JOURNAL OF CHEMICAL INFORMATION AND MODELING

Article

pubs.acs.org/jcim

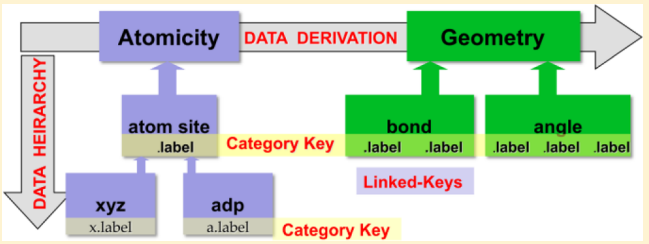
DDLm: A New Dictionary Definition Language

Nick Spadaccini* and Sydney R. Hall*

School of Chemistry and Biochemistry, The University of Western Australia, Nedlands 6009, Australia

Supporting Information

ABSTRACT: A previous paper [Spadaccini and Hall *J. Chem. Inf. Model.* doi:10.1021/ci300074v] details extensions to the STAR File [Hall *J. Chem. Inf. Comput. Sci.* **1991**, *31*, 326−333] syntax that will improve the exchange and archiving of electronic data. This paper describes a dictionary definition language (DDLm) for defining STAR File data items in a domain dictionary. A dictionary that defines the ontology and vocabulary of a discipline is built with DDLm, which is itself implemented in STAR, and is extensible and machine parsable. The DDLm is semantically rich and highly specific; provides strong data typing, data enumerations, and ranges; enables relationship keys between data items; and uses imbedded methods written in dREL [Spadaccini et al. *J. Chem. Inf. Model.* doi:10.1021/ci300076w] for data validation and evaluation and for refining data definitions. It promotes the modular definition of the discipline ontology and reuse through the ability to import definitions from other local and remote dictionaries, thus encouraging the sharing of data dictionaries within and across domains.



INTRODUCTION

Precise quantitative definitions for structural chemistry data[4] as a domain-specific dictionary in STAR file format[1,2] were first suggested by Tony Cook.[5] This gave rise to the development of a dictionary definition language (DDL) used primarily for providing metadata within the crystallographic information framework (CIF),[6] which uses a subset of the STAR file syntax. The expression of data as a CIF and definition of its data items in a dictionary written in DDL, has been adopted by the International Union of Crystallography as the preferred format for manuscript and data submission to their publications,[7] and is used for data depositions to the crystallographic databases. The first version of the DDL, referred to as DDL1,[5,8] was limited in its scope but sufficiently expressive to enable the development of a dictionary for chemical crystallography. For macromolecular crystallographic data, the dictionary language was extended to more directly support a relational model and to provide stronger typing. This led to the development of a second DDL, known as DDL2.[9,10] While both DDL versions remain the basis for a number of crystallographic dictionaries[4] and provide quantitative information for CIF validation tools,[11] the meta-data of any scientific domain can be expressed in either one of these DDLs.

A 2009 editorial in Nature entitled *Data's Shameful Neglect*[12] highlighted the lack of technical, institutional, and cultural frameworks for supporting open data access and archiving. The increasing need of scientific communities for precise data definition to support comprehensive and extensible tools for data exchange, archiving, and validation underpins new approaches to data definition and has been the major motivation in the development of DDLm.

A number of key objectives were set for this development. A logical one was to raise the level of definition precision provided by existing STAR/CIF DDLs as well as dictionary languages used outside the crystallographic domain. A conspicuous requirement was a more comprehensive set of *type* attributes. It was clear at the outset of this development, however, that the goals of the DDLm development needed to go well beyond the simple expansion of existing attributes and provide for more automatic approaches to process and validate data. We will show that DDLm achieves this and opens up new paradigms for data management and definition.

A pivotal goal for the DDLm development is the ability to treat a text dictionary as a machine-executable form that can be populated with a particular instance of data and facilitate the automatic evaluation of derivable data values directly from other data and definition information embedded in a DDLm dictionary. This is achieved by the inclusion of methods attributes that specify data relationships in a canonical language. A description of the methods attributes is given below in part 5 of the section DDLm Features.

Another core objective of the DDLm design was to allow definitions to be shared across local and remote dictionaries, obviating the need for redundant definitions in related domains. In the management of global data, it is of increasing importance that the definitions of data across different dictionaries, domains, and countries are shared to ensure that the definition of equivalent data items is coordinated and seamless. DDLm facilitates definition sharing with the provision of the *import* attribute. This is described below in part 4 of section DDLm Features.

Received: February 7, 2012
Published: June 22, 2012

THE EXPRESSION OF DATA

Since the introduction of the STAR File[2] syntax in 1991, significant advances have been made to information management technologies, particularly through the use of the Internet.

It is now especially important to record the exact quantitative relationships between *measured or observed data* and *derived data*, and to place this information into universally accessible dictionaries that can allow a much higher level of data validation in any domain. Moreover, an ability to store interdependencies and relationships between data and allow their quantitative evaluation, will promote improved approaches to archiving by enabling access to derivative data generated solely from a process that can be executed via a dictionary.

Figure 1 contains a typical data record for a crystallographic study; similar data records exist in all scientific domains. In

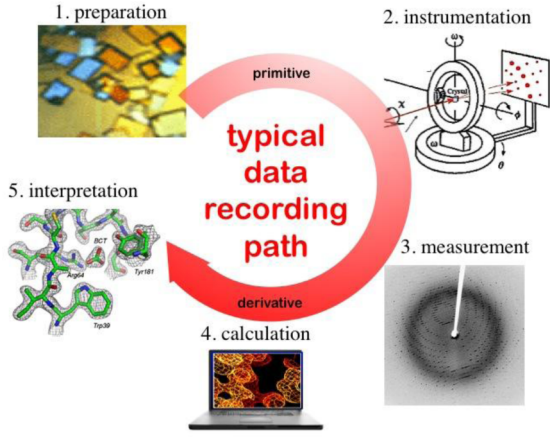


Figure 1. Typical data-recording pathway for a scientific experiment.

crystal structure analyses, the basic data accumulation steps are preparation of the crystals for the material under study; alignment and characterization of the crystal; measurement of the diffraction intensities; conversion of the diffraction image into electron density, and finally the interpretation of this density as a molecular image. Most scientific experiments follow a similar sequence of preparation, instrumentation, measurement, calculation, and interpretation.

In the record of a study, each step generates and introduces measured, observed, and assigned (i.e., *primitive*) data on one hand, or derived and transformed (i.e., *nonprimitive*) data on the other. Derivative data can be directly evaluated from other data, and assuming *all* relevant information for a study is recorded in a *quantifiable way*, many data items will be of this type. Bookkeeping data is used only to classify and organize data into lists and containers and have no bearing on the interpretation of the archived data. The rest may be considered as raw or primitive data. Primitive data are measurements and observations, such as instrument settings, or decisions made in nonlinear processes, etc. That is, information that usually cannot be deduced without repeating the entire experiment. As a study progresses, the data recorded tends to become increasingly derived. In some domains, especially in those where data archiving and definition practices are underdeveloped, the distinction between "primary" and "primitive" is often not made; primary information being that data which a discipline may prefer to be archived even though it is derivable; as opposed to primitive data which cannot be derived and comes directly from experimental observation (and if not recorded is lost forever).

The distinction between primitive and nonprimitive data has significant implications for its precise definition and for archival processes. The inter-relationships of derivative data can be included in its definition and expressed using a methods scripting approach. Relational expressions define, and can be used to automatically evaluate, derived data values in terms of other items in the data record. From a definition viewpoint, the methods approach serves to both simplify and quantify the characterization of derivative items and the component data on which a derivative item depends. From a data archival and validation viewpoint, a data definition can now provide executable methods for cross-verifying an archived value. Details are given in a following paper[3] where we show that these relationships can be expressed in a canonical form.

For database archiving, the presence of derivative items in the experimental record has both advantages and disadvantages. In a well-defined domain, derivative data is in a sense redundant since it can be recalculated; on the other hand the original derivative values do provide a means to validate related data. This is not the case for primitive data, though most databases do have ad hoc tests for checking the internal consistency of measurements. When data interdependencies are precisely recorded in a dictionary, the option to not archive the derivative data exists. In the future it may be considered more reliable to service requests for derived data values by recalculating them, and the dictionary methods makes this a possibility. Moreover, if archived primitive data were time-stamped with the dictionary version used for its initial validation, derived data could be generated according to the methodologies current at the time of recording, or for any later period as techniques and methodologies evolve. Improved methods for analyzing and interpreting data are always emerging in science, and this can mean that past derivations, and the interpretations drawn from these, are no longer valid. Retaining the time-stamped dictionaries serves as a chronological record of the evolution of methods used in the domain. The role therefore of the archived primitive data, assuming of course that they are also valid by current standards, is of absolute importance to the flexibility and precision of future data farming and mining.

This is a long-term goal and, realistically, "smart" dictionaries are unlikely to be used in this way for database searches at a time when access speed, user convenience, and storage costs are the major concern. Computing advances will make this goal increasingly achievable. As we shall see, the principal advantages offered by dictionary languages such as DDLm are a capacity to accurately represent complex interdependencies and relationships between data, and the functional dependencies between data and their derivation. These are essential to achieving the level of data precision increasingly needed in science.

THE RATIONALE OF DDLM

Data dictionaries written in DDLm contain precise, machine-parsable definitions of data and therein provide a stable knowledgebase for the reliable exchange and validation of data. Moreover, the existence of comprehensive and precise definitions within a data dictionary serves to provide a globally consistent view of specific data items, within and across domains. The automatic and seamless exchange and validation of data electronically depends implicitly on the level of usable semantic information available, and open-access dictionaries are

much better repositories for this information than the customized and closed software approaches currently in use.

We adhere to the rationale that the ability of a dictionary definition language to impose precise data typing, and to express the formulaic and symbolic relationships between data, enables dictionaries to perform a greater role in data validation and evaluation. The significant advantage of using a semantically rich dictionary for this purpose, particularly if adopted internationally by an official scientific body, is the assurance that standard well-understood definitions are uniformly adhered to across the domain. Scientific data, by its very nature, are in a state of continual change, and this necessitates efficient management processes that are in step with the evolution and creation of new data definitions. Semantically rich and readily updatable data dictionaries allow this to happen.

The DDLm approach offers a comprehensive set of definition attributes and provides a hierarchical, concise, and implicit mechanism for building electronic dictionaries for current and future data needs. In particular, definition attributes for including canonical expressions based on the symbolic language dREL[3] changes the way that derived data items may be archived.

THE DDLM DATA MODEL

The DDLm dictionary structure described below provides a simple mechanism to define data as either a *hierarchical* or a *class-object* model. The same structure is employed to represent either model interpretation.

DDLm requires data items to be grouped into categories (i.e., classes). There are two types of categories; those items with single values are in *Set* categories, and those with multiple values are grouped into *Loop* categories that are recorded as a looped list. A looped list is equivalent to a relational table. Categories may be within categories allowing for a hierarchical interpretation, or equally as a class and subclass model.

Categories of the Loop class must have a key, and any child categories of Loop class may automatically be *joined* on the key values. Joins within and across the category hierarchies are allowed by the presence of a specific attribute in the category definition. These can be seen equivalently as message passing from one category class to another.

The organization of a dictionary written in DDLm is straightforward. The definition of each data item is wholly contained within a *save frame*. All item definitions of the same category are contained within the definition save frame of the category. A child category is wholly contained in the save frame that defines its parent category and so on. These definitions are stored within a single *data block* in a dictionary file.

This arrangement allows for a variety of data models to be represented. The detail of the structure and the syntax for definitions is given below.

THE DDLM DICTIONARY STRUCTURE

The terminology and syntax of the metadata needed to define STAR data items are relatively straightforward and have been embodied in our *dictionary definition language* (DDLm). This language is a prescribed set of *attributes* in which each attribute serves as a taxonomic identifier of a particular characteristic of data. The definition of each data item is achieved by assigning values to appropriate attributes that are needed to describe its precise nature, and its relationship to other items. A collection of data definitions is referred to as a *domain* (or *discipline*) dictionary. A collection of data items (i.e., tags with ascribed values) is a *data file* or an *instance document*, e.g. a CIF.[4] These terms are used throughout this paper.

The DDLm construction template for a dictionary is as follows. Each data item, each class of items (referred to as a *category*), and each dictionary are defined using a sequence of appropriate attributes. A total of 60 DDLm attributes, grouped into 12 major categories (Figure 2), are used in definitions. A description of all attributes defined in the *DDLm Reference Dictionary*, is given in the Supporting Information.

Figure 2. Major attribute groupings in the DDLm.

There is an important distinction between the DDLm Reference Dictionary in which the DDLm attributes are defined and the domain dictionaries where the discipline-specific data definitions, written in DDLm, reside. The DDLm Reference Dictionary defines the DDLm language attributes and cannot be altered; definitions in a domain dictionary, constructed with DDLm attributes, are the responsibility of the domain. The majority of attributes in the DDLm repertoire are intended for the definition of domain specific data items, their categories, and dictionaries. A few, however, are reserved specifically for use in the DDLm Reference Dictionary.

Not every attribute is meaningful or needed in every definition; some will be mandatory but the use of most will depend on the nature of the item or category being defined. The validity of attributes for item, category, and dictionary definition is specified in the DDLm Reference Dictionary and given in the Supporting Information.

The construction and organization of a dictionary written in DDLm conforms to the extended Star File syntax.[1] As alluded to in the model description above, each data item definition is wholly contained within a save frame (in this instance referred to as a *definition frame*) and is expressed as a particular sequence of attributes that define that item. Item definitions are grouped into their data categories and each category has its own definition frame. A category definition is wholly contained within a save frame, within which all of its related item and child category definition frames reside. Parent categories contain the definition frames of child categories as *nested* frames. The principal category in a dictionary, known as the *Head* definition frame, contains all other category definitions in the file. In turn, the Head definition resides in a single data block, known as the *dictionary block*, along with dictionary-specific attributes that specify dictionary-only information. Additional dictionary-only audit information is also placed in this block.

We illustrate how dictionary definitions are constructed using DDLm attributes with some simple example definitions for data

items common to the crystallographic domain. To introduce these data, we show them, in Figures 3 and 4, as two trivial

```
loop_
_atom_site.label
_atom_site.fract_x
_atom_site.fract_y
_atom_site.fract_z
  o1   .5501(6)   .6371(6)   .1601(13)
  c1   .4170(8)   .6931(9)   .4965(18)
  c2   .3144(8)   .6702(9)   .6420(19)
  c3   .2789(9)   .7494(10)   .838(2)
```

Figure 3. List of atomic coordinates in a crystal unit cell.

```
loop _
_atom_site.label
_atom_site.fract_xyx
  o1   [.5501(6), .6371(6), .1601(13)]
  c1   [.4170(8), .6931(9), .4965(18)]
  c2   [.3144(8), .6702(9), .6420(19)]
  c3   [.2789(9), .7494(10), .838(2)]
```

Figure 4. Same data as in Figure 3, but with coordinates expressed as a vector.

instance documents expressed in CIF format.[4] In each instance there is a loop list containing data describing the positions of atomic sites in a crystal structure and belonging to the ATOM_SITE category. The atom_site.label item is used to uniquely identify an atom in the loop and hence is the *key* to the loop packet containing data relevant to this site—in this case, the three fractional coordinates for the atom in the crystal unit cell.

Since DDLm allows for the definition of compound items, such as the *vector* of the three coordinate items in Figure 3, this data may be expressed more precisely as in Figure 4.

Using these example data instances, we now describe the different definition levels present in a data dictionary written in DDLm: the definitions of individual items, the definitions of categories of items, and the definitions of the dictionary itself.

1. Item Definition. Thirty-five DDLm attributes are available for defining individual data items. For example the single item atom_site.fract_x in Figure 3 would typically be defined as shown in Figure 5.

In this definition all attributes, except definition.update and description.text, are mandatory because they are essential to the unique identification and characterization of the data item. The value of this item is type-specified as a Derived Single Real Measurand. In this domain, although the atomic coordinates are derived, they are often considered primary and consequently no evaluation method is included in this definition. Atomic coordinates are deduced from one or more complex processes, sometimes iterative and nonlinear, that determine the electron density from the diffraction images. Such processes are difficult express in a methods script, and their exclusion is the sort of definitional pragmatism accepted by a domain. It is for each domain to determine where primitive or primary data sits in the data-recording pathway and from where other data can then be derived.

```
save_atom_site.fract_x
  _definition.id          '_atom_site.fract_x'
  _definition.update       2010-06-29
  _description.text
;
   Atom site coordinate x as fraction of the cell
   length value.
;
  _name.category_id          site
  _name.object_id            fract_x
  _type.purpose              Measurand
  _type.source               Derived
  _type.container            Single
  _type.contents             Real
save
```

Figure 5. Domain dictionary definition frame for the atom_site.fract_x data item, written in DDLm.

In the data instance shown in Figure 4, the item atom_site.fract_xyz is used. This item is defined in Figure 6. This definition states that item atom_site.fract_xyz is a three element vector (i.e., container type *Matrix*; dimension [3]) that has been assembled from the three individual coordinate measurands. This is specified explicitly using the methods attributes to stipulate the dREL constructor of a relationship between the single position vector and its component coordinates. The component items must also be defined in the dictionary. An important implication of the methods scripts, which are discussed briefly in part 5 of the section DDLM FEATURES and in more detail elsewhere,[3] is that they facilitate the automatic construction of this vector measurand from the individual coordinate measurands if it is requested and not present in the instance data file (as is the case in Figure 3). Of course, if the coordinate items are also missing from the instance file, the request will not be satisfied.

```
save_atom_site.fract_xyz

  _definition.id          '_atom_site.fract_xyz'
  _definition.update       2010-12-14
  _description.text
;
   Vector of atom site coordinates projected onto the
   crystal unit cell as fractions of the cell lengths.
;
  _description.common      'AtomSiteFractXyz'
  _name.category_id         site
  _name.object_id           fract_xyz
  _type.purpose             Measurand
  _type.source              Assembled
  _type.container           Matrix
  _type.contents            Real
  _type.dimension           [3]
   loop_
    _method.purpose
    _method.expression
     Evaluation
;
  With a  as  atom_site

  atom_site.fract_xyz = [a.fract_x, a.fract_y, a.fract_z]
;
save_
```

Figure 6. Domain dictionary definition frame for the atom_site.fract_xyz data item, written in DDLm.

A comparison of definitions in Figures 5 and 6 illustrates the importance of richer data typing. In crystallography, the practice of treating the x, y, and z coordinates as three separate items in instance documents and in dictionaries exists only because previous syntax and definition approaches did not adequately describe vector items. A precise specification of

metadata opens the way for more powerful algorithmic approaches to manipulating data. This is the case within the DDLm methods attributes because the dREL scripting language is capable of handling any complex compound data items expressed in algebraic form.[3]

2. Category Definition. Fourteen DDLm attributes are available for defining a category of data items. The atomic site items in the instance data shown in Figures 3 and 4 each belong to the same category. A specific requirement of items within the same loop list is that they belong to the same category. In this case the category is ATOM_SITE, and its definition is shown in Figure 7.

```
save_ATOM_SITE
  _definition.id          ATOM_SITE
  _definition.scope       Category
  _definition.class       Loop
  _definition.update      2010-02-06
  _description.text
;
  The CATEGORY of data items used to describe atomic site
  information used in crystallographic structure studies.
;
  _description.common     'Atom Site List'
  _name.category_id       ATOM
  _name.object_id         SITE
  _category.key_id        '_atom_site.label'


save_atom_site.label
    <definition of _atom_site.label>
save_

    <all other definition frames for atom_site items>

save_  #---- close of ATOM_SITE category definition
```

Figure 7. Domain dictionary definition frame for the ATOM_SITE category, written in DDLm.

In the definition of the category ATOM_SITE, the attribute definition.class is set to the value Loop. This stipulates that ATOM_SITE is a loop category and that all the data items in this category will appear in the same loop list of a data instance. The definition in Figure 7 stipulates that the attribute category.key_id is atom_site.label; that is, the values of this item are the unique keys to packets in the loop for this category. The specification of a key item is mandatory for every category in the Loop class.

The values assigned to the attributes name.object_id and name.category_id and to that of definition.id, appear to be the same. However, as is discussed further in part 2 of section DDLM FEATURES, this need not be the case. The methods scripts utilize the hierarchical class relationships of data, and this is strictly adhered to for the values of name.object_id and name.category_id. The definition.id attribute specifies the data name used in instance documents, and this need not adhere to the class hierarchy (though in this instance, it does). In the definition of the ATOM_SITE category in Figure 7, the internal category name is SITE and it is a child member of the parent category ATOM. Although the parent information is implicit in the nesting level of categories, its explicit declaration with name.category_id provides a useful visual cue for each category definition.

3. Dictionary Definition. Eighteen DDLm attributes are available for defining a dictionary. A dictionary file is composed of a data block containing a sequence of *dictionary-specific* attributes (i.e., attributes in the DICTIONARY category) that determine its identity and characteristics. Figure 8 provides an example of the definition attributes used at the beginning of the CORE_STRUC dictionary.

```
data_CORE_STRUC

  _dictionary.title            CORE_STRUC
  _dictionary.class            Instance
  _dictionary.version          1.1.05
  _dictionary.date             2010-02-12
  _dictionary.uri              "www.iucr.org/cif/dic/core_struc.dic"
  _dictionary.ddl_conformance  3.7.09
  _dictionary.namespace        CoreStruc:
  _description.text
;
  This dictionary contains the definitions of data items
  as considered CORE to the description of STRUCTURE data.
;
```

Figure 8. Dictionary specific attributes for the domain dictionary CORE_STRUC, written in DDLm.

The dictionary block also contains a definition for the *Head* class category. The save frame of the Head category encompasses *all other definitions in the dictionary*. A typical definition is shown in Figure 9 for STRUCTURE data items in the crystallography domain.

```
save_STRUCTURE
  _definition.id          STRUCTURE
  _definition.scope       Category
  _definition.class       Head
  _definition.update      2010-02-12
  _description.text
;
  The DICTIONARY group encompassing the STRUCTURE data
  items defined and used with in the Crystallographic
  Information Framework (CIF).
;
  _name.category_id       CIF_CORE
  _name.object_id         STRUCTURE

<definition frames of all child categories of STRUCTURE>

save_
```

Figure 9. Domain dictionary definition frame for the *Head* category STRUCTURE, written in DDLm.

```
loop_
  _dictionary_audit.version
  _dictionary_audit.date
  _dictionary_audit.revision

  1.0.01   2010-12-12
;
  Initial version of the TEMPLATES dictionary created
  from the definitions used in CORE_3 dictionary version
  3.5.02
;
  1.0.02   2011-02-12
;
  Remove dictionary attributes from a save frame.
  Change category core_templates to template
;
```

Figure 10. Typical dictionary audit trail.

In summary, the dictionary block is composed of

- a leading data_*blockcode* statement, where the *blockcode* provides the dictionary identity;
- all the dictionary-specific attributes;
- the definition frame of the Head category, which encompasses all other category and item definitions in the dictionary;
- the audit trail of the dictionary revision history.

An example of a dictionary audit record, which is usually placed at the end of a dictionary file, is shown in Figure 10.

dx.doi.org/10.1021/ci300075z | J. Chem. Inf. Model. 2012, 52, 1907−1916

DDLM FEATURES

This section describes the main features of DDLm.

1. Strong Data Typing. DDLm provides an extensive set of data typing options, referred to as the *intrinsic* DDLm data types. These serve to precisely define the *structure* and *origin* of data items. Five separate attributes are used to define data type information; they are the **container**, **contents**, **purpose**, **source**, and **dimension** attributes.

The **container** attribute `type.container` is used to specify the *composition* or *compound structure* of a data item. The allowed container classes are *Single, Multiple, List, Matrix, Array, Table, Ref-table,* and *Implied*. The definition of this attribute in the DDLm Reference Dictionary is shown in Figure 11.

```
save_type.container
  _definition.id          '_type.container'
  _definition.update      2011-06-18
  _definition.class       Attribute
  _name.category_id       type
  _name.object_id         container
  _type.purpose           State
  _type.container         Single
  _type.contents          Code
   loop_
  _enumeration_set.state
  _enumeration_set.detail
   Single     'a single value'
   Multiple   'values related by , | & ! * : operators'
   List       'comma separated values bounded by []'
   Array      'comma separated values bounded by []'
   Matrix     'comma separated values bounded by []'
   Table      'comma separated key:value bounded by {}'
   Ref-table  'STAR reference table bounded by ${}$'
   Implied    'implied by associated value'

  _enumeration.default         Single
save_
```

Figure 11. DDLm dictionary definition frame for the `type.container` attribute, written in DDLm.

The **contents** attribute `type.contents` is used to specify the nature of the individual value or values, in terms of numerical and character descriptors that are enumerated as generic state codes in Figure 12. Note that additional facilities for applying type descriptors, specific to a particular domain and defined in a separate dictionary, are part of the DDLm approach and are discussed in part 6 below.

The **purpose** attribute `type.purpose` is used to specify the purpose or function of a data item. This information is important at several levels; it indicates, for example, if a numerical value is expected to have a standard uncertainty value or not; if the item value serves a special function, such as a loop key or a STAR File reference-value; or is used to initiate the importation of definitions from other sources. Identifying an item as a `Measurand` (with an associated standard uncertainty) or an exact `Quantity` (numeric or text) enables it to be handled appropriately within the method expressions and in some cases automatically. The definition of the `type.purpose` attribute is given in Figure 13.

The **source** attribute `type.source` is used to specify the *origin* of a data item, and therefore if it is primitive or nonprimitive. This information has implications for the reuse and archiving of the data. The definition of this attribute is given in Figure 14.

2. Item Names and Aliased Names. A domain dictionary written in DDLm is intended to be, first and foremost, a text

```
save_type.contents
  _definition.id          '_type.contents'
  _definition.update       2011-06-22
  _definition.class        Attribute
  _description.text
;
  Syntax of the item value
;
  _name.category_id        type
  _name.object_id          contents
  _type.purpose            State
  _type.container          Multiple
  _type.contents           Code
   loop_
  _enumeration_set.state
  _enumeration_set.detail
    Inherited   'type inherited from referenced item'
    Text        'a case-sensitive string/lines of text'
    Name        'case-insens name of alpha-num chars & _'
    Tag         'case-insens name starting with a _'
    Filename    'name of an external file'
    Code        'case-insensitive code to index data'
    Digit       'unsigned single digit number'
    Count       'unsigned integer number'
    Index       'unsigned non-zero integer'
    Integer     '+ or - integer number'
    Float       'floating-point real number'
    Real        'floating-point real number'
    Imag        'floating-point imaginary number'
    Complex     'complex number R+jI'
    Binary      'binary number \b<n>'
    Hexadec     'hexadecimal number \x<n>'
    Octal       'octal number \o<n>'
    Date        'ISO date format yyyy-mm-dd'
    Uri         'universal resource indicator'
    Version     '<major>.<version>.<update>'
    Range       'Inclusive range min:max'
save_
```

Figure 12. DDLm dictionary definition frame for the `type.contents` attribute, written in DDLm.

```
save_type.purpose
  _definition.id          '_type.purpose'
  _definition.update       2012-05-07
  _definition.class        Attribute
  _description.text
;
   The primary purpose a defined data item serves in a
   dictionary or a specific data instance.
;
  _name.category_id        type
  _name.object_id          purpose
  _type.purpose            State
  _type.container          Single
  _type.contents           Code
   loop_
    _enumeration_set.state
    _enumeration_set.detail
   Import      'import definitions from dictionaries'
   Method      'express methods relating defined items'
   Audit       'audit information about dictionary'
   Identify    'identify item, save frame or file names'
   Extend      'extend enumeration states'
   Describe    'Textual description'
   State       'value in enumerated states'
   Key         'key to packets in Loop category'
   Link        'links packet to another category'
   Composite   'Quantity with separate parts'
   Quantity    'An exact value numerical or text'
   Measurand   'Numerical estimate with its SU.'
   SU          'Standard uncertainty'
save
```

Figure 13. DDLm dictionary definition frame for the `type.purpose` attribute, written in DDLm.

document that is easily human- and machine-readable. For example, the data names specified with the attributes `definition.id`, and `alias.definition_id` can employ the full STAR character set. These attributes specify the data names that are used as identifiers in instance documents, and these names may include characters beyond the usual ASCII character set.

Each item definition also specifies a unique *internal* data name, and this is the identifier *always* employed in methods scripts. It is constructed from the values of the `name.category_id` and `name.object_id` attributes, and these *category* and *object* values form the internal name

```
save_type.source
  _definition.id              '_type.source'
  _definition.update          2012-05-07
  _definition.class           Attribute
  _description.text
;
   The origin or source of the defined data item, indicating
   by what recording process it has been added.
;
  _name.category_id           type
  _name.object_id             source
  _type.purpose               State
  _type.container             Single
  _type.contents              Code
   loop_
    _enumeration_set.state
    _enumeration_set.detail

 Assigned  'assigned to model data        PRIMITIVE'
 Observed  'recorded by observation.      PRIMITIVE'
 Measured  'number with its SU value.     PRIMITIVE'
 Derived   'derived from other items.     NON PRIMITIVE'
 Selected  'for administrative tagging.   NON PRIMITIVE'
 Assembled 'assembled from other items    NON PRIMITIVE'
save_
```

Figure 14. DDLm dictionary definition frame for the `type.source` attribute, written in DDLm.

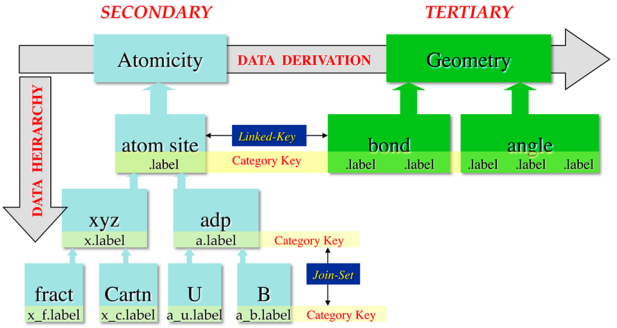


Figure 15. Hierarchical and derivative relationships for `ATOMICITY` and `GEOMETRY` data.

<category>.<object>. Apart from the strict adherence of this identifier to the hierarchy of definitions, which is essential to the algebraic functionality of the methods language dREL, it provides an important separation from the *external* data identifier (given by the value of `definition.id`) which serves to accommodate colloquial data tags used in instance documents that need not conform to the dictionary category hierarchy.

In this way, data names used in `definition.id` and `alias.definition_id` ensure compatibility with past naming practices, while the internal name conforms to the strict hierarchical requirements. That is, the internal data name satisfies the hierarchical naming conventions required in the methods scripts, whereas the external tag specified by `definition.id` need not. To enable these scripts to be easily convert into programming languages in dictionary applications, the character set for the values specified for `name.category_id` and `name.object_id` is restricted. Only the ASCII alphanumeric characters (U+0030 to U+0039; U+0041 to U+005A; U+0061 to U+007A) and an underscore (U+005F) are permitted. The period character (U+002E) is not permitted because it is introduced automatically as an object and category separator when constructing the internal name.

As stated earlier, because items defined in a DDLm dictionary may be aliased to multiple external names expressed in the *full* STAR File character set, any written language supported within UNICODE may be used for name construction, and this directly supports internationalization. In particular, for the crystallography domain, the `ALIAS` attributes enable complete compatibility with name definitions used in existing dictionaries defined using DDL1[5] and DDL2,[4] while, at the same time, ensuring complete functionality for these items within the dREL methods scripts.

3. Hierarchical Data Relationships.

The DDLm definition approach ensures that the hierarchical relationships between items and categories of items are recorded and are available for data derivation and validation processes. The nature of these relationships is illustrated for molecular structure data in Figure 15. The systematic specification of these relationships in a dictionary is essential to the precise characterization and organization of data. In this section, we explain how data hierarchies are defined and applied within the DDLm.

(a). Parent and Child Categories. All data items are divided into groups known as data categories. These groupings allow for the usual taxonomic organization of data (e.g., all the atomic data together, all of the crystal habit data together, and so on). Categories, as well as data items, can be grouped into categories, and this leads to the nesting and a hierarchy of categories. The outermost category is referred to as the "parent" category, and the categories it contains are its "child" categories.

Each data block in a dictionary contains a Head category that is the parent of all other categories in the dictionary. Viewed as a hierarchical structure, the head category is at the top of the tree and all other categories in the dictionary file are directly or indirectly its children. Categories in DDLm are divided into four classes: *Head*, *Loop*, *Ref-loop*, and *Set*. Items belonging to a *Loop* category must appear in an instance document as a loop (see the `ATOM_SITE` example above). Items in a *Set* category have only single values in the data cell enclosing the category. A *Head* category is a set-type category that is only used for dictionary definition.

The ref-loop category serves a special purpose. In an instance document, a ref-loop category is expressed as a loop (as shown in Figure 16) in which the key is a *reference* to a save frame containing data items defined as child categories of the ref-loop category. In a programming sense, the ref-loop is equivalent to an array of references to class objects. In this case the class objects are save frames containing data from the child categories. This construction enables loop categories to have child categories without resorting to nested loops. The loop key of a ref-loop category is always a `Ref-table type`. A data instance example of this approach is shown in Figure 16. Here the ref-loop categories are `EXPERIMENTS` and `EXPTL_CRYSTALS` and the data item `*.key` ref-table value serves to enable the `EXPTL` and `EXPTL_CRYSTAL` category of items to be repeated. Save frames may be nested and so can the use of ref-loop categories.

Note that child loop categories within a parent loop category share loop key values and in an instance document may be expressed either as separate loop lists, **or** as a single merged (i.e., joined) loop list containing one or both of the defined keys. In all other circumstances, loop categories can only be defined as children of a *Set* or a *ref-loop* category, as described above.

(b). Loop Categories and Keys. In each loop at least one data item must have unique values so that a row may be unambiguously addressed. These rows of values are referred to as loop "packets". The item that is designated to have "unique" values in a category is the "category key". The *key* in the

dx.doi.org/10.1021/ci300075z | J. Chem. Inf. Model. 2012, 52, 1907−1916

```
data_study_XYZ

<other data for study_XYZ>

loop_  _experiments.key
                ${"frame":"expt_blue"}$
                ${"frame":"expt_red"}$
save_expt_blue

<other data for experiment “blue”>

loop_  _exptl_crystals.key
                ${"frame":crystal_1}$
                ${"frame":crystal_2}$

save_crystal_1   < data for blue crystal 1>   save_
save_crystal_2   < data for blue crystal 2>   save_
save_

save_expt_red

<other data for experiment “red”>

loop_  _exptl_crystals.key
                ${"frame":crystal_1}$
                ${"frame":crystal_2}$

save_crystal_1   < data for red crystal 1>   save_
save_crystal_2   < data for red crystal 2>   save_
save_
```

Figure 16. The ref-loop invocation of the `experiments` category. This example records two experiments, each involving two crystals. An experiment has its own save frame and contains nested save frames for each crystal.

`ATOM_SITE` category (see Figures 4 and 7) is `atom_site.label`, and each value of this item provides the access to the other items in the packet. In a dictionary the category key is specified in the category definition with the `category.key_id` attribute.

(c). Linked Loop Keys. A category may contain data items whose values are *foreign keys* to a different category or even to the same category. In DDLm this is referred to as a *linked loop key*, and this key serves to give access to data values that can be meaningfully shared within and across categories. We illustrate both cases in the two loop lists shown in Figure 17.

```
loop_
_atom_site.label
_atom_site.type_symbol
_atom_site.fract_xyz
_atom_site.calc_attached_atom
  C1  C  [.41520, .69430, .49560]  .
  C2  C  [.31850, .66960, .63180]  H1
  C3  C  [.27660, .75080, .84370]  .
  H1  H  [.41000, .72000, .47000]  .

loop_
_atom_type.symbol
_atom_type_scat.dispersion_real
_atom_type_scat.dispersion_imag
_atom_type_scat.source
  O   .047  .032  'Int Tables Vol IV Tables 2.2B and 2.3.1'
  C   .017  .009  'Int Tables Vol IV Tables 2.2B and 2.3.1'
  H    0    0     'Stewart and Davidson'
```

Figure 17. Data instance containing two loop lists.

The first case is a link between items in the same category. In the first loop, the value H1 of the item `atom_site.calc_attached_atom` is a foreign key to the `atom_site.label` within the same loop. This link provides access to the position coordinates of the attached atom. The link is specified in the `atom_site.calc_attached_atom` definition in Figure 18 where `name.linked_item_id` is set to the parent key `atom_site.label`.

In the second case we show a linked key relationship across categories. The value of `atom_site.type_symbol` is a

```
save_atom_site.calc_attached_atom
  _definition.id          '_atom_site.calc_attached_atom'
  _definition.update       2010-11-03
  _description.text
;
   The _atom_site.label of the atom site to which the
   'geometry-calculated' atom site is attached.
;
  _description.common      'Atom Site Parent Atom'
  _name.category_id        site
  _name.object_id          calc_attached_atom
  _name.linked_item_id     '_atom_site.label'
  _type.purpose            Link
  _type.source             Assigned
  _type.container          Single
  _type.contents           Label
save
```

Figure 18. Definition of `atom_site.calc_attached_atom`.

key to a loop in the `ATOM_TYPE` category and equivalent to the `atom_type.symbol` item.

This connects the atomic properties and the atom's location in the crystal. Again, the existence of the parent link is specified in the definition of `atom_site.type_symbol` using the attribute `name.linked_item_id` as shown in Figure 19.

```
save_atom_site.type_symbol
  _definition.id          '_atom_site.type_symbol'
  _definition.update       2010-11-03
  _description.text
;
   A code to identify the atom specie(s) occupying this
   site. This code must match _atom_type.symbol. This code
   is optional if component_0 of the _atom_site.label is used
   for this purpose. See _atom_type.symbol
;
  _description.common      'Atom Site Type Symbol'
  _name.category_id        site
  _name.object_id          type_symbol
  _name.linked_item_id     '_atom_type.symbol'
  _type.purpose            Link
  _type.source             Assigned
  _type.container          Single
  _type.contents           Code
save
```

Figure 19. Definition of `atom_site.type_symbol`.

4. Sharing Definitions. The ready exchange and reuse of data definitions will in the future be critical to the efficient management of domain dictionaries. For dictionaries written in DDLm, definition *sharing* is achieved with the data item `import.get`. This is used to substitute requested external definitions in its place. The `import.get` value arguments are a List of Tables, shown in Figure 20, which identify save frames containing definitions to be imported.

Importing unique definition material, rather than replicating it in one or more dictionaries, has a number of advantages. First and foremost, it encourages common definitions to be shared within and across domains. Second it enables much better

Definitions are imported as a List of Tables

```
import.get [
  {'file':<file name>,          # source dictionary file
   'save':<frame code>,         # definition save frame code
   'mode':<import mode>,        # Full or Contents
   'dupl':<duplicate action>,   # if duplicates encountered
   'miss':<missing action>}     # if missing definitions
   , {…}, …]
```

Figure 20. Arguments of the getter item `import.get`.

dx.doi.org/10.1021/ci300075z | J. Chem. Inf. Model. 2012, 52, 1907−1916

management and maintenance practices. Dictionaries often contain long sequences of related definitions that have identical attributes differing only by one or two values. This repetition tends to mask small differences in definitions and unnecessarily obfuscates the dictionary. The use of `import.get` allows a definition *template* to be specified in an external dictionary, and for the template attributes to be injected at instantiation time. In this way definition maintenance is simplified because identical changes to repetitive definitions need only be made in one place. Figure 21 illustrates a typical sequence in the import process.

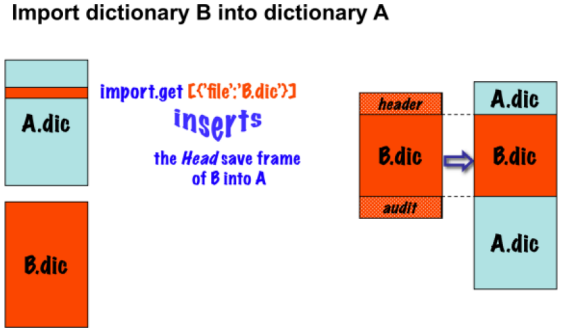


Figure 21. Illustration of the importation process sequence.

In summary, the DDLm import ability promotes the modularization of domain dictionaries that can be maintained by discipline experts. Definitions of the commonly used data items are then shared across specialist areas. The primary goal here is to avoid the duplication of definitions and ensure that only one recorded definition of a data item exists and that this is available to all domains. The organizational advantages of this are clear—the possibility of conflicting definitions for the same item in different dictionaries is avoided; dictionary maintenance of definitions need only be done in one place, and the latest definition of an item can be sourced from one place at the time of instantiation.

Because the importation *getter* processes are recursive (i.e., an imported dictionary may also import definitions), arguments in the `import.get` attribute indicate the action to be taken if duplicate or missing definitions are encountered. Figure 22 shows how the importation process takes place for nested imports.

5. Methods. DDLm supports user-defined methods written in a canonical language dREL tailored for the DDLm definition model. Only a brief overview of the dREL language will be given here: a full description and specifications are published separately.[3]

The dREL language has a *Python-like* syntax that enables algorithmic relationships to be easily defined without the structural complexities and bookkeeping necessary in most programming languages. dREL scripts are both canonical and intuitive, and as such permits derivative relationships to be written with little or no prior knowledge of programming, or the need to handle data structures, pointers, indices, etc.

Three classes of methods are supported. The class types *Evaluation*, *Validation*, or *Definition* are specified with the attribute `method.purpose`.

An *Evaluation* method describes, algorithmically, the derivation of a data item in terms of other data items defined in the same dictionary.

A *Validation* method is for checking the value data item against other related item values, and it may invoke its *Evaluation* method.

A *Definition* method is applied to change the definition of a data item in the context of the current instance values. For example, aspects of a definition (e.g., the allowed enumeration states) may be restricted according to certain data dependences. This facility can greatly aid validation processes.

A prototype interpreter of dREL methods scripts has been developed and is a *proof-of-concept* model for a full DDLm implementation.[3]

6. Domain Extensions to DDLm. DDLm provides the common data types that are enumerated in `type.contents`. Domains requiring specialized data representation can extend the `types.contents` beyond those defined in DDLm. These extensions are defined within an *extensions save frame* and contained in the domain dictionary.

If a data item's `type.contents` is set to a value that is not standard, its meaning, and translation into a preferred data structure, must be specified in the *extensions* save frame. In Figure 23 we illustrate the definition of a `type.contents` state value of **CText**.

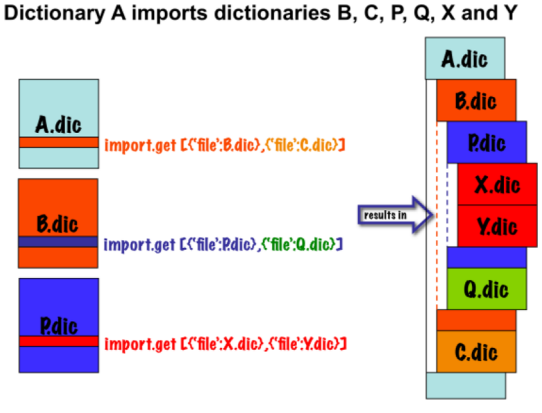


Figure 22. Hierarchy of nested imports.

```
save_CText
  _definition.id              'CText'
  _definition.update          2012-03-31
  _description.text
;
   A data definition that has
            _type.contents   CText
   will use this evaluation method below to translate
   the string value to the required data structure.
   Here the solidus is mapped as in the C language.
   eg. map \n ::= <newline>  \t ::= <tab>
;
  _name.category_id           contents
  _name.object_id             CText
  _type.purpose               Extend
  _type.source                Assigned
  _type.contents              Text
   loop_
  _method.purpose
  _method.expression
   Evaluation
;
   this.value = toCString(this.inputValue);

#  toCString() will be defined in the FUNCTIONS category
;
save_
```

Figure 23. Definition of the domain-defined **CText**.

dx.doi.org/10.1021/ci300075z | J. Chem. Inf. Model. 2012, 52, 1907−1916

This domain-defined type extension includes an *Evaluation* method to generate the value from the original input value, which is the raw string. In this case **CText** is the text string in which any escape (reverse solidus) sequences are converted in to the single character encodings, according to the specification of the C language. In this way a user may extend the DDLm `type.contents` to suit their specific needs.

Equally, defining domain-specific functions will simplify dREL scripts significantly. In Figure 24 we illustrate a domain-defined function. In crystallography *AtomType()* is used to return the identity of the atomic element used as part of the name of an atomic site (i.e., `atom_site.label`) in a crystal structure. The convention is to include the element name and charge when assigning labels to each atom site.

```
save_AtomType
  _definition.id            'AtomType'
  _definition.update        2011-06-30
  _description.text
;
   The function
                      r  =  AtomType( s )
   returns an atom type symbol (element name) from the
   atom site label.
;
  _name.category_id         function
  _name.object_id           AtomType
  _type.purpose             Extend
  _type.source              Assigned
  _type.container           Single
  _type.contents            Text
   loop_
  _method.purpose
  _method.expression
     Evaluation
;
   Function AtomType( s :[Single, Label])  {  # atom label

      m = Len(s)
      n = 1
      If (m > 1 and s[1] not in '0123456789') n = 2
      If (m > 2 and s[2]     in '+-'        ) n = 3
      If (m > 3 and s[3]     in '+-'        ) n = 4

      AtomType =  s[0:n]
    }
;
save
```

Figure 24. Domain-definition of the function `AtomType`.

EPILOGUE

The definition facilities of the DDLm language described in this paper provide the higher level of metadata needed for modern-day archiving and data mining. The language is flexible and extensible and, therefore, capable of expansion to meet yet unforeseen definition needs, without generating compatibility issues for existing archived STAR data.

ASSOCIATED CONTENT

Supporting Information

(Appendix A) a glossary of DDLm attributes. (Appendix B) *DDLm Reference Dictionary* as a STAR File. This material is available free of charge via the Internet at http://pubs.acs.org.

AUTHOR INFORMATION

Corresponding Author

*E-mail: Nick.Spadaccini@uwa.edu.au (N.S.); Sydney.Hall@uwa.edu.au (S.R.H.).

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

We wish to thank John Westbrook for the help and support he has given to the DDLm development. We also thank Brian McMahon and James Hester for their feedback and advice during the preparation of this publication. We gratefully acknowledge the support of this work by the Australian Research Council Discovery Grant DP0344560.

REFERENCES

(1) Spadaccini, N.; Hall, S. R. Extensions to the STAR File Syntax. *J Chem. Inf. Model.* **2012**, DOI: 10.10.21/ci300074v.

(2) Hall, S. R. The STAR File: A New Format for Electronic Data Transfer and Archiving. *J. Chem. Inf. Comput. Sci.* **1991**, *31*, 326−333.

(3) Spadaccini, N.; Castleden, I. R.; du Boulay, D; Hall, S. R. dREL: A Relational Expression Language for Dictionary Methods. *J Chem. Inf. Model.* **2012**, DOI: 10.10.21/ci300076w.

(4) *International Tables for Crystallography. Vol. G: Definition and exchange of crystallographic data*; Hall, S. R., McMahon, B., Eds; Springer: Dordrecht, The Netherlands, 2005.

(5) Hall, S. R.; Cook, A. P. F. STAR Data Definition Language: Initial Specification. *J. Chem. Inf. Comput. Sci.* **1995**, *35*, 819−825.

(6) Hall, S. R.; Allen, F. H.; Brown, I. D. The Crystallographic Information File (CIF): A New Standard Archive File for Crystallography. *Acta Crystallogr., Sect. A: Found. Crystallogr.* **1991**, *A47*, 655−685.

(7) Strickland, P. R.; Hoyland, M. A.; McMahon, B. Small-molecule crystal structure publication using CIF. In *International Tables for Crystallography. Vol. G: Definition and exchange of crystallographic data*; Hall, S. R., McMahon, B., Eds; Springer: Dordrecht, The Netherlands, 2005; Chapter 5.7.

(8) Hall, S. R.; Cook, A. P. F. Specification of the core CIF dictionary definition language (DDL1). In *International Tables for Crystallography. Vol. G: Definition and exchange of crystallographic data*; Hall, S. R., McMahon, B., Eds; Springer: Dordrecht, The Netherlands, 2005; Chapter 2.5.

(9) Westbrook, J. D.; Hall, S. R. A Dictionary Language for Macromolecular Structure DDL 2.1.11 http://mmcif.rcsb.org (accessed May 25, 2011).

(10) Westbrook, J. D.; Berman, H.; Hall, S. R. Specification of the core CIF dictionary definition language (DDL2). In *International Tables for Crystallography. Vol. G: Definition and exchange of crystallographic data*; Hall, S. R., McMahon, B., Eds; Springer: Dordrecht, The Netherlands, 2005; Chapter 2.6.

(11) Berman, H.; Bernstein, H. J.; Ellis, P. J.; Feng, Z.; Hall, S. R.; Hoyland, M. A.; McMahon, B.; Spadaccini, N.; Strickland, P. R.; Westbrook, J. D.; Yang, H. APPLICATIONS. In *International Tables for Crystallography. Vol. G: Definition and exchange of crystallographic data*. Hall, S. R., McMahon, B., Eds; Springer: Dordrecht, The Netherlands, 2005; Section 5.

(12) Data's Shameful Neglect *Nature* **2009**, *461*, 145.