriv…
10 ADAE     Adverse Events …    10 SEX   Sex   Prede… FALSE NA      Req   ADSL.…
# ℹ 16 more rows
# ℹ 5 more variables: codelist <chr>, format_type <chr>, format_length <int>,
#   format_display <chr>, comment <chr>
```

Automation

- Goal: One source of truth
- YAML specs drive both code generation and regulatory documentation
- How do we do this?



Deterministic process from metadata to ADaM datasets and define.xml!

Final Study Workflow

1. Define metadata:

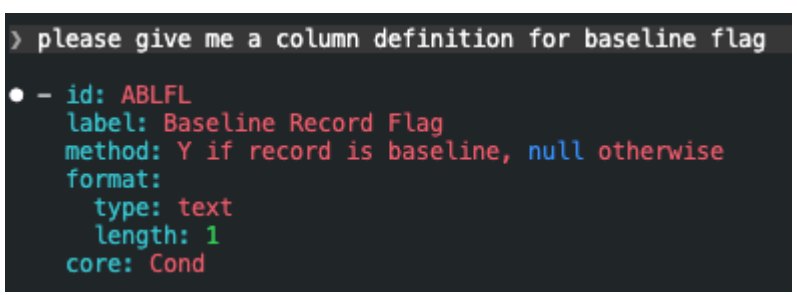
Edit YAML directly:

```yaml
columns:
+ - id: ABLFL
+   label: Baseline Record Flag
+   method: Y if baseline
+   core: Cond
```

Use {mighty.metadata} tooling:

```
study$ADVS |>
  add_column(id = "ABLFL",
             method = "Y if baseline")
```

Use your AI assistants



2. Run automation workflow:

```
# Load study metadata
study <- mighty_study("path/to/my/study") |>
  populate_core() |>
  populate_sparse()

# Create scripts
study |>
  mighty::generate_adam_code() |>
  mighty::write_adam_programs(
    dir = "path/to/adam/code"
  )

# Run scripts to create ADaM
whirl::run("path/to/adam/code")

# Create define.xml
mighty.toolbox::generate_define_xml(
  metadata = study,
  output_dir = "path/to/define/folder"
)
```

Getting Started

Install {mighty.metadata}:

```
# From CRAN
install.packages("mighty.metadata")

# Development version:
pak::pak("NovoNordisk-OpenSource/mighty.metadata")
```

Documentation and code:

mighty.metadata

NovoNordisk-OpenSource/mighty.metadata

Learn more about {mighty}:

mighty

NovoNordisk-OpenSource/mighty

Questions?

Create an issue

oath@novonordisk.com

> *Deterministic Automation Meets Generative AI: ADaM Programming with the Mighty Framework Gives the Best of Both Worlds*
>
> *— Matthew Phelps, Novo Nordisk*
>
> *Session 7, Track B: AI Espresso Shot @ 14.30*