Why Validator?

Information is your most valuable resource and applying standards effectively is a vital step to protect your data investment. ***Validator*** helps:

- Codify and disseminate meta-data standards.
- Test object labels for standards adherence.
- Detect similarities and overlaps in your databases.
- Suggest valid abbreviated labels and generate plain English descriptive names.
- Make the work of multiple project teams consistent.

The RESULT:

More consistent data that is accessible and shareable

KISMET
Analytic Corporation

P.O. Box 3218
Arlington VA 22203

www.kismeta.com
info@kismeta.com

703-243-5445

Introducing:

kisMETA
Validator™

Meta-data Standards Setting and Enforcing Aid

Object Label Standards

A guide to Meta-data Data Element and Object Label Standards Enforcement

Object Label Standards: A Primer

Validator is designed so that it may be used by all individuals responsible for system and database design and administration, as well as data management specialists. This paper provides a basic background in formal object label standards administration, and also conveys the philosophy behind Validator.

Why Object Label Standards?

The value of information as business capital is now widely understood. The function of data administration - and object label standards - is to make information available, comprehensible and functional. We use the phrase "object label standards" rather than data standards because each object from element to entity to code module to system has a meta-data label that benefits from naming standards.

Object/Data "administration" means classification and assembly into structures which can be maintained and from which information can be derived. A clerk setting up a filing system who clearly labels and organizes the folders according to a meaningful scheme, and then makes each project team member aware of the existence and organization of this scheme, is a genuine data administrator. In fact, every system development professional has some data administration responsibilities.

If we carry the filing metaphor further, we can see that this system will be of the most use if the pool of potential users share a common understanding of how files will be ordered and labeled. Even a visitor from a sister office would know which drawer contained the desired document and would also recognize the wording of the folder labels, since every office's files are identically arranged. Also, if the files are arranged alphabetically by employee name, most users would be surprised and probably disconcerted to find them arranged by the first name rather than the last. This is an example of a "implicit" object label standard that need not be made explicit, but these are in short supply in the real world. Most standards will have to be worked out as an explicit standards "set", established by decree or by consensus agreement of the parties involved.

Unfortunately, naming conventions are typically inconsistent among software applications. There have been several attempts to establish a universal rules set for object label standards, most of which ride on the same elementary structure, which we will elaborate shortly. Let us simply preface our discussion by stating that *by standardized object label, we mean information that is described briefly, accurately, and uniquely.* We strongly believe that *all object labels should be administered with both sharing and reuse in mind.*

> ***An Observation**:*
> *The point of standards is more about finding similarities than finding*

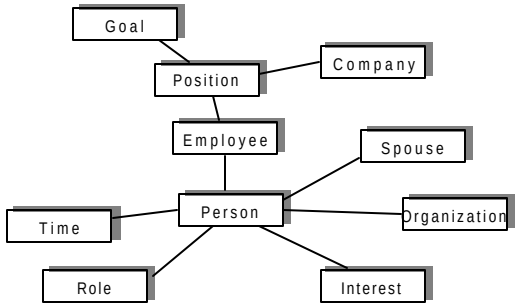
XML and EDI illustrate some of the mechanisms that the industry has adopted in the quest for inter-organizational communication. XML is often described as a syntax solution to meta-data, however it is neither better nor worse at that task than, for example, the ODBC standard. Within XML, the meta-data label itself must be understood between the two communicating systems, syntactically and semantically, before there can be an exchange of information. The techniques described here are urgently needed in an XML context.

Few means exist which can have as powerful influence on information accessibility and system user-friendliness as good object label standards. In addition, standards are central to the smooth accommodation of heterogeneous technologies and facilitate object sharing that can dramatically reduce the cost of systems development and maintenance.

What do Standards Standardize?

The focus of Validator-supported standardization is creating meaningful and consistent meta-data object labels. This helps with a much bigger and more important problem: knowing what information is contained in the objects labeled. This bigger problem is beyond the scope of just object labels, which cannot be both sufficiently expressive *and* concise and simple enough to be practical. A name (object label) can only symbolically stand in for TRUE MEANING.

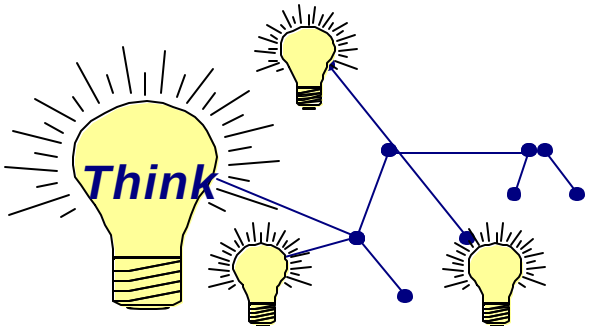
It is useful to consider a mechanism often used to understand an object's TRUE MEANING: a semantic network. The general idea is that a concept does not stand alone, but is understood only in the context of other concepts. Consider a person, what is she? The answer is: Employee, Spouse, Attorney, Mother,



Trivial Semantic Network

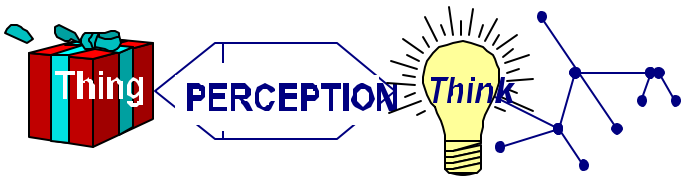
Georgian, Veteran, Class of '75... the list can be very long. A complete understanding is impossible without understanding all of these other concepts, and the concepts that they in turn relate to. Each piece of information is attached to the real world by dozens of strands.

> *No two people see the external world in exactly the same way. To every separate person a thing is what he thinks it is - in other words, not a thing, but a think.*
>
> *- Fitzgerald Penelope*



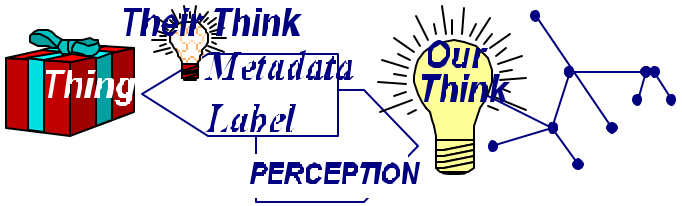
Semantic Network
a.k.a Conceptual Schema, Ontology, Frame of Reference, Viewpoint ....

In light of this, we can see that a scheme whereby a package of information is labeled with one "prime word" such as "Employee" is rather lame if our expectation is that one standard label will work for every function of a small company, let alone for one that would work in a huge and diverse environment such as the US Government. Yet, as a symbol it works. Sometimes standardization efforts chase the dream of making a name somehow capture enough meaning that anyone looking at it, in whatever context, would fully understand the meaning intended. These efforts are likely to fail because of their impractical scope. The first job for



Perception Relates *Things* to *Thinks*

a label is to be a symbolic placeholder for an idea; a possible and proper additional ambition of a standard object label is to modestly aid the beholder's process of perception - associating a thing with a concept.



The *Best* Meta-data Label is Still Somebody's Else's *Think*, but Still Manages to Aid Perception

Full meta-data requirements include a:

- good name
- data type and size
- imperative indicator (optional/ mandatory)
- description
- place in conceptual schema/semantic net.

Philosophies of Naming Standards

Context Independent Whole Labels - versus - Context Dependent Component Assembly

In the ***context independent*** approach, organizations attempt to establish standards by publishing lists of valid labels that are to be used by all applicable systems. Good examples include:

SSN	=Social Security Number
FIP	=Financial Information Pointer (Account #)
EMP_ID	=Employee Identifier

The intent is that the whole label is a context independent piece of information of real value to the organization; the examples above are highly effective tools for achieving goals of object label standards.

Unfortunately, the results are less effective when the concept is pushed further. For example, the following is an actual example from a corporate object label standard:

"Payroll hrs + month chge amt"

This might be a useful fact for the payroll or personnel function, but it is difficult to conceive of this information capsule being of widespread utility across multiple systems and departments. It is precisely universal

utility that is the test of the value and appropriateness of a standard. The problem is that there are a great many useful idea combinations for which information can be collected. If you identify thousands, that leaves millions unidentified.

The ***context dependent*** component assembly approach manages standards through four sets of rules: lexical, semantic, syntactic, and procedure rules.

- **Lexical rules** limit the terms allowed for use to a pre-established lexicon of valid terms. The objective is usually to limit the common vocabulary to somewhere between 400 and 1000 well defined terms.

- **Semantic rules** clarify the role of certain terms in labels based on their meaning . For example, "first" is an adjective/modifier/qualifier, and "code" is a datatype class / representation term.

- **Syntactic rules** recommend relationships among term types and state the intended inference to be drawn from word order.

- **Procedure rules** state the obligation, if any, to follow precedents.

Given these rules, the user is able to construct the elements that meet his or her needs.

Context independence implies context inheritance. If a "class" is a more general or "higher level" concept, it is the confluence of classes that uniquely define an object. In universal terms, the set of data elements is the set of all class interaction combinations. A standard example of context inheritance is as follows:

Business Area/Subject: *Personnel*
Entity: *Employee*
Element: *Last Name*

(Last name is understood to be of an Employee within the Personnel business area)

Context Dependence of Data

Can every data object be labeled so completely as to permit each fact to "stand alone?" Consider: "Acceleration Due to Gravity on the Surface of the Earth," an example of relative data independence - a piece of information that is self contained and self explanatory.

Clearly, substantial context information is bound up in this label: acceleration is different, for example, on the Moon, or 20 miles above the Earth's surface. Hence, a ten-word sentence is needed to describe this simple fact. Unfortunately, information only half as context-independent as this example is exceedingly rare in business systems settings. For practical purposes, there is no such thing as data independence, since part of the meaning is always derived from the specific business context, and may further depend on the particular user's perspective.

In a physical database, the full implied label of every element by rights consists of at least the entity identity, which is to say each identifying primary key. After that, an additional prime word assignment is redundant.

Taken literally, this implies that elements do not need an explicit reference to an entity (i.e. prime word) to be meaningful, as long as the context is clear. Nevertheless, whenever context is not clear, as in the case of foreign keys, entity references (prime words) are required. But even in this latter case, the entity reference need not be a permanent part of the logical element. The value of prime words in an element is a mater of debate; we argue that they can be profitably be done without in most cases. This results in far fewer elements and more emphasis placed on the proper use of entity and other context information.

If we could change one thing about IDEF1X, or at least how it is implemented by tools such as ERwin, it would be to allow elements to be named with context dependence. For example Entity Project*' unique ID would be simply "Code" - implicitly, due to its context, it would mean "Project_Code". Any time it was used outside of context, for example as a foreign key, it would automatically appear as "Project_Code". This would restore the Entity to its proper role as the key to object meaning, and would allow elements - properly akin to words in*

Drawing from this model, elements should be treated as data objects that inherit characteristics from class membership. Another way of stating this: an object is defined by its position within a conceptual schema (semantic network).

A Third Philosophy: Conceptual Schema Labels

The final approach is one used heavily throughout the world for meta-data labeling of books and the like. The Dewey Decimal System and Library of Congress (LOC) system are arbitrary, highly simplified, and flattened master conceptual schemas. Each piece of information is forced into a slot in the scheme. If the fit is good in several slots or poor in any slot, a likely slot is picked arbitrarily; one unique catalog number is always assigned.

The system partially accommodates other conceptual schemas through a virtual mechanism: cross indexing - having multiple "cards" for every book, one in each major subject area. (e.g. books on historic shoes in 'fashion' with a chapter on shoe repair may get a secondary reference under 'crafts'.)

This is more-or-less the intent behind the use of prime terms. (e.g. just as the "library science" slot might not be perfect for data element naming guides, "Employee" might not be the perfect entity for "Employee_Home_Phone_Code" but it is the best fit available.) Unfortunately, the required supporting conceptual schema, if it exists at all, is often feeble and poorly understood compared to the well defined structure of the LOC.

É

The makers of Validator recommend depending on a flexible but well regulated context-dependent approach, supplemented with a very small number of context-independent standard elements with business-critical identifying characteristics, such as "SSN". This recommended hybrid approach, the pure context-independent approach, and the pure context-dependent approach are all supported by Validator.

We further recommend development of a conceptual schema describing the organization, and assigning elements to this schema as a way to identify "true meaning".

A Note on Terminology

Several different words are sometimes used to describe each of a number of basic objects in data design, a fact which may cause great and justifiable confusion.

*The distinctions being made are useful to distinguish between different phases of the design process. For example, though the terms **entity** and **record type** are nearly synonymous, we will use the term entity when speaking in more abstract, "logical" frameworks, and the term "record type" when referring to a physical database table. In general, entities represent business concepts with specific semantic meanings and domains. Similarly, we have tried to be consistent in our use of the word **attribute** to refer to one of the descriptive characteristics of a record (i.e., a label for a column), and the word **element** to mean a component of an entity; we have avoided using the term **field** because of its frequent application in describing a single cell in a table.*

Semantic Classification

The three broad classifications of data are:

Prime/Mod/Class		ISO11197
1. (Data Type) Class	•	Representation term
2. Subject/Prime/Topic	•	Object Class Term
3. Modifier	•	Qualifier Term

The Prime/ Modifier/Datatype Class scheme, popularized by W.R. Durrell, has been adopted in many data standards, including DOD8320. A competing but similar standard devised recently is ISO11197.

Datatype Class words are used to identify and describe the general purpose (or use) of a data element. Examples include code, amount, age, ID, and name. They are roughly related to the physical data type of the database element, e.g. alphanumeric, real, memo, etc. ISO1197 uses the expression "representation term" to stress the form of the data being described.

Prime words represent a topic or subject area. They are often the names of entities on a logical model. Examples include "customer," "invoice," and

"employee." A prime word is the most important identifier of the element; it anchors the semantic and conceptual meaning of the object being defined. ISO11179 uses the expression "object class term."

Modifying words, such as ordinals and other adjectives, are used to round out an element name, providing any further detail. ISO11179 uses the expression "qualifier term."

Syntactic Classification

By this model, then, every element should consist of one class word, one prime word, and one or two modifying words. Though hardly universal, the standard syntax for an element label is:

modifier(s) + datatype class word

or

prime + modifier(s) + datatype class word

or

prime + prime-modifier + class modifier + datatype class word

A convenient consequence of placing the prime word first is that related elements are grouped together when in alphabetical order.

When class modifier terms are used, they are usually:

- Ordinal: first, last, new... or
- Type/Role: rate, angle, money, weight, percent... or
- Unit: day, month, meter, room, dollar...

And the datatype class is tightly defined as a

- Representation: code, amount, ID, text....

Components of the Relational Database

Data designers distinguish between so-called "flat-file" databases, in which most or all relevant data for a particular purpose is kept in a single table, and "relational" databases, in which the data is kept in multiple tables which are connectable by shared elements. The flat file designer seeks consolidation; the relational designer seeks to avoid redundancy and inconsistency.

A "relationship" is evidence of a meaningful association between two entities. Relationships are described by cardinality (e.g. one-to-many) and other aspects of the business rule they represent.

A relational data structure identifies a set of architectural components that reflect the prime/modifier/datatype class scheme.

First, a hierarchy of **subjects** (or topics) is specified:

1. Function/domain/database: = High level subject
2. Record type/file/entity: = More detailed subject/topic
3. Primary key: = Specific topic
4. Foreign key: = Related or subordinate topic
5. Main term of element: = Minor subordinate topic
6. Secondary term of element: = Minor subordinate topic

Second, **data types** are classified:

1. Class words within elements
2. Data type field content

Third, **elements** are detailed using ordinals and other adjectives as modifiers:

1. Personnel - Employee - (last)name
2. Personnel - Employee - spouse(last)name

[All of these structures can be described as relationships within an ontology, or semantic network, the proper construction and maintenance of which is the role of *kisMeta Analyst* and *Schemer*.]

The Element Specification

The aspects of the element may be taken collectively as the "specification." Each specification contains three types of information: conceptual, internal (physical) and external (logical).

1. *Conceptual*: The fundamental meaning of the element, including the element name and the description, map to conceptual schema, business rules, and domain.

2. *External*: How the information is represented to the user, primarily the data type and display format, and name on form.

3. *Internal (Physical)*: How the values are stored in a field, usually consisting of the datatype and length, the required value, the range of values, and the default value. This may be extended to include data systems characteristics.

Naming Conventions

Two types of names are typically assigned to elements: the business (long) name and the physical (short) name. The **business name** is the foundation

for an entity, data element and attribute. The **physical name** is the abbreviated form of the business name, and is what typically appears in a physical database table. In each case, they should consist of the minimum number of words that adequately identify the data element.

A **lexicon of standard terms** is essential to any naming standard. The terms should have full names for identification purposes, as well as standard abbreviations. Term names will be used in the business name of an element, while abbreviations will be used in the physical representations of that element, which must be shorter. In addition, each term should be given a description, to avoid confusion where a single term is common to two or more business functions but has different meanings for each. Minus the definitions, your lexicon becomes the Valid Terms List as used by Validator.

Model Business Naming Conventions

The following rules for devising business names are common to a number of standard sets. They are offered here as a collection of "model" data element naming standards that may provide inspiration. Where Validator specifically assists with the enforcement of a standard, the rule is followed by "**SPEC**" plus the number of the specification screen page on which the option is found. Where Validator does the process automatically, or optionally generates an issue message, the rule is followed by "**VAL**."

Semantic Rules

- Business names should be clear, accurate and self-explanatory.

Please observe that some of these "standards" are contradictory and overlapping. This is only a collection of ideas!

- They should be named according to logical, and not physical considerations, that is, they must not contain the names of organizations, computer or information systems, directives, forms, rows or columns of screens, or reports. They should be "context-free" **(SPEC - 2)**.
- They should not include redundant concepts **(SPEC - 3)**.
- They should not express multiple concepts; they must be a single primary concept. **(SPEC - 2)**.

Syntax Rules

- Names should consist of the minimum number of words that categorize the data element, often a maximum of 50 characters **(SPEC - 3)**.
- Abbreviations or acronyms should be avoided, except for universally accepted abbreviations or acronyms as well as acronyms that are officially approved for usage; abbreviations may be used according to standard especially when length restrictions must be met **(SPEC - 3)**.
- Only alphabetic characters, valid separator characters (e.g. a space or an underscore), and in rare cases, numbers (0-9) are permitted **(SPEC - 1)**.
- Prepositions are not permitted unless essential for clarity **(VAL)**.
- Articles, conjunctions, plural words **(SPEC - 3)** and possessive forms of words are not permitted **(VAL)**.
- Verbs should be avoided **(VAL)**.
- Terms with synonyms and homonyms should be avoided.

Structuring Conventions

- The business name consists of a prime word, optional modifiers, and a class word **(SPEC-3)**.
- Each element will have one and only one definition and domain. The business name will be unique among all other data elements. If similar elements have different domains or definitions, a separate element will be defined. The prime word will normally be first **(SPEC - 2)**.
- Elements should be context free to the extent possible. Prime terms should not be used if possible.
- The prime word identifies the object to which the business name refers; hence, a data name must contain one and only one prime word **(SPEC -2)**.
- Prime words should be discrete and non-overlapping.
- For a prime word, modifiers are used to differentiate an entity from other similar entities. For a class word, modifiers are used to differentiate the properties of an entity from other similar properties of that entity.
- The use of modifiers should be restricted, since the use of multiple modifiers in a name for purposes of uniquely identifying the business name may be an indication that multiple concepts are represented. The result may be limited potential for reuse and sharing.
- Class words are normally the last word of a business name and follow the prime word and its modifiers **(SPEC - 2)**.
- A data name must contain one and only one class word **(SPEC -2, SPEC -3)**.

- Class words are reserved and should not be used as prime words or modifiers.
- Class words should be discrete and non-overlapping.

Physical Naming Conventions

Physical attribute names are constructed by abbreviating the business name, using standard abbreviations. They should conform to the restrictions of the programming language and the DBMS being used **(SPEC - 1)**. (For example, SQL Server imposes an 18 character limit.)

Guideline for Abbreviating Names

- Use abbreviations from the approved abbreviation list.
- Business and logical names should be abbreviated from right to left to meet the length restriction. All words in the physical name longer than five characters, with few exceptions, must be abbreviated **(SPEC - 1)**.
- Abbreviated words must be in the same sequence as the words in the business name. Begin with the same letter as the word being abbreviated, and the order of letters must parallel the sequence in the business name **(VAL)**.
- Generally, an abbreviation is formed by eliminating the vowels from a word, unless the word begins with a vowel **(VAL)**.
- Generally, if a double consonant appears in the abbreviation, drop one of the consonants **(SPEC - 2)**.
- Avoid lengthy abbreviations **(SPEC - 2)**.
- Repeatability and consistency of abbreviation must be promoted across systems.
- Abbreviations should be unique **(SPEC - 3)**.
- Designers should strive for abbreviations of 4 or fewer characters whenever possible to support repeatability and consistency, since all future usage combinations cannot be known **(SPEC - 2)**.
- When a physical data element name is designed from approved abbreviations and the length still exceeds the character limit, shorten or drop the least meaningful component of the element's name until the length restriction is met **(VAL)**.

Disseminating and Enforcing Standards

Issues include:

Proper Scope. Standards must be set for an entire organization or enterprise; there is no room for separate standards in multiple divisions of a company.

Acceptance and support. Effective standards require "buy-in" from both management and technical development staff.

Availability. Standards must be published and disseminated through paper documentation and training, and ideally as part of an automated validation tool such as Validator.

Enforceability. Relevant standards, kept simple and understandable. An easy-to-use testing method must be available which can be used by any local developer who is designing a data structure. Beyond a few generalities, all standards must be expressed as specific criteria which can be tested: e.g., "not more than 50 characters" in contrast to "not too long".

Responsibilities for Maintenance. Standards must be monitored and maintained by a central function such as DA, DBA, or system architecture. Individuals responsible for using standards - such as database developers, must provide information feedback that must be used to ensure that the standards remain practical and relevant in a changing technological and business environment.

Responsibilities for Enforcement. Standards should also be enforced centrally, in the course of system design reviews; however, standards must be primarily enforced as part of normal business practice by development and design staff. Roles include:

- Top management - Support and reward concept of efficient and effective systems, including accessibility and reusability through standards enforcement
- Central DA/DBA/System Architecture staff - Support enforcement by development staff through provision of standards information and validation methods and tools; perform formal standards validation reviews as part of critical milestone reviews.
- Management - insist on and reward standards application, and monitor compliance
- Development/Design Staff - understand and use standards; perform internal standards validation as quality control.

Timeliness. Standards must be applied when systems are designed for the first time, not as part of some formal review after-the-fact when coding is half-done and databases are populated with test data. That is too late, and is very costly in resource use and in delayed schedules, and may even result in standards which are ignored for expediency. For this reason, standards enforcement cannot be left solely in the hands of a central group such as DA, but must be part of every developer's responsibility.

Conclusion

Object label design and naming is a big job with invaluable consequences. Adhering to standards is a necessary part of the job if information is to be easily accessed and shared. We have offered some opinions about what makes some standards better or easier to use than others, but ultimately the best standard is one that is actually used. Validator can alleviate the burden by testing labels and offering suggestions for modifications based on standards specified by the user, and can enforce consistency.

Best of all, Validator can be placed in the hands of the people who need it: system designers and database architects doing new systems or system reengineering work. Validator's Enterprise version is designed to be centrally maintained, but to be used by every project team in an organization.

NOTES: