INTERNATIONAL STANDARD

ISO 18629-11

First edition
2005-10-01

Industrial automation systems and integration — Process specification language —

Part 11: **PSL core**

Systèmes d'automatisation industrielle et intégration — Langage de spécification de procédé —

Partie 11: Noyau PSL

Reference number
ISO 18629-11:2005(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Figures

Table

Foreword

The International Organisation for Standardisation (ISO) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organisations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in ISO/IEC Directives, Part 2.

Draft International Standards (DIS) adopted by technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75% of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this part of ISO 18629 may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 18629-11 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration,* Sub-committee SC4, *Industrial data.*

A complete list of parts of ISO 18629 is available from the Internet.

http://www.tc184-sc4.org/titles

Introduction

ISO 18629 is an International Standard for the computer-interpretable exchange of information related to manufacturing processes. Taken together, all the parts contained in the ISO 18629 Standard provide a generic language for describing a manufacturing process throughout the entire production process within the same industrial company or across several industrial sectors or companies, independently from any particular representation model. The nature of this language makes it suitable for sharing process information related to manufacturing during all the stages of a production process.

This part provides a description of the core elements of the language defined within the International Standard.

This part of ISO 18629 and all other parts in ISO 18629 are independent of any specific process representation or model proposed in a software application in the domain of manufacturing management. Collectively, they provide a structural framework for improving the interoperability of these applications.

Industrial automation systems and integration -- Process specification language -- Part 11: PSL core

1 Scope

This part of ISO 18629 provides a representation of the concepts of PSL core using a set of axioms written in the basic language of ISO 18629.

The following is within the scope of this part of ISO 18629:

— the representation of the concepts common to all processes.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated basic only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 8824-1, *Information technology - Abstract Syntax Notation One (ASN.1) - Specification of basic notation*

ISO 10303-1, *Industrial automation systems and integration - Product data representation and exchange - Part 1: Overview and fundamental principles*

ISO 15531-1, *Industrial automation systems and integration - Industrial manufacturing management data - Part 1: General overview*

ISO 15531-42:2005, *Industrial automation systems and integration - Industrial manufacturing management data - Part 42:Time model*

3 Terms, definitions, and abbreviations

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1.1
axiom
well-formed formula in a formal language that provides constraints on the interpretation of symbols in the lexicon of a language

[ISO 18629-1]

3.1.2
conservative definition
definition that specifies necessary and sufficient conditions that a term shall satisfy and that does not allow new inferences to be drawn from the theory

[ISO 18629-1]

3.1.3
core theory
set of predicates, function symbols and individual constants, associated with some axioms, the primitive concepts of the ontology

3.1.4
data
a representation of information in a formal manner suitable for communication, interpretation, or processing by human beings or computers

[ISO 10303-1]

3.1.5
defined lexicon
set of symbols in the non-logical lexicon which denote defined concepts

NOTE Defined lexicon is divided into constant, function and relation symbols.

EXAMPLE terms with conservative definitions.

[ISO 18629-1]

3.1.6
extension
augmentation of PSL-Core containing additional axioms

NOTE 1 The PSL-Core is a relatively simple set of axioms that is adequate for expressing a wide range of basic processes. However, more complex processes require expressive resources that exceed those of the PSL-Core. Rather than clutter the PSL-Core itself with every conceivable concept that might prove useful in describing one

process or another, a variety of separate, modular extensions need to be developed and added to the PSL-Core as necessary. In this way a user can tailor the language precisely to suit his or her expressive needs.

NOTE 2 All extensions are core theories or definitional extensions.

[ISO 18629-1]

3.1.7
grammar
specification of how logical symbols and lexical terms can be combined to make well-formed formulae

[ISO 18629-1]

3.1.8
individual
element of an interpretation domain, in logic, considered as not divisible without loss of identity

EXAMPLE an individual constant is a symbol that is used to refer to some fixed individual object : it is the equivalent in logic of the "name" in the everyday language. In first-order logic, arguments of predicates are always individual constants.

NOTE 1 for further information, see [5].

NOTE 2 this term is commonly used in formal logic.

NOTE 3 in first order logic, the only individuals are individual constants.

3.1.9
information
facts, concepts, or instructions

[ISO 10303-1]

3.1.10
interpretation
universe of discourse and assignment of truth values (TRUE or FALSE) to all sentences in a theory

NOTE See annex C for an example of an interpretation.

3.1.11
language
combination of a lexicon and a grammar

[ISO 18629-1]

3.1.12
lexicon
set of symbols and terms

NOTE The lexicon consists of logical symbols (such as Boolean connectives and quantifiers) and non-logical symbols. For ISO 18629, the non logical part of the lexicon consists of expressions (constants, function symbols, and relation symbols) chosen to represent the basic concepts of the ontology.

[ISO 18629-1]

3.1.13
linear ordering
set of elements with a binary relation between any two elements that is transitive, irreflexive and antisymmetric

EXAMPLE the less-than relation in mathematics : $3 < 5$.

3.1.14
manufacturing
function or act of converting or transforming material from raw material or semi-finished state to a state of further completion

[ISO 15531-1]

3.1.15
manufacturing process

structured set of activities or operations performed upon material to convert it from the raw material or a semifinished state to a state of further completion

NOTE Manufacturing processes may be arranged in process layout, product layout, cellular layout or fixed position layout. Manufacturing processes may be planned to support make-to-stock, make-to-order, assemble-to-order, etc., based on strategic use and placements of inventories.

[ISO 15531-1]

3.1.16
model
combination of a set of elements and a truth assignment that satisfies all well-formed formulae in a theory

NOTE 1 The word "model" is used, in logic, in a way that differs from the way it is used in most scientific and everyday contexts: if a sentence is true in a certain interpretation, it is possible to say that the interpretation is a model of the sentence. The kind of semantics presented here is often called model-theoretical semantics.

NOTE 2 A model is typically represented as a set with some additional structure (partial ordering, lattice, or vector space). The model then defines meanings for the terminology and a notion of truth for sentences of the language in terms of this model. Given a model, the underlying set of axioms of the mathematical structures used in the set of axioms then becomes available as a basis for reasoning about the concepts intended by the terms of the language and their logical relationships, so that the set of models constitutes the formal semantics of the ontology.

[ISO 18629-1]

3.1.17
ontology
a lexicon of specialised terminology along with some specification of the meaning of terms in the lexicon

NOTE 1: structured set of related terms given with a specification of the meaning of the terms in a formal language. The specification of meaning explains why and how the terms are related and conditions how the set is partitioned and structured.

NOTE 2: The primary component of a process specification language such as ISO 18629 is an ontology The primitive concepts is the ontology according to ISO 18629 are adequate for describing basic manufacturing, engineering, and business processes.

NOTE 3: The focus of an ontology is not only on terms, but also on their meaning. An arbitrary set of terms is included in the ontology, but these terms can only be shared if there is an agreement about their meaning. It is the intended semantics of the terms that is being shared, not simply the terms.

NOTE 4: Any term used without an explicit definition is a possible source of ambiguity and confusion. The challenge for an ontology is that a framework is needed for making explicit the meaning of the terms within it. For the ISO 18629 ontology, it is necessary to provide a rigorous mathematical characterisation of process information as well as a precise expression of the basic logical properties of that information in the ISO 18629 language.

[ISO 18629-1]

3.1.18
point in time

location of something noticeable within a time domain

EXAMPLE 1 Wednesday, 15th of March, 2003.

EXAMPLE 2 9.30 a.m.

[ISO 15531-42]

3.1.19
primitive concept
lexical term that has no conservative definition

[ISO 18629-1]

3.1.20
primitive lexicon
set of symbols in the non-logical lexicon which denote primitive concepts

NOTE Primitive lexicon is divided into constant, function and relation symbols.

[ISO 18629-1]

3.1.21
process

structured set of activities involving various enterprise entities, that is designed and organised for a given purpose

NOTE The definition provided here is very close to that given in ISO 10303-49. Nevertheless ISO 15531 needs the notion of structured set of activities, without any predefined reference to the time or steps. In addition, from the point of view of flow management, some empty processes may be needed for a synchronisation purpose although they are not actually doing anything (ghost task).

[ISO 15531-1]

3.1.22
product
a thing or substance produced by a natural or artificial process

[ISO 10303-1]

3.1.23
proof theory
set of theories and lexical elements necessary for the interpretation of the semantics of the language

NOTE It consists of three components: the PSL-Core, the Outer Core and the extensions.

[ISO 18629-1]

3.1.24
PSL-Core
set of axioms for the concepts of activity, activity-occurrence, time-point, and object

NOTE The motivation for PSL-Core is any two process-related applications shall share these axioms in order to exchange process information, and hence is adequate for describing the fundamental concepts of manufacturing processes. Consequently, this characterisation of basic processes makes few assumptions about their nature beyond what is needed for describing those processes, and the PSL-Core is therefore rather weak in

terms of logical expressiveness. In particular, PSL-Core is not strong enough to provide definitions of the many auxiliary notions that become necessary to describe all intuitions about manufacturing processes.

[ISO 18629-1]

3.1.25
theory
set of axioms and definitions that pertain to a given concept or set of concepts

NOTE this definition reflects the approach of artificial intelligence in which a theory is the set of assumptions on which the meaning of the related concept is based.

[ISO 18629-1]

3.1.26
universe of discourse

the collection of concrete or abstract things that belong to an area of the real world, selected according to its interest for the system to be modelled and for its corresponding environment.

[ISO 15531-1]

3.2 Abbreviations

For the purpose of this part of ISO 18629, the following abbreviations apply:

— **BNF** Backus-Naur form

— **KIF** Knowledge Interchange Format

— **PSL** Process Specification Language

4 ISO 18629 general

ISO 18629 specifies a language for the representation of process information. It is composed of a lexicon, an ontology, and a grammar for process descriptions.

NOTE 1 PSL is a language for specifying manufacturing processes based on a mathematically well defined vocabulary and grammar. As such, it is different from the schemas and product representations provided in the standards ISO 10303, ISO 13584 [2], and ISO 15926 [3], it is also different from the representation provided by ISO 15531, but strongly related to it and complementary . In the context of an exchange of information between two software applications, PSL identifies each process independently of its behaviour. For example, an object identified as a resource within one process can be recognised as the same object even though it is identified as a product within a second process.

NOTE 2 PSL is based on first order logic; as such, it follows significantly different methods for specifying semantics for modelling concepts than those used in ISO 10303. The meaning of the concepts defined within PSL follows from sets of axioms and definitions provided within each extension of PSL-Core. A set of supporting notes and examples are provided within each part to aid the understanding of the language.

The parts 1x of ISO 18629 specify foundational theories needed to give precise definitions and the axiomatization of the primitive concepts of ISO 18629. The foundational theories enable precise semantic translations between different schemes.

The following are within the scope of the 1x series of parts of ISO 18629:

— the representation of the basic elements of the language;

— the provision of standardised sets of axioms that correspond to intuitive semantic primitive concepts adequate to describe basic processes;

— the set of rules to develop other foundational theories or extensions in compliance with PSL-Core.

The following is outside the scope of the 1x series of parts of ISO 18629:

— the representation of information involving concepts that are not part of foundational theories.

5 Language specification syntax

This clause specifies the definition of the ISO 18629 language using an extended Backus-Naur form (BNF) (see Annex D). The syntax described in this section is built on the syntax of the KIF format provided in the Annex B.

NOTE for further information, see [7 and [8].

5.1 Basic symbols and Syntactic Categories

The set of basic symbols that shall be used to specify any PSL process description are defined in the following BNF :

— <uc-letter> ::=A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

— <lc-letter> ::=a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

— <letter> ::= <uc-letter> | <lc-letter>

— <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

— <oper> ::= - | ~ | # |$ | * | + | /

— <punct> ::= _ | - | ~ | ! | @ | # |$ | % | ^ | & | * | (|) | + | = | ` | : | ; | ' | < | > | , | . | ? | / | | | [|] | { | } | <space>

An expression is any string of basic symbols.

There are five basic categories of expression :

— <b-con> ::= {<lc-letter> | <digit>} {<letter> | <digit>}* {{ _ | - } {<letter> | <digit>}}*

— <b-var> ::= ?<b-con>[']

— <b-func> ::= {<oper> | <uc-letter>} {<letter> | <digit>}* {{ _ | - } {<letter> | <digit>}}*

— <b-pred> ::= {<uc-letter>} {<letter> | <digit>}* {{ _ | - } {<letter> | <digit>}}*

— <doc-string> ::= " {<letter> | <digit> | <punct> | \" | \\ }* "

An expression derived from the nonterminal <b-con> is a string of alphanumeric characters, dashes, and underscores that begins with a lower case letter or digit and in which every dash and underscore is flanked on either side by a letter or digit. A <b-var> is the result of prefixing a <b-con> with a question mark and, optionally, appending a single quote (a "prime"). A <b-func> is similar to a <b-con> except that it must begin with either an <oper>, a <punc>, or an upper case letter, and a <b-pred> is similar to a <b-con> except that it must begin with an upper case letter. (Every <b-pred> is thus a <b-func>.) A <doc-string> is the result of quoting any string of symbols; double quotes and the backslash can be used as well as long as they are preceded by a backslash.

5.2 Lexicon

A first-order PSL language L is given in terms of a lexicon and a grammar. The lexicon provides the basic "words" of the language, and the grammar determines how the lexical elements may be used to build the complex, well-formed expressions of the language.

A PSL *lexicon* λ consists of the following:

— The expressions <space>, `(`, `)`, `not`, `and`, `or`, `implies`, `iff`, forall, and exists;

— A denumerable recursive set V_λ of <b-var> expressions known as the *variables* of λ;

— A recursive set C_λ of <b-con> expressions, known as the *constants* of λ which includes at least the strings inf-, inf+, max-, and max+;

— A recursive set F_λ of <b-func> expressions, known as the *function symbols* of λ, which includes at least the strings `beginof` and `endof`.

— A recursive set P_λ of <b-pred> expressions known as the *predicates* of λ, which includes at least the strings `=` activity, activity-occurrence, object, timepoint, is-occurring-at, occurrence-of, exists-at, before, and participates-in.

5.3 Grammar

Given an PSL lexicon λ, the grammar based on λ is given in the following BNF :

— <con> ::= a member of C_λ

— <var> ::= a member of V_λ

— <func> ::= a member of F_λ

— <pred> ::= a member of P_λ

— <term> ::= <atomterm> | <compterm>

— <atomterm> ::= <var> | <con>

— <compterm> ::=(<func> <term>)

— <sentence> ::= <atomsent> | <boolsent> | <quantsent>

— <atomsent> ::= (<pred> <term>+) | (= <term> <term>)

— <boolsent> ::=(not <sentence>) | (and <sentence> <sentence>+) | (or <sentence> <sentence>+) | (implies <sentence> <sentence>) | (iff <sentence> <sentence>)

— <quantsent> ::= ({forall | exists} {<var> | (<var>+)} <sentence>)

— <psl-sentence> ::= <sentence>

5.4 Language

The PSL language L_λ based on a lexicon λ is the set of expressions that can be derived from the nonterminal <psl-sentence> in the above grammar. The members of L_λ are the *sentences* of L_λ. User defined constraints shall be defined using the PSL language .

Note Example of user defined constraints are provided in annex C.

6 Basic elements of the PSL Core

This clause specifies the basic elements from which the ISO 18629-11 PSL-Core is composed.

6.1 Basic features

The basic elements of the language are :

— the primitive lexicon;

— the defined lexicon with supporting definitions;

— the axioms.

Since all these concepts provide the core elements of the language, no additional theory is needed.

The meaning of the following terms follows either from the axioms or from the supporting definitions.

6.2 Primitive lexicon of the PSL-Core

The primitive lexicon is made of the following :

— primitive categories;

— individuals;

— primitive relations;

— primitive functions.

NOTE In logic, primitive concepts are not given formal definitions within an ontology.

6.2.1 Primitive categories of the PSL-Core

The primitive categories are presented here with informal descriptions.

6.2.1.1 activity

The KIF notation for activity is :

(activity ?a)

The informal description for activity is :

(activity ?a) is TRUE in an interpretation of PSL-Core if and only if ?a is a member of the set of activities in the universe of discourse of the interpretation.

6.2.1.2 activity_occurrence

The KIF notation for activity_occurrence is :

(activity_occurrence ?occ)

The informal description for activity_occurrence is :

(activity_occurrence ?occ) is TRUE in an interpretation of PSL-Core if and only if ?occ is a member of the set of activity occurrences in the universe of discourse of the interpretation. An activity occurrence is associated with a unique activity and begins and ends at specific points in time.

NOTE An activity occurrence is not an instance of an activity. Although there may exist activities that have no activity occurrence, all activity occurrences must be associated with an activity

EXAMPLE: The activity denoted by the term (paint House1 Paintcan1) is an instance of the class of Painting activities:

(Painting (paint House1 Paintcan1))

There may be multiple distinct occurrences of this instance:

(occurrence_of Occ1 (paint House1 Paintcan1))

(occurrence_of Occ2 (paint House1 Paintcan1))

(= (beginof Occ1) 1100)

(= (endof Occ1) 1200)

(= (beginof Occ2) 1500)

(= (endof Occ2) 1800)

There may be another instance of the class of Painting activities

(Painting (paint House1 Paintcan2))

that has no occurrences.

6.2.1.3 object

The KIF notation for object is :

(object ?x)

The informal description for object is :

(object ?x) is TRUE in an interpretation of PSL-Core if and only if ?x is a member of the set of objects in the universe of discourse of the interpretation. An object is anything that is not a timepoint, nor an activity nor an activity-occurrence.

NOTE Intuitively, an object is a concrete or abstract thing that can participate in an activity. Objects can come into existence (be created) and go out of existence (be "used up" as a resource) at certain points in time. In such cases, an object has a begin and an end point. In some contexts it may be useful to consider some ordinary objects as having no such points either.

EXAMPLE 1 The most typical examples of objects are ordinary, tangible things, such as people, chairs, car bodies, NC-machines. Abstract objects, such as numbers are also objects.

EXAMPLE 2 Some objects, for instance numbers, do not have finite begin and end points.

6.2.1.4 timepoint

The KIF notation for timepoint is :

(timepoint ?t)

The informal description for timepoint is :

(timepoint ?t) is TRUE in an interpretation of PSL-Core if and only if ?t is a member of the set of timepoints in the universe of discourse of the interpretation.Timepoints form a linear ordering.

6.2.2 Individuals of the PSL-Core

6.2.2.1 inf-

The informal description for inf- is :

(= ?t inf-) is TRUE in an interpretation of PSL-Core if and only if ?t is the unique timepoint that is before all other timepoints in the linear ordering over timepoints in the universe of discourse of the interpretation.

NOTE inf- plays the role of negative infinity. It is needed to specify objects that have not been created within the universe of discourse of the interpretation.

6.2.2.2 inf+

The informal description for inf+ is :

(= ?t inf+) is TRUE in an interpretation of PSL-Core if and only if ?t is the unique timepoint that is after all other timepoints in the linear ordering over timepoints in the universe of discourse of the interpretation.

NOTE inf+ plays the role of positive infinity. It is needed to specify objects that have not been destroyed within the universe of discourse of the interpretation.

6.2.3 Primitive relations of the PSL-Core

6.2.3.1 before

The KIF notation for before is :

(before ?t1 ?t2)

The informal description for before is :

(before ?t1 ?t2) is TRUE in an interpretation of PSL-Core if and only if the timepoint ?t1 is earlier than ?t2 in the linear ordering over timepoints in the interpretation.

NOTE In PSL-Core, the set of timepoints is not dense (between any two distinct timepoints there is a third timepoint), although in PSL-Core the set of timepoints is infinite. Denseness can be added by a user as an additional postulate. Time intervals are not included among the primitives of PSL-Core; intervals can be defined with respect to timepoints and activities.

EXAMPLE the concept of intervals made of timepoints is a fundamental element of the parts ISO 15531-42 (Time model) and ISO IS 10303-41 (Product description and support).

6.2.3.2 occurrence_of

The KIF notation for occurrence_of is :

(occurrence_of ?occ ?a)

The informal description for occurrence_of is :

(occurrence_of ?occ ?a) is TRUE in an interpretation of PSL-Core if and only if ?occ is a particular occurrence of the activity ?a. occurrence_of is the basic relation between activities and activity occurrences. Every activity occurrence is associated with a unique activity. An activity may have no occurrences or multiple occurrences.

6.2.3.3 participates_in

The KIF notation for participates_in is :

(participates_in ?x ?occ ?t)

The informal description for participates_in is :

(participates_in ?x ?occ ?t) is TRUE in an interpretation of PSL-Core if and only if ?x plays some role that is not pre-specified in an occurrence of the activity occurrence ?occ at the timepoint ?t in the interpretation. An object can participate in an activity occurrence only at those timepoints at which both the object exists and the activity is occurring.

6.2.4 Primitive functions of the PSL-Core

6.2.4.1 beginof

The KIF notation for beginof is :

(beginof ?x)

The informal description for beginof is :

The domain of the beginof function is the set of activity occurrences and objects.

If ?x is an activity occurence in the universe of discourse of an interpretation of PSL-Core, then (beginof ?x) has the value t if and only if t is the timepoint at which the activity occurrence ?x begins.

EXAMPLE (= 10 (beginof milling_occurrence)) where milling_occurrence begins at timepoint 10.

If ?x is an object in the universe of discourse of an interpretation of PSL-Core, then (beginof ?x) has the value x if and only if t is the timepoint at which the object ?x becomes possible to participate in an activity.

EXAMPLE (= 15 (beginof screw)) where the object screw can participate in an activity at timepoint 15.

6.2.4.2 endof

The KIF notation for endof is :

(endof?x)

The informal description for endof is :

The domain of the endof function is the set of activity occurrences and objects.

If ?x is an activity occurrence in the universe of discourse of an interpretation of PSL-Core, then (endof?x) has the value x if and only if t is the timepoint at which the activity occurrence ?x ends.

EXAMPLE (= 25 (endof milling_occurrence)) where milling_occurrence ends at timepoint 25.

If ?x is an object in the universe of discourse of an interpretation of PSL-Core, then (endof?x) has the value x if and only if t is the timepoint at which the object ?x becomes no longer possible to participate in an activity.

EXAMPLE (= 30 (endof screw)) where screw can no longer participate in an activity at timepoint 30.

6.2.5 Theories required by the PSL-Core

No other theory is required for the primitive lexicon.

6.3 Defined lexicon of the PSL-Core

6.3.1 Formal definitions of the PSL-Core

NOTE The lexicon 'defrelation', 'exists', 'forall', 'and', 'or', 'not', '=', '⇔', and 'implies' are defined in the KIF Reference Manual [8]

6.3.1.1 between

Definition 1: Timepoint t2 is between timepoints t1 and t3 if and only if t1 is before t2 and t2 is before t3.

```
(forall (?t1 ?t2 ?t3) (iff (between ?t1 ?t2 ?t3)
    (and  (before ?t1 ?t2) (before ?t2 ?t3)))))
```

6.3.1.2 beforeEq

Definition 2: Timepoint t1 is beforeEq timepoint t2 if and only if t1 is before or equal to t2.

```
(forall (?t1 ?t2) (iff (beforeEq ?t1 ?t2)
    (and  (timepoint ?t1) (timepoint ?t2)
          (or (before ?t1 ?t2) (= ?t1 ?t2))))))
```

6.3.1.3 betweenEq

Definition 3: Timepoint t2 is between or equal to timepoints t1 and t3 if and only if t1 is before or equal to t2, and t2 is before or equal to t3.

```
(forall (?t1 ?t2 ?t3) (iff (betweenEq ?t1 ?t2 ?t3)
    (and  (beforeEq ?t1 ?t2)
          (beforeEq ?t2 ?t3)))))
```


6.3.1.4 exists_at

Definition 4: An object exists at a timepoint t1 if and only if t1 is between or equal to its begin and end points.

```
(forall (?x ?t) (iff (exists_at ?x ?t)
    (and   (object ?x)
           (betweenEq (beginof ?x) ?t (endof ?x))))))
```

6.3.1.5 is_occurring_at

Definition 5: An activity is occurring at a timepoint t1 if and only if t1 is between or equal to the activity occurrence's begin and end points.

```
(forall (?occ ?t) (iff (is-occuring-at ?occ ?t)
(and   (activity_ooccurrence ?occ)
       (betweenEq (beginof ?occ) ?t (endof ?occ)))))
```

6.4 Axioms

The following axioms (axiom 1 to axiom 17) are the fundamental axioms of the PSL-Core.

NOTE The lexicon 'defrelation', 'exists', 'forall', 'and', 'or', 'not', '=', 'iff', and 'implies' are defined in the KIF Reference Manual [8]

6.4.1 Axiom 1

The before relation only holds between timepoints.

```
(forall (?t1 ?t2)
    (implies    (before ?t1 ?t2)
                (and        (timepoint ?t1)
                            (timepoint ?t2))))
```

6.4.2 Axiom 2

The before relation is a total ordering.

```
(forall (?t1 ?t2)
    (implies    (and        (timepoint ?t1)
                            (timepoint ?t2))
```

```
(or             (= ?t1 ?t2)
                (before ?t1 ?t2)
                (before ?t2 ?t1))))
```

6.4.3 Axiom 3

The before relation is irreflexive.

```
(forall (?t)    (not (before ?t ?t)))
```

6.4.4 Axiom 4

The before relation is transitive.

```
(forall (?t1 ?t2 ?t3)
    (implies        (and    (before ?t1 ?t2)
                            (before ?t2 ?t3))
                    (before ?t1 ?t3)))
```

6.4.5 Axiom 5

The timepoint inf- is before all other timepoints.

```
(forall (?t)
(implies        (and    (timepoint ?t)      (not (= ?t inf-)))
    (before inf- ?t))
```

6.4.6 Axiom 6

Every other timepoint is before inf+.

```
(forall (?t)
(implies        (and    (timepoint ?t)      (not (= ?t inf+)))
    (before ?t inf+))
```

6.4.7 Axiom 7

Given any timepoint t other than inf-, there is a timepoint between inf- and t.

```
(forall (?t)
    (implies        (and            (timepoint ?t)
```



```
                    (not (= ?t inf-)))
          (exists   (?u)
                    (between inf- ?u ?t))))
```

6.4.8 Axiom 8

Given any timepoint t other than inf+, there is a timepoint between t and inf+.

```
(forall (?t)
    (implies    (and        (timepoint ?t)
                            (not (= ?t inf+)))
                (exists     (?u)
                            (between ?t ?u inf+))))
```

6.4.9 Axiom 9

Everything is either an activity, object, activity occurrence or timepoint.

```
(forall (?x)
    (or     (activity ?x)
            (activity_occurrence ?x)
            (object ?x)
            (timepoint ?x)))
```

6.4.10 Axiom 10

Objects, activities, activity occurrences, and timepoints are all distinct kinds of things.

```
(forall (?x)
(and  (implies      (activity ?x)
            (not (or (activity_occurrence ?x) (object ?x) (timepoint ?x))))
      (implies      (activity_occurrence ?x)
            (not (or (object ?x) (timepoint ?x))))
      (implies      (object ?x)
```



```
        (not (timepoint ?x))))
```

6.4.11 Axiom 11

The occurrence relation only holds between activities and activity occurrences.

```
(forall (?a ?occ)
    (implies    (occurrence_of ?occ ?a)
        (and        (activity ?a)
                    (activity_occurrence ?occ))))
```

6.4.12 Axiom 12

An activity occurrence is associated with a unique activity.

```
(forall (?occ ?a1 ?a2)
    (implies    (and        (occurrence_of ?occ ?a1)
                            (occurrence_of ?occ ?a2))
        (= ?a1 ?a2))))
```

6.4.13 Axiom 13

The begin and end of an activity occurrence or object are timepoints.

```
(forall (?x)
    (implies    (or     (activity_occurrence?x)    (object ?x))
                (and    (timepoint (beginof ?x))
                        (timepoint (endof ?x)))))
```

6.4.14 Axiom 14

The begin point of every activity occurrence or object is before or equal to its end point.

```
(forall (?x)
    (implies        (or     (activity_occurrence ?x )
                            (object ?x))
                    (beforeEq (beginof ?x) (endof ?x))))
```


NOTE The case in which the beginof and endof time points are equal supports software applications that represent instantaneous events.

6.4.15 Axiom 15

The participates_in relation only holds between objects, activity occurrences, and timepoints, respectively.

```
(forall (?x ?occ ?t)
    (implies    (participates_in ?x ?occ ?t)
        (and        (object ?x)
                    (activity_occurrence ?occ)
                    (timepoint ?t))))
```

6.4.16 Axiom 16

An object can participate in an activity occurrence only at those timepoints at which both the object exists and the activity is occurring.

```
(forall (?x ?occ ?t)
    (implies    (participates_in ?x ?occ ?t)
                (and  (exists_at ?x ?t)
                      (is_occurring_at ?occ ?t))))
```

6.4.17 Axiom 17

Every activity occurrence is an occurrence of an activity.

```
(forall (?occ)
    (implies    (activity_occurrence  ?occ)
                (exist  (?a)
                        (and  (activity  ?a)
                              (occurrence_of  ?occ  ?a)))))
```


7 Conformance to PSL-Core

7.1 Conformance of Ontologies

An ontology is in conformance with PSL-Core if and only if the concepts provided by the ontology are defined in a way that is consistent with the axioms and definitions of PSL-Core in Clause 6.

7.2 Conformance of Process Descriptions

A process description is in conformance with PSL-Core if and only if the ontology of the process description is in conformance with PSL-Core and the grammar of the process description satisfies Clause 5.

Annex A
(Normative)
ASN.1 Identifier of ISO 18629-11

To provide for unambiguous identification of an information object in an open system, the object identifier

```
iso standard 18629 part 11 version 1
```

is assigned to this part of ISO 18629. The meaning of this value is defined in ISO/IEC 8824-1 and is described in ISO 18628-1.

Annex B
(Informative)
KIF Syntax and Semantics

Like many computer-oriented languages, KIF has two varieties. In linear KIF, all expressions are strings of ASCII characters and, as such, are suitable for storage on serial devices (such as magnetic disks) and for transmission on serial media (such as phone lines). In structured KIF, the legal "expressions" of the language are structured objects. Structured KIF is of special use in communication between programs operating in the same address space [8].

Fortunately, there is a simple correspondence between the two varieties of KIF. For every character string, there is exactly one corresponding list structure; and, for every list structure, there is exactly one corresponding character string (once all unnecessary white space is eliminated).

In what follows, we first define the mapping between the linear and structured forms of the language; and, thereafter, we deal exclusively with the structured form.

B.1 Linear KIF

The alphabet of linear KIF consists of the 128 characters in the ASCII character set. Some of these characters have standard print representations; others do not. The characters with standard print representations (93 of the 128) are shown below.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

0 1 2 3 4 5 6 7 8 9 ( ) [ ] -- Ø ! ?

= + - * / ` ` & ð ~ ' ` " ù @ # $ % : ; , . ! ?

KIF originated in a Lisp application and inherits its syntax from Lisp. The relationship between linear KIF and structured KIF is most easily specified by appeal to the Common Lisp reader. In particular, a string of ASCII characters forms a legal expression in linear KIF if and only if (1) it is acceptable to the Common Lisp reader and (2) the structure produced by the Common Lisp reader is a legal expression of structured KIF (as defined in the next section).

B.2 Structured KIF

In structured KIF, the notion of *word* is taken as primitive. An expression is either a word or a finite sequence of expressions. In our treatment here, we use enclosing parentheses to bound the items in a composite expression.

<word> ::= a primitive syntactic object

<expression> ::= <word> | (<expression>*)

The set of all words is divided into the categories listed below. This categorization is disjoint and exhaustive. Every word is a member of one and only one category. (The categories defined here are used again in the grammatical rules of subsequent tables.)

<indvar> ::= a word beginning with the character ?

<seqvar> ::= a word beginning with the character @

<termop> ::= listof | setof | quote | if | cond | the | setofall | kappa | lambda

<sentop> ::= = | /= | not | and | or | implies | <= | iff | forall | exists

<ruleop> ::= =>> | <<= | consis

<defop> ::= defobject | defunction | defrelation | := | :implies | :&

<objconst> ::= a word denoting an object

<funconst> ::= *a word denoting a function*

<relconst> ::= *a word denoting a relation*

<logconst> ::= *a word denoting a truth value*

From these fundamental categories, we can build up more complex categories, viz. variables, operators, and constants.

<variable> ::= <indvar> | <seqvar>

<operator> ::= <termop> | <sentop> | <ruleop> | <defop>

<constant> ::= <objconst> | <funconst> | <relconst> | <logconst>

A *variable* is a word in which the first character is ? or @. A variable that begins with ? is called an *individual variable*. A variable that begins with an @ is called a *sequence variable*. Individual variables are used in quantifying over individual objects. Sequence variables are used in quantifying over sequences of objects.

Operators are used in forming complex expressions of various sorts. There are four types of operators in KIF -- *term operators, sentence operators, rule operators*, and *definition operators*. Term operators are used in forming complex terms. Sentence operators are used in forming complex sentences. Rule operators are using in forming rules. Definition operators are used in forming definitions.

A *constant* is any word that is neither a variable nor an operator. There are four categories of constants in KIF -- *object constants, function constants, relation constants, and logical constants*. *Object constants* are used to denote individual objects. *Function constants* denote functions on those objects.

Relation constants denote relations. *Logical constants* express conditions about the world and are either true or false.

Some constants are *basic* in that their type and meaning are fixed in the definition of KIF. All other constants are *non-basic* in that the language user gets to choose the type and the meaning. All numbers, characters, and strings are basic constants in KIF; the remaining basic constants are described in the remaining chapters of this document.

KIF is unusual among logical languages in that there is no way of determining the category of a non-basic constant (i.e. whether it is an object, function, relation, or logical constant) from its inherent properties (i.e. its spelling). The user selects the category of every non-basic constant for himself. The user need not declare that choice explicitly. However, the category of a constant determines how it can be used in forming expressions, and its category can be determined from this use. Consequently, once a constant is used in a particular way, its category becomes fixed.

There are four special types of expressions in the language -- *terms, sentences, rules, and definitions*. Terms are used to denote objects in the world being described; sentences are used to express facts about the world; rules are used to express legal steps of inference; and definitions are used to define constants; and forms are either sentences, rules, or definitions.

The set of legal terms in KIF is defined below. There are ten types of terms -- *individual variables, object constants, function constants, relation constants, functional terms, list terms, set terms, quotations, logical terms, and quantified terms*.

Individual variables, object constants, function constants, and relation constants were discussed earlier.

<term> ::= <indvar> | <objconst> | <funconst> | <relconst> | funterm> | <listterm> | <setterm> |

<quoterm> | <logterm> | <quanterm>

<listterm> ::= (listof <term>* [<seqvar>])

<setterm> ::= (setof <term>* [<seqvar>])

<funterm> ::= (<funconst> <term>* [<seqvar>])

<quoterm> ::= (quote <expression>)

<logterm> ::= (if <sentence> <term> [<term>]) | (cond (<sentence> <term>) ... (<sentence> <term>))

<quanterm> ::= (the <term> <sentence>) | (setofall <term> <sentence>) |

(kappa (<indvar>* [<seqvar>]) <sentence>*) | (lambda (<indvar>* [<seqvar>]) <term>)

A *functional term* consists of a function constant and an arbitrary number of *argument* terms, terminated by an optional sequence variable. Note that there is no syntactic restriction on the number of argument terms -- the same function constant can be applied to different numbers of arguments; arity restrictions in KIF are treated semantically.

A *list term* consists of the listof operator and a finite list of terms, terminated by an optional sequence variable.

A *set term* consists of the setof operator and a finite list of terms, terminated by an optional sequence variable.

Quotations involve the quote operator and an arbitrary list expression. The embedded expression can be an arbitrary list structure; it need *not* be a legal expression in KIF.

Remember that the Lisp reader converts strings of the form ‘σ into (quote σ).

Logical terms involve the `if` and `cond` operators. The `if` form allows for the testing of a single condition only, whereas the `cond` form allows for the testing of a sequence of conditions.

Quantified terms involve the operators the, setofall, kappa, and lambda. A *designator* consists of the the operator, a term, and a sentence. A *set-forming term* consist of the setof operator, a term, and a sentence. A *relation-forming term* consists of kappa, a list of variables, and a sentence. A *function-forming term* consists of lambda, a list of variables, and a term. Strictly speaking, we do not need kappa and lambda -- both can be defined in terms of setof; they are included in KIF for the sake of convenience.

The following BNF defines the set of legal sentences in KIF. There are six types of sentences. We have already mentioned logical constants.

<sentence> ::= <logconst> | <equation> | <inequality> | <relsent> | <logsent> | <quantsent>

<equation> ::= (= <term> <term>)

<inequality> ::= (/= <term> <term>)

<relsent> ::= (<relconst> <term>* [<seqvar>]) | (<funconst> <term>* <term>)

<logsent> ::= (not <sentence>) |

(and <sentence>*) |

(or <sentence>*) |

(implies <sentence>* <sentence>) |

(<= <sentence> <sentence>*) |

(iff <sentence> <sentence>)

<quantsent> ::= (forall <indvar> <sentence>) |

(forall (<indvar>*) <sentence>) |

(exists <indvar> <sentence>) |

(exists (<indvar>*) <sentence>)

An *equation* consists of the = operator and two terms.

An *inequality* consist of the /= operator and two terms.

A *relational* sentence consists of a relation constant and an arbitrary number of argument terms, terminated by an optional sequence variable. As with functional terms, there is no syntactic restriction on the number of argument terms in a relation sentence -- the same relation constant can be applied to any finite number of arguments.

The syntax of *logical sentences* depends on the logical operator involved. A sentence involving the not operator is called a *negation*. A sentence involving the `and` operator is called a *conjunction*, and the arguments are called *conjuncts*. A sentence involving the `or` operator is called a *disjunction*, and the arguments are called *disjuncts*. A sentence involving the implies operator is called an *implication*; all of its arguments but the last are called *antecedents*; and the last argument is called the consequent. A sentence involving the <= operator is called a reverse implication; its first argument is called the consequent; and the remaining arguments are called the antecedents. A sentence involving the iff operator is called an *equivalence*.

There are two types of *quantified sentences* -- a *universally quantified se*ntence is signalled by the use of the forall operator, and an *existentially quantified sentence* is signalled by the use of the exists operator.

B.3 Semantics

An interpretation for a First-Order KIF language is a tuple $I = \langle O, ext, \sigma, \tau \rangle$ specified as follows :

B.3.1 Extension Function

Within an interpretation I, the extension function *ext* has the following properties:

- $O = E \cup R$;
- $E \cap R = \varnothing$;
- If $s \in E$, then $ext(s) = \varnothing$;
- If $s \in R$, then $ext(s) \subseteq O^*$.

B.3.2 Semantic Value Function for Terms

Within an interpretation I, the semantic value function σ has the properties specified in the following clauses.

B.3.2.1 Semantic Value of Object Variables

The semantic value of an object variable v is the object assigned to that variable in the interpretation I :

$\sigma(v) \in O$

B.3.2.2 Semantic Value of Constants

The semantic value of a constant s is the object assigned to that constant in the interpretation I:

$\sigma(s) \in O$

B.3.2.3 Semantic Value of Function Terms

If $\sigma(F)$ is functional, then $\sigma((F\ t_1 \langle\ t_n))$ is the unique $e \in E$ such that $cc(\sigma(t_1), \langle\ \sigma(t_n), e) \in ext(\sigma(F))$ in the interpretation I;

otherwise, $\sigma((F\ t_1 \langle\ t_n)) = \sigma(F)$.

B.3.3 Satisfaction Function for Sentences

Within an interpretation I, the satisfaction function τ has the properties specified as follows :

B.3.3.1 Equations

An equation is true if and only if the terms in the equation refer to the same object in the universe of discourse of the interpretation I.

$$\tau((=\ t_1\ t_2)) = \begin{cases} true & \sigma(t_1) = \sigma(t_2) \\ false & otherwise \end{cases}$$

B.3.3.2 Atomic Sentences

An atomic sentence is true if and only if the tuple of objects formed from the values of the arguments is a member of the set of tuples in the extension of the relation denoted by the relation constant in the interpretation I.

$$\tau((R\ t_1 \langle\ t_n)) = \begin{cases} true & cc(\sigma(t_1), \langle, \sigma(t_n)) \in ext(\sigma(R)) \\ false & otherwise \end{cases}$$

NOTE: An atomic sentence of the form (R) will be true just in case the empty sequence $\langle\rangle \in ext(\sigma(R))$.

B.3.3.3 not

A negation is true if and only if the negated sentence is false in the interpretation I.

$$\tau((not\ \varphi)) = \begin{cases} true & \tau(\varphi) = false \\ false & otherwise \end{cases}$$

B.3.3.4 or

A disjunction is true if and only if at least one of the disjuncts is true in the interpretation I.

$$\tau((or\ \varphi_1 \lessdot \varphi_n)) = \begin{cases} true & \tau(\varphi_j) = true\ for some\ j,\ 0 \le j \le n \\ false & otherwise \end{cases}$$

NOTE: (or) is vacuously true under any interpretation.

B.3.3.5 and

A conjunction is true if and only if every conjunct is true in the interpretation I.

$$\tau((and\ \varphi_1 \lessdot \varphi_n)) = \begin{cases} true & \tau(\varphi_j) = true\ for all\ j,\ 0 \le j \le n \\ false & otherwise \end{cases}$$

NOTE: (and) is vacuously false under any interpretation.

B.3.3.6 implies

If every antecedent in an implication is true, then the implication as a whole is true if and only if the consequent is true in the interpretation I. If any of the antecedents is false, then the implication as a whole is true, regardless of the truth value of the consequent in the interpretation I.

$$\tau((implies\ \varphi_1 \lessdot \varphi_n\ \varphi)) = \begin{cases} true & for\ some\ j,\ \tau(\varphi_j) = false\ or\ \tau(\varphi) = true \\ false & otherwise \end{cases}$$

B.3.3.7 iff

An equivalence is true if and only if all arguments have the same truth value in the interpretation I.

$$\tau((iff\ \varphi_1\ \varphi_2)) = \begin{cases} true & \tau(\varphi_1) = \tau(\varphi_2) \\ false & otherwise \end{cases}$$

B.3.3.8 exists

An existentially quantified sentence is true if and only if the embedded sentence is true for some value of the variables mentioned in the first argument.

$$\tau((exists(v_1 \mathrm{K}\ v_n)\varphi)) = \begin{cases} true\tau(\varphi((v_1,...v_n))) = true under some I - \mathrm{var}iant \\ falseotherwise \end{cases}$$

B.3.3.9 forall

A universally quantified is true if and only if the embedded sentence is true for every value of the variables mentioned in the first argument.

$$\tau((\mathit{forall}\,(\nu_1 \lessdot \nu_2)\ \varphi)) = \begin{cases} \mathit{true} & \tau(\varphi((\nu_1,\ldots\nu_n))) = \mathit{true}\ \mathit{under}\ \mathit{every}\ I - \mathrm{var}\mathit{iant} \\ \mathit{false} & \mathit{otherwise} \end{cases}$$

Annex C
(Informative)
Example of process description using PSL-Core

The purpose of this annex is to provide a detailed scenario in which the ISO 18629 PSL is used in a information-sharing effort which involves multiple manufacturing functions.

This scenario is an "interoperability" manufacturing scenario. This means that its goal is to show how PSL can be used to facilitate the communication of process knowledge in a manufacturing environment. Specifically, this scenario is centered around the exchange of knowledge from a process planner to a job shop scheduler.

The following text illustrates the manufacturing context in which this exchange occurs[1] :

“A manufacturing firm makes a number of products. Associated with each product is a unique part number and one or more process plans. One process plan may be the preferred alternative, but the other alternatives are equally valid and yield acceptable parts. The process plan has a list of operations that must be performed in that sequence. The process plan has certain information about each operation: the operation name, the required resources, the planned run time, the planned setup time, the various parameters, etc.

At a given point in time, the firm has a set of customer orders. Each order has a due date and requests a certain number of some parts. Completing the order means completing every operation in some process plan for that product.

The production scheduling person must schedule the shop. That is, this person must create a schedule that states, for each customer order and each necessary operation, the work-center that should perform the operation, when the operation should begin, and when it should complete. The schedule is given to the manufacturing employees, who attempt to follow the schedule.”

This scenario is concerned in particular with the exchange of the process plan knowledge described above. It is assumed that each manufacturing function has its own “native” representation which is suited to the particular tools which are specialised for that function (i.e. a process planner uses a process planning tool and a scheduler uses a scheduling tool). Obviously, interoperability is a trivial act if the co-participants natively share a common language/representation. The “interesting” aspect of this communication is the translation involved.

The following interoperablity steps are proposed to facilitate this communication:

a) a process plan representation is translated into PSL and

b) this PSL representation is translated into a scheduling representation.

[1] This text is adapted from a scenario description submitted to the PSL scenario group by Jeffrey Herrmann, University of Maryland, College Park.

Alternatively, a translator could be written directly between the two (source and target) representations. This has the disadvantage of increasing the number of translators needed from O(n) to $O(n^2)$ (where n = number of translators) when translation with more representations are required (e.g. between the scheduler and floor control system).

C.1 Source and Target Representations

The representations used for this example are IDEF3 [9] and a scheduler from the commerce for the process planning and scheduling functions, respectively.

— IDEF3 was created specifically to capture descriptions of sequences of activities and has become a well-known representation for process knowledge.

— This scheduler is an add-on to a commercial solver library which is designed for finite capacity scheduling applications.

C.2 The Camile Motor Works Factory

The product, process and factory knowledge presented in this scenario has been taken from a planning and control reference case study. The aim is to provide a detailed account of a fictitious factory, the Camile Motor Works (CMW). Each characteristic and concept documented for this factory is based on factors which have been observed to be critical for decision making (predictive or reactive) in one or more “real” factories.

This case study covers normal manufacturing planning and control activities from the time of work order acceptance, including due date negotiation, to the time when final products are stored or shipped. CMW is described as a factory which produces a line of scale model automobiles which are constructed using :

— a number of purchased parts used during assembly

— purchased parts which have additional internal processing (assembling, painting, etc.)

— internally fabricated components transformed from raw materials (various bodies, name plates, gear shifts, frames, engine block, spark plug wires, tires, rims, etc.).

CMW has a wide variety of manufacturing processes at the factory and can be considered a hybrid environment.

There are eight major departments comprised of 43 primary resources. The factory employs approximately 50 direct labour personnel which operate the plant and maintain the resources.

C.3 The Product: GT-350

While three products are presented in the source factory description (GT-200, GT-250, GT-350), it is sufficient for this interoperability scenario to restrict the scope to only address the most complex of the three. The GT-350 is described as a sophisticated running scale model automobile that is marketed via

direct mail to company executives with optional license plates (various types to choose from). The product can be further personalised with the buyer's choice of 2 to 6 characters placed on an ID plate. The model comes in red or white (batteries not included). This high priced item is not stocked and is built only when firm orders are given. Delivery times of 4 to 6 weeks are indicated to buyers.

The product structure for the GT-350 is shown in table C.1. The indenting of names is used to indicate the part sub-component structure. An unique part number is assigned to each component. The method of manufacturing for each component is listed in the right-most column. Some parts are manufactured internally or are assembled from other parts, while others are purchased or sub-contracted externally.

GT-350 Product Structure			
Part Name	**Quantity**	**Part Number**	**Method**
Chassis	1	350–CHASSIS	assembled
Doors	2	x50–DOORS	purchased
Unibody	1	x50–BODY	internal
Frame	1	350–FRAME	internal
Engine	1	350-ENGINE	assembled
Block	1	350–BLOCK	internal
Harness	1	350–HARNESS	internal
Wires	1	350–WIRE-SET	internal
partial wire	12	350–WIRE	internal
partial wire	1	x50–WIRE	internal
Drive	1	350–DRIVE	assembled
Motor	1	350–MOTOR	Purchased
Electronics	1	350–PCB	internal
Interior	1	350–INTERIOR	assembled
Cockpit	1	350–COCKPIT	sub–contract
dash	1	350–DASH	purchased
lights	2	350–LIGHTS	purchased
Seats	1	350–SEATS	Purchased
gear shift	1	x50–GEAR	internal
Trim	1	350–TRIM	assembled
Options	1	350–OPTIONS	assembled
license plates	2	350–LICENSE	internal
Wheels	4	350–WHEELS	Assembled
tires	1	350–TIRES	internal
rims	1	350–RIMS	internal
Decals	1	350–DECALS	purchased
id–plate	1	350–PLATE	internal
Stand	1	350–STAND	internal

Table C.1: GT-350 Product Structure Table

C.4 The Departments

The departments of the CMW Factory are the following (for further details, see [9]) :