Gellish Syntax for Formalized Languages

Definition of the Gellish Expression Format

including a definition of

the Gellish Contextual Facts

Version 7.4

November 2020

Author: Dr. Ir. Andries van Renssen
Gellish.net
http://www.gellish.net
e-mail: info@gellish.net

The Gellish family of formal languages is defined in two parts:

1. This document describes its syntax.
2. The Gellish Taxonomic Dictionary-Ontology of Formal English, or one of the formal Gellish Dictionaries in other formalized natural languages, which defines the semantics.

Furthermore, the language definition is described in the book 'Semantic Information Modeling in Formalized Languages' and its application is described in the book 'Semantic Information Modeling Methodology'.

The latest version of the Taxonomic Dictionary-Ontology of Formal English and Formal Dutch can be licensed via the web shop: http://www.gellish.net/
The Taxonomic Dictionary-Ontology contains definitions of concepts, including also definitions of kinds of relations. The definitions include definition models as well as textual definitions expressed in natural English. The concepts are arranged in a subtype-supertype hierarchy (taxonomy). This is the semantics part of the definition of the formal language.
Formal language expressions can be stored in universal Semantic Databases and can be exchanged between systems using message files or query messages, all expressed in the Gellish Expression format.

This document defines the structure of expressions (the definition of the syntax) in Gellish.

Change history

Version	Date	Modification relative to previous version
4.0	March 2003	Definition of subset models defined.
5.0	August 2007	Gellish database description clarified.
5.1	June 2008	Queries and table header definition extended.
5.2	February 2011	Implementation guidelines and unique key discussion added. Subset Business Model extended.
6	December 2012	Contextual facts description improved. Universal Gellish Expression Format tables described as integration of Naming Tables and Ideas Tables. UID of intention, inverse indicator, author of copy and addressee added.
6.1	March 2014	Restructured
6.2	Nov 2015	Extent, probability and exponents added as contextual facts
6.3	March 2016	Phrase type UID added as language independent indicator of first and second roles and role player columns (base and inverse phrases).
7.0	October 2017	Non-numeric UIDs added, JSON support added. Columns for multiple languages added.
7.1	October 2018	First header row extended with a prefix and with keys.
7.2	January 2020	Key based first header line; UIDs for numbers and percentages.
7.3	November 2020	UIDs for roles of individual things in relations.

Table of Content

1. Introduction

Interoperability of systems requires primarily data exchange between systems in a neutral common formalized language with a standard, yet extensible vocabulary. The Gellish family of formalized natural languages is developed for providing that role.
Semantic modeling is the methodology for expressing knowledge, requirements as well as information about individual things in such a formal language and its interpretation.

Semantic modeling and semantic databases and messages are based on the principle that the meaning of expressions is explicitly represented in the models themselves and is not stored as separate documentation about the models, databases or messages. Semantic modeling is therefore based on a formal semantic modeling language that enables the explicit modeling of meaning.
The formal languages are a formalization of natural languages. The core of the formalizations is the discovery that meaning can be expressed as collections of basic semantic units (BSUs), each of which is basically a binary relation, extended with a number of contextual facts. The definitions of the concepts, especially the definitions of kinds of relations and the syntax (the sentence structures) of that language determine its semantic expression capabilities. It appears that a formal language can be defined and applied using universal basic semantic patterns for all its expressions. Those patterns specify that is necessary and sufficient for the expression and interpretation of any expression in the formal languages. The formal languages enable making expressions of any kind of knowledge, requirements and definitions as well as any idea about individual things, and they also enable expressing questions and answers, promises, commands, etc. These universal basic semantic patterns enable defining universal semantic databases as well as a universal formal language for data exchange messages (see also the Gellish wiki on semantic modeling).

This document specifies the Gellish Expression Format, which is the syntax of the expressions in the formal languages, which means that it defines the structure of the expressions. This syntax is the basis for expressing information in data exchange messages between systems, APIs, queries as well as for data storage and data integration in databases. Together with the semantics of the formal languages it enables expressing and interpreting information that is expressed in one of the Gellish formalized languages. The semantics of the formal languages is defined in the Gellish Taxonomic Dictionary-Ontology, which is also expressed in the Gellish Expression Format.

Semantic models in Gellish are unambiguously interpretable by computers. Therefore, the terminology that is used in semantic expressions that make up semantic models is not free, but is predefined in the form of formal definitions of formalized terms and concepts in the electronic Gellish taxonomic dictionary. That dictionary is more than an ordinary dictionary, because it not only contains terms and definitions of concepts, but it includes also explicit relations between the concepts as in a taxonomy and an ontology, whereas the definitions and relations are computer interpretable. Thus for English, the Formal English taxonomic dictionary-ontology defines the Formal English language. The dictionaries are extensible by the users. Extensions of the dictionary with other natural languages only need to relate new terms to existing concepts, because the relations are language independent and can thus be re-used from the taxonomy-ontology that first defined the concepts and the relations.
When in this document the term 'the dictionary' or 'the formal dictionary' is used, this Taxonomic Dictionary-Ontology is meant or an equivalent one of another formalized natural language, such as the taxonomic dictionary of Formal Dutch ('Taxonomisch Woordenboek van Formeel Nederlands').

The Gellish Expression Format that is defined in this document can be expressed in any basic format, such as CSV or JSON. It is only one of the possible syntaxes for the formal languages. Other syntaxes may be based for example on RDF. To enable the definition of alternative syntaxes the definition of the concepts is accompanied by a definition of the relations between the various columns in the tabular Gellish Expression format.

The Gellish formal languages are standardized structured subsets of natural languages that can be understood by humans as well as by computers. They are not programming languages, nor a data definition languages. Formal languages use ordinary words, terms and expressions from natural languages. They do not invent their own terminology, such as Esperanto did. However, formal languages restrict the way in which sentences (expressions) may be made and therefore they standardized a large number of phrases and terms. For example, the Taxonomic Dictionary of Formal English includes ordinary words, such as car and wheel, length and color, buying and transporting, etc. and also includes a large number of standard English phrases, such as ‘is a part of’, ‘is located in’ and ‘can have as part a’. With those phrases sentences can be made, such as: a car can have as part a wheel. The standard phrases are defined and documented in the Base Ontology section (the upper ontology) of the Taxonomic Dictionary.

The taxonomic dictionaries of formal languages are special and powerful, because they not only specify the meaning (semantics) of the components (words and phrases) in formal language expressions, but they also relate synonym terms, abbreviations and codes to the unique identifiers (UIDs) of their common concepts and they distinguish homonyms (the same terms with different meanings in their ‘language communities’) by allocating different UIDs to them. The dictionaries can also include pictures to support or illustrate definitions. The concepts in a Gellish dictionary are arranged in a subtype-supertype hierarchy that is compliant with their definition and the dictionary relates concepts to other concepts when such relations can be derived from their definition. Therefore the Taxonomic Dictionary is also called a Taxonomy.

Semantic expressions are not only expression of *base semantic units (ideas, facts, opinions, conditions, consequences, questions, commands, etc.),* but each semantic unit needs to be accompanied by a number of *contextual facts* in order to provide a context for interpretation. Base semantic units (or ‘ideas’ for short*)* are the core units of meaning that one want to store or communicate. For example, the fact that ‘the Eiffel tower is located in Paris’ is expressed as a base semantic unit (an assertion). Contextual facts are facts that provide the context of a base semantic unit and adds details about the base semantic unit, such as the intention with which it is communicated, its status, who expressed it, when that was done, in which context it is valid, etc., etc. Those contextual facts are optional, because in many applications have an implicit context that is considered sufficient for its interpretation.

This document specifies how basic semantic units *(such as statements and opinions)* are expressed in one of the formal languages of the Gellish family of formalized languages and it specifies a standard collection of obligatory and optional contextual facts: *the Gellish collection* of *kinds of contextual facts*. Base semantic units (ideas) and their contextual facts may be stored and exchanged in Gellish Expression formal tables or in any public or proprietary syntax.

Most conventional database and message data structures only enable storing subsets of formal expressions and do not enable support the rich semantic expression capabilities of the complete formal language. This document also specifies universal data structures for Semantic Databases and Data Exchange Messages. Databases and messages that are based on this specification are suitable to store or exchange semantic units of any kind, such as facts about products, knowledge representations, rules, E-Business messages and database queries and responses.

Conventional databases typically consist of some or many tables, each of which is composed of a number of columns. The definition of those tables and columns determine the storage capabilities of the database, whereas the relations between the columns define the kinds of facts that can be stored in such a database. Those columns and relations determine the database structures that defines the expression capabilities of the databases. Similar rules apply for information that is exchanged between systems in electronic data files.

This conventional database technology has some major limitations:

- When data was not covered during the database design and thus is not included in the data model, then such data cannot be stored in the database nor exchanged via such a data file structure.
- Different databases have different data structures, which causes that data in one database cannot be integrated with data from other databases nor exchanged between databases without dedicated data conversion.
- A database modification or extension requires a redesign of the database table structures and modification of the software, which is a complicated and costly exercise.

New Semantic Web technologies, such as RDF or OWL based data storage and data exchange have more flexibility. They allow any kind of relation and have no constraints on terminology, thus they do not standardize the language that is used in the databases and messages. This means that those technologies require additional standards to define a common language for use in different databases and between parties. Formal languages are designed to act as such common languages, which can be used on their own, or in combination with 'meta languages' such as RDF and OWL.

Semantic databases and message files for data exchange that apply formal languages do not have the limitations of conventional databases, nor the unconstrained conventions of languages such as RDF, XML and OWL. Formal languages provide flexibility by specifying primarily the general semantics of expressions and they standardize the languages, by providing standard relation types and definitions of concepts and terminology, together with guidelines for (proprietary) extensions. In addition to that this document specifies a number of contextual facts about each base semantic unit (also called meta data about each idea). These expressions can be implemented in any meta language, including for example RDF or OWL. However, formal languages can also be implemented in the powerful tabular Gellish Expression Format, in which any opinion or fact can be expressed, together with their contextual facts, as is specified in this document.

This document specifies how semantic units, such as ideas, facts and queries should be expressed in formal languages. It defines a syntax for *formal expressions* as well as the *Gellish collection of kinds of contextual facts,* presented in the form of the definition of the *Gellish Expression Format* for databases, message files and queries that can be filled with data written in a *formal language*.

1.1 Gellish data structures for formalized languages

Gellish expressions and semantic interpretation rules have a universal data structure and syntax, which implies that information, knowledge and requirements that are expressed in a formal language can be presented in multiple files that all have the same data structure, such as table columns and column definitions. The format is suitable for Messages as well as Databases and Queries. That universal format is called:

- *The Gellish Expression Format*

The Expression Format basically consists of one kind of table: an *Expression Table* or an equivalent form for expressing the same content, such as the *Gellish RDF Expression Format*. The Expression Format enables the use of multiple aliases and languages for the same concepts and enables to keep track of which alias or translation is used in which expression. Therefore we will explain the format as if it consists of two kinds of tables (that are combined into one *Expression Table*):

- *Naming Dictionary*
  A Naming Dictionary consists of a table that is intended to contain the relation between UIDs and terms and aliases, such as synonyms, abbreviations, codes, etc. in multiple languages and language communities.

- *Expressions of Ideas*
  Expressions of Ideas consists of a core that contains basic semantic units and its extension with their contextual facts, optionally using UIDs and no names.

Queries are modeled in Gellish in the same way as product and process models, except that in queries *unknowns* (undefined objects) may be used. This makes that queries can be expressed using the same Gellish Expression Format as for expressions of statements or other intention. The expression format can optionally be extended for queries with a specification of 'string commonality' requirements. Such requirements specify to which extent searched terms should correspond with a search string. Thus a Gellish Expression Format extended with a few optional fields defines a:

- *Gellish Query Format*
  Queries can be expressed using the same format extended with a few query specific fields.

Formal language databases are typically implemented as object oriented databases (O-O databases) that include a Naming Dictionary, but they may also be implemented by one or more tables conform the Gellish Expression Format, or as triple stores. The database may be either centrally managed or may be distributed in a collection of cooperating databases. In the latter case the distributed sections may operate independent of each other, provided that the allocation of UIDs and the usage of UID ranges is coordinated. This coordination process is described in the book Gellish Information Modeling Methodology.

A Gellish Expression Format table has three fundamental characteristics:

1. A Gellish table enables storing information about *any kind of thing*. The identity of individual things of any kind are introduced as new concepts in the language dictionary through allocation of a UID as representative of the individual thing throughout the language and by explicit classification of the individual things by any of the kinds of things that are already defined in the Gellish Dictionary. This differs from conventional databases that predefines a limited number of object types or entity types and a particular collection of attribute types in their data model.
2. A Gellish table enables storing *ideas of any kind (possible facts, options, etc.)* about things or kinds of things. This is enabled through the expression of ideas as (collections of) binary relations between things, whereas all relations are explicitly classified by kinds of relations that can either be selected from the standardized kinds of relations that are defined in the Dictionary or from kinds of relations that are added to the Dictionary as proprietary extensions. Furthermore, the expressions can express any kind of intentions, because each expression of an idea is accompanied by an intention (an illocutionary force), which indicates whether the expression is an assertion, a question, an answer, a promise, a command, a condition, a consequence, etc.
3. A Gellish table enable to store *kinds of contextual facts* about each idea, such as the natural language in which it is expressed, its applicability context, intention, scale, status, successing idea when replaced, date of start or end of applicability, author, etc.

As a consequence, Universal Semantic Databases or Message files have a data structure that does not need to be modified or extended when the scope of an application changes. Furthermore, different Universal Semantic Databases or files can be combined, merged and integrated, and act as one distributed database, whenever required without a need for data conversion, provided that the unique identifiers are managed.

The Gellish Expression Format data structure is simple and even human readable. Furthermore, it supports the simultaneous use of multiple languages, because it contains the option to express for each idea in which language it is expressed. This is supported by the fact that Gellish is a family with various natural language specific variants that share the same unique identifiers (UIDs) for the same concepts. For example, Formal English is a formalized subset of natural English and thus uses natural English terminology, whereas Formal Dutch is a formalized subset of natural Dutch and thus uses natural Dutch terminology, but they both recognize the same concepts represented by the same UIDs.

1.2 Natural language variants and automated translation

In principle, there is a variant language for each natural language, depending on the availability of a translation of the terms by which the concepts are denoted. For example, The Formal English Dictionary defines Formal English and the 'Formeel Nederlands Woordenboek' defines Formeel Nederlands (Formal Dutch). International terminology (such as most units of measure and mathematical concepts) is included in the Dictionary as International language. As languages can be combined in Gellish messages and databases, it is possible to use e.g. English for the kinds of relations and the bases ontology, while using e.g. Chinese for the terminology in a particular application domain. In such a case, the Chinese dictionary does not need to introduce new concepts, but only needs to relate Chinese names of concepts to the existing unique identifiers (UIDs).

This is a main benefit of the fact that Gellish formal languages use a language independent UID for each thing. This includes also user objects, concepts that are defined in the Dictionary as well as ideas and kinds of relations. This also enables the use of synonyms and homonyms. As a consequence it enables that a computer can automatically translate expressions in a certain language into expressions in other languages, provided that Gellish dictionaries for those languages are available or are provided

by the user. Such an automated translation is possible because of the fact that the meaning of a formal expression is captured as a relation between the unique identifiers, so that the meaning is expressed in a language independent way.
This adds power to cooperation, such as via the Semantic Web, because messages can be created e.g. in Formal English whereas they can be presented in other formal language variants, while the computer software has invisibly done the translation.

1.3 Documentation

Gellish formal languages are defined in the Gellish Taxonomic Dictionary-Ontology. The core of that dictionary can be free downloaded.
The full language definition can be licensed from Gellish.net. The database itself is written in Gellish. The Gellish formal languages are compliant with ISO 16354 – Guidelines for Object Modeling and Knowledge Modeling. The Gellish languages are a further development of the ISO 10303-221 and ISO 15926 standards from which also ISO 12006-3 standard is derived.

Guidance on how to use formal languages for the expression of information, knowledge and requirements, can be found in the following documentation:

- The Gellish website at http://www.gellish.net/.
- The books 'Semantic Information Modeling in Formalized Languages' and 'Semantic Information Modeling Methodology'.

1.4 Formal languages syntax and semantics

The definition of formal languages includes a semantic part and a syntax part:

- **Semantics**
  The definition of the *semantics* (the meaning) of the concepts in the formal language, which is needed to interpret formal expressions, consists of expressions that form a definition model conform a universal basic semantic pattern. Each concept also has a textual definition and a specification of terms (names, codes, numbers, synonyms, etc.) and phrases that denote those concepts in various languages and language communities. Each concept is represented in Gellish by a unique identifier (UID). A definition model comprises a collections of expressions (basic semantic units with their contextual facts). The names and phrases that denote the concepts form the vocabulary of the language and thus provide the terminology that may be used in formal expressions. In addition to that the users of the language can add terms by attaching them to the UIDs to existing concepts or to the individual things or kinds of things that they define. The semantics of concepts (the definition models) is contained in the Taxonomic Dictionary. The basic collection of definition models is provided in the base ontology section.
  This document defines the kinds of contextual facts that may accompany expressions of base semantic units. The collection of kinds of contextual facts is called the *Gellish collection of kinds of contextual facts*[1].

- **Syntax (the Gellish Expression Format)**
  The structure of formal expressions (their *syntax*) is defined in the form of Gellish Expression Format. The *Gellish Expression Format* is a tabular format that has a header of three lines and a body that can have various columns that are selected from the definitions in this document and an unlimited number of rows. A file header should be compliant with the specification in par. 4.2.1. The header specifies the columns in the body. Because the columns are identified by a language independent identified, the sequence of the columns in the table are free.

  The tables can be implemented in any tabular notation, such as in SQL or in CSV or JSON or even as tables in a Spreadsheet. However, they can also be expressed in other formats. For

1The Gellish set of kinds of contextual facts about core ideas is comparable with the Dublin Core of meta data (contextual facts or attributes) about 'resources', where a 'resource' typically is a file (document) or web page, but usually not an idea or fact. In Gellish, references to files and documents and ideas or facts about them are included in expressions of ideas.

example as collections of Triples (as in RDF/XML or OWL) or Quads (as in Trix) as described in ISO 15926-11.

This document provides the definitions of the Gellish syntax in the form of Gellish Expression Format tables for databases, messages and queries and gives recommendations for their RDF and XML equivalent representations.

For an extended description of formal languages and its capabilities see the Gellish Wiki and the above mentioned books.

2. Expression of ideas

Expressions of ideas consist of collections of elementary binary relations. We distinguish between relations that express a core idea, which specify the topic that is addressed in the expression and relations that express auxiliary facts and contextual facts about that topic. The latter categories form the context that is required for a correct interpretation of an expression.

Gellish allows for a free choice of the elements that form an expression, with a minimum of three (the Minimum subset) and with the constraint that the elements are selected from the ones that are defined in this document. Chapter 3 describes the Minimum subset and various recommended subsets. Chapter 5 describes the possible expression components.

2.1 The core of an expression

An expression of the core of an idea that capture the core of its meaning (the semantics) consists of a number of elementary relations that relate 'expression components' to each other.

1. The *idea* itself

   The idea itself is represented by a unique identifier (the UID of the idea).

2. A *binary relation* that relates two objects that are involved in the relation.

   The relation is represented by *two related things*, each playing a particular role in the idea. Thus each of them is related to the idea by a particular kind of involvement relation. One of the is the player of the first role that is by definition played in the relation (usually[2] the left hand object) and the other is the player of the second role that is by definition played in the relation (usually the right hand object). The objects are represented in the relation by their respective unique identifiers (UIDs).

3. A *classification* of the relation.

   This is a fact that classifies the relation by a (standard) kind of relation. This classifying relation is represented by a pair of things: the UID of the idea and a UID of the kind of relation that classifies the relation.
   The binary relation and its classification together declare the topic of the expression.

4. A qualification or quantification of the *extent* in which a statement is the case – when applicable.

   This is an auxiliary fact that specifies the fraction for which the main expression is the case. Typically a fraction (possibly expressed as a percentage weight or volume) that expresses the extent in which a part in a composition relation is a fraction of the whole, or the fraction of a mixture for which a classification by substance holds.

5. A classification of a quantification relation by a scale (a *unit of measure*) – when applicable.

   This is an auxiliary fact that classifies a relation by a (standard) kind of scale (also called a unit of measure). Typically this classifies a quantification relation, but it may also classify an extent in which a composition relation or a classification relation is the case (see below on 'extent'). This auxiliary fact is represented by a pair of things: the UID of the fact and a UID of the kind of scale that classifies the relation.

6. A qualification of the expression by the *intention* with which the expression is communicated.

   This is an auxiliary fact that expresses with which intention the core idea is communicated. For example, it may express that the fact is communicated as a statement or as a denial, a confirmation, a command or a question.

2In Formal English expressions it is allowed that inverse phrases for kinds of relations are used, but in a Fact Table, inverse phrases may not be used. Thus in a Fact Table the left hand object is always the player of the first role.

Any expression of a core idea in a formal language should therefore consists of the above relations, whereas those relations relate six component. Those expression components can be represented by UIDs that are independent of any natural language. The components are presented in Table 1.

Expression component ID (column ID)	Description of object
1	UID of an idea (fact)
2	UID of a left hand object
60	UID of a relation type
15	UID of a right hand object
30	UID of an extent
66	UID of a scale (unit of measure)
5	UID of an intention

Table 1, The core components of an expression

When those six component are arranged in a syntactic structure that defines the above described relations between them, then that structure forms an expression. Thus the expression of a the core of an idea is an arrangement of seven components, consisting of seven UIDs in a syntactic structure.

2.1.1 Core Table

The simplest syntactic structure for implementation of the core components of an expression is a table. A tabular syntactic structure implies the relations between the columns in the table which define the relations between the components of the expression. Thus the above expression components and their relations can be implemented in a Core Table and thus enables the expression and interpretation of the meaning of the core of expressions of ideas. The Core Table is therefore defined by seven columns, each of which is identified by a column ID. Each row in a Core Table represents a combination of the seven components, each represented by its own UID.

For example, Table 2 is a Core Table that illustrates the expression of a statement with UID 201.

1	2	60	15	66	5	30
UID of an idea	**UID of a left hand object**	**UID of a kind of relation**	**UID of a right hand object**	**UID of a UoM**	**UID of an intention**	**UID of an extent**
201	101	5026	102	570423	491285	928419

Table 2, A Core Table with the identification and expression of the core of one idea

Note that the sequence of the columns in the table is free, because from the standardized IDs of the columns their meaning can be deduced (and a sequence can be constructed when required). Furthermore, the names of the columns on the second row is free, because they follow from the definitions of the IDs.

Each object that is represented in column 2 of Table 2 is called a left hand object and denotes the player of a ‘first role’ in a relation. The kind of the first role is defined in the definition model of the kind of relation (as is defined in the base ontology section of the Taxonomic Dictionary). By analogy column 15 denotes a right hand object, which is the player of the ‘second role’ in the relation of a specified kind.

The UIDs in Table 2 represent objects (things). The terms (names, etc.) of those objects in natural language are specified in expressions of separate auxiliary facts, which are called naming relations, as is specified in the paragraph 2.5. Those naming expressions are typically provided in a Naming Dictionary (see par 2.4.1.1). Replacing or accompanying UIDs by columns with terms that denote the UIDs in some language delivers a human readable equivalent for Table 2.

Usually a Naming Dictionary will contain various synonyms terms for the objects that are denoted only by a UID in a Core Table. Therefore, in principle it is necessary to record which term is used in which expression. This can be done by creating a separate Term Usage Table that contains a specification of which terms are used for each UID that appears on a row in a Core Table. However this issue is solved by using integrated Expression Format tables (see par. 4).

The following paragraphs discusses other expression components and the relations between them. Precise definitions of the expression components and the kinds of relations between them are provided in chapter 5.

2.2 Expression of roles of role players

Each object that is involved in a relation plays a role of a particular kind in that relation. Often such a role of a physical object is called a usage or an application and such roles of aspects or characteristics are called intrinsic aspects or intrinsic characteristics. Kinds of objects are then used or applied in a particular way, which does not imply that a subtype of object is involved, but it implies that an object of that kind has a kind of usage or a kind of application. Thus each binary relation implies two different roles played by the related objects. This also holds for subtypes of roles, such as kinds of usage or kinds of application.
Those roles often may remain implicit in expressions, because the kind of relation implies particular kinds of roles. For example, the kind of relation that is denoted by the phrase <is a part of> (a composition relation) implies the kinds of roles 'part' and 'whole'. The definition of such kinds of roles follows from the *definitions* of the kinds of relations (as specified in the base ontology section of the Dictionary).

In a number of cases it is required to model those roles explicitly. This especially holds for the modeling of the values of intrinsic characteristics of standard product types and for the modeling of constraints. For example, assume that an 'xyz-cable' <is by definition made of> PVC, then it would be too generic to specify that 'material' <is by definition> PVC, but we should specify that the intrinsic characteristic 'xyz-cable material' <is by definition> PVC.

Note: Many syntaxes or notations use brackets after an identification of a physical object to denote that intrinsic characteristics are meant even though names of generic characteristic are mentioned. Such brackets specify e.g. that if we find after 'article: xyz-cable' {'material' <is by definition> PVC}, then we should interpret that as: 'its material' <is by definition> PVC. However, in fact we mean 'material of an xyz-cable' <is by definition> PVC and the latter is the way in which it is specified in Gellish. Thus, by making roles explicit, Gellish makes the meaning of such brackets explicit and thus makes the expressions independent of each other and independent of a particular sequence.

Roles can be made explicit by modeling a role as a separate object in either of two ways:

1. By treating a role in the same way as a role player. This means that a statement that an object plays a particular role in a relation, is explicitly expressed by two relations:
   - A relation between the object and the role that is played by that object.
   - A relation between a relation and the role of the kind that is required by that relation.
   These kinds of expressions of facts about roles can be stored in the same way as all other expressions of ideas. It only requires the recording of the roles as separate objects and explicit classification of those roles and defining kinds of roles in their own subtype-supertype hierarchy (taxonomy).

7. By inserting a left hand and right hand kind of role in an orderly expression. This implies that for each idea four contextual facts are defined: two that specify that the two objects play roles of those kinds and two that specify that those kinds of role roles are subtypes of the kinds of roles that are by definition required by a relation of such a kind.

In a tabular form this means that a Core Table is extended with two additional columns; one for the left hand kind of role and another for the right hand kind of role, whereas also the names of the kinds of roles need to be defined in a Naming Dictionary. The kinds of roles classify the played roles and are implied subtypes of the kinds of roles that are by definition involved in the kind of relation. This is implemented in a Core Table as follows:

1	2	72	60	74	15	66	5	30
UID of an idea	UID of a left hand object	UID of a left hand kind of role	UID of a kind of relation	UID of a right hand kind of role	UID of a right hand object	UID of a UoM	UID of an intention	UID of an extent
201	101	301	5026	401	102	570423	491285	928419

Table 3, Core Table extended with explicit kinds of roles

Thus the explicit modeling of the kinds of roles implies an extension of a Core Table with the following columns:

Expression component ID (column ID)	Description of object
72	UID of a left hand kind of role
74	UID of a right hand kind of role

Table 4, Extension of a Core Table with kinds of roles

2.3 Expression of queries

The modeling of a dialogue between systems typically requires modeling the various communicative acts as separate occurrences. The questioning, answering, confirmation, etc. are modeled as activities that are classified by kinds of activities. However, even with or without modeling the dialogue itself, it is also required to model a question or query as a message. The kinds of communicative acts are specified by the intention, which is discussed before. Thus for a query the intention is ‘question’ on the first line and ‘query specification’ on the following lines. Optionally, the collection of query lines can be grouped together by specifying that the lines are elements of an explicit collection of expressions in the collection columns (see par. 5.35). The general model of a query message is further discussed in detail in the book on Semantic Modeling Languages.

The extent of correspondence between a search string and a candidate string to become a match can be indicated by a string commonality, as is described in more detail in par. 5.36 etc. The three components of an expression that express contextual facts for terms in queries are specified in Table 5.

Expression component ID (column ID)	Description of object
80	Left hand string commonality
81	Right hand string commonality
82	Relation type string commonality

Table 5, String commonality columns in a Query Table

These components imply additional relations between these components and the left hand term (character string), the right hand term and the name of the relation type respectively that should be interpreted from the syntactical structure of the expression.

A tabular implementation enables interpreting the relations from the definition of relations between the columns. This makes that a query can be implemented as a Query Table. Such a Query Table is an Expression Table that is extended with three additional columns (80, 81 and 82) in which the commonality criteria for the left hand and the right hand term and the name of the kind of relation can be specified.

2.4 Expression of contexts

A proper interpretation of the meaning of an expression (or proposition) requires not only that the core idea (the topic) is expressed with the terms in the user preferred language and language community, but it also requires that the expression includes information about the context in which an expression is made. Therefore, semantic modeling not only requires expressions of the core ideas themselves, but it also requires that each expression is accompanied by additional expressions of facts about the core idea. Such additional facts are called 'contextual facts' about a core idea.

Contextual information is also required for the management of information. For example, for proper interpretation as well as for proper information management it should be recorded who has created an expression, when that was done, what the status of the expression is, since when it is outdated or replaced by another expression of a fact, in what language it is expressed, etc.

Each expression of an idea shall be accompanied by such contextual facts. Standard kinds of contextual facts are discussed below.

2.4.1 Language and language community contexts

Terms (names) of things and phrases are language and language community context dependent. Formalized languages use natural language independent UIDs for representing things, ideas and facts as well as for the components in expressions of those ideas. Not only the objects and their aspects, but also the mean meaning is expressed in a language independent way.

The relations between language independent UIDs and natural language and language community dependent terms and phrases (names) are specified through naming relations.

Naming of an object requires four elementary relations. A naming relation (1) is a relation between a term (possibly being a code or a phrase that can be used in any expression) and the thing (UID) that is denoted by that term, whereas the naming relation is classified (through a classification relation (2)) by a kind of naming relation. Every naming relation requires a relation (3) with a language community that is the contexts in which the naming relation is based and declared valid. Every term also requires a relation between the term and a natural language (4), which specifies that the term belongs to the vocabulary of that language. Those four relations form a collection of contextual binary relations that forms a pattern for naming something, also called a naming model.

The components of a naming model that includes the language and language community contexts are given in Table 6.

Expression component ID (column ID)	Description of object
69	UID of a natural language
71	UID of a language community
101	Term (name, phrase, abbreviation, code, URI, number or symbol)
60	UID of the kind of naming relation
2	UID of a thing that is denoted by the 'term' in the language, as originated in the language community
64	Partial textual definition

Table 6, Components for a pattern for naming something

The relations between these components are implemented through the syntactical structure or format of expressions.

The definitions of the language and language community are given in the following paragraphs. The definition of the components 101, 60 and 2 were already provided in par. 2.1)

2.4.1.1 Naming Dictionary

In a tabular implementation the relations between the components are defined by the definition of the relations between the columns in the table. For example, each contextual relation in a collection of contextual relations can be represented in tabular form as one Naming Dictionary table, provided that the relations between the columns in that table represent the kinds of relations for that collection.

Such a Naming Dictionary Table therefore has the following table header:

69	71	101	60	2	65
UID of a language	UID of a language community	Term	UID of kind of relation	UID of a named thing	Partial definition

Table 7, Header of a Naming Dictionary table

The columns 69, 71 and 101 together form a unique key, which means that a combination of those three items may occur only once in the table..

The columns have their own column ID's that uniquely identifies the columns, independent of a natural language. This enable that the column titles are free text descriptions that can vary per language or user preference.

69	**54**	**71**	**16**	**101**	**60**	**2**	**65**
UID of the language of the term	**Name of the language**	**UID of the language community**	**Name of the language community**	**Term (name)**	**UID (Name of kind of relation)**	**UID of named thing**	**Partial definition**
910036	English	192936	technology	pump	5117 (is a name of)	130206	that is ...
910037	Dutch	192936	technology	pomp	5117 (is a name of)	130206	die ...
910038	German	192936	technology	Pumpe	5117 (is a name of)	130206	welchem..
910036	English	190668	linguistics	German	5117 (is a name of)	910038	...
910038	German	190668	linguistics	Deutsch	5117 (is a name of)	910038	...
910037	Dutch	190668	linguistics	Duits	5117 (is a name of)	910038	...
910036	English	193259	ontology	assembly relation	5117 (is a name of)	1190	..
910036	English	492015	Formal English	is a part of	1981 (is a base phrase for)	1190	...
910036	English	492015	Formal English	has as part	1986 (is an inverse phrase for)	1190	...
910036	English	492015	Formal English	is a whole of	1986 (is an inverse phrase for)	1190	...

Table 8, Naming Dictionary with UIDs and names of a concept in various languages

The use of a Naming Dictionary table is illustrated in Table 8 on three examples:

1. The concept represented by UID 130206 is denoted in English as pump, in German as Pumpe and in Dutch as pomp. The language community where these names originate is 'technology'. Table 8 illustrates how those various names in those three languages are allocated to the concept that is denoted by UID 130206. It also illustrates that each language requires its own textual definition, whereas also language communities may add their own textual definition, provided that the meaning of the concept is respected. Furthermore, the definition is called a partial definition, because the Gellish methodology prescribes that each definition of a concept should start with a reference to a supertype (or classifier) of the defined concept. Thus the partial definition of a concept C is assumed to be preceded by a phrase such as: a C 'is a S' (where S stands for the supertype), which thus is typically followed by the partial textual description 'that is ...'.
2. Table 8 also illustrates an example of how names of languages differ in various languages. For example, the name of the German language, expressed in German is Deutsch and in Dutch it is Duits. Table 8 illustrates how the various names of the German language are related to the concept that is denoted by UID 910038.
3. The third example gives the names of a kind of relation, its denoting base phrase as a synonym for the name, its inverse phrase and an alternative for the phrase.

Table 8 should be interpreted as follows:

- The table has two header rows. The numbers in the first row, 69, 54, 71, 16, 101, 60 and 2, are standardized natural language independent identifiers of the table columns. They refer to standard columns in Expression Format tables as is described later in this document. The texts on the second line are not-standardized names of those columns.

- The second and the fourth columns (54 and 16, in red) are added for clarification, but are semantically superfluous and are not part of a standard Naming Dictionary table.
- The UID of the language in the first column (69) specifies the language in which the term in column (101) is expressed. Thus the number 910036 on the first row, which is the Gellish UID of the English language, specifies that the term ‘pump’ is an English term for concept 130206. Similarly, UID 910037 denotes the Dutch language and UID 910038 denotes the German language.
  Note that the fourth line specifies that the term ‘English’ is the English name of the language that is represented by the UID 910036.
- Column 60 denotes a UID of the kind of naming relation. In order to facilitate the readability of the example table the name of that relation type is given in addition, although that name is superfluous and does not belong to a Naming Dictionary table. Note that the UID of the kind of relation could also indicate other kinds of naming relations, such as ‘is an abbreviated name of’ or ‘is a code for’ and some other variations. If the UID is 1986, then the ‘name’ consists of an inverse phrase, which denotes that in a relation the left and right hand terms are switched to express the same idea as when base phrases are used.
- The columns 69, 71 and 101 together form a unique key for the table.

Not only all dictionary concepts, but also each user-defined concept or individual thing (user defined object[3]) that is used in formal expressions of core ideas shall have a Gellish UID. Each user defined object UID shall be unique and shall be allocated conform the rules for allocation of UIDs (as described in the book ‘Semantic Information Modeling in Formalized Languages’.

2.4.2 Multi-language dictionaries

The above described language and language community expression components allow for usage of various languages in one collection of expressions. They also allow for the specification of aliases, synonyms and translations, on separate lines. However, this is inconvenient for the specification of large numbers of translations.

To facilitate translations as well as textual definitions in various languages, the Gellish Expression Format is extended with two optional additional columns per language: one for a name in some language and the other for a textual description in that language. The column ID in the Expression table for the name (the naming column) is the integer value of the UID of the applicable language in the Gellish Dictionary; the column ID for the textual description is the ID of the naming column plus 1000000.
An example table in English, German and French that expresses that a centrifugal pump is a kind of pump, thus will contain the following columns:

Expression component ID (column ID)	Description of object
910038	Name of left hand object in German
1910038	Textual partial definition in German
910039	Name of left hand object in French
1910039	Textual partial definition in French

Note that the text on row 3 in an Expression table is free, and thus may be formulated in German, French or whatever.

The Expression table will become:

3In this document the unqualified term ‘object’ is used as synonym for the term ‘anything’.

54	16	101	910038	1910038	910039	1910039	3	201	65
Name of a languag e	Name of a language community	Name of left hand object EN	Name auf Deutsch	Beschrei -bung (DE)	Nom en Francai s	Définitio n (FR)	Name of kind of relation	Name of right hand object	Partial definitio n
English	substance	water	Wasser	das ...	eau	c'est ...	is a kind of	substance	that is ...

A name and textual description in additional languages is only useful in expressions that define concepts or that specify aliases. Thus they are only applied on lines that express specialization relations, classification relations and alias relations and their subtypes. Names and texts on other lines may be ignored by software that reads and interprets the expressions.

2.4.3 Contextual facts

Each core idea is accompanied by a number of prime and secondary contextual facts. Together that collection of facts is called the *expression context*, which is a set of kinds of contextual facts. Each of the contextual facts (which are specified below) is expressed as a binary relation that relates a pair of objects and a classification of that relation. The classification relation and the classifying kind of relation that classifies the relation may remain implicit in implementations (for example in a tabular implementation where they are defined by the definitions of the columns and the relations between the columns that make up an expression). However it depends on the kind of implementation whether the contextual relations can be interpreted from these relations and thus whether they should be made explicit in order to enable semantic interpretation. The latter is for example the case in triple stores and RDF implementations.

2.4.3.1 Prime contextual facts

The objects that are specified in the following table imply relations that express prime contextual facts. Definitions of these contextual facts as well as those in the next table are given in the following paragraphs.

Expression component ID (column ID)	Description of object
44	A pair of left hand object minimum and maximum simultaneous cardinalities.
45	A pair of right hand object minimum and maximum simultaneous cardinalities.
76	A UID of the accuracy of a quantification.
70	A UID of a pick list for the qualification of aspects.
19	A UID of the applicability context for an idea.
65	A partial definition in natural language of a concept or individual thing.
4	A full definition in natural language of a concept or individual thing.
42	A textual description of a core idea.
14	Remarks on the expression of a core idea.
8	Approval status of the expression of a core idea.

Table 9, Prime contextual facts

The definitions of these components of an expression are given in the following paragraphs.

2.4.3.2 Secondary contextual facts

The secondary contextual facts are facts that do not directly contribute to the semantic interpretation of the facts, but are added for administrative reasons. They include the facts in the following table.

Expression component ID (column ID)	Description of object
24	Reason for latest change of status.
67	UID of the successor of the idea, in case the idea has the status ‘replaced’.
11	UID of creator of fact.
9	Date-time of start of applicability of the fact.
23	Date-time of start of availability of the expression.
22	Date-time of creation of copy.
10	Date-time of latest change of the expression.
6	UID of author of latest change of the expression.
78	UID of addressee of the expression.
13	References.
53	UID of the expression of the fact. (Line UID)
50	UID of a collection of facts to which the fact belongs.
0	A sequence in which the expressions are presented. (Presentation sequence)

Table 10, Secondary contextual facts

Uniqueness constraints are implementation constraints that intent to prevent that a database contains identical expressions in which also the contextual facts are identical. It depends on the scope of a database which expressions including context are considered to be identical. For example, in an extreme situation two identical expressions about the same fact, thus semantically having the same meaning, but expressed by different persons (originators), may be considered to be two different expressions in one context, whereas they are considered to be the same expression in another context. This means that it might be required to add the originator to the uniqueness constraint. Similarly, when a requirement is stated to be valid in multiple applicability contexts, then this means that there are multiple requirements, each with its own ‘fact UID’. This implies that the ‘applicability context UID’ should be added to the second uniqueness constraints.

2.5 Naming relations for objects in expressions of ideas

In principle, every UID that is used in an expression of a core idea, or in an expression of a contextual fact, is denoted in a human readable expression by a term (name, etc.), or by more than one term in case of synonyms. The terminology is recorded in naming relations between UIDs and terms.

In Databases all the naming relations of UIDs can be recorded in a separate Naming Table. However, it is also possible that they are included in an integrated Expression Table (see par. 4). In an integrated Expression Table the UIDs as well as the terms are included in the table itself.

Table 11 specifies all the names that imply naming relations (expressions of additional contextual facts) that are required to allocate names (terms) to the UIDs that are used to express core ideas and contextual facts.’ Note that ‘name’ stands for a character string that can be a term, a code, a phrase, a number, a URI, etc.

Expression component ID (column ID)	Description of object
101	The name of a left hand object.
201	The name of a right hand object.
3	The name of a kind of relation.
31	The name of an extent (typically a number)
7	The name of a scale (UoM).
54	The name of a language.

16	The name of a language community.
73	The name of a left hand role.
75	The name of a right hand role.
43	The name of an intention.
12	The name of an author of latest change.
77	The name of an accuracy of quantification.
20	The name of a pick list.
68	The name of a collection of facts.
79	The name of an addressee of the expression.
83	The name of a creator of a fact.

Table 11, Naming columns in an Expression Table

3. Subsets of expression components & context

Expressions in messages or databases may consist of the full set of expression component, ideas and contextual facts as defined in this document. It may also consist of a subset of them.

The definition of these subsets implicitly also define subset Expression Tables.

Depending on the application, users may decide to use a flexible subset or one of the predefined standard subsets of the collection of contextual facts.

The following subsets of facts are defined, each with its equivalent subset Expression Table:

- Subset Minimum subset
- Subset Flexible subset
- Subset Nomenclature
- Subset Dictionary
- Subset Taxonomy
- Subset Product Model
- Subset Business Model (recommended)
- Query tables

These standard subsets are defined in the following paragraphs.

The subsets require the presence of all elements that are specified for the chosen subset and the elements shall be arranged in the indicated sequence, with as only exception the Flexible subset.

The default subset is the *Flexible subset*.

3.1 Subset: Minimum subset

A *Minimum subset* is intended for exchanging statements.

A Minimum subset has powerful expression capabilities and is suitable for usage in not too complex applications in closed communities. The subset allows for the use of synonyms, but does not provide a mechanism for the explicit distinction between homonyms. It can be used in any language, but does not make an explicit distinction between languages. It is suitable for expressing statements and queries, but does not distinguish other intentions such as promises, denials, commands, etc. It assumes that statements are true and timeless, because it does not provide for contextual facts, such as an approval status, source and timing information about the expressed facts.

Users of Minimum subsets should ensure that the terms (names) of objects in the messages are unique or that the distinction between homonyms is apparent from the context in which the terms are used and that synonyms are explicitly declared to be synonyms.

A Minimum subset consists of a *triple*. It expressed only three elements of an expression of a core idea, expressed in formalized natural language terms. Such a minimum subset consists of the following three expression components:

Expression component ID (column ID)	Description
3	A name of a relation type (= formal language phrase)
101	A name of a left hand object
201	A name of a right hand object

Table 12, Minimum subset

There are two extensions of the minimum subset of interest:

1. An extension with a unique identifier for the idea (the statement). Such an expression is called a quad.
2. An extension with an optional unit of measure.

3.1.1 The Minimum subset in Gellish Expression Format

Minimum subsets may be expressed (implemented) in various ways (syntactic structures or formats).

The Minimum subset Gellish Expression Format contains only the three columns: 101, 3 and 201. An example of such a table is:

101	3	201
Name of left hand object	**Name of relation type**	**Name of right hand object**
the Eiffel tower	is located in	Paris

Table 13, Minimum subset Expression Format table

The Gellish Expression Format can be exchanged e.g. as a CSV file or a JSON File, preferably encoded in Unicode UTF-8.

The Extended Minimum subset is illustrated in tabel14.

1	101	3	201	7
UID of idea	**Name of left hand object**	**Name of kind of relation**	**Name of right hand object**	**Name (symbol) of unit of measure**
101	the height of the Eiffel tower	has on scale a value equal to	324	m

3.1.2 Other formats

3.1.2.1 Function Notation

Another format is a *Function Notation* which uses a kind of binary relation as the name of a function and the names of the two related objects as its arguments. Thus in the following form:

relation type (left hand object, right hand object)

The expression in Table 13 expressed in function notation becomes:

is_located_in (the Eiffel tower, Paris)

Note that the spaces in the name of the kind of relations are replaced by underscores.

The function notation allows for extensions

3.1.2.2 RDF

Minimum subset expressions are triples of expression components. They form the basis is triple stores and graph databases. The format is compatible with the RDF (Resource Description Framework) standard of the World Wide Web Consortium (W3C) and has similarities with its Notation 3 (N3) format.

Note: A more elaborate Expression Table with additional columns can also be represented as collections of triples and can also be expressed in RDF or Notation 3 RDF as is described in the last chapter.

3.1.2.3 Quads

An Extended Minimum subset adds a UID of the statement. The Extended Minimum subset is compatible with 'named graphs', N-Quads and the TriX (Triples in XML) serialization format for RDF graphs.

3.2 Subset: Flexible subset

A **Flexible subset** is a subset that contains at least the non-optional expression components. The non-optional components are: 2, 101, 1, 60, 3, 15, 201, 8, 9 and 10 as described in Table 14.

Expression component ID (column ID)	Description
2	UID of left hand object
101	Name of left hand object
1	UID of an idea
60	UID of relation type
3	Name of relation type
15	UID of right hand object
201	Name of right hand object
8	Approval status
9	Date-Time of start of applicability
10	Date-time of latest change
etc	Free choice of additional columns (in any sequence)

Table 14, Minimum expression components for flexible subset

Note: Expressions that consists of more than three expression components can be represented as collections of triples. For example, when they are expressed in RDF or Notation 3 RDF extended with an indicator for the collections (such as in TRIX). Such a format is described in ISO 15926-11.

The selection of additional optional columns as well as the sequence of the columns is free. The sequence of the columns in an Expression Table is semantically irrelevant, because the columns shall be uniquely identified by their column identifiers and the relations between the columns are defined independent of their position in the table.

A Flexible subset may even include non-standard additional columns, which columns are then treated as comment from a formal language perspective.

3.3 Subset: Nomenclature, Lexicon or Vocabulary

A *Nomenclature subset*, *Lexicon subset* or *Vocabulary subset* (Nomenclature for short) is intended to specify terminology. A specification of terminology implies names, synonyms, codes, abbreviations, translations, etc. that are used to denote something that is represented by a UID.

A Nomenclature subset represents a list of particular terms as 'names' of things and their unique identifier, together with the language in which the names are expressed and the language community in which the term for the thing originates.

A Nomenclature list typically includes names of concepts, but may also include names of individual things such as countries and other standard geographical objects. Organizations or projects will often maintain the nomenclature of individual things or collections of individual things. For example as represented in equipment lists, line lists, inventories, etc.

A Nomenclature subset includes contextual facts as well. For example the approval status and date-time values, sources, etc. A Nomenclature subset consists of the following expression components in the indicated sequence:
0, 69, 54, 71, 16, 2, 101, 1, 8, 67, 9, 10, 12 and 13. These expression components are given in Table 15.

Expression component ID (column ID)	Description
0	Presentation key
69	UID of natural language
54	Name of natural language
71	UID of language community
16	Name of language community
2	UID of left hand object

101	Name of left hand object
1	UID of an idea
8	Approval status
67	UID of succeeding idea
9	Date-Time of start of applicability
10	Date-time of latest change
12	Name of author of latest change
13	UID of creator of fact

Table 15, Expression components for a vocabulary

A collection of such expression components require a syntactical structure to define the relations between the components. For example, a tabular implementation implicitly defines as contextual fact a naming relation between the UID and a term (name of thing) in the vocabulary. This relation is of the type 'is called' (or 'is referenced as'). For example:

130206 is called pump.

Such a table also expresses a contextual fact that defines the language context in which the naming is done. This fact is of the type 'is presented in' (English).

The Nomenclature subset also allows defining the language community (sub-culture) where a name originates (component 71 and 16). For example, the name 'pump' may be declared to originate in the 'mechanical engineering' domain.

Misspellings and a pointer to the correct spelling can also be recorded in the nomenclature table. Misspellings can be indicated by a status (column 8) 'replaced' as well as an 'identifier of successor of the idea' (column 67), which refers to the idea UID that defines the correct spelling.

Preferred terms are terms which use is preferred in a particular language community. When an organization wants to specify its own list of preferred terms it might specify them within their own language community, even specifying terms that are identical to terms that are already specified for another language community.

When a Nomenclature (or Lexicon or Vocabulary) is represented in tabular form it can be represented in a Nomenclature subset of an Expression Table. Table 16 is an example of the main columns in a Nomenclature table.

54	**16**	**2**	**101**	**1**	**8**	**67**
Language	**Language community (discipline)**	**Gellish UID**	**Name of thing**	**UID of idea**	**Status**	**UID of successor of the idea**
English	mechanical technology	130206	pump	201	accepted	
Deutsch	Maschinenbau	130206	Pumpe	202	proposed	
Nederlands	werktuigbouwkunde	130206	pompe	203	replaced	204
Nederlands	werktuigbouwkunde	130206	pomp	204	accepted	

Table 16, Nomenclature subset example

Table 16 illustrates that the same concept, represented in the formal language by UID 130206 is denoted in English as 'pump' and in other languages by different terms, whereas the spelling 'pompe' in Dutch is a misspelling that should be replaced by 'pomp'.

Multi lingual vocabularies may add additional columns to the table as is described in par. 2.4.2.

3.4 Subset: Dictionary

A *Dictionary subset* is intended to provide textual definitions of things, especially of concepts, as an addition to the Nomenclature and Taxonomy subsets. This implies a relation between the thing and the text that defines the thing.

The following is an example of the core columns in a Dictionary subset of an Expression Table.

54	2	101	1	4	8
Language	**UID of defined thing**	**Name of thing**	**UID of fact**	**Textual definition**	**Status**
English	130206	pump	205	is a rotating equipment item intended to increase pressure in a liquid.	accepted
Nederlands	130206	pomp	206	is een apparaat met roterende delen dat bedoeld is om de druk in een vloeistof te verhogen.	accepted

Table 17, Dictionary subset core example.

A full *Dictionary subset* consists of a Vocabulary subset (Table 15) plus two additional components: the full definition and an option for adding remarks.

Expression component ID (column ID)	Description
4	Full definition (natural language text)
14	Remarks

Thus a Dictionary subset comprises the following components in the indicated sequence:
0, 69, 54, 71, 16, 2, 101, 1, **4**, **14**, 8, 67, 9, 10, 12 and 13.

Note 1: It is possible to record definitions for the same concept in multiple languages.

Note 2: *Definition models* are definitions that are expressed as collections of relations between concepts. Those relations require at least a Product Model subset.

Note 3: *Verbal (spoken) or pictorial definitions* require a relation to a sound or picture (or combination of them). However the textual definition (column 4) is meant for a string in ASCII or Unicode only. Therefore, such other definitions require at least a 'Product model' subset, as described below.

Multi lingual dictionaries may add additional columns to the table as is described in par. 2.4.2.

3.5 Subset: Taxonomy

A *Taxonomy subset* is a specialization hierarchy of concepts, also called a subtyping hierarchy (sometimes erroneously called a classification hierarchy). This implies that there are subtype-supertype relations between the concepts. A subtype concept is a specialization of a supertype concept. The inverse of that relation expresses the same fact in another way, namely that a supertype concept is a generalization of a subtype concept.

Table 18 illustrates the core columns in a Taxonomy table.

54	2	101	1	15	15	8
Language	**UID of left hand object**	**Name of left hand object**	**UID of fact**	**UID of right hand object**	**Name of right hand object**	**Status**
English	130206	pump	7	130227	rotating equipment item	accepted
Nederlands	130206	pomp	7	130227	apparaat met roterende delen	ignored duplicate

Table 18, Taxonomy subset example

A specialization relation implies that the subtype concept inherits all the aspects that are intrinsic to the supertype concept.

Note that the left hand object name and the right hand object name, as well as the language, are strictly speaking superfluous, but they are added to support the readability of the table. If they are ignored it becomes clear that the two lines in the above example define the same fact, which is the reason why the UIDs of the facts are identical and the status of the latter one is set at 'duplicate'.

A *Taxonomy subset* is an extension of a Dictionary subset by including expression components for the UIDs and names of supertype concepts.

Expression component ID (column ID)	Description
15	UID of right hand object
201	Name of right hand object

Thus a Taxonomy subset consists of the following expression components in the indicated sequence:

0, 69, 54, 71, 16, 2, 101, 1, **15**, **201**, 14, 8, 67, 9, 10, 12 and 13.

3.6 Subset: Product Model

A *Product Model subset* is intended for use in practice of data exchange to describe individual objects (including occurrences) during their lifecycle as well as knowledge about kinds of things.

A Product Model subset consists of the following expression components in the indicated sequence:

0, 69, 54, 71, 16, 2, **44**, 101, 1, **60, 3**, 15, **45**, 201, **65, 4, 30, 31, 66, 7**, 14, 8, 67, 9, 10, 12, 13, **50** and **68**.

The expression components are presented in Table 19.

Expression component ID (column ID)	Description
0	Presentation key
69	UID of natural language
54	Name of natural language
71	UID of language community
16	Name of language community
44	Left hand object cardinalities
2	UID of left hand object
101	Name of left hand object
1	UID of the idea
60	UID of relation type (kind of relation)
3	Name of relation type
45	Right hand object cardinalities
15	UID of right hand object
201	Name of right hand object
65	Partial definition
4	Full definition
30	UID of extent
31	Name of extent
66	UID of unit of measure
7	Name (symbol) of unit of measure (UoM)
14	Remarks
8	Approval status
67	UID of succeeding idea
9	Date-Time of start of applicability

10	Date-time of latest change
12	Name of author of latest change
13	UID of creator of fact
50	UID of collection of facts
68	Name of collection of facts

Table 19, Expression components of a Product Model subset

For definitions of the components and implied relations see par. 5.

3.7 Subset: Business Model

A *Business Model subset* is intended for use in practice of data exchange to describe propositions. This includes business communication about both designs (imaginary objects) as well as real world objects (observed individual objects) during their lifecycle and about enquiries, answers, orders, confirmations, etc. This subset is a superset (indicated in **bold**) of the Product Model subset, so it can also be used for storage and exchange of knowledge about kinds of things.

A Business Model subset is a subset that consists of the following expression components in the indicated sequence:
0, 69, 54, 71, 16, **39**, **5**, **43**, 44, 2, 101, **72**, **73**, **19**, **18**, 1, **42**, 60, 3, **85**, **74**, **75**, 45, 15, 201, **34, 35,** 65, 4, 30, 31, **32, 33,** 66, 7, **76, 77, 70**, **20**, 14, 8, **24**, 67, 9, **23, 22**, 10, **11, 83, 6**, 12, **78, 79,** 13, **53**, 50, 68.

The expression components in a Business Model are presented in Table 20.

Expression component ID (column ID)	**Description**
0	Presentation key
69	UID of natural language
54	Name of natural language
71	UID of language community
16	Name of language community
39	Reality
5	UID of intention
43	Name of intention
44	Left hand object cardinalities
2	UID of left hand object
101	Name of left hand object
72	UID of left hand kind of role
73	Name of left hand kind of role
19	UID of applicability context
18	Name of applicability context
1	UID of the idea
42	Description of the idea
60	UID of kind of relation (relation type)
3	Name (phrase) of kind of relation
85	Phrase type
74	UID of right hand kind of role
75	Name of right hand kind of role
45	Right hand object cardinalities
15	UID of right hand object
201	Name of right hand object
34	UID of exponent
35	Name of exponent
65	Partial definition

4	Full definition
30	UID of extent
31	Name of extent
32	UID of probability
33	Name of probability
66	UID of unit of measure
7	Name (symbol) of unit of measure (UoM)
76	UID of accuracy of quantification
77	Name of accuracy of quantification
70	UID of pick list
20	Name of pick list
14	Remarks
8	Approval status
24	Reason
67	UID of succeeding idea
9	Date-Time of start of applicability
23	Date-time of start of availability of expression
22	Date-Time of creation of this copy of expression
10	Date-time of latest change
11	UID of creator of idea
83	Name of creator of idea
6	UID of author of latest change
12	Name of author of latest change
78	UID of addressee of expression
79	Name of addressee of expression
13	References
53	UID of expression
50	UID of collection of ideas
68	Name of collection of ideas
82	Name of file in which expressions reside

Table 20, Expression components for a Business Model

The above-indicated sequences of expression components are defined as a handy sequence for human interpretation of a tabular content. There is no semantic meaning in that sequence, because the semantics of the relations between the components are defined explicitly in chapter 5.

3.8 Query subsets

A *Query subset* consists of one of the other subsets, extended with expression components for the specification of string commonality criteria.

In a tabular form a Query subset is a subset that is extended with the expression components 80. 81 and 84.

Expression component ID (column ID)	Description
80	Left hand string commonality
81	Right hand string commonality
84	Relation type string commonality

4. Implementation in the Universal Format (Syntax)

All semantic expressions, of any 'arity', can be expressed in various syntaxes. For example in RDF triples, also called graphs. However, for representing ideas, terms as well as contextual facts, such triples should be extended with an identifier that enables recognizing collections of triples. Such extended triples are usually called 'named graphs'. This can be done for example by using TRIX as is specified in ISO 15926-11.

A powerful and more direct and efficient implementation is the tabular Gellish Expression Format syntax. Such a table is suitable for describing any facts and ideas as well as queries about individual things or occurrences, requirements for things or knowledge about things in general.

Typically a statement or question about an individual thing is modeled by a relation that is classified by a kind of relation (a relation type) that is denoted by a phrase that starts with "is" or "has". A requirement phrase starts with "shall" and must specify a applicability context (in column 18). A statement that expresses knowledge about possibilities typically uses a relation type that is denoted by a phrase that starts with "can have" or "can be". This is illustrated in Figure 1.

101	18	1	3	45	201
Name of left hand object	**Applicability context for the idea**	**UID of idea**	**Name of relation type**	**Cardinalities**	**Name of right hand object**
I-1		101	is a part of		P-1
impeller	handover to operations	102	shall have as aspect a		diameter
centrifugal pump		103	can have as part a	1,n	pump impeller
impeller		104	has by definition as part a	2,n	vane

Figure 1, Example of Product data, a Requirement and Knowledge in one Expression Table

The example in Figure 1 illustrates four kinds of statements. The first one states that a particular impeller is a part of a particular pump. The second one states that information about any (model of an) impeller that is handed over to operations shall include a diameter. The third statement describes the general knowledge that any centrifugal pump can have (and at least has) one impeller. The minimum and maximum number of simultaneous instances (individual impellers for individual pumps) is indicated by the cardinalities. The last expression states that an impeller has by definition 2 or more vanes. Figure 1 demonstrates that all such kinds of statements can be expressed in the same table or in tables that have the same columns and have a single common definition.

4.1 Universal Databases and Messages – Expression Format

Messages that are exchanged between systems will each consist of a header and a body of one or more expressions of ideas (facts). In order to support readability for verification and human communication the expressions should also contain the names of the things that are referred to by the UIDs. This combination enables that different parties use their own terminology as synonyms, whereas a corresponding party can verify the terminology, while replacing it by his own terminology.

Universal Semantic Databases can import and export these integrated Expression Tables or their equivalents and can internally store the information in such tables or in an object oriented structure or in triples, all depending on the opinion of the database designer.

The core of an Expression Table consists of a combination of the content of a Core Table (Table 2) and Naming Dictionary table (Table 11). A partial combination is illustrated in Table 21.

54	43	1
Language	**Intention**	**UID of idea**
English	statement	201
English	statement	202

2	101	60	3	15	201	4
UID of left hand object	**Name of left hand object**	**UID of relation type**	**Name of relation type**	**UID of right hand object**	**Name of right hand object**	**Full definition**
101	P-1	1225	is classified as	102	cycle-pump	for office-1
102	cycle-pump	1146	is a specialization of	130206	pump	intended to inflate cycle tires.

Table 21, Example of an Expression Format table (selection of columns)

Databases can also consist of a combination of various Expression Format tables, each with the same structure, whereas such tables may be stored in a distributed way, thus forming distributed databases.

Expression Format tables allow that different naming conventions (synonyms) for the same objects are used in the same table or in a combination of Expression Format tables. This has the advantage that each organization can keep using its own terminology, provided that they use common UIDs while the dictionaries explicitly specify the synonyms.

When an organization does not allow for the use of synonyms, then an Expression Format table may be considered to include redundancy, as it then (re)specifies (uses) the names of things multiple times. For such database implementations it is possible to eliminate the redundancy by implementing only Naming Dictionary tables and Core Tables.

A data exchange message will contain one or more Expression Tables that are sent to another party as a file (a message, embedded in an 'envelope') in some chosen format. The format that may be chosen is not prescribed by the formal language, the only constraint is that the format represents the same meaning as the expressions in a tabular structure such as in Expression Format tables, without constraints on the characters in the cells, except for a tab, and that the receiving party possesses software to read such a format.

An Expression Format table is a neutral (software independent) tabular format that can be implemented in the structure of various proprietary or open file or database formats. For example, an Expression Format table for data exchange can be implemented as a neutral ASCII or Unicode text file (.txt) or may be implemented as a spreadsheet table (.xls), provided that single tabs separate the fields.

An Expression Format table can be composed of various subsets of expression components, each with its own application area and corresponding number of columns. Recommended subsets are defined in chapter 3.

4.2 Expression Format table - definition

4.2.1 The Expression Format table header definition

Each Expression Format table file has in principle a table header that defines table columns (with defined and implied relations between the columns) and a body that contains rows with fields that represent cells for values of the expression components. Each row represents an expression of an idea or fact and accompanying contextual facts as described in chapter 2.
An Expression Format table can consist either of a complete set of columns (according to a combination of a Business Model subset and a Query subset) or of one of the pre-defined subsets of expression components as defined in chapter 3.

Each column has a column ID and a column name (which are the same as the expression component ID and name). A value in a column field is a value for the expression component. Most expression components imply a contextual fact, which means that the value has implied relations with one or more values in other columns. *Those relations define the ideas about the objects!*

An Expression Format table body shall be preceded by header information, which consists of three collections of 'fields'. Each collection of fields may be implemented on a separate line (row) at the top of a table (such as in a spreadsheet table), or may be included in a database definition.

1. The first collection of fields consists of a sequence of fields A1 through An. The fields are name based. The fields shall contain the following content:

A1 = The string ‘Gellish’, which specifies that the table that follows conforms to the Gellish Expression Format. This includes the definition of its table header and column definitions and the subsequent expressions in the table rows.

A2 = The string ‘language’ (or ‘taal’ in Dutch) indicating the language in which the table is expressed. Note that each line may indicate a language that deviates from this language and indicates the language for the left hand term on that line. Thus e.g. ‘Name=English’ or ‘Nederlands’ or ‘international’ or any other name of a formalized natural language that is used for the terms (vocabulary) in the table as a whole.

A3 = The optional version specified by character string ‘Version=’ or the equivalent term in the natural language indicated in the specified language, followed by the version of the formal language defining ontology (dictionary) that is required for the interpretation of the expressions in the file. (especially the version of the upper ontology section).

A4 = The optional date of the release of the collection of expressions in this table (optional), indicated as ‘date=..’ with a string as date.

A5 = The optional category, specified by the string ‘category=...’ that characterizes the collection of expressions in the table. Standard categories are: Base ontology, Domain dictionary, Knowledge, Requirements, Product information, Process information, Product and process information, Query and model. Default is ‘category=model’.

A6 = An optional path, indicated by the string ‘path=’, which specifies a path to the location where the source of the table is located in a network (such as the Internet).
For example: http://example.gellish.net/Base_ontology.

A7 = An optional file name indicated by the string ‘file=...’, which specifies the name of the file or a description of the content of the table (optional).

A8= Prefix=prefix specifies a prefix code. Each file with Gellish expressions shall have a unique prefix code. The prefix code can be used for generating unique identifiers (UIDs) for new concepts and ideas in the current table by concatenating the code and a colon (:) and the next free sequence number in the range for that prefix for Obj_uid and Idea_uid respectively. For example the prefix ‘pre’ and a sequence number can be used by software for generating the following sequence of UIDs: pre:1, pre:2, pre:3, etc., whereas the software should first search for the current highest value in the range of the prefix.

A9 = An optional object id range indicated by the string ‘Obj_uid=n:m’ which is the specification of the numeric range within which (by default) new UIDs can be allocated to new objects within this set. New UIDs should not duplicate UIDs that appear already in the table. New object UIDs shall be preceded by a prefix and a colon, in case a prefix is specified. For example, Obj_uid=1:99.

A10= An optional idea id range indicated by the string ‘Idea_uid=n:m’ which is the specification of the numeric range within which (by default) new UIDs can be allocated to new ideas (facts) within this set. New idea UIDs shall be preceded by a prefix and a colon, in case a prefix is specified. For example, Idea_uid=100:199. The range for ideas shall not overlap with the range for objects.

A11= Optional references by means of IRIs, indicated by the string ‘Ref_iris=(name$_1$[,name$_i$])’. These references refer to zero or more Expression Format tables that are required to be used in combination with this table as a prerequisite for a proper interpretation.

2. The second collection of fields contains the sequence of expressions component ID’s (column ID’s), whereas each ID is a standard number that denotes a particular defined expressions component.
Note that the ID numbers are arbitrarily chosen. These ID’s allow the expressions components (table columns or parameters) to be presented in a different sequence without loss of meaning (the numbers in the table below correspond to those expressions component ID’s).

3. The third collection contains human readable text for every expressions component (column field) in the second collection, providing (short) names of the table columns. These names are free text. They are typically expressed in the natural language that is indicated in field A2.

If an Expression Format table is implemented in a spreadsheet, CSV or JSON file in ASCII or Unicode encoding, then the table starts with a header of three lines that represent the above three collections in the same sequence.

If an Expression Format is implemented in a parameter driven form, then the header consists of three header lines: header-1, header-2 and header-3, each with a set of parameter values conform the above three collections.

4.2.2 The Expression Format table columns

The lines in Expression Format tables are independent of each other and thus the lines may be sorted in any sequence, without loss of meaning (different sequences should be semantically identical). For example time dependency should be modeled explicitly and should not be inferred from line sequences.

Each line (row) in the body of an Expression Format table (which in a spreadsheet, ASCII or Unicode table starts on the fourth line) expresses an idea, which consists of the expression of a *core idea* and a number of *contextual facts*.

Unique identifiers (UIDs)

Several columns contain unique identifiers (UIDs). Each UID is a string of characters. Standard Gellish concepts are represented by strings that are represented by whole numbers**,** whereas only positive values are used as UIDs. For user defined concepts Gellish concepts may be represented by alpha-numeric UIDs, optionally preceded by a prefix of followed by a reserved postfix. The reserved prefix for numbers is # (without a colon) and the reserved postfix for percentages is %. Numbers are identified by a mantisse as a whole number, optionally followed by the character 'E', followed by an exponent indicating the number of decimals. For example the number 3.25 (English notation) has as UID: #325E-2. Note that the latter is a natural language independent Gellish notation. (e.g. that number in German is written with a comma, as 3,25). Furthermore 325E-2% should be interpreted as 3.25 %.
UIDs for roles of individual things in (individual) relations are composed of the UID of the role player, folled by a comma and a space, followed by the UID of the idea. For example, the role of John (UID=pers:1) as being father of Mary (UID=pers:2), in a father-daughter relation (UID=rel:1) will be UID=pers:1, rel:1), whereas the role of Mary in that relation will be UID=pers:2, rel:1).

String values in Unicode

All columns contain character *string values*. For database implementations it is indicated whether they have a fixed or variable length (*nvarchar* of *varchar*) or whether the string is externally stored (data types *ntext* and *text*). Basically all cells contain Unicode UTF-8 encoded values, enclosed by double quotes and separated by semicolons, although applications may accept other coding conventions.

Fields in columns that are indicated as optional may be left empty, in which case the indicated default value is applicable. Otherwise a field value is obligatory.

The data type, optionality and default value of each expression component (or table column in an Expression Format table) are specified in Table 22. Note: the Expression component numbers (Comp ids) correspond with the column IDs in an Expression Format table.

Comp id	Expression component name (name of table column)	Data type, Optionality, Default value
0	Presentation key (Sequence)	string (optional), Unicode, varchar(32), default null
69	UID of natural language (LanguageUID)	string (optional), Unicode, varchar(32), default null
54	Name of language of left hand object (Language)	string (optional), Unicode, nvarchar(255), default null
71	UID of language community (UID-7) (LHContextUID)	string (optional), Unicode, varchar(32), default null
16	Name of language community (LHContextName)	string (optional), Unicode, nvarchar(255), default null
39	Reality (LHReality)	string (optional), Unicode, nvarchar(255), default null
5	UID of an intention (IntentionUID)	string (optional), varchar(32), default '491285'

43	Name of intention (Intention)	string (optional), Unicode, nvarchar(255), default ‘statement’
44	Left hand object cardinalities (LHCardinalities)	string (optional), Unicode, varchar(32), default null
2	UID of left hand object (UID-2) (LHObjectUID)	string, Unicode, varchar(32)
101	Name of left hand object (LHObjectName)	string, Unicode, nvarchar(255), default = ‘nameless’
72	UID of left hand kind of role (LHRoleUID)	string (optional), Unicode, varchar(32), default null
73	Name of left hand kind of role (LHRoleName)	string (optional), Unicode, nvarchar(255), default null
19	UID of applicability context (AppContextUID)	string (optional), Unicode, varchar(32), default null
18	Name of applicability context (AppContextName)	string (optional), Unicode, nvarchar(255), default null
1	UID of core idea (UID-1) (IdeaUID)	string (optional), Unicode, varchar(32)
42	Description of core idea (template text) (IdeaDescription)	string (optional), Unicode, nvarchar(255), default null
85	Phrase type (PhraseType)string (optional), Unicode, varchar(32)	string (optional), Unicode, varchar(32)
60	UID of kind of relation (RelTypeUID)	string (optional), Unicode, varchar(32)
3	Name of kind of relation (RelTypeName)	string, Unicode, nvarchar(255)
74	UID of right hand kind of role (RHRoleUID)	string (optional), Unicode, varchar(32), default null
75	Name of right hand kind of role (RHRoleName)	string (optional), Unicode, nvarchar(255), default null
45	Right hand object cardinalities (RHCardinalities)	string (optional), non-Unicode, varchar(32), default null
15	UID of right hand object (UID-3) (RHObjectUID)	string (optional), Unicode, varchar(32)
201	Name of right hand object (RHObjectName)	string, Unicode, nvarchar(255), default = ‘nameless’
65	Partial definition (PartialDefinition)	string (optional), Unicode, ntext, default null
4	Full definition (FullDefinition)	string (optional), Unicode, ntext, default null
30	UID of extent (ExtentUID)	string (optional), Unicode, varchar(32)
31	Name of extent (ExtentName)	string, Unicode, nvarchar(255)
66	UID of Unit of measure (UoMUID)	string (optional), Unicode, varchar(32), default null
7	Name of Unit of measure (UoM) (UoMName)	string (optional), Unicode, nvarchar(32), default null
76	UID of accuracy of quantification (AccuracyUID)	string (optional), Unicode, varchar(32), default null
77	Name of accuracy of quantification (AccuracyName)	string (optional), Unicode, nvarchar(255), default null
70	UID of pick list (DomainUID)	string (optional), Unicode, varchar(32), default null
20	Name of pick list (DomainName)	string (optional), Unicode, nvarchar(255), default null
14	Remarks (Remarks)	string (optional), Unicode, ntext, default null
8	Approval status of core idea (ApprovalStatus)	string, non-Unicode, varchar(64)

67	UID of succeeding idea (SuccessorUID)	string (optional), Unicode, varchar(32), default null
24	Reason (Reason)	string (optional), Unicode, ntext, default null
9	Date-time of start of applicability (EffectiveFrom)	date-time, stored as a real value in the '1900 date system'[4]
21	Date-time of end of applicability (EffectiveUntil)	date-time, stored as a real value in the '1900 date system'[5]
13	UID of creator of idea (CreatorUID)	string (optional), Unicode, varchar(32), default null
10	Date-time of latest change (end of applicability) (LatestUpdate)	date-time, stored as a real value in the '1900 date system'
6	UID of author of latest change (AuthorUID)	string (optional), Unicode, varchar(32), default null
12	Name of author of latest change (Author)	string (optional), Unicode, nvarchar(64), default null
22	Date-time of creation of copy (CopyDate)	date-time, stored as a real value in the '1900 date system'
23	Date-time of start of availability of expression (AvailabilityDate)	date-time, stored as a real value in the '1900 date system'
78	UID of addressee of expression (AddresseeUID)	string (optional), Unicode, varchar(32), default null
79	Name of addressee of expression (AddresseeName)	string (optional), Unicode, nvarchar(64), default null
13	Reference (Reference)	string (optional), Unicode, nvarchar(255), default null
53	UID of expression (UID-5) (ExpressionUID)	string (optional), Unicode, varchar(32), default null
50	UID of collection of ideas (UID-4) (CollectionUID)	string (optional), Unicode, varchar(32), default null
68	Name of collection of ideas (CollectionName)	string (optional), Unicode, nvarchar(255), default null
80	Left hand string commonality (LHCommonality)	string (optional), non-Unicode, nvarchar(64), default null
81	Right hand string commonality (RHCommonality)	string (optional), non-Unicode, nvarchar(64), default null
82	File name	string (optional), Unicode, nvarchar(64), default null

Table 22, Expression components in Expression Format tables

Note: All values for UIDs are specified as integers values. However, they shall be implemented as character string values with as default an empty string (''). The standard UIDs in the Gellish Dictionary have string values that represent whole numbers conform the specified integers, but other UIDs may be alpha-numeric strings.

4See http://support.microsoft.com/kb/q180162/

5See http://support.microsoft.com/kb/q180162/

5. Definitions of expression components and implied relations

5.1 Natural language

A UID of a natural language (69) is the unique identifier of the natural language or 'International' (for language independent codes) in which the name of the left hand object (see column 101) and, if present, in which the definition (see column 63 and 4) is spelled. The language is a context for the origin of the referencing relation between the UID and the string that is the name of the left hand object and it implies that the name belongs to the vocabulary of the formal language variant for that language.

Note: the name of the right hand object may be any name or alias that is a defined name for the right hand object UID. The language of fields in other columns is determined by the language name in the first field in the first header line.

A name of a natural language (54) of the left hand object name indicates the name of the language for which a UID is given in column 69 and that is a context for the name of the left hand object (see column 101).
The allowed values for 'language name' and UID are the names and UIDs defined in the Formal Dictionary (or a private extension). Currently the dictionary contains names of natural languages and of (artificial) programming languages.

For example:

- natural language has as qualitative subtype English, French (francais), German (Deutsch), etc.
  The language 'International' shall be used to indicate strings that are natural language independent, such as codes.

Column 54 implies a fact that can be expressed as a naming relation between the UID of the language (69) and a name of that language (54).
For example:

910036	is called	English

If the columns for language UID and name are missing, then by default 910036 and English are assumed for all lines.

5.2 Language community

The language community UID (71) provides the uniqueness context within which the left hand object *name* (101) is a unique reference to the object id in column 2, in addition to the language context (see component 69 and 54).

The context is superfluous (and is for human clarification only) on all lines other than lines with a specialization, a qualification a classification or an alias relation and their subtypes, because only there the left hand objects, identified by their UID, are *defined* to have a name. If no context is given on a definition line, then the name for the left hand object is unique in the whole (natural) language (column 54) and no homonyms are then allowed (in the Dictionary).

A name of a language community (16) for the associated name of the left hand object is a name for the uniqueness context of which the identifier is given in column 71.

The name is optional (and is for human clarification only) because the context UID in column 71 shall be a reference to a context that is defined on another line, where its UID and name appears in columns 2 and 101 respectively.

This column represents a fact that can be expressed as a relation between the UID of the language community (71) and a name of the community in which a term originates (16).

Example: 1193707 is called engineering

If the columns for language community UID and name are missing, then by default 492015 and Formal English are assumed for all lines.

5.3 Core idea (fact)

A UID of a core idea or fact (1) is an identifier that represents a unique idea in the expression on the line and in the whole language. There can be not multiple UIDs that represent an idea, although there can be multiple expressions, such as in different languages. A core idea is of a kind as is indicated in column 60 and 3 'name of kind of relation'.

A 'description of a core idea' (42) is a character string that is a description of a core idea (1). If present, such a description is typically meant to be presented to users as a heading of one or more fields in the user interface of an application system. The text is intended as an aid for human interpretation of the meaning of the core idea in its context and may imply an instruction to a user for what should be filled in (typically for filling in a value for the left and/or right hand term in an expression) or what should be selected from a pick list in order to finalize a idea or group of ideas. The text might appear on a user interface (e.g. a fill-in-the-blanks form or data sheet) and supports human understanding of the meaning of the idea(s) and the intention of the object in column 15 and 201 and optionally the UoM in column 7.
For example: the text 'temperature of the fluid at inlet' suggests that a value and a unit of measure should be supplied.

Column 42 implies a fact that can be expressed by a relation between a UID of an idea (1) and its description (42). Such an idea is basically only used to provide a textual description when being specified for a user interface template.

Example: 201 is described as Length of the pipe

If column 42 is missing, then an empty field is assumed for all lines.

5.4 Applicability context

The UID of a applicability context (19) for a core idea identifies the context within which the uid of the idea, given in column 1, represents an applicable requirement (a valid idea). If not given, the requirement is applicable in all contexts.

This column represents a fact that can be expressed by a relation between a core idea (1) and a UID of a applicability context (19). The fact is basically only used to express in which context a requirement is applicable. Thus it is only applicable for 'shall be...' and 'shall have...' relations.

Example: 201 is applicable in the context of 202 (e.g. ISO 16739)

The applicability context name (18) provides a name of the context that is identified in column 19.

5.5 Intention

A UID of an intention (5) is the UID of which the name is specified in column 43.
An intention is the intention with which the expression of a core idea is communicated. It indicates the extent to which the core idea is the case according to the author of the proposition. An intention includes also a level of conviction about the truth of the idea. If a line expresses a proposition or communicative fact, then the intention qualifies the proposition. If a line expresses an opinion about a possible fact, then the intention indicates whether the expression of an idea is one of the following example intentions:

- *statement (491285)*
- *assertion (declaration of truth) (553991)*
- *denial (790595)*
- *question (790665)*
- *confirmation (790598)*
- *promise (492036)*
- *denial (790595)*
- *probability (551573)*
- *acceptance (declaration of being agreed) (553992)*

If the columns are missing, the default value = ‘statement’, which means a qualification of the expression: this “is the case” according to the opinion of the author of latest change (see below).

A name of an intention (43) represents a fact that can be expressed as a relation between the UID of an intention (5) and a name of an intention with which an expression is communicated (43).

Example:	491285	is called	statement

5.6 Reality

The reality (39) of left hand object is a classification of the left hand object, being either

- imaginary or
- real (= materialized)

This indicates that the object is either a product of a mind or an object whose existence is based in the physical world, either as natural or as artificial object.
If not specified, then the reality shall be interpreted from the context or from an explicit classification relation. Kinds of things (classes) are by definition imaginary. For example, a design activity of a pump will create an imaginary (although realistic) object; a fabrication process will create a real (observable) object. Note that an object cannot be imaginary and real. An installation relation or a materialization relation relates an imaginary object to a real object.

5.7 Left hand cardinalities

For relations between kinds of things this column contains the *simultaneous cardinalities for the left hand object kind of thing*. This means that it indicates the minimum and maximum number of individual things of the specified kind that can or may be related at the same time with an individual thing of the kind specified as the right hand.
The cardinalities may be specified by:

- A comma separated list of two integers that indicate the lower and upper limit cardinalities. The upper limit may be the character ‘n’ to indicate that the upper limit is unlimited.

The table column represents a fact that can be expressed as a relation between the UID of the core idea (1) and the left hand cardinalities (44).

Example:	201	has as left hand cardinalities	1, n

5.8 Left hand object

A UID of left hand object (2) is the identifier of the main object about which the line expresses an idea. That core idea is expressed as a relation between two objects mentioned in column 2 and 15. The external identifier (name) of the object in column 2 can be given in column 56 with its text attribute in column 101 ‘name of left hand object’.

A UID is an artificial sequence number, provided it is unique in a managed context. For example, the UID 4724 is a reference number of a telephone extension in the context of my company in The Hague. An identical number may refer to a different object in a different context, such as the extension with UID 4724 in the context of your company. The uniqueness context is given in column 16 (subject area). Such a context itself is defined on a separate line in an Expression Format table.

Note, that a fact represented by an association or relationship is also an object.

A name of a left hand object (101) is a string, which is a term such as a textual name, phrase, code, number, URI, etc. that denotes the object identified in column 2 and associated with it via an “is called” relation in a language context referred to in column 69 and a language community context referred to in column 71.

For example, a tag name or some other code or proper name or class name.

The string may also consist of a phrase consisting of multiple terms or a sequence of terms, each separated by a term separator, such as a comma or semicolon followed by a space. For example, a string may consist of a list of numeric values. It may also consist of a function name with a sequence of UIDs and terms as arguments in brackets.

The name is intended as a human reference or computer readable file name or address to denote the object represented by a UID in column 2. The name facilitates when the lines are sorted in a different sequence later. Normally the *name* has no UID (the object has one), but if the string would have a UID, then this name can be regarded an attribute of the encoded information identified in column 56.

Nameless objects are allowed, which implies that there is no instance in column 56 and the name in column 101 for the object in column 2 should be 'nameless'. Note, a nameless object (with a UID) can be uniquely referenced indirectly. For example it can be referenced by a combination of its kind and the assembly of which it is a part. For example, the impeller of P-1201 can be uniquely referenced as a nameless thing that is classified as an impeller and is a part of P-1201.

This column represents a fact that can be expressed as a naming relation between a UID of a left hand object (2) and a name of the left hand object (101).

Example: 301 is called P-1

5.8.1 Names and definitions in other languages

Names and textual definitions may be added in additional columns to the table as is described in par. 2.4.2. Such columns imply additional naming relations and defining relations similar to those of columns 101 and 65 (partial definition). The language community is assumed to be the same as the language community of the name in column 101, although in a different language.

5.9 Left hand kind of role

A UID of left hand kind of role (72) identifies the kind of role that classifies the role that is played by the left hand object in column 2. This kind of role is implicitly a subtype of the first or second kind of role that is required by the kind of relation in column 60.

A name of left hand kind of role (73) is the name of the kind of role in column 72.

This column represents a fact that can be expressed as a relation between the UID of a left hand role (72) and a name of the kind of role (73).

Example: 401 is called vessel assembly

5.10 Kind of relation (relation type)

A UID of a kind of relation (60) is unique ID for the kind that qualifies the idea in column 1, whereas a name of the kind of relation is given in a formal language in column 3.

A name (or phrase) of a kind of relation (3) is a name of one of a subtypes of relation or one of its base phrases or one of its inverse phrases. Allowed phrases are defined in the base ontology. Each phrase is defined in a language and language community. A language community determines the preferred phrases for that community, such as a preferred term in 'Formal English' (see column 71 and 16).

The base ontology consists of a collection of expressions that define among others the base phrases and inverse phrases for kinds of relations that are allowed in any expression in the formal language. The base ontology itself uses only 'bootstrapping' kinds of relations. Bootstrapping kinds of relations are the few kinds that need to be interpreted by software in order to enable interpreting the first imported base ontology file that defines the kinds of relations of the formal language. Those bootstrapping kinds of relations are in English:

1. 'is a kind of' or its synonym 'is a specialization of' (1146), which phrases specify subtype-supertype relations,
2. 'has by definition as first role a' (5944), which defines the kind of role of a first role player in a binary relation,
3. 'has by definition as second role a' (5945), which defines the kind of role of a second role player in a binary relation,
4. 'is a base phrase for' (6066), which specifies a base phrase for a kind of relation in an expression, which implies that the first role player appears at the left hand side of the phrase in the expression,

5. 'is an inverse phrase for' (1986), which specifies an inverse phrase for a kind of relation in an expression, which implies that the second role player appears at the left hand side of the phrase in the expression,
6. 'is a synonym of' (1981), which specifies a synonym name for a concept,
7. 'is by definition a role of a' (5343), which specifies for a (first or second) kind of role which kind of role player is allowed to play such a role in a relation of the applicable kind.

5.11 Phrase type UID

A phrase type UID (85) is a language independent UID of a kind of phrase, being either base phrase (UID 6066) or inverse phrase (UID 1986) that represents the word and phrase sequence in the expression. A phrase type UID with value 6066 indicates that a first role player UID and name is given in columns (2 and 101) and that a second role player is given in columns (15, 201), whereas a phrase type UID with value 1986 indicates the inverse. Thus, if a base phrase is used, then the left hand object (column 2, 101) shall be the object that is the player of the first role (role-1) according to the definition of the relation type. Then the right hand object (column 15, 201) shall be the player of the second role (role-2). When an inverse phrase is used, then the left hand object shall be the player of role-2 and the right hand object shall be the player of role-1. The idea that is expressed is independent of the phrase that is used, provided that the sequence of the objects is in line with the phrase.

Column 85 implies a fact that can be expressed as a relation between the UID of the relation type (60) and a phrase or name (3).

Example:	1225	has as base phrase (6066)	is classified as a

Note: the phrase types can be derived from the used phrases, but they can be included in data exchange files for enabling a natural language independent interpretation of the expressions.

5.12 Right hand kind of role

A UID of right hand kind of role (74) identifies the kind of role that classifies a role that is played by the right hand object in column 15. This kind of role is implicitly a subtype of the first or second kind of role that is required by the kind of relation in column 60.

A name of right hand kind of role (75) is the name of the kind of role in column 74. Typically the name of a kind of right hand role is a concatenation of the right hand name, the string " of a " and the left hand name. For example, in the expression "pump <has as part a> bearing", the right hand role can be called "bearing of a pump". Similarly, in the expression "pipe <has as aspect a> diameter", the right hand role name can be called "diameter of a pipe".

This column represents a fact that can be expressed as a relation between the UID of a right hand kind of role (74) and a name of the kind of role (75).

Example 1:	501	is called	vessel part
Example 2:	502	is called	member of A1

5.12.1 Right hand object

A UID of a right hand object (15) is the UID of the object that is related to the object in column 2. The name of this right hand object can (optionally) be given as right hand term in column 201. The name of an object that has a name is defined only on a line where the fact type indicates a referencing association to the object. On other lines a filled in name is only meant to support human readability.

For dates in column 15/201 the Gellish convention includes that the UIDs for dates between the year 1500 and 3000 are integer numbers that are concatenations of four digits for the year, two for the month and two for the day, whereas two zero's are used for the month when a whole year is meant and two zero's for the day when a whole month is meant. For example, January 2006 has UID 20060100.

For numbers the Gellish convention includes usage of the *formalized scientific notation* as is described in Appendix A of the book 'Semantic Information Modeling in Formalized Languages' [Ref. 1]. This convention implies that column 15 and 201 are used either for the whole number or for the integer significand only whereas in the latter case column 34 and 35 are optionally used for the corresponding exponent.

A name of a right hand object (201) is a character string, which is a term such as a textual name, phrase, code, number, URI, list of numbers separated by semicolons, etc. that denotes the object identified in column 15. It is associated with the object in column 2 that has a name in column 101.

For example, a tag name or some other code, numeric value, class name or a description that also is a name. A string that may appear in column 101 (left hand object name) may also appear in this column (201, right hand object name).

This column represents a fact that can be expressed as a naming relation between a UID of a right hand object (15) and a name of the right hand object (201). This fact in an expression requires a consistency check, because every (right hand) object shall appear (as a left hand object in a classification or specialization relation where is receives its name in the language and language community context (or is nameless).

Example: 302 is called Length of P-1

5.13 Right hand cardinalities

Right hand object cardinalities (45) for relations between concepts are a pair of values, separated by a comma, that specify the *simultaneous* cardinalities for the right hand object concept. This means that it specifies the minimum and maximum number of individual things of that kind that can or may be associated with an individual thing of the left hand object kind at the same time. The cardinalities may be specified in the same way as the cardinalities for the left hand object.

The column represents a fact that can be expressed as a relation between the UID of the core idea (1) and the right hand cardinalities (45).

Example: 201 has as right hand cardinalities 1, n

5.14 Partial definition

A partial definition (65) of the left hand object is a description in natural language that together with the relation type name (column 3) and the right hand object name (column 201) forms a full definition of the left hand object as presented in column 4. A partial definition is only useful as an intermediate to generate a full definition in column 4.

This column represents a fact that can be expressed by a relation between a left hand object (2) and its partial textual definition (65) that complements a classification or specialization relation. The fact is basically only used to provide a textual definition on a line that specifies a classification or a specialization relation.

Example: 101 is partially defined as that ...

5.15 Full definition

A full definition (4) of the left hand object is a textual description in natural language of the characteristics that define the left hand object or things of the kind specified by the left hand object concept. Typically this is a concatenation of three components: the term “is a” or “is an”, the right hand object name and the text in column 65 (partial definition), separated by spaces. A full definition is only applicable on lines where the left hand object is an individual thing that is classified or a kind of thing that is defined to be a subtype of another kind of thing (i.e. with relation types <is classified as a> or <is a specialization of> or one of their subtypes.

This column represents a fact that can be expressed by a relation between a left hand object (2) and its full textual definition (4). The fact is basically only used to provide a textual definition on a line that expresses a classification or a specialization of the left hand object.

Example: 101 is defined as a circular hollow profile that ...

5.16 Extent of being the case

A UID of an extent (30) specifies the extent to which the core idea (1) is the case. Typically in a relation between two individual things it specifies the fraction or percentage on a scale of a part in a composition relation with a whole. The scale is to be specified in the UoM column 7. For example expressed on a scale in weight percentage, %wt. Then the fraction (or percentage) is the fraction of the whole individual thing that forms the component individual thing. In a classification of a part relation it specifies a fraction or concentration of a component of a kind or a substance of a kind in a mixture for which a classification is the case. Then the fraction is the fraction of the whole that is classified by the kind. Note that the value may be denoted by a numeric UID, starting either with a # or a % character.

A 'name' of an extent (31) is a denotation for the extent value, typically being a decimal encoded number.

The UID column represents a fact that can be expressed as a relation between the UID of the core idea and the UID of the extent.

Example:	Sample-1 of seawater	has as part	Sample-1 of salt	3.0	wt%
	Sample-1 of seawater	is classified by substance as	water	97	wt%

The extent can also be an upper or lower limit value for an extent. In such cases a subtype of the relation type should be used to express how the value should be interpreted. For example the relation types <has with a minimum ratio as part> (6089) and <is classified by substance as at least> (6085) indicate that the extent value is a lower limit value.

5.17 Probability of being the case

The UID of a probability of being the case (32) represents a qualitative value or a quantitative value on a scale that indicates the probability that the core idea in column 1 is the case. The character string that denotes the probability value can be expressed in column 33. If a probability is quantified then the scale or unit of measure (66) classifies the quantification of the probability as a value on that scale, except when the core idea is about a quantification relation. In that latter case a quantitative probability shall be in percentage. For example, the probability may specify a % chance that the core idea is true. A denotation ('name') of a probability (33) is typically a string in decimal notation.

For example, the specification that some state, called State-1, is open is 97 %. This can be expressed as follows:

State-1 <is qualified as> open 97 %

5.18 Unit of measure (UoM)

A UID of a unit of measure (66) identifies the scale used for interpretation of the numeric value of a property in column 201. In case column 201 contains a concept of property name, the indicated UoM UID in column 66 indicates the default.

A name of a unit of measure (7) is a name of the scale used for interpretation of the numeric value of a property in column 201. In case column 201 contains a concept of property name, the indicated UoM in column 7 is a name of the default.

This column represents a fact that can be expressed as a relation between the UID of the scale (66) and a name or symbol (7).

Example: 570423 is called mm

5.19 Accuracy of a quantification

A UID of an accuracy of a quantification mapping (76) identifies a numeric range that is defined by two tolerances. The tolerances are defined relative to a common pivot value. The range defines the accuracy of the quantification mapping relation between a property and a numeric value on a scale, where the numeric value has a role as pivot value. For example, a range that is defined from –0.3 to +0.4 can be used to indicate the accuracy of a diameter. When the diameter maps on scale as equal to 5 mm with an accuracy that range around the pivot value (5), then this means that the diameter is between 4.7 mm and 5.4 mm.

The column represents a fact that can be expressed by a relation between a core idea (1) and a UID of an accuracy of mapping (76).

Example:	201	has as accuracy	590001

A name of an accuracy of quantification (77) is a name for the concept in column 76. For example, an accuracy of –0.3 and +0.5 around a pivot value.

This column represents a fact that can be expressed as a relation between the UID of an accuracy of quantification (76) and a name of a range that expresses the accuracy or tolerance (77).

Example:	590001	is called	-0.3, + 0.5

5.20 Picklist

The unique identifier for the collection of objects or pick list from which values for instances of the right hand term may be selected in the context of an instance of the left hand term. This holds within the applicability context (if specified).

Note, this column (together with column 20) is meant as a short-cut for subtyping a (right hand) aspect type in the context of the left hand object and adding an additional line which defines that the value for a subtype "shall be one of the" pick list collection of aspect values.
For example,

model X	shall have a color from the list of	model X colors

is a short cut for:

model X	shall have a	color of model X
color of model X	is a specialization of	color

and

color of model X	shall be one of the	model X colors

The column represents a fact that can be expressed by a relation between a core idea (1) and a UID of a domain (70) that is a collection of qualitative aspects from which aspect values may be selected. The fact is only used to denote a list of qualitative values.

Example:	201	has as pick list	590002

The name of a pick list or domain identified by the Picklist UID in column 70. The name of the pick list shall be unique in the same context as the context for the right hand term (column 201) as defined in column 16 on the line where the right hand term is defined and occurs as a left hand term.

This column represents a fact that can be expressed as a relation between the UID of a pick list (70) and a name of a pick list (20).

Example:	590002	is called	Model X colors

5.21 Remarks

A remarks (14) component is intended for comments related to the fact or the existence of the left hand object, its definition or status.
Formally this is qualitative information that is a qualitative subtype of 'remark'.

A remarks column represents a fact that can be expressed by a relation between a UID of a fact (1) and a Remark (14).

Example:	201	has as remark	To be checked with John

5.22 Approval status (Status)

An approval status (8) indicates the status of the expression of the core idea as determined by an applicable authority. The status of the other facts on a line can be derived from the status of the core idea. A status can be any of the qualitative subtypes of the concept 'approval status' in the Dictionary. Typically in English: proposed, issue, deleted, proposed to be deleted, ignored (ignore), agreed, accepted, accepted idea (= only the core idea is accepted, but not the definition), history, or replaced (see also the book 'Semantic Information Modeling Methodology' about the consequences of these statusses). The status 'replaced' indicates that the core idea is deleted and that a succeeding idea (see column 64) exists. The reason of a status may be clarified in the remarks column (see column 14). See also the 'reason' (24) below.

This column represents a fact that can be expressed by a relation between a UID of an idea (1) and an approval status (8).

Example: 201 has approval status accepted

5.23 Succeeding idea

A UID of a succeeding idea (67) is a UID of the idea by which this line, and especially the core idea which UID is given in column 1, is replaced when the status in column 8 is "replaced". It indicates that there exists a succession relation between the two ideas that points to the succeeding idea.

Note: If the kind of relation of a replaced idea is the last classification relation or specialization relation for the left hand object, then the life of the left hand object is terminated and replaced by the left hand object of the succeeding relation, when the UIDs are different.

This column represents a fact that can be expressed as a relation between a UID of an idea (1) and a UID of another idea (67). The fact specifies that the idea (1) is succeeded by the idea (67). When there is a succeeding UID mentioned, then the status of idea (1) should be 'replaced' and vice versa.

Example: 201 is succeeded by 301

5.24 Reason

A reason (24) describes the reason why an expression of a fact is created and changed and including the reason why the approval status became what it currently is. Typically a reason why a status became qualified as 'deleted' or 'replaced' or 'historic'.

A reason column represents a fact that can be expressed as a relation between a UID of a fact (1) and a reason for latest change of status (24).

Example: 202 is changed for reason duplicate of fact 201

5.25 Date-time of start of applicability

A date-time of start of applicability (9) is the moment at with the period of the applicability or validity of the core idea begins. It is implicitly associated with the core idea via a "is valid since" relation. The '1900 date system' enables very accurate timestamps, for example for the recording of moments of measurement
(See http://support.microsoft.com/kb/q180162/). If no date-time value is given, it is assumed that the core idea has been valid always.

This column represents a fact that can be expressed by a relation between a UID of an idea (1) and a number that indicates a date and time (9) registered according to the 1900 time system that specifies when the idea became valid. This is either the time of measurement, when the measured value is recorded or it specifies an (approximate) date and time that the idea was invented, created or proposed for the first time.

Example: The fact that The Eiffel Tower is located in Paris is fact that became applicable in the year 1887. This can be expressed as follows:

201 has as date of start of applicability 1887

5.26 Creator of idea (originator of idea)

A UID of a creator of an idea (11), specifies the person who invented, created or (first) proposed the idea. That person may differ from the one who created or changed the expression. The latter is the author of the latest change of the expression (6).

Column 11 implies that there exists a fact that can be expressed as a <has as originator> (6023) relation between a UID of an idea (1) and a UID of a person who is the originator (or creator or first proposer) of the idea or fact (11). The name of the originator can be recorded in column id (83).

Example: the fact that The Eiffel Tower is located in Paris is a fact that is created possibly by Gustave Eiffel. The expression in a particular database may be created by John Doe, indicated by UID 123457 (see author of latest change (6)).

5.27 Date-time of latest change

A date-time of latest change (10) specifies the timestamp of the latest change of the expression or of one of the contextual facts. If the status in column 8 is 'deleted', 'replaced' or 'history' and there is no data-time of end of validity (column 21) specified, then the date of latest change specifies the ***date-time of end of validity*** of the core idea. Then it is assumed to be related to the core idea by an 'is valid until' relation.

This column represents a fact that can be expressed by a relation between a UID of an expression (53) and a number that represents a date and time registered according to the 1900 time system that specifies when the expression of the fact was modified for the last time. When no change of the expression took place yet, this date is the same as the date-time of start of applicability.

Example:

222001	has as date of latest change	August 1, 2011

5.28 Date-time of end of validity

A date-time of end of validity (21) specifies the timestamp of the latest moment until which the core idea is valid or applicable. The timestamp is assumed to be related to the core idea by an 'is valid until' relation, also called an 'has as latest validity' relation.

This column represents a fact that can be expressed by a relation between a UID of an idea (1) and a number that represents a date and time registered according to the 1900 time system that specifies until when the idea is valid.

Example:

222001	has as latest validity	August 1, 2011

5.29 Author of latest change

A UID of the author of the latest change (6) is the party which name is given in column 12.

This fact is expressed by a relation between the UID of an expression (53) and the UID of a party (6) that is responsible for the changed version of the expression. When the expression is in its original unchanged state, then the UID of the author (party) shall be the same as the 'UID of creator of fact'. This author (party) is taken as the issuer (sender) of the message line, making the statement, or asking the question, or commanding the command, etc.

Example:	222001	is changed by	401

A name of the author of a latest change (12) is a name of the person who is the originator of the proposition or of the expression of the fact and who has at least some responsibility for the content of the line; especially its latest change. It is good practice to provide this information, although strictly speaking it is optional. It is recommended that the allowed names are limited to unique references to persons of which the UID and additional information is in the database.

This column represents a fact that can be expressed as a relation between the UID of a party (6) and a name of a party that is author of the latest change (12).

Example:	491286	is called	John Doe

5.30 Date-time of creation of copy

A date-time of creation of copy (22) is a creation date-time that specifies when this copy of the expression was created. It is distinguished from the date of start of applicability of fact. The latter specifies when a fact became the case, whereas this date-time specifies when this expression was recorded.

This column represents a fact that can be expressed as a relation between the UID of an expression (the line UID, 53) and a number that represents a data and time registered according to the 1900 time system, that specifies when this copy of the expression was created (22). It is distinguished from the date of start of applicability of fact. The latter specifies when a fact became the case, whereas this date-time specifies when this expression was recorded.

Example, the fact that The Eiffel Tower is located in Paris is a fact that is valid since 1889, its date of start of applicability of the fact. This fact may be recorded in a particular fact database e.g. at September 12, 2011, its date-time of creation of the (second) copy of the formal expression.

Example:

201	has as date of creation of copy	August 26, 2011

5.31 Date-time of start of availability of expression

This availability date-time (23) specifies when the expression was incorporated in a system for the first time. Other copies of the expression may be included in other data sets at later dates (see 22).

This fact is expressed as a relation between a UID of an expression (a Line UID, 53) and a number that represents a date-time (23) registered according to the 1900 time system that specifies when the expression of the fact was incorporated in a system for the first time. Other copies of the expression may be included in other data sets at later dates (see 22).

Example:

201	has as date of start of availability	August 26, 2011

5.32 Addressee of expression

A UID of an addressee of an expression (78) is the UID of the party that is the intended addressee for the expression. When the expression is not changed yet, the UID of the author (party) shall be the same as the 'UID of creator of fact'. This author (party) is taken as the issuer (sender) of the message line, making the statement, or asking the question, or commanding the command, etc.

This fact is expressed by a relation between the UID of an expression (53) and the UID of a party (78) that is the intended addressee for the expression. When the expression is not changed yet, the UID of the author (party) shall be the same as the 'UID of creator of fact'. This author (party) is taken as the issuer (sender) of the message line, making the statement, or asking the question, or commanding the command, etc.

Example:	222001	is addressed at	402

A name of the addressee of an expression (79) is the name of the party which UID is specified in column 78.

This column represents a fact that can be expressed as a relation between the UID of a party (78) and a name of a party that is the addressee of the expression (79).

Example:	491286	is called	John Doe

5.33 References

References (13) are one or more names of organizations, persons or positions in organizations or (parts of) documents that act as a source or point of reference for the core idea. If more than one reference is provided the references are separated by semicolons. It is good practice to provide this information, although strictly speaking it is optional. It may include URI strings. It is recommended that the allowed names are limited to unique references to parties or documents, preferrably of which the UID and additional information is contained in the database.

This column represents a fact that can be expressed by a relation between a UID of a fact (1) and a character string (13) that describes one or more references that form supporting evidence for a fact or its description or expression.

Example:	201	has as reference	ISO 1000

5.34 Complete expression

A UID of an expression (53) is the identifier for a single row in an Expression Format table. It indicates the collection of (contextual) facts (or 'cloud' of related things) in which the core idea and the contextual facts on one single line in an Expression Format table are included. It may be used to distinguish different expressions of the same idea (with the same idea UID (see column 1). For example, for distinguishing the same idea expressed in different languages.

This column represents a fact that can be expressed as a relation between a UID of an idea (1) and a local UID of an expression (53), which expressions including the expression of the accompanying contextual facts.

Example: 201 is expressed by 222001

5.35 Collection of ideas

A UID of a collection of ideas (50) is a unique identifier that represents a collection of ideas in which the idea as identified in column 1 is included. This column is intended to indicate a collection of which the elements are ideas that are identified by the above mentioned unique core idea identifiers (UID-1). A plural idea name is typically used as an *identifier* of a *model* or *(sub) template* or *view*.
When a plural idea identifier is filled-in, it implies the existence of an inclusion relation (<is an element of>) between the core idea on this line identified in column number 1 and the collection of ideas identified in column number 50. The name of the collection may be given in column 68. Collections may appear as left hand or right hand objects in core ideas, for example in the definition of larger collections.

This column represents a fact that can be expressed as a relation between a UID of an idea (1) and a UID of a collection of ideas (50) that specifies that the idea administratively belongs to the collection.

Example: 201 is an element of 601

A name of a collection of ideas (68) specifies a name of the collection of core ideas to which this idea belongs. This collection is identified by the UID in column 50. The idea in the collection might be managed together. The core idea on the line is an element of the collection. The collection may indicate for example an area of responsibility of a peer group, the content of a table or the statements on a data sheet.

This column represents a fact that can be expressed as a relation between the UID of a collection of ideas (50) and a name of a collection of ideas (68) of which the core idea is an element.

Example: 590003 is called Ideas about buildings

5.36 Left hand string commonality

A left hand string commonality (80) specifies for a query in which way a search string has or shall have commonality with a target string that is a left hand object name.

A string commonality may have one of the allowed values that are specified as qualifications of a string commonality in the Dictionary. The allowed string commonality values are defined in the Dictionary. They include:

- case sensitive identical
- case insensitive identical
- case sensitive partially identical
- case insensitive partially identical
- case sensitive front end identical
- case insensitive front end identical
- case sensitive not including
- case insensitive not including
- equal
- unequal
- less than or equal
- great than or equal

The default in a query is 'case sensitive partially identical'. The last four are mathematical comparisons. For example, 1.0 is equal to 1, whereas they are not case (in)sensitive identical.

This column represents a fact that can be expressed as a commonality relation between a name of a left hand object (101) (the search string) and the names of objects in Expression Format tables that are searched and that match the expression of the fact.

Example:

P	has commonality	case sensitive partially identical

5.37 Right hand string commonality

A right hand string commonality (81) specifies for a query in which way a search string has or shall have commonality with a target string that is a right hand object name. The allowed values are the same as for the left hand string commonalities.

This column represents a fact that can be expressed as a commonality relation between a name of a right hand object (201) (the search string) and the names of objects in expressions that are searched and that match the expression of the fact.

Example:

1	has commonality	case sensitive partially identical

5.38 Relation type string commonality

A relation type string commonality (84) specifies for a query in which way a search string has or shall have commonality with a target string that is a relation type name. The allowed values are the same as for the left hand string commonalities.

5.39 File name

A file name (82) specifies the name of the file from which the expression originates. A file name is recommended to include the date or date-time of the latest change of the file (or when not available the date-time of import of the file). For example, a CSV file called 'Model' with a latest modification date of 30 jan 2015 might be: Model 30jan2015.cvs.

Note: This component is only applicable for databases and not for data exchange message files.

5.40 Presentation key

A presentation key indicates a position or field in a presentation structure, such as a spreadsheet or a list of lines. It can support *sorting* the content of an Expression Format table. It has no contribution to the meaning of the facts represented on the line. The presentation key does not affect the meaning of the lines. This column can be arbitrarily filled-in for use in a specific context. Identical strings indicate that there is no preference in presentation sequence of the lines.

This fact is expressed as a relation between a UID of an expression (53) and a presentation sequence (0).

6. Implementation of formal languages

A Universal Semantic Database or Data Exchange Message consists of one or more expressions of core ideas and their accompanying contextual facts. Those facts shall be stored in Gellish Expression Format tables (e.g. in SQL) or in one of their equivalent formats, such as collections of RDF triples. Each of those formats shall contain at least the obligatory facts (table columns) that are defined in this document and the definitions of those facts shall be compliant with the definitions in this document.

A database in which the content of several files with Gellish Expression Format tables are combined into one Gellish Expression Format table data store shall be extended with an additional column in which the file name is recorded from which an expression (a line) originates. The database application shall manage the addition of new data and the consistency of the various data sets as well as the consistency of data stores in a family of data stores.

6.1 Unique keys

Typically in databases the uniqueness of expressions is managed by determining a 'unique key'. That raises the question: when are expressions considered to be duplicates and what makes an expression unique?
Typically each core idea (a possible fact plus an intention) is unique and thus has a unique idea UID. However, one and the same idea may be expressed in multiple languages. This means that in a multi-language Database it is allowed that the UID of an idea is not unique, but then the combination of language UID, language community UID and idea UID is unique. Within the same database this may be equivalent with a unique Expression UID (line UID, 53). For particular kinds of applications, such as discussion forums where several people issue proposals for concepts, ideas and their names and definitions, it may even be allowed that in one language multiple expressions about the same fact are allowed, provided that that status of the expression is 'accepted'. In such a database the author of latest change and the status shall be added to the unique key.

So, depending on the objective of a database the unique key may be:

1. Single language database: idea UID
2. Multi-language database: language UID, language community UID and idea UID
3. Development database: for status 'accepted' the same as for multi-language database. For non-accepted status: language UID, language community UID, idea UID and author of latest change.

6.2 Distributed Semantic Databases

A Distributed Semantic Database consists of several data stores, whereas each data store is a Semantic Database.

Different data stores shall use the same formal language definition and shall use the same core of (main and contextual) definitions. In addition to that, data stores may also use one or more of the optional contextual facts. Preferred collections of contextual facts are defined in chapter 3, Subsets, in which subsets of contextual facts are defined. When data stores are based on the same set of definition of kinds of contextual facts it enables that data from different data stores can be easily combined, merged or integrated, provided that the other conventions of the formal language are also adhered to. This also enables for example to combine the results of a query to various independent data stores, which then act as a distributed database.
This is illustrated in Figure 2.

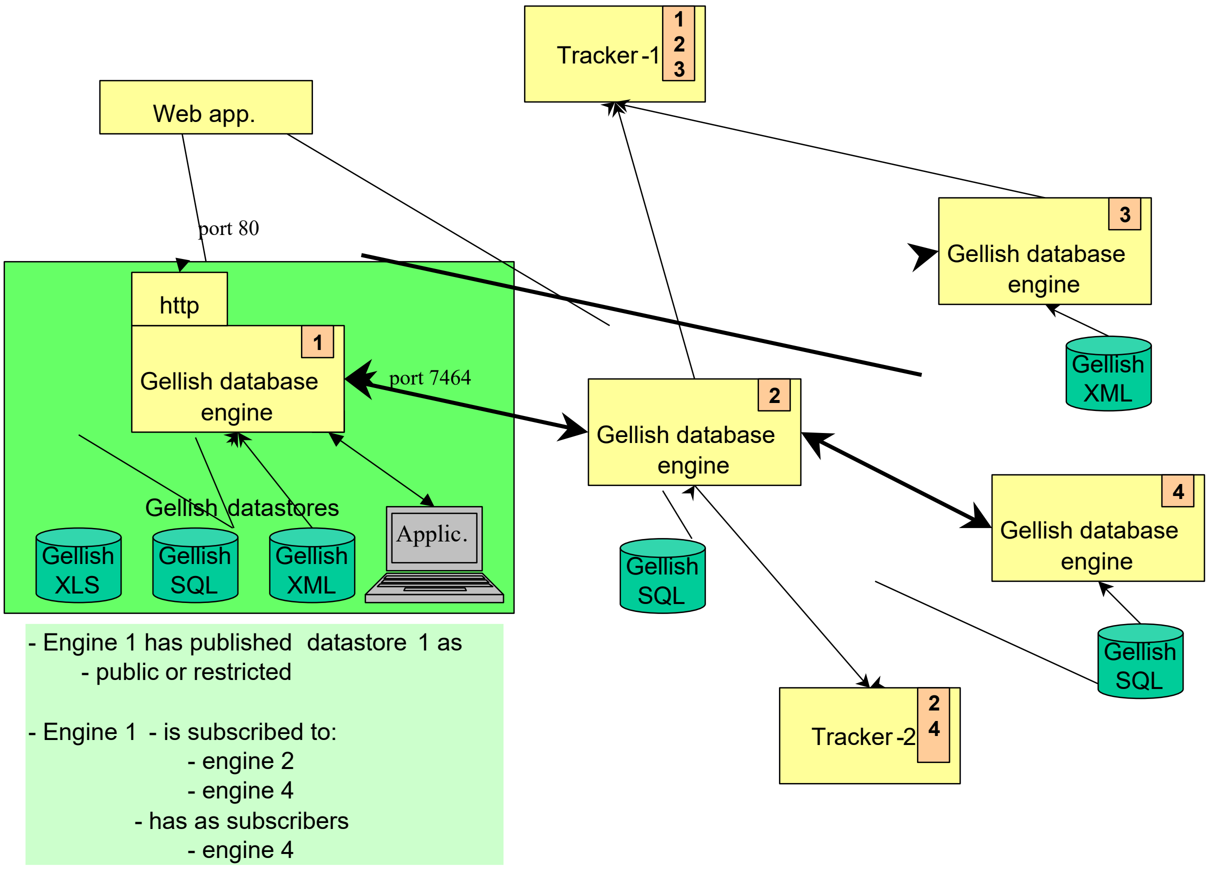

Figure 2, A Distributed Universal Semantic Database

The left hand of Figure 2 illustrates a database engine that manages three Data stores (in different formats) that can be read and updated via an Application Software system or via a web application (API). The engine can also communicate with other engines in the network to issues queries and to provide answers on queries from those other engines. Engines can be made known to engine trackers. An engine can publish a data store either as public or as restricted to subscribers. Other engines can subscribe to data stores. The engines exchange formal language messages, for example according to the SOAP protocol, which consists of a standard Header and a Message Body. The structure of the message bodies (queries as well as answers) shall include all required contextual facts that are defined in this document.

6.3 Formal languages in other syntaxes (formats)

A collection of formal expressions (expression of core ideas and their contextual facts) can be implemented in several standard, system independent formats:

1. The standard *Gellish Expression Format tables*, implemented as an object oriented database, in any SQL-based database tables (e.g. MySQL, MS-Access, Oracle or DB2), Unicode UTF-8 CSV, JSON or in a spreadsheet table format, such as an MS-Excel table format (XLS).
2. The *Gellish STEPfile format*, an ISO 10303-21 implementation format of a Gellish Expression Format.
3. The *Gellish XML format*, an XML implementation format of a Gellish Expression Format, defined in an XML Schema (see http://gellikx.com/2009/ns/2.0/GellishSchema/).
4. The *Gellish RDF* (e.g. as defined in *ISO 1526-11) possible in combination with TRIX (as defined in ISO 15926-11) or in RDF(S)/OWL.*

The above-mentioned formats are semantically equivalent. In other words, the meaning of all five ways of expressions of facts is identical and defined unambiguously by the semantics of the formal language.

This separation between form and content definition gives a freedom to choose their preferred or the most appropriate format in their context. It also enables computer software to interpret and process a message content automatically and unambiguously in whatever form it is.

The Dictionary itself is documented using a Gellish Expression Format table, as it can be downloaded in the form of Unicode UTF-8 CSV or JSON files. The integration of the various tables forms one large virtual Gellish Expression Format table (seen either as one or as a collection of tables), in which various subsets can be distinguished as collections of ideas.

6.4 Software and support

The Gellish Communicator software is Open Source demonstration software on GitHub from which various tools, such as mapper tools can be derived. Various suppliers may deliver formal languages enabled software. Such software should be able to process any data in a standard neutral format, as far as the software can store and retrieve such data. Such software forms a component in the Semantic Web. Such software can also be made to search for product data in a semantic database or to browse the Dictionary or knowledge that is expressed in a formal language. An example of such an application independent browser is the *Gellish Search Engine* (which can be acquired via www.gellish.net/), which supports the creation, import and export, validation and browsing of any data in a Gellish Expression Format table. Software can become 'certified Gellish enabled' by Gellish.net or one of its accredited partners.

7. Gellish Expression Format implementations

7.1 A Gellish Expression Format table

Gellish Expression Format tables are typically implemented as Unicode UTF-8 CSV or JSON Files. However they can be implemented directly in any tabular file format. For example they can be implemented in spreadsheets or SQL based database tables, or in XLS format of MS-Excel, in MDB format of MS-ACCESS or in an Oracle or DB2 database table. And as such it can be written, exchanged and read.

7.2 The Gellish STEPfile format

The Gellish STEPfile format is a way to express the content of a Gellish Expression Format in a form that is compliant with the STEP physical file standard (ISO 10303-21), also called a "part 21" file format. A file in this format is indicated by file extension '.G21'.
ISO 10303-21 requires that the entities that are instantiated in a STEP compliant file are defined in a data model, written in EXPRESS (ISO 10303-11). This is defined in the following paragraph.

7.2.1 Gellish Expression Format subset Product Model defined in EXPRESS

The *Gellish Expression Format subset Product Model* as defined in EXPRESS is presented in the 3rd column of Table 23.

		SCHEMA Gellish_Data_Table_subset_Product_Model;
		ENTITY gellish_fact;
0	Sequence	presentation_sequence_key: OPTIONAL string;
54	Language	language_name: string;
71	LHContextUID	context_UID_for_left_hand_object_name: OPTIONAL integer;
16	LHContextName	context_name_for_left_hand_name: OPTIONAL string;
2	LHObjectUID	left_hand_object_UID: integer;
44	LHCardinalities	left_hand_cardinalities: OPTIONAL LIST(2) of integer;
101	LHObjectName	left_hand_object_name: string;
1	FactUID	fact_UID: integer;
60	RelTypeUID	relation_type_UID: integer;
3	RelTypeName	relation_type_name: string;
15	RHObjectUID	right_hand_object_UID: integer;
45	RHCardinalities	right_hand_cardinalities: OPTIONAL LIST(2) of integer;
201	RHObjectName	right_hand_object_name: string;
65	PartialDefinition	definition: OPTIONAL string;
4	FullDefinition	full_definition: OPTIONAL string;
66	UoMUID	uom_UID: OPTIONAL integer;
7	UoMName	uom_name: OPTIONAL string;
14	Remarks	remarks: OPTIONAL string;
8	ApprovalStatus	status: string;
67	SuccessorUID	successor_of_fact_UID: OPTIONAL integer;
9	EffectiveFrom	date_of_creation: real;
10	LatestUpdate	date_of_latest_change: real;
12	Author	originator_of_change: OPTIONAL string;
13	Reference	source: OPTIONAL string;
50	CollectionUID	collection_of_facts_UID: OPTIONAL integer;
68	CollectionName	collection_of_facts_name: OPTIONAL string;
		UNIQUE
		ur1: fact_UID;
		ur2: left_hand_object_name, right_hand_object_name, relation_type_name;
		END_ENTITY;
		END_SCHEMA;

Table 23, The Gellish subset Product Model defined in EXPRESS

The first column in the figure refers to the column number in a Gellish Expression Format table.
The second column provides standard column names for database implementations.

A row in a Gellish Expression Format table corresponds directly with an instance of this “gelish_fact” entity.

The following example is an illustration of the body of a G21 file in ISO standard format for subset Product Model. The fact expresses that P-101 is classified as a centrifugal pump.

```
#1   gellish_fact(,‘english’,,’project A’,10000001,,,’P-101’,11000001,’is classified as’,
                  130058,,,’centrifugal pump’,,,,,,’accepted’,,20Feb2003,20Feb2003,’AvR’,’AvR’,)
```

When this is represented in a Gellish Expression Format table, not showing the empty columns and the last four columns, it becomes:

54	**16**	**2**	**101**	**1**	**3**	**15**	**201**	**8**
Language	**Language community for left hand object name**	**UID of left hand object**	**Name of left hand objec t**	**UID of fact**	**Name of relation type**	**UID of right hand object**	**Name of right hand object**	**Status of fact**
English	project A	101	P-101	201	is classified as a	130058	centrifugal pump	accepted

7.2.2 Gellish Expression Format table subset Business Model data model

A *Gellish subset Business Model* of a Expression Format table, as defined in EXPRESS, is presented in the third column of Table 24.

ID	Column name	Long name, optionality, type and default (EXPRESS schema)
		SCHEMA Gellish_Message_Table_Format_subset_Business_Model;
		ENTITY gellish_fact;
0	Sequence	presentation_sequence_key: OPTIONAL string; default null;
69	LanguageUID	language_UID: OPTIONAL integer; default null;
54	Language	language_name: OPTIONAL string; Unicode; nvarchar(255); default null;
71	CommunityUID	community_UID_for_community_name: OPTIONAL integer; default null;
16	CommunityName	community_name_for_LHObject_name: OPTIONAL string; default null;
39	LHReality	reality_of_left_hand_object: OPTIONAL string; default null;
2	LHObjectUID	left_hand_object_UID: integer;
44	LHCardinalities	left_hand_cardinalities: OPTIONAL LIST(2) of integer; default null;
101	LHObjectName	left_hand_object_name: string; default nameless;
72	LHRoleUID	left_hand_role_UID: OPTIONAL integer; default null;
73	LHRoleName	left_hand_role_name: OPTIONAL string; default null;
5	IntensionUID	intention_UID: OPTIONAL integer; default null;
43	Intention	intention: OPTIONAL string; default null;
19	AppContextUID	applicability_context_UID: integer; default null;
18	AppContextName	applicability_context_name: string; default null;
1	FactUID	fact_UID: integer;
42	FactDescription	description_of_main_fact: OPTIONAL string; default null;
60	RelTypeUID	relation_type_UID: integer;
3	RelTypeName	relation_type_name: string;
74	RHRoleUID	right_hand_role_UID: OPTIONAL integer; default null;
75	RHRoleName	right_hand_role_name: OPTIONAL string; default null;
15	RHObjectUID	right_hand_object_UID: integer;
45	RHCardinalities	right_hand_cardinalities: OPTIONAL LIST(2) of integer; default null;
201	RHObjectName	right_hand_object_name: string; default nameless;
65	PartialDefinition	definition: OPTIONAL string; default null;
4	FullDefinition	full_definition: OPTIONAL string; default null;
66	UoMUID	uom_UID: OPTIONAL integer; default null;
7	UoMName	uom_name: OPTIONAL string; default null;
76	AccuracyUID	accuracy_UID: OPTIONAL integer; default null;
77	AccuracyName	accuracy_name: OPTIONAL string; default null;
70	DomainUID	domain_UID: OPTIONAL integer; default null;
20	DomainName	domain_name: OPTIONAL string; default null;
14	Remarks	remarks: OPTIONAL string; default null;
8	ApprovalStatus	status: string;
24	Reason	reason: string; default null;
67	SuccessorUID	successor_of_fact_UID: OPTIONAL integer; default null;
9	EffectiveFrom	date_of_creation: real;
10	LatestUpdate	date_of_latest_change: real;
6	AuthorUID	originatorUID: OPTIONAL integer; default null;
12	AuthorName	originator_of_change: OPTIONAL string; default null;
78	AddresseeUID	addresseeUID: OPTIONAL integer; default null;
79	AddresseeName	addressee: OPTIONAL string; default null;
13	Reference	source: OPTIONAL string; default null;
53	LineUID	line_UID: OPTIONAL integer; default null;
50	CollectionUID	collection_of_facts_UID: OPTIONAL integer; default null;
68	CollectionName	collection_of_facts_name: OPTIONAL string; default null;

		UNIQUE
		ur1: fact_UID, language_UID, intention, originator_of_change;
		ur2: left_hand_object_UID, right_hand_object_UID, relation_type_UID, language_UID, intention, originator_of_change;

		END_ENTITY;
		END_SCHEMA;

Table 24, The Gellish subset Business Model defined in EXPRESS

The second column provides standard column names for database implementations. Yellow marked columns indicate that those columns do not appear in a Gellish Fact Table.

The above example expressed as a STEP Physical File, compliant with GTF subset Business Model and ISO 10303-21 is as follows:

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION((),'2;1');
FILE_NAME('gellish_table_format_subset_business_model','2003-05-02T23:18:26',('B.J.H. de
Boer'),('TLO Holland Controls b.v.'),'EXPRESS Data Manager version 20020107',$,$);
FILE_SCHEMA(('GELLISH_TABLE_FORMAT_SUBSET_BUSINESS_MODEL'));
ENDSEC;

DATA;
#1= GELLISH_FACT($,'english',$,'project A',$,10000001,$,'P-101',$,11000001,'is classified as',
$,130058,$,'centrifugal pump',$,$,$,$,$,'accepted',$,300000.,300000.,'AvR','AvR',$);
ENDSEC;

END-ISO-10303-21;
```

7.2.3 Subset Extended Model data model

The *Gellish Expression Format subset Extended Model* in EXPRESS is presented in Table 25.

		SCHEMA Gellish_Data_Table_subset_Extended_Model;
		ENTITY extended_gellish_fact;
0	Sequence	presentation_sequence_key: OPTIONAL string;
69	LanguageUID	language_UID: OPTIONAL integer;
54	Language	language_name: string;
71	LHContextUID	context_UID_for_left_hand_object_name: OPTIONAL integer;
16	LHContextName	context_name_for_left_hand_name: OPTIONAL string;
17	LHUniqueContext	uniqueness_context_left_UID: OPTIONAL string;
50	PluralFactUID	plural_fact_UID: OPTIONAL integer;
38	LHObjectType	left_hand_object_type: OPTIONAL string;
39	LHReality	reality_of_left_hand_object: OPTIONAL string;
2	LHObjectUID	left_hand_object_UID: integer;
56	LHTermUID	left_hand_term_UID: OPTIONAL integer;
44	LHCardinalities	left_hand_cardinalities: OPTIONAL LIST(2) of integer;
101	LHObjectName	left_hand_object_name: string;
43	Intention	intention: string;
19	AppContextUID	applicability_context_UID: integer;
18	AppContextName	applicability_context_name: string;
1	FactUID	fact_UID: integer;
60	RelTypeUID	relation_type_UID: OPTIONAL integer;
3	RelTypeName	relation_type_name: string;
52	RHUniqueContext	uniqueness_context_right_UID: OPTIONAL string;
15	RHObjectUID	right_hand_object_UID: integer;
45	RHCardinalities	right_hand_cardinalities: OPTIONAL LIST(2) of integer;
55	RHUnContextName	uniqueness_context_right_name: OPTIONAL string;
42	FactDescription	description_of_main_fact: OPTIONAL string;
201	RHObjectName	right_hand_object_name: string;
65	PartialDefinition	definition: OPTIONAL string;
4	FullDefinition	full_definition: OPTIONAL string;
66	UoMUID	uom_UID: OPTIONAL integer;
7	UoMName	uom_name: OPTIONAL string;
70	DomainUID	domain_UID: OPTIONAL integer;
20	DomainName	domain_name: OPTIONAL string;
14	Remarks	remarks: OPTIONAL string;
8	ApprovalStatus	status: string;

67	SuccessorUID	successor_of_fact_UID: OPTIONAL integer;
9	EffectiveFrom	date_of_creation: real;
10	LatestUpdate	date_of_latest_change: real;
12	Author	originator_of_change: OPTIONAL string;
13	Reference	source: OPTIONAL string;
53	LineUID	line_UID: OPTIONAL integer;
50	CollectionUID	collection_of_facts_UID: OPTIONAL integer;
68	CollectionName	collection_of_facts_name: OPTIONAL string;

		UNIQUE
		ur1: fact_UID, language_UID, intention;
		ur2: left_hand_object_name, right_hand_object_name, relation_type_name, language_UID, intention, originator_of_change, date_of_creation;

		END_ENTITY;
		END_SCHEMA;

Table 25, The Gellish subset Extended Model defined in EXPRESS

The second column provides standard column names for database implementations.

The second uniqueness rule expresses that a person can express a proposition more than once. If somebody expresses the same proposition twice (e.g. on different moments), then these expressions are considered to be different propositions (with different fact_UID).

7.3 The Gellish XML format (GXL)

An XML representation of a Gellish Expression Format may according to Ref [2], but may also be conform ISO 10303-28. The latter means that it is defined as a representation of a Gellish data model in EXPRESS as defined in the previous paragraph, although presented in XML, compliant with the conversion rules defined in ISO 10303 part 28.

An automated conversion procedure can converts a Gellish Expression Format implemented as an Excel spreadsheet table (XLS) into an XML file.

8. References

1 Andries van Renssen,: Semantic Information Modeling in Formalized Languages, Gellish.net, ISBN 978-1-304-51359-5, http://www.lulu.com/commerce/index.php?fBuyContent=14146523

2 http://www.gellish.net

3 http://gellikx.com/2009/ns/2.0/GellishSchema/

9. Index