# Applying DDI-CDI in a CDIF Use Case: XAS

*July 17 2025 – Arofan Gregory, Deirdre Lungley, Steve Richard, Flavio Rizzolo*

## Background

This document is the result of discussions among various members of the DDI-CDI and CDIF communities in response to questions coming from the EOSC/WorldFAIR+ XAS-4-CDIF project. In this project, the use of CDIF to enable better FAIR exchange of data between applications and data sources within the XAS community is being explored. While these are several “standard” formats within this community, these tend to be heavily influenced and specific to particular applications, and the ability to easily exchange XAS data still does not exist. There is, however, a strong interest in being able to do so.

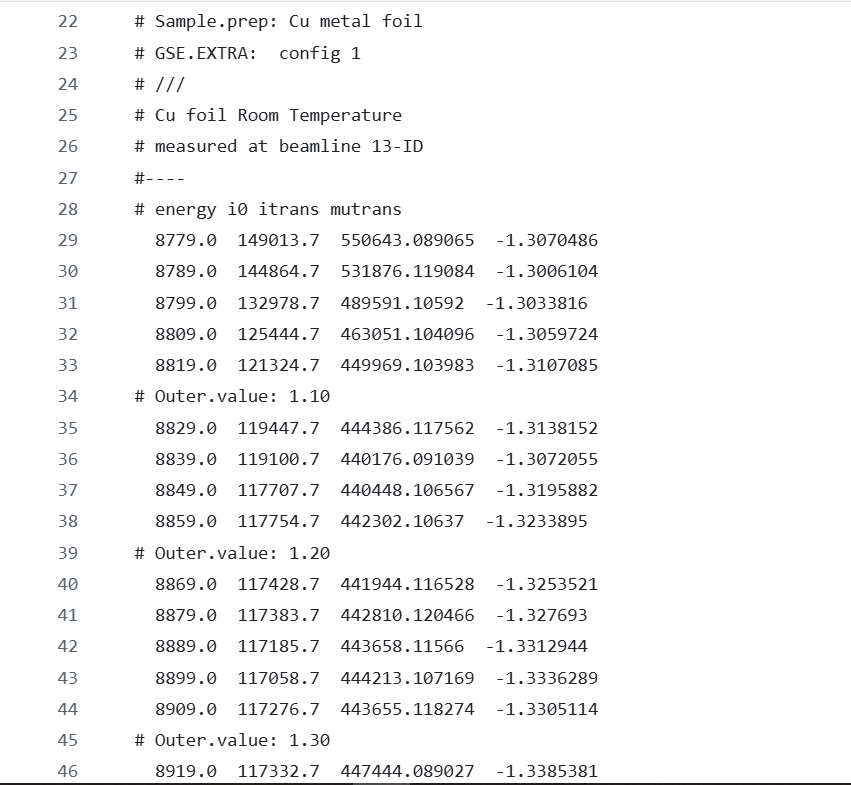
It has been expected that this project might result in new requirements being produced which would result in changes to the DDI-CDI Model or to CDIF, which uses only a subset of the features of DDI-CDI in its Data Description profile. CDIF also uses other standards, and the XAS use case might well highlight needed applications of or changes to the other profiles within CDIF, such as Discovery or the now-developing approach to provenance and data context.

## The “Data”

The example of XAS data provided by the project is shown below. Please note that terms such as “data” and “metadata” are easily confused, as they have different connotations and definitions across different domains. We will try to use the blanket term “information” as much as possible here, but it should be understood that “data” to one domain might be termed “metadata” in another. In the end, such differences are not important so long as the needed information is exchanged without loss or misinterpretation.

A screenshot of a computer

AI-generated content may be incorrect.



This data file (like many “XDI” files and similar measurements taken using sophisticated equipment) has a “header” section, providing configuration information regarding the lower section of the file, which contains observational data. (You can see the specification of the XDI format at: <https://github.com/XraySpectroscopy/XAS-Data-Interchange/blob/master/specification/spec.md> .) The “header” lines are preceded by a “#” symbol. Below that is a set of observational data in a traditional “wide” table, where each row is a set of measurements relating to the specific “unit of interest” (in the case the “outer value”, and indication of energy levels **[IS THIS CORRECT?]**) The observational data has a set of regular columns.

It should be noted that there are two different record structures (logical and physical) within this example: the header is essentially a set of key-value pairs, and the observational section is a set of repeated measurements. The information in these two area is related: the key-value pairs describe information needed to understand the observations recorded below. We can term this (generically) “configuration” information, although that term may not be fully descriptive of all the information expressed. What is important is that the header information is linked to the observational information, and that there is a dependency between them. Without the header information, we cannot correctly understand or process the information in the observational section.

DDI-CDI asks that we reduce the internal sub-structure into a regular set of records to the greatest extent possible. In part, this is done to allow for easier normalization of the information, since the intention is to integrate the data with other data, a process which often involves deconstructing it and re-assembling it into another structure. Here, we can see that the lower observational section is neatly divided into small sub-tables with four rows in each one, preceded by a specific value for the outer value. We could eliminate this sub-structure by repeating the outer value as the first value in each of the columns. As will be shown below, this type of change may be used to simplify the record structure when applying CDIF to this example.

## Requirements for Interchange

When we consider the purpose for which the information in the data file is exchanged, we see that it is meaningful only to those who understand the format. This style of file may well represent a domain standard, being based on existing, agreed domain ontologies, but it lacks sufficient information to be meaningful outside of the specific world of the operators of the machines which make XAS measurements (and similar).

While the way in which information is combined in the example file may be useful within the context of a domain exchange, it may be problematic for FAIR interchange. XAS measurements are used in a wide variety of other domains, and these domains do not necessarily support the same standards as practitioners within the XAS world do. The value of the XAS data lies in its integration with other domain data for these users. Thus, there is a need for the information we see in the example to be more clearly contextualized. Even if such non-XAS users do not fully understand the meaning of the structures and terms, they should at least be given a basic understanding of what they are looking at, and pointers to a formal expression of ontologies and other standards which they can then consult further to gain the needed understanding.

Given the wide range of potential users, both within the XAS “domain” and outside of it, it is not unreasonable to rely on transformations to specific standards which will be meaningful within the domain of any particular user. The diagram below shows the intention of CDIF to act as a “lingua franca” in such a cross-domain scenario:

A diagram of a map

AI-generated content may be incorrect.

Here, we see the comparison of an approach where all of the existing standards are mapped against other standards in other domains. CDIF helps to address this problem by massively reducing the number of mappings needed: in the first diagram, any given standard will be included in 18 mappings to and from each of the other standards, with a possible 90 combinations. In the second diagram, there are only 10 such combination in total.

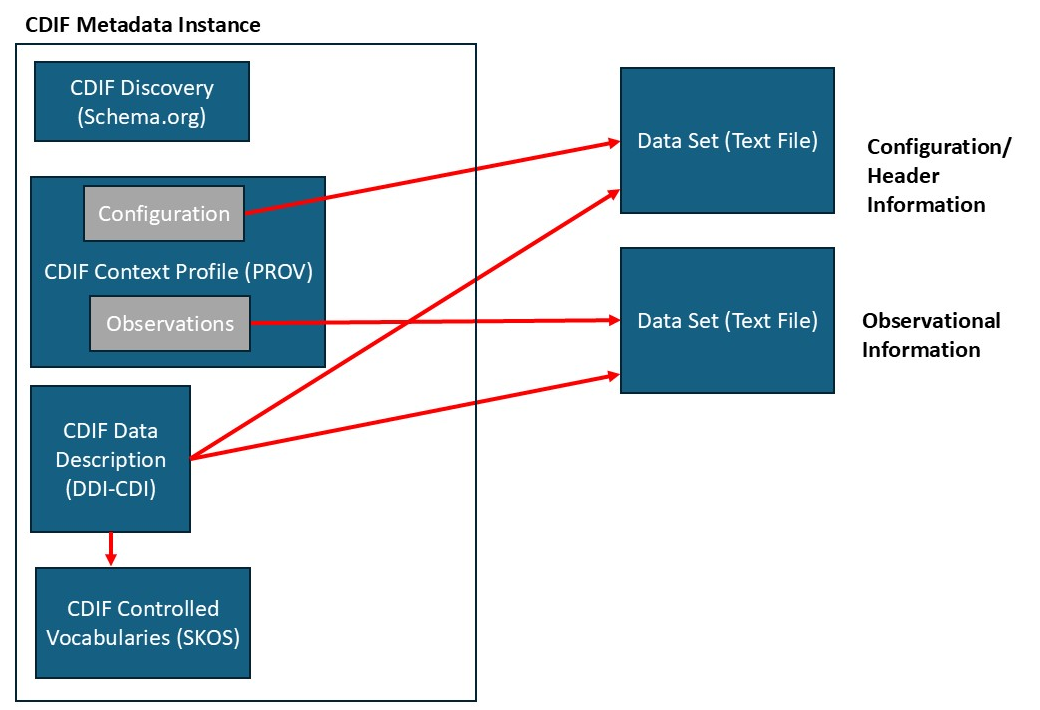
This dynamic can be applied both within and across domains in the case where there are multiple duplicative standards. XAS would seem to offer us such a case, with XDI being an example of one popular “standard” format, with lots of variations and other similar ones needing to be supported even within the community. **[SITE THE LANDSCAPE PAPER]**

A minimal subset of DDI-CDI is used in CDIF to describe data formats, the idea being that such a description provides a sufficient core for data interchange, but one which can be enhanced/extended as needed. This is designed to align with the capabilities of modern RDF-based technologies. The general approach tends to be one of disassembling structures into their atomic components, or enabling such a deconstruction, so that these same components can be reassembled as needed for integration with other data. This aligns well with the basic capabilities of RDF technology.

## The CDIF Structure

CDIF offers us a number of different profiles for different functions. In the XAS case, only a subset of these will be used. It would be possible to add other profiles, but that is beyond the scope of this document.

The diagram below shows the suggested parts of the resulting package intended for FAIR exchange and reuse:



This diagram provides an overview which we will use to navigate the overall package of information to be exchanged. Note that everything inside the “CDIF Metadata Instance” box is RDF expressed as JSON-LD, according to the CDIF profiles. Of these, the “CDOF Context Profile” has not yet been published, but is an area where there is currently much work ongoing. This example will also employ some parts of the DDI-CDI standard which are not currently recommended by CDIF. It is expected that changes will be made as needed to accommodate the XAS case and similar cases found in other domains.

It should further be noted that what is shown here is the result of a transformation applied to the XDI file shown above. It has been re-organized to agree with the “lingua franca” of the earlier diagram. As such, it has been pulled apart into a larger number of pieces, which are more completely self-describing, in order to better meet the needs of the broad range of users across domains. Each of these will be described in the following sections.

## Data Discovery

This part of the information set is designed to allow for discovery and cataloguing of the data resource, and may not be needed in every case (e.g., for internal exchange within an organization of in a direct-reporting scenario). In many cases, however, this basic metadata set is a basic requirement of FAIR metadata. Here, we have a subset of Schema.org fields which are recommended as core for the purposes of CDIF. The alternate here is to use the DCAT vocabulary. For which a corresponding subset is identified in the CDIF recommendations.

We will not go into detail here, as this topic has already been covered elsewhere, and examples provided (the CDIF recommendations themselves provide a sufficient example). **[NOTE: We may wish to flesh this out with a worked example once we have the Nectar code working to support CDIF Discovery.]**

## Describing the Data Structures and Necessary Definitional Metadata

There are three aspects to be considered when we describe this data for exchange: (1) the structure of the header information; (2) the structure of the observational information; and (3) the definitions of the fields and values found in these two sections. CDIF gives us a way of describing all three of these aspects. The first two are expressed in DDI-CDI as part of the CDIF Data Description. The third uses SKOS, a vocabulary for concept systems, which is integrated with the DDI-CDI descriptions but is also heavily used in its own right within the FAIR community. We will address each of these separately.

It should be noted that while DDI-CDI is capable of describing data files with multiple internal record structures, this tends to become very complex, and the fields needed to do this are not currently part of the CDIF Data Description profile. It has been decided in the interests of simplicity, and in the spirit of “deconstruction-for-reuse” which informs CDIF, to separate these two parts of the file, each into its own simpler file, with a single record structure.

The basic overall pattern we see in DDI-CDI for describing data is shown below. DataSet or DataStore are basically interchangeable for our purposes, the main difference being that a DataStore is always a queryable data source, and so could describe either a file storage or a service such as a database. For our purposes, the distinction is not meaningful, and we will use subclasses of both (see below).

A diagram of data set or service

AI-generated content may be incorrect.

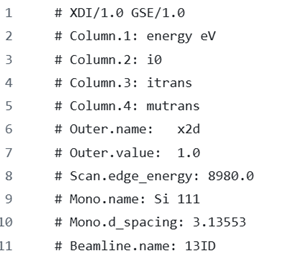
We will be describing the structure of our data, and will be describing the logical contents of the different data. In DDI-CDI, the variables which make up the data are a key player in the recomposability of data, and so we will be describing these for all of our data as well.

### Describing the Header/Configuration Information

There are two approaches to describing the configuration portion of the example file. The first is to use the “Key Value” features of DDI-CDI, as these seem om the surface to be best-suited to describing the kind of information we are dealing with. This portion of DDI-CDI is not currently in the CDIF Data Description profile, but it could be added.

The alternative is to describe the information as a “wide” data table with only two columns. This is an approach which is perhaps less intuitive but may prove to be somewhat simpler, and it is the one which we will use to describe the observational data. This approach is currently supported by the CDIF data Description Profile. We will consider both approaches here.

The diagram below shows the classes involved in describing the Key-Value data structure in DDI-CDI. We essentially have two columns in our example: the first one gives the name of an element in the configuration (e.g., “Column.1”, “Outer.Name”, “Beamline.Name”, etc.) which is separated from second column by a colon. Following the colon, we have the values for each element.

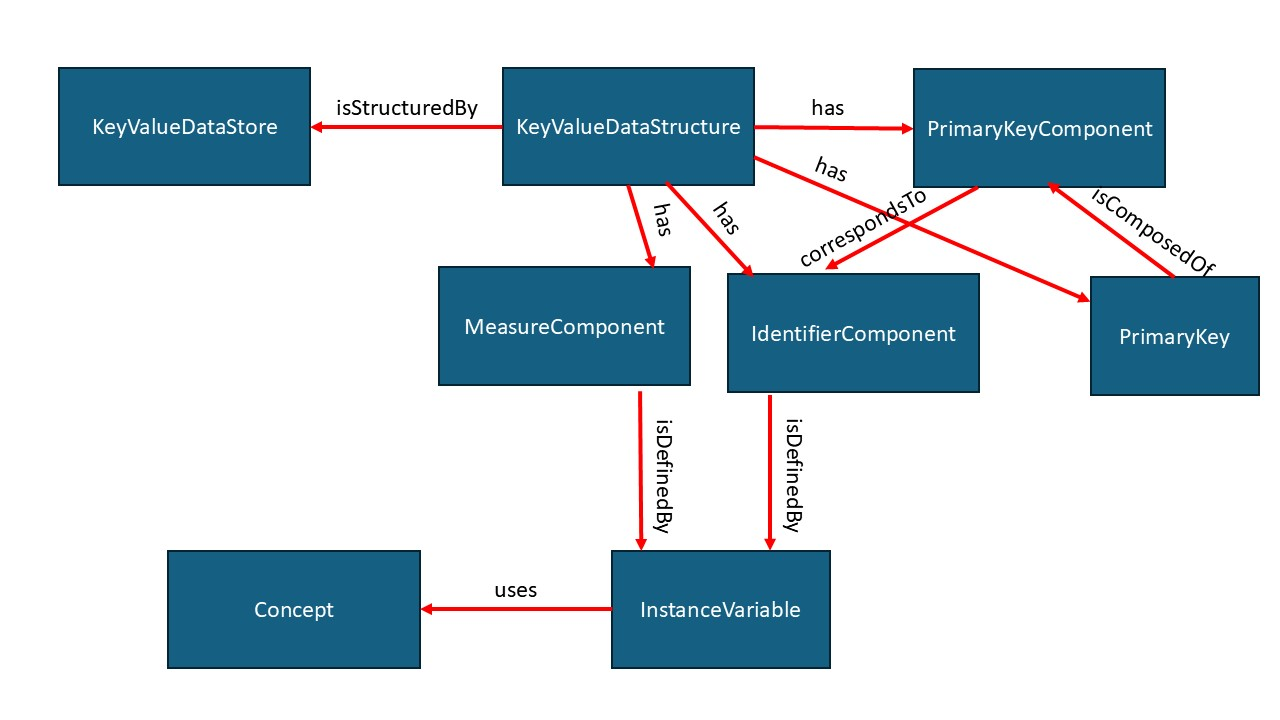


Note that the first line is meta-information about the XDI file which can be probably be discarded for the purposes of interchange. The element “Column.1, “Column.2”, “Column.3” and “Column.4” are structural information which is redundant, so these could also potentially be discarded (they could be generated from the structural description of the observational information as will be seen below). It may be convenient to persist them, however, to make transformations easier, and to retain the connection between the two different data structures.

The two columns in our structure – the elements and the element values – will be described as InstanceVariables in DDI-CDI, in this case with the names “Element” and “Element Value”. They could take their definitions from a Concept declared in a SKOS Concept Scheme (see below) although this is not a hard requirement for these particular variables. The representation of the Element variable would, however, need to come from an agreed formally defined list, expressed as a SKOS ConceptScheme (see below for how this should be done).

There are two possible ways to define the roles of the InstanceVariables we have identified. In one approach, the names of the elements are seen as Identifiers, giving us a way of pointing to the specific field for which a value is being assigned. The other approach is to see the list of configuration fields as possible variables in the Observational data (the lower portion of the file). In this case, there is a special construct in DDI-CDI which allows us to capture all the potential relationships, using what are termed “variable descriptors” and “value descriptors”.

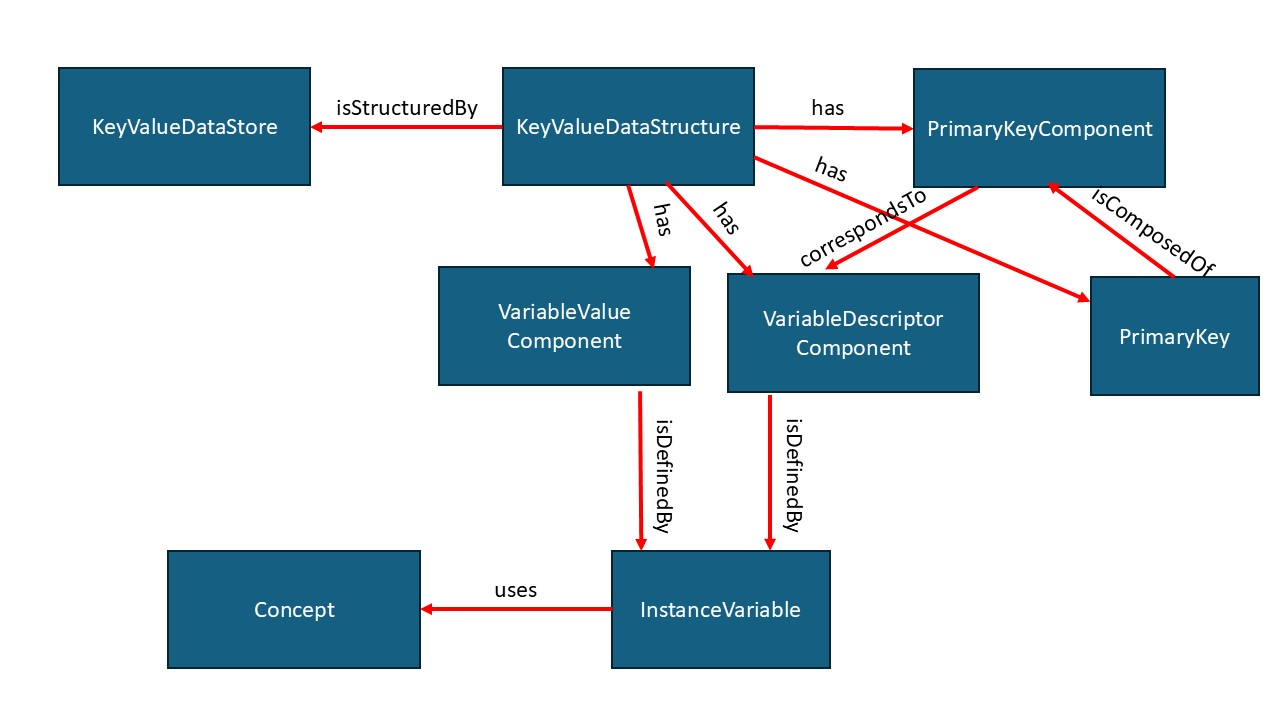
Both of these approaches are Key-Value structures in DDI-CDI: what changes are the roles assigned to the variables. This assignment of roles if done using what are termed “data structure components”. A component is an object which characterizes the role played by the variable within the context of a particular data structure. (Since variables can perform different roles in different structures when integrated/reused, it is important to separate the variable role from the variable description itself.)

In oyr first approach, where the “Element” column is described as an identifier, we will have a diagram which looks like this:

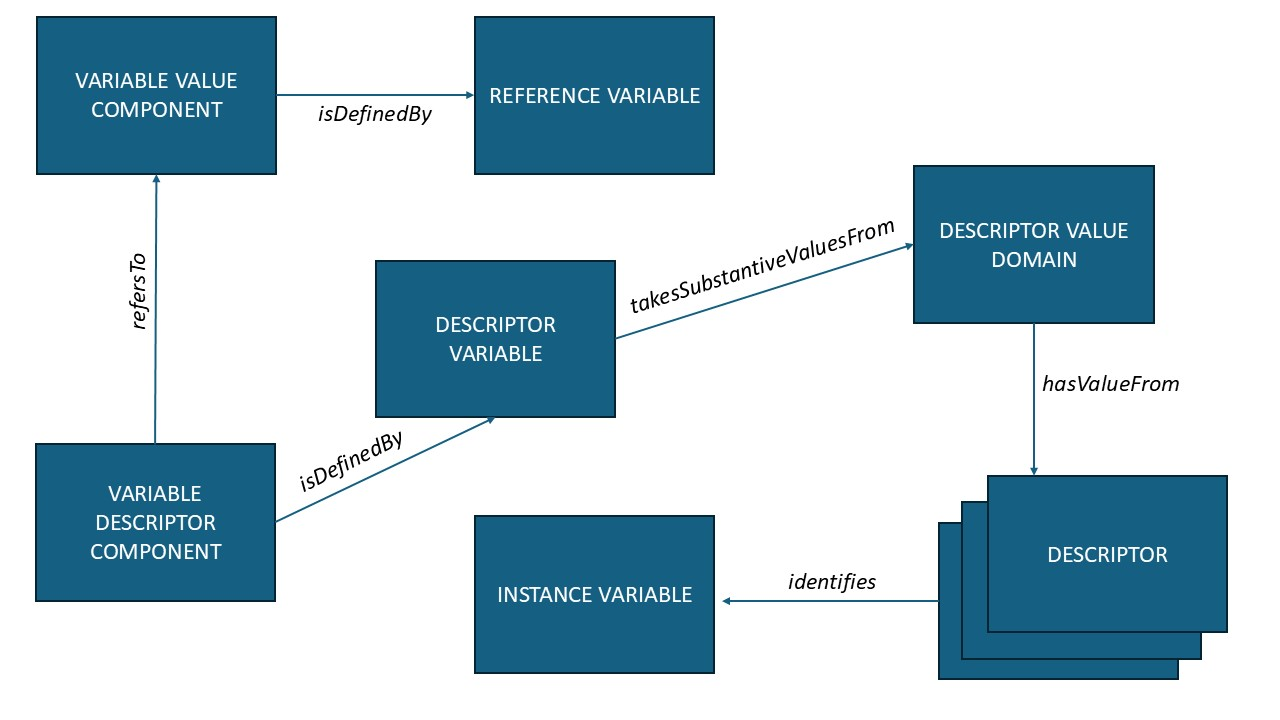
Here, the “Element” variable defines a component (that is, takes the role of) an IdentifierComponent, and the “Element Value” variable takes the role of (that is, defines) a MeasureComponent. The nice thing about this approach is that it is simple. The downside is that if different elements have different data types – that is, they have constrained values which are specific to the particular element being populated – then we do not have a description of that constraint.

The alternate approach treats the “Element” and “Element Value” variables as what are termed “presentational” variables in CDIF. Thet is, they are only declared for the purposes of structuring a table in reference to a larger set of logical variables. In this approach, each Element would be described as an InstanceVariable (technically a RepresentedVariable in DDI-CDI, but in CDIF these functions are collapsed into a single InstanceVariable object) and those Instancevariables would have their representations fully described.

The roles assihned to the “presentational” variables would be a VariableDescriptorComponent to define the “Element” variable, and a VariableValueComponent for the “Element Value” variable. This is shown in the diagram below:



These components have a special relationship to the logical variables which have been declared for each Element in our data set: the value of a VariableDescriptorComponent is a code which is associated with one of the logical variables; the VariableValueComponent holds a value for the associated logical variable. The diagram below shows how this works (in this case, for a Long Data Structure but it is the same for Key0-Value Data Structures):



The ReferenceVariable here is the specialization of an Instancevariable to hold the values of a range of other Instancevariables (our logical variables) – in this case, it is the Element Value variable.

Regardless of which approach is used, other classes will remain fairly consistent. The KeyValueDataStore represents the file which holds the data, and would have a property with the URL of the text file. It would have an object representing the key-value structure, which would have a PrimaryKeyComponent object, comprising the PrimaryKey of the structure. The PrimaryKeyComponent in this case is very simple – it is made up of only the IdentifierComponent or the VariableDescriptorComponent, because these have unique values within the data set. This is reflected by the fact that the PrimaryKeyComponent corresponds to the IdentifierComponent or the VariableDescriptorComponent, which itself describes the role of the Element InstanceVariable in this structure. (Note that this is probably overkill for the very basic requirements of this use case, but is designed to handle multi-part keys composed of values from several variables.)

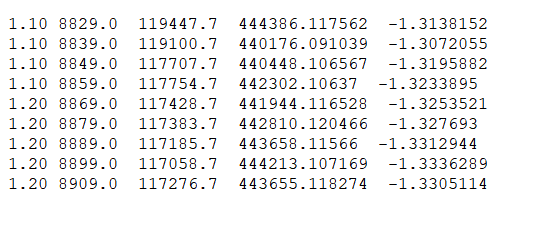
This covers the logical description of the key-value structure, which can then be bound into the physical representation in the text file. This is the same for all of our data, so we will describe this once at the end of this section.

It is felt that the use of the presentational variables (using the VariableDescriptorComponent and VariableValueComponent classes) is preferable, because it retains the typing information for the data in the Element Value column.

To describe the information as a WideDataSet, we would use the approach described below for the observational information, but we would have only two columns in our table instead of five. Again, our primary key would be made up of our IdentifierComponent, the “Element” InstanceVariable. This may arguably be a little bit simpler, but is felt that the use of the Key-Value structure is more intuitive and thus to be preferred.

### Describing the Observational Information

The lower portion of the file – the observational section – would have the “# Outer.value: 1.10” (and similar lines) normalized by repeating this value for each of the four following lines. This, we would end up with a data set which looked like this (for the “Outer.value: 1.10” and “Outer.value: 1.20” sections):



Each row can be uniquely identified by the values of the first two columns. This will become our Primary Key for each record (that is, each row). The five InstanceVariables we will declare will be: Outer.value, Energy, iO, itrans, and mutrans. Each will be represented by a suitable numeric type (see below). Outer.value and Energy are associated with IdentifierComponents, while iO, itrans, and mutrans are associated with MeasureComponents. Both our MeasureComponents correspond to PrimaryKeyComponents, which together make up our PrimaryKey. As before – although not shown here – the InstanceVariable would also have a “uses” association with a Concept, coming from a SKOS ConceptScheme. For these variables, that is not optional, but would be required, as these concepts may link elements of the key-value structure to the column measurements here. This is indicated when the Concept used by an Instancevariable is the same as the Concept used as the Element in a key-value pair within the configuration information.

A diagram of components of a component

AI-generated content may be incorrect.

### Describing the Needed Definitional Metadata

This section looks at two aspects of the description of our data files which are technically implemented in the same way but perform different functions. Both use SKOS Concepts and ConceptSchemes as an implementation, but one is concerned with the representation of the values of variables, and the other is concerned with the use if Concepts to define the variables themselves. To make this distinction clear, a simple example we often see is Marital Status: this can be defined as a variables using a concept “Marital Status” which might be defined as “Whether an individual is or has been married or not, according to the laws of the country in which they reside.” The valid values might be: “Married”, “Divorced,” and “Never Married.” Each of these values could also be defined using a Concept with the appropriate definition. In this example, Marital Status is the Concept which would be used to define a variable (a column in our data set) which could hold a code associated with any one of our valid values, each of which is also associated with a Concept. In DDI-CDI, there would be a discrete SKOS ConceptScheme which held only the Concepts used to represent that variable. Concepts used only to define variables could appear in a more general ConceptScheme, or could even appear in multiple SKOS ConceptSchemes and have “sameAs” relationships between the instances of the same Concept.

The diagram below shows how InstanceVariables have their representations described in the CDIF profile of DDI-CDI:

A diagram of a network

AI-generated content may be incorrect.

Substantive values are the values which are intended to populate the variable. Sentinel values (if any) are flags indicating a missing value or other condition. In our example, we may not need any of these. Our SubstantiveValueDomain could either point to a numeric type expressed as ab XSD data type, or would reference a SKOS ConceptScheme which contained only the valid categorical values for the variable it is used to represent. Additionally, we would associate a Concept from a more general scheme with each variable to provide its definition, with a “uses” association.

The ConceptScheme for this second application would be the SKOS representation of a more general ontology describing all of the possible elements to be found in XAS data and encoded in the configuration of XDI files. Ideally, this would be based on an agreed ontology or classification within the domain, as the use of specific agreed values here is important for the purposes of interoperability between systems.

It is significant that in some cases, the Concept used as a value of the Element variable in our configuration data set will be the same Concept used as an InstanceVariable in our observational data set. This connection is very important, which is why the correct use of SKOS here will be critical. Without these connections, the receiving application may not know what to do with the configuration information vis-à-vis the data in an automated fashion.

### Physical Representation of the Data

In DDI-CDI, the description of the logical contents of the data are separated from the description of their physical storage. The diagram below shows how the physical description is done I DDI-CDI, and how it connects to the logical structure of the data.

Associated with each of our data structures will be a a LogicalRecord, which is an object that acts to group the set of Instancevariables appearing in each record. For our key-value data, this would be a two-variable set including the Element and the Element Value variables. For our observational data, this will be a five-variable set including the Outer.value, Energy, iO, itrans, and mutrans variables. Note that these sets are not ordered, as their ordering is not important in understanding the logical content of each record.

A diagram of a data flow

AI-generated content may be incorrect.

Each record has a PhysicalSegmentLayout which uses a ValueMapping object to associate one of the InstanceVariables with its position in the physical record, using a ValueMappingPosition object. Properties such as the delimiter or the position of the variable in a fixed-width structure are captured on the PhysicalSegementLayout object and the ValueMapping object as literal values in the RDF. (See the appendix in the DDI-CDI Guidelines for the set of properties included in the core CDIF profile for these classes.)

In our example, the header information data set is delimited using a colon as the delimiter. The observational data set is fixed-width. In each case, the data structure would be of the appropriate type: KeyValueDataStructure for the header information, and WideDataStructure for the observational data. These would otherwise be treated in exactly the same fashion for describing how their InstanceVariables are mapped into a PhysicalSegmentLayout.

## Describing the Context of the Data

We are still faced with the challenge of conveying to the user of our data the dependency between the two data sets we have described. These need to be placed in a suitable relationship to one another so that a receiving application can process them appropriately, and so that the user can understand what each data set contains. As we mentioned earlier, within a domain, a user might be familiar with the intended use of the different sections of an XDI file, but outside of the XAS domain this may not be the case.

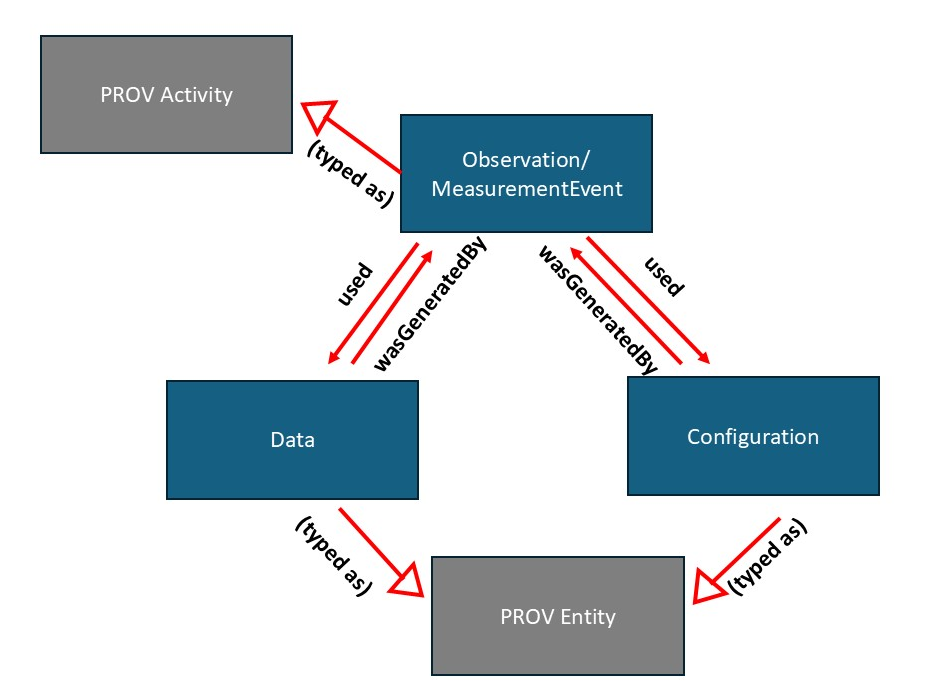
Within the CDIF Working Group there has been discussion of how to produce a meaningful description of the context of research data across domains. One idea which has gained some traction is the use of PROV to describe an agreed set of generic classes which can be applied as descriptors to all research across domains. These could then be used to qualify more domain-specific provenance descriptions, or could be used on their own for strictly cross-domain contextualization. This work is still ongoing, but is based on a bottom-up analysis of research in different domains over the last two years, primarily at the Dagstuhl workshops **[CITE THE COBEXT DRAFT HERE].**

Three of the generic classes which have been identified are especially relevant to what we are describing here with the XAS example, although more could be applied. These are the Observation/Measurement Event, the Configuration class, and the Data class. Each of these is a refinement of a base class in PROV. Here are the base classes:

A diagram of a diagram

Description automatically generated

In our model, the Observation/Measurement Event is a PROV Activity, and the Data and Configuration are PROV Entities. The diagram below shows how this would appear in the proposed CDIF Context profile:



Note that RDF is capable of expressing both subclass/superclass relationships, and straight “type” relationships, and this could be done in either fashion, The correct approach has not yet been identified, but either approach could express the needed association. What is important is that the “Data” object in our example would reference the observational data set, and the Configuration object would reference the header information data set. This additional context information would indicate that there is a connection between these two PROV entities. It might also be desirable to introduce am RDF property which would explicitly describe this dependency.

For clarity’s sake, some proposed definitions of these extensions go PROV are provided here. The full set of extensions is still being drafted and defined – these definitions are just one possibility, for the sake of illustration amd c;arity:

|  |  |
| --- | --- |
| **PROV Extension** | **Definition/Description** |
| Data | A set of observations or measurements which describe the phenomenon or object (including samples) being studied. These are primary inputs to analysis. |
| Configuration | Settings employed in the observation, measurement, analysis, processing, or calculation of data. These may refer to instrument configurations, constant values employed by specific methodologies, or other supporting information. |
| Observation/Measurement Event | This is an event which produces data regarding a phenomenon or object of study. This includes samples. Data is a necessary output of such an event. |