DDI-CDI Use Case Overview: Programmatically Documenting a Simple Wide Data File Described in DDI Codebook

*Arofan Gregory, 1 July 2023, version 0.3*

# Overview

This document attempts to identify what one typical case would look like when a wide data file with a single record segment, either fixed-width or delimited, is mapped into DDI-CDI. Study-level metadata is ignored, as are summary statistics. The focus here is on providing enough structural metadata that the data file itself could be taken and automatically acted on to perform a structural transformation (into, for example, a long data file, or a structural description in another non-DDI standard).

The goal of this review is to identify if the current model can capture this information in a clear and reasonable fashion: is the “simple” case actually simple, or is it too baroque? Initial publication of reference subsets of DDI-CDI are desired to support short-term adoption, and these will – in the first instance – cover the simplest cases such as the one proposed here.

To establish a reference point, the DDI-CDI subset of classes used by the UKDA in their test implementation will be taken as a starting point (see below). While this implementation does not cover the physical description of the data file (these are composed on demand in this implementation) it does indicate a set of classes for describing the metadata payload at the logical level and above.

# Use Case Description

This test case is based on the file “ZAF\_2021\_QLFS-Q1\_v01\_M.xml” which was downloaded from the DataFirst portal (<https://www.datafirst.uct.ac.za/dataportal/index.php/catalog/872/study-description>) after being discovered using the IHSN Survey Catalogue (<https://catalog.ihsn.org/catalog/9937>). It describes a quarterly labour force survey conducted in South Africa in 2021. If you register with DataFirst you can download the microdata – it is public data coming from the statistical office of South Africa.

This use case focuses only on providing a sufficient structural description of the data to support transformation of the data into other structures without loss. It ignores all the documentation at higher levels, and also omits statistics, which could be calculated from the data itself.

The narrative is essentially this: the disseminator of this data set wishes to render it more FAIR by conforming with a minimal profile for describing data structures, as dictated by the CDIF recommendation for data integration. Because they already have this metadata in the form of DDI Codebook, they wish to automatically produce the CDIF output. The profile they wish to publish does not require any information at the study level (provenance, methodological information, etc.) as this is covered by other profiles.

The needed information which is to be expressed can be understood as follows:

1. **Variables:** What variables does the file contain? How are they named and defined in human-readable terms? How are they represented?
2. **Records:** Where in the file can the variables be found? How are they encoded? What is the structure of the records?
3. **Logical Structure:** What is the logical shape of the data in the file (it is a unit-record data file: “wide” in DDI-CDI terms). What are the roles of the variables in relation to this structure?
4. **Physical File Description:** Where can the file be found, and how does it encode the variables? What is the spacing of variable values within the records? What are the delimiters?

# The Reference DDI-CDI Profile: UKDA Wide Data

At UKDA, a large number of data sets have been described logically at the variable level to support a user-driven data-selection application which functions across data sets, with all the data stored in a centralized “big data” repository. As such, the UKDA use case is not identical to ours, but some requirements are similar, and it can act as a guide to what is actually useful to implementers.

A profile of DDI-CDI has been used which describes the variables for the application. While this profile is not a sufficient one for our use case, lacking a description of the data file, it does give us an implemented starting point for the logical description of the contents of a data set.

## The UKDA Subset

The diagram below shows the classes used to describe wide data sets in the UKDA implementation. It contains the following classes from the DDI-CDI model:

DataSet

WideKey

DataStructure

IdentifierComponent

MeasureComponent

AttributeComponent

RepresentedVariablke

InstanceVariable

SubstantiveValueDomain

CodeList

Code

Notation

Category

ConceptualVariable

SubstantiveConceptualDomain

ConceptSystem

Concept

A picture containing text, diagram, parallel, screenshot

Description automatically generated

## Number of Classes in the Subset

This profile uses 17 classes out of the DDI-CDI model, and this is seen as a not-unreasonable number. It will be noticed that for the purposes of the current use case, some obvious classes are missing: the structure of the wide data set is not present (only the DataStructure class at a generic level) and the classes which describe the physical records in the data file. We can expect, therefore, that a larger number of classes will be needed. The target expectation is that the number of additional classes will be no more than the number shown above, roughly: if we can use 30 classes to fulfill our use case requirements, that would be acceptable. If we need many more classes than that, we must begin to ask if the model is not too complex.

# Mapping Requirements to DDI Codebook and DDI-CDI

This section provides an overview into the source of the information to be expressed in DDI-CDI within the DDI Codebook standard. While reference will be made to the example file as a practical example, other common cases will be considered as needed.

This is not a field-level mapping, but a survey of the metadata and how such a mapping could be usefully constructed to meet the requirements of the use case. Detailed mapping could then be developed in a practical way, rather than in the abstract.

## Variables

The variables in DDI Codebook are found in the <dataDscr> element, as a list of <var> elements. An example of one from our sample is shown below:

<var ID="V4" name="Q13GENDER" files="F1" dcml="0" intrvl="discrete">

<labl>Gender</labl>

<location StartPos="22" EndPos="22" width="1"/>

<valrng>

<range UNITS="REAL" min="1" max="2"/>

</valrng>

<sumStat type="vald">45702</sumStat>

<sumStat type="invd"/>

<sumStat type="min">1</sumStat>

<sumStat type="max">2</sumStat>

<catgry>

<catValu>1</catValu>

<labl>Male</labl>

<catStat type="freq">21222</catStat>

</catgry>

<catgry>

<catValu>2</catValu>

<labl>Female</labl>

<catStat type="freq">24480</catStat>

</catgry>

<varFormat type="numeric" schema="other"/>

</var>

This element gives us a mixed set of information about the variable: on the one hand, it tells us the identification and the label for the variable, and shows that it is represented with a set of categories, which are spelled out. On the other hand, it provides information about where in the physical record the values for the variable can be found, and how they are types/formatted. Further, it provides some information useful for statistics, which are not needed in our use case.

In DDI-CDI, these are InstanceVariables – RepresentedVariables used in the specific context of a data set. Given the variable cascade, I can thus implicitly determine a range of the needed metadata:

**InstanceVariable:**  The variable as presented in the Codebook XML. I may need to qualify the identifier with the data set ID and the agency ID, and I may need to provide versioning information (which could be defaulted to “1.0” if absent). Name, DisplayLabel, and Definition properties will all rely on the contents of the Codebook <labl>. While “Gender” is not a lot to work with, this is a typical example, and it conveys a minimal idea of what is in the variable. This is a realistic case. The InstanceVariable will correspond one-for-one with the implicit RepresentedVariable (see below).

**RepresentedVariable:** Implicit in the existence of an InstanceVariable is the existence of a RepresentedVariable – in this case, we are given the set of categories and codes which are valid values for the variable. Again, the information we need is contained in the <var> element: we can come up with a naming convention to distinguish RepresentedVariables from InstanceVariables with a prefix (“inst\_” and “rep\_” or something similar) if needed.

The codelist which is described here will need to be externalized, again by making explicit the metadata which is implied by the Codebook XML. This will involve creating the SubstantiveValueDomain (if that is really needed), the Codelist, the Codes, the Categories, and the Notations from what we are given in the Codebook instance, again relying on naming conventions where there is no explicit object in the source.

Further, if useful, Concepts could be generated for the Categories and for the ConceptualVariable (as could the ConceptualVariable itself). This may not be useful, given that we are literally working in this example with the string “Gender” for our ConceptualVariable, and the labels “Male” and “Female” for our Categories.

Other non-categorical representations are simpler – they will need to be mapped against the set of available datatypes found in DDI-CDI, but this should not be problematic.

## Records

The UKDA reference profile does not cover this part of the DDI-CDI model, so we should briefly describe the classes which may be of interest.

There is only a single type of record in the data file, and it is very consistent. Thus, we can look to the PhysicalSegment and the accompanying PhysicalSegmentLayout. These acts as a way of connecting the LogicalRecord (see below) with the physical layout of that record in the file, as expressed by mapping the variables to specific locations within the record. **[Note that we think there is a “typo” here in the model – ValueMappings need to connect to InstanceVariables, not (only?) DataPoints!]**  ValueMapping allows me to assign a sequence to the InstanceVariables in the record with ValueMappingPosition, and lets me describe the “width” of a fixed-length record. The PhysicalSegmentLayout tells me whether I am dealing with a fixed-width or delimited file, and other, similar information.

All of the detail needed here is available in the Codebook XML <var> element. The set of variables includes all those found in the XML instance, and they describe their physical sequencing and the details of their encoding in the file.

## Logical Structure

The logical structure of my data set operates on two levels: (1) How are the variables grouped by my records in the data file? (2) What roles do the variables play in terms of the data set structure?

The first question is not addressed by our UKDA reference profile, and the second question only partially.

The grouping of variables from a logical perspective is modelled in DDI-CDI using the LogicalRecord class. This connects the DataSet with the PhysicalSegementLayout and the InstanceVariables. It acts as a grouping mechanism for the InstanceVariables which need to be mapped into the physical record structure.

The role of the RepresentedVariables within the data structure requires a class not found in the UKDA reference profile: WideDataStructure. This is a sub-class of DataStructure, and effectively replaces it for the purposes of our use case. The role this plays is to provide a context for the set of RepresentedVariables, which are assigned roles by the Component classes. Note that there is nothing in the DDI Codebook XML which tells us what these roles are with complete accuracy. While we can assume that the first column is a case identifier, this will always need to be verified as a (probably manual) step.

Once the roles of the RepresentedVariables – the Components – have been determined, the WideKey can be described so that the values in the data set can be programmatically referenced.

For all but the Components – the roles played by the variables described in the DDI Codebook XML – we have the ability to programmatically create the objects needed for the description of the data set’s logical structure.

## Physical File Description

A few important details remain: where is the data set found (that is, how can I retrieve it with a URL or similar mechanism)? How can it be identified?

The UKDA reference profile uses DataSet, when we actually need only the sub-class WideDataSet for our use case, so we will need to make that replacement.

The relevant section of the DDI Codebook XML is the file description section:

<fileDscr ID="F1">

<fileTxt>

<fileName>qlfs-2021-q1-worker-v1</fileName>

<dimensns>

<caseQnty>9999</caseQnty>

<varQnty>161</varQnty>

</dimensns>

<fileCont>2021 first quarter labor force survey data.</fileCont>

</fileTxt>

</fileDscr>

In our example, we do not have a URI attribute normally used to provide a file location, but it is standard practice to use this field, and the program doing the transformation could presumably gather this information (here, it is missing because the catalog application has its own way of providing the data on request from the landing page where this metadata is downloaded).

We are given identifying information regarding the data file, as well as a general description of it, which could be placed in CatalogDetails or elsewhere (this is beyond the scope of this use case – DDI Codebook provides a wealth of useful study-level metadata which would also be good to publish).

The <fileTxt> element also has fields describing the format of the data file and other details, which are often populated. This information could be very useful to have in the DDI-CDI description of the file.

In the DDI-CDI model, the WideDataSet – which is a logical construct – is shadowed by a PhysicalDataSet object. **[How do I include the URL of the file’s location? This seems to be missing, or buried too deep in CatalogDetails!]** As a “companion” class, there is the DataStore, which may be very useful for describing how the file can be located, navigated, and retrieved.

## Connecting the Pieces

The diagram below shows the basic approach taken in this use case to rendering the DDI-CDI:

A picture containing text, screenshot, diagram, line

Description automatically generated

This diagram is somewhat abbreviated, but it shows the basic navigational paths between the objects in a general sense. This is a somewhat complicated picture, but is it more complicated than it needs to be?

# How Bad Is It?

When we compare the classes needed to support our use case with the reference UKDA profile, we end up with a couple of substitutions (WideDataStructure for DataStructure, WideDataSet for DataSet) and a list of additional classes:

PhysicalSegment

PhysicalSegmentLayout

ValueMapping

ValueMappingPosition

PhysicalDataSet

LogicalRecord

DataStore

This increases the list of 17 classes to a list of 23. In terms of the increase, this seems very reasonable.

If we consider how verbose this way of describing data is, we must remember that the value of DDI-CDI in general terms lies in its ability to support connections between data at a granular level. Many of the “verbose” classes here are actually most useful in the context of a larger data store, where many individual data sets may exist, and their contents may be re-combined at a variable level. What happens when we start reusing LogicalRecords in different physical data encodings?

So, the basic perspective here is that the use case can be supported in a reasonable fashion, using a limited profile of DDI-CDI. While some manual work is required to indicate the roles played by the variables (the Components) and possibly to rectify some missing information in the metadata record (in this case, the location of the file such as a URL), this is fairly minimal.

What this exercise does show is how the fairly limited metadata captured in the DDI Codebook instance can be “exploded” by making the implicit metadata explicit. This provides a rich set of contact points for formally describing how the data might fit into an integration scenario, even beyond the simple ability to transform the structure on an as-needed basis (an example is the externalization of Concepts, which, although not very fully described, can still be the basis of useful automated functionality which is much more difficult when these are embedded in the implicit metadata record).

A few issues have been identified (see the **[boldface]** notes in the text above). These will need to be reported as issues and resolved by the DDI-CDI WG.