WARWICK
THE UNIVERSITY OF WARWICK

A Thesis Submitted for the Degree of PhD at the University of Warwick

Permanent WRAP URL:
http://wrap.warwick.ac.uk/108798/

Copyright and reuse:
This thesis is made available online and is protected by original copyright.
Please scroll down to view the document itself.
Please refer to the repository record for this item for information to help you to cite it.
Our policy information is available from the repository home page.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk

warwick.ac.uk/lib-publications

An Object Oriented approach to Automating the Product Specification Concept in the Automotive Industry.

by

Panayiotis S. Markomichalis

Submitted in fulfilment of the degree of Ph.D.

Department of Engineering

University of Warwick

February 1991

VOLUME 2

TABLE OF CONTENTS

VOLUME 2

APPENDIX 1: KNOWLEDGE AND OBJECT ORIENTED PROGRAMMING.

APPENDIX 2: STATICE.

APPENDIX 3: LISTING OF THE VAX TAPE DATA AND THE PARSING PROGRAM.

APPENDIX 4: LISTING OF THE VALIDATION ALGORITHM.

APPENDIX 5: LISTING OF THE STATISTICAL ANALYSIS PROGRAM.

APPENDIX 6: THE STATICE DATABASES DEFINITION.

APPENDIX 7: LISTING OF THE TRUTH MAINTENANCE MECHANISM.

APPENDIX 8: LISTING OF THE INTELLIGENT NETWORKS.

APPENDIX 9: SAMPLE OF THE INFORMATION CREATED DURING THE RUN OF THE VALIDATION ALGORITHM.

APPENDIX 10: SAMPLE RUN OF **ROOVESP**.

APPENDIX 11: INSTRUCTIONS TO RUN **ROOVESP**.

APPENDIX 1

Knowledge and Object Oriented Programming

Introduction

In Information Technology a distinction usually exists between "data" and "information". Data consists of raw figures or files which do not necessarily answer the questions posed from their user. Information, on the other hand, is considered to be the refinement of such crude data ie. their statistical analysis, interelationship or graphical tables, etc. which may answer more easily the queries arize with their use.

In A.I., as well, a third level is identified: the "knowledge" []. This is the ultimate organisation of information and much of it is shared by most people as it relates to everyday processes (ie. how to turn on the television or eat a meal). Other knowledge is more specialized and known only by few experts, such as how to find the result of an integral equation or repair a car.

Regardless, however, of its nature, knowledge in its general form can be represented as a collection of objects, relationships between these objects and facts. These elements when combined with heuristic rules generate inference procedures. In this perspective, the overall mechanism of knowledge can be viewed as

the ultimate manipulation of static information (facts, relationships) from procedural courses (rules and control structures). Similarly, computer systems, when programed with such a paradigm could be viewed to behave in a manner equivalent to the humans *knowledgeble* way and the degree of their *intelligence* might be measured by their efficiency to manipulate the domain's data. The latter refers to the broader AI issue of representing domain knowledge. That is, the organisation of the information inside the memory of the computer in a such way that a program could access it and consulted, add, modify or delete the data structures relevant to its inference procedures with the maximum efficiency and flexibility.

Theories and techniques about knowledge representation have undergone intense investigation the last years. Also, the actual understanding of the topic is still incomplete; knowledge representation is the most active topic in AI research.

The following section of this chapter discusses the elements of *knowledge* as they are derived from both psycology and AI research. This discussion then serves to justify the reason these techniques were used in the application methodology.

The elements of knowledge

The anatomy of knowledge is based on AI and Psycology research which has been carried out mainly to simulate the way human

memory and thinking processes operate.

- ***Objects*** Generally, humans think of knowledge in terms of facts about objects: The *door* is closed. *Birds* fly. *Robin* is a bird. A *man* has two legs. *Mary* borrowed my book etc.

For example, the importance of nouns in language. Thus, knowledge could be built on primitive knowledge types; objects.

For memory economy the relevant knowledge is linked through groups *(or classes)* of objects with strong similarities. For example, the piece of information "Robin flies" is build on the class "Birds" which includes the object "Robin", rather of the object itself. This characteristic classification makes the access and retrieval of information from the memory easier as it is not necessary to know only the features of the class rather of each object individually.

- ***Relationships*** Objects or classes of objects are linked together with relationships inside the human (or computer) memory in order for the stored information to be related and make sense. Thus the relationship "Father(X, Y)" which can be read as X is the father of Y, can link independent objects such as George, Paul, Nick together eg. Father(George, Paul) , Father(Paul, Nick).

- ***Facts or events*** Humans use various events of the world to increase their knowledge or they take from granded simple commonly known facts from their environment eg. It is raining today. The water is liquid. Peter married Ann.

Rules or Heuristics This type of knowledge enables humans (or computers) to infer new instances of a class or relation or even create new classes or relations within the domain based on the existing facts and relationships of the objects. For example, the relationship of "Father" as mentioned earlier and the fact that everybody is linked to somebody with the relationship "Father" creates a new relationship between the objects George and Nick: Grandfather(George, Nick). This heuristic rule in a computer program it may look like:

```
Grandfather(X, Y) IF
                    Father (X, Z) and Father(Z, Y)
                       OR
                    Father (X, Z) and Grandfather(Z, Y)
```

- ***Control Structure*** So far the main elements of storing and manipulating knowledge have been discussed. In addition to these, humans use a complex psychological way of understanding the world and learning new concepts. In an expert system such a cognitive process is only elementarily simulated with a special algorithm that *directs* the inference paths towards the solution; the *control structure*. In the field of AI, the nature of a control structure is literaly dependent on the way the domain knowledge is represented within the memory of the computer.

- ***Meta-knowledge*** This type of Knowledge helps humans to conceive information of the world into its highest *abstract* level and consequently have a better understanding of the knowledge they possess. In other words, Meta-knowledge is used by humans in

order to reason on what they already know. For example, humans (or computers could) know some facts about the knowledge they poccess such as the reliability of its source, its depth, its presentation etc. Meta-knowledge is very important in AI applications, in particular to expert systems that tackle fuzzy knowledge eg. uncertain or unreliable information (probabilistic reasoning).

The elements of knowledge, have been modeled in various ways by knowledge engineers to simulate *thought* process in AI applications. The most common techniques have been used are: production rules, procedural representation, semantic networks, frames and scripts, predicate calculus, analogical reasoning etc.

A detail discussion of knowledge representation techniques, however, is beyond the scope of this chapter (further detail can be found in [], [], [],). Only a brief introduction to the basic concepts of the Object Oriented paradigm is attempted in the next section, as it represents the software platform on which the system has been built on.

OBJECT ORIENTED PROGRAMMING

Object oriented programming is a specific way of grouping information. One way to understand the difference of Object

Oriented programming from conventional programming is by considering the implementations of *generic functions* within a program. A *generic function* is a function which will perform a specific action on its argument. Consider a knowledge engineer is writing a program to manipulate assembly structures and a piece of code is needed to delete a link from the assembly. In the conventional *action-oriented style* he writes a generic *delete* function which applies different procedures according to the kind of object is involved:

To <u>delete</u> an object from the assembly:

If the object is a *physical part*, then apply procedure A
If the object is a *sub-assembly* then apply procedure B. etc.

Similarly, the information is coded into the program for a generic function that *adds* a new object in the assembly:

To <u>add</u> an object in the assembly:

If the object is a *physical part*, then apply procedure K
If the object is a *sub-assembly* then apply procedure L. etc.

The programmer thinks in terms of the actions involved in his applications and constructs an IF-THEN generic function of all the possible cases for each of them. Consequently, in such a paradigm the information about each action is grouped whereas the information about the objects is scattered.

The *object-oriented programming* tackles the same information from

a different perspective:

If it is for an Physical part:

To *delete* the object apply procedure A

To *add* the object apply procedure K, etc.

If it is a sub-assembly:

To *delete* the object apply procedure B

To *add* the object apply procedure L, etc.

The programmer thinks in terms of the kinds of the *objects* which are involved in his application and groups their possible actions around them. Consequently, the information about the objects is grouped whereas the information about the actions is scattered.

The action-oriented programming languages (Pascal, Fortran) support their style by providing primitives for expressing conditionals and apply subactions (subroutines). On the other hand, object-oriented programming languages (Flavors, CommonOrbit) support their pohilosophy by providing primitives to create objects and accessor functions on the objects (they will be discussed shortly).

As it has been shown, the basic philosophy of object oriented programming is to provide a formalism that enable programmers to model the phenomena of a domain as discrete entities, ie. objects.

Instantiation (An-instance-of)

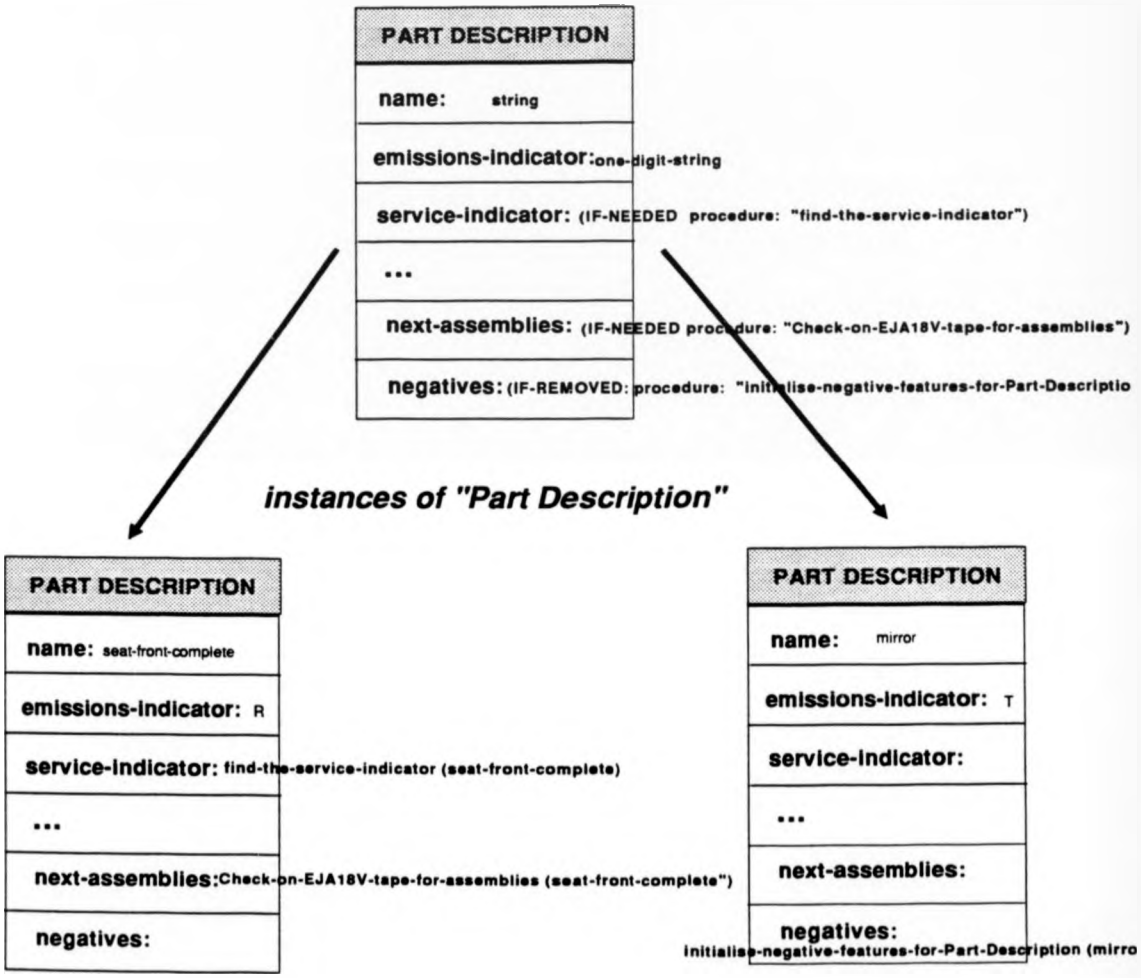


Figure A1: Two instances of the object "Part Description".

Two types of entities can be distinguihed within the object oriented programming: *classes* and *instances*. For example, the Rover APN catalogue at its lowest level can be viewed as a collection of entities of the class *part description* . A specific part description, then, such as "seat front complete manual-passenger" it will be an *instance* of this class (figure 1).

Another term for the object which could be used on the same context here is *frame*. The instances of this class are created by *instantiating* the general frame part description. The process of instantiating the class involves creating a new object by allocating memory for it, linking it to the class, such as by filling in a special "instane-of" attribute, and filling in the remaining attributes with particular information for the part description in question. An attribute may be provided with information and/or default values. It may also contain procedures attached to the slots called procedural attachments. These are generally of three types: The **if-needed** type are procedures to be executed when an attribute value is needed. The **if-added** type is run for procedures to be executed when a value is to be added to an attribute. Finally, the **if-removal** type is run whenever a value is to be removed from the attribute.

Methods

The fundamental characteristic of the object oriented programming

is that objects are directly associated with the procedures which are applicable to them. These procedures are called *methods*. Methods are similar to conventional procedures in that they have names, take arguments, perform computation, return values, produce side-effects. They basically control the behaviour of the objects they apply to. A useful property of an object's methods is that they have direct access to that object's local variables. This means that the variables are protected from the rest of the program and can only be accessed or altered through a set of methods. The methods therefore can be considered to be the communication *protocol* of the objects with the rest of the program.

More concepts of the object oriented programming are discussed in the following section using as a paradigm the FLAVORS programming language which represents the software tool (and philosophy) of the actual implementation of the system.

- *the FLAVORS system*

FLAVORS is a specialised object oriented programming system implemented in ZETALISP on the Lisp Machines using special microcode support. A class of objects is described as a Flavor type entity within the system. For example, the following function defines a specific information structure as Flavor

object (*class*) of type *part description* with attributes: name, record-no, main-group, emissions-indicator, design-reponsibility, year-of-issue, etc .....

```
(defFlavor  part-description (name
                              record-no
                              main-group
                              emissions-indicator
                              design-responsibility
                              year-of-issue (1986)
                              .....)
  ()
:initable-instance-variables
:readable-instance-variables))
```

Notice the *default* 1986 value of the "year-of-issue" attribute and the declarations that the attributes of this Flavor can be initialised and read from the user. The primitive function **make-instance** creates an instance (object) of this Flavor type (class) by allocating memory reference to it and *"moulding"* its structure on the basis of the Flavor prototype. .

```
-> (make-instance 'part-description :name "seat-front-complete"
                                    :record-no 456
                                    :design-responsibility R&D)

-> #<SEAT-FRONT-COMPLETE 143789634>
```

accessor functions

Methods (or procedures) can be assigned to the Flavor *part-description* to provide certain behaviour for all the instances of this particular Flavor. The method following, for example, specifies the way that an instance of the *part decription* Flavor prints itself:

```
;;; print the name of the part description Flavor

(defmethod (print part-description) ()
     (print (part-description-name self))
```

The creation of a Flavor type part-description has defined internally in the memory of the computer functions which can access locally the values of the attributes of any instance of this type. These are called the *accessor functions*. Their names are syntactically structured by the name of the Flavor, followed by a dash (-), followed by the name of the attribute they access.

Other methods can be defined, as well, which can use these functions and change the information within the data structure of each instance ie. add, delete or update the local variables etc., or control the communication of the part-description Flavor with the outside world.

Message passing

Message passing represents the old Flavors mechanism (see [] and [] for more details) of *methods* implementation which still exists and has been used inside the system. Message passing is like an indirect procedure call, but instead of invoking a procedure on an object, a message is sent to the object. The following message clears the screen of an object window:

```
(:send window :clear-history)
```

Generalisation (A-Kind-Of)

Flavor types (classes) are related to one another in supertype/subtype hierarchies. Given a pair of hierarchically related types, the more general one is termed the *supertype*, the most specific one the *subtype*. For example, *physical-part* is a new Flavor type. In addition, *physical part* is the subtype of part-description (figure 2).

What gives this notion its power is that it serves as the base for an automatic inheritance mechanishm.

Specialisation/Inheritance

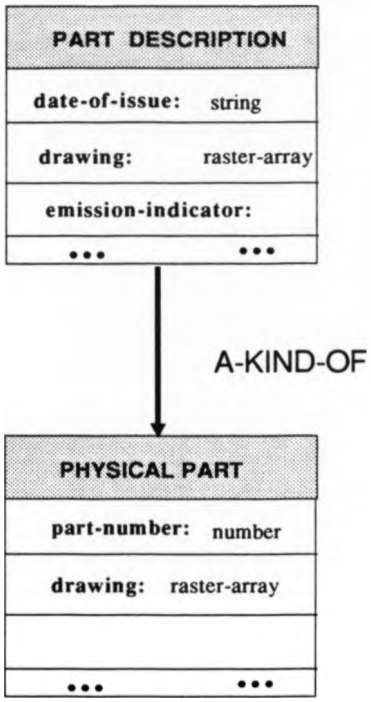


Figure A2: **Generalisation and Inheritance**

The most of the power of the object oriented programming comes from specialisation (or inheritance). In general, specialisation allows a more specific object to inherit data and methods from a more general class (Flavor type).

For example, the previously mentioned Flavor *physical-part* inherits all the knowledge of the part-description Flavor and additional information of its drawing:

```
(defFlavor   physical-part ()
     (part-number number)
     (drawing  MATRIX(40, 90))
          (part-description)
:initable-instance-variables
:readable-instance-variables)
```

The inheritance mechanism is invoked by inserting **part-description** inside the parenthesis of the Flavor in the third row of its definition. This means that the physical-part Flavor inherits the local variables such as the name, record-no, emission-indicator etc. of the part-description Flavor. Notice, *physical-part* inherits all the methods of the *part description* Flavor, as well. eg. the printing method.

In general, specialisation allows a more specific object to inherit data and methods from a more general object.

However, if an object needs to rewrite a method it has inherited from a more general object for some reason, the newly written local method will override the more general one that has been inherited.

For example, a new printing method can be defined for the *physical part* Flavor if the programmer thinks that such an inheritance is not efficient for his application:

```
;;; print the drawing of the physical-part Flavor
```

```
(defmethod (print physical-part)
   (print (physical-part-DRAWING  self)))
```

Notice however, the ordering of the inheritance mechanism can be altered according to the needs of the knowledge engineer. Thus, methods can be inherited using the *most generic* concept or another number of method combinations or other facilities available from the FLAVORS system eg. daemons :before and :after, :and and :or combination methods, whopers, wrappers etc. A detailed discussion of changing the behaviour of inheritance in a Flavor hierarchy can be found to [3] and [4].

Multiple Inheritance

Multiple inheritance is supported from the FLAVORS programming language and expands the concept of strict hierarchical inheritance described above. Basically, multiple inheritance provides the mechanism for a class to share the methods and data with any arbritary class. The real power of Flavors is embedded in multiple inheritance. More specifically, in the facility of producing new Flavors by combining existing ones.

Figure 3 depicts the multiple inheritance concept. In the example, the class of *physical-part* objects could be derived by combining two parent Flavors *part-description* and *physical-object* and make their in-line definition obsolete. Besides, in

this way, both philosophies of physicality and functionality merge and make the physical-parts of the vehicle behave in a similar way to the knowledge engineer's natural way of thinking. A detailed discussion of multiple inheritance and problems met and solved can be found in [7] and [8].

presentation types

Historically, the user interface component has often turned out to be the largest part of any larged-sized computer system. The part of the program devoted to problem solving was often overshadowed by the part that took care of presenting data, reading responses, and handling input and menu display.

The use of Flavor objects in a system can ease the implementation of higher-level user interfaces. As the data are coded as objects of specific classes and methods which control their behaviour, these methods could also control the way in which these objects are presented in the screen. ie. the printing method of the part-description Flavor object, which discussed earlier.

Hence, as the use of Flavors determines a way of discrimination among the objects of the screen, using the mouse to sensitise specific objects programs can be written more easily.

Lisp environment supports *presentation types* of Flavor objects. Presentation types are the means of organising the way data is

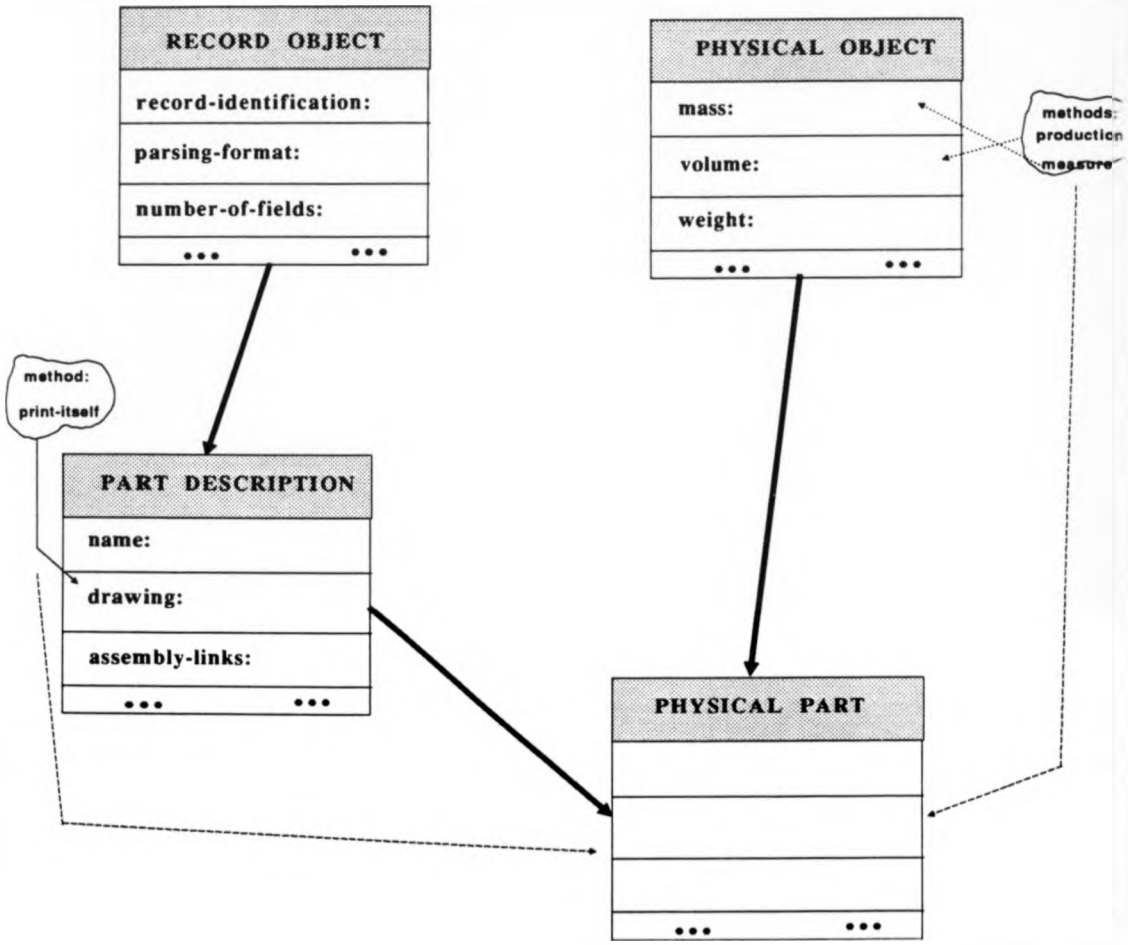


Figure A3: Inheritance of attributes and methods.

collected from and returned to the user. ie. the user-visible representations of objects. Their implementation in Lisp Machines came from a Doctorate thesis at MIT university [4].

In summary, object oriented programming environment offers aa alternative way of organising information. A number of *features* of such an environment have been introduced in this chapter. However, some others also exist, such as *delegation, percolation,* etc. which appear with the growing number of programming environments supporting object-oriented programming style. FLAVORS is one of the oldest object-oriented environments (together with CommonOrbit and CLOS) and has become the standard platform in Lisp Machines. It merely represents *functional* objects and it is build on the Daplex philosophy [].

APPENDIX 2

The STATICE environment

Object-oriented database systems are a new phenomenon, only recently emerging from the research world. Up till now Lisp objects were confined to virtual memory. They could not be used for long term information storage, because they vanished when the machine booted, or if the power failed. To provide persistent storage, the user must arrange to copy information out to files.

In files, however, the interconnected structure of diverse objects must be flattened out, and somehow encoded into simple bytes and characters. This encoding adds complexity to the application and takes time to run. As the files become large, saving out data and reading it back in becomes slow. Because information is being copied, the user has to worry about the copies becoming inconcistent. The benefits of object oriented programming are lost.

Statice is one of the first object-oriented databases available anywhere. It extends the power of objects by providing *persistent* storage. Actually, it gives the user the ability to store information outside of the virtual memory of the computer, in a real data base on disk, without losing object-orientd programming.

For example, one can build a database of objects describing the design of a vehicle with thousands of parts of many different types, and the complex relationships between them. The database lasts as long as the knowledge engineer needs it. And he can still use object oriented programming techniques directly on the information.

Object-oriented databases, it can be said that they are the epitome of the *object-oriented Flavor* in the field of Artificial Intelligence. In particular, Statice supports all the characteristics of the Flavors environment: methods, multiple inheritance, dynamic schema modification together with the rest of a real data base system ie. indexes, default optimistic concurrency control at the page level, with pessimistic optional, data security and concistency. In more detail , Statice is built on the techniques developed and refined over the years by traditional database management systems such as *transactions* Statice keeps the data safe by using a log for *recovery* in case of failures (such as power outrages). It prevents multiple users fron unexpectedly interfering with each other by providing *concurrency control* using locking. Transactions are *isolated* from one another and they can be nested Recovery and concurrency control provide Statice with true transactions.

Integrity Checking : Statice's *Integrity Checking* assures that invalid data is rejected from the data base. *Efficient data access:* Statice uses powerful database acess methods to provide efficient access to information stored on disk.

Data Independance : A non procedural query language, with a

query optimiser, provides the same data independence that makes relational databases flexible.

Statice also provides new capabilities, beyond those offered by relational systems.

Statice's flexible system of *entity types* and *relationships* is ideal for representing complex structures.

Multiple Inheritance allows modular sharing and flexible reuse of code, making programs easier to maintain.

Methods let the knowledge engineer attach procedures to entities and generally lets him associate programs and data.

From the application prospective Statice has no hard limit on the number of entity types, the number of attributes in a type, or the length of names of entity types and attributes. Similarly there is no hard limit on the number of users per single server. In addition, it supports a SQL-like language for queries at the top level and the user can view the actual objects by entering assertions to match a pattern.

The next section discusses in detail the Statice environment because it shows research interest as OODBs represent a new software technology and the most of such systems are currently under experimentation rather than real life apllications. Besides it familiarise the reader with the characteristics of the databases that have been designed to support **ROOVESP**. APN

catalogue entities are used as examples in order to relate the discussion to the **ROOVESP**,s design

The Statice OODB system

Statice's componentss

Every Statice database is stored in its own file in a special type of file system called a *statice file system*. The logical creation of a new statice database system is formed with a new *schema* definition. The schema is a description of everything that appears in the database. Each database schema is made up of entities which are analogous to the Flavor objects discussed earlier. Their attributes ,as well, correspond to the attributes of the Flavor objects. They express properties of the entities and/or relationships between the entities.

The below piece of code represents the APN catalogue schema (data base) which is made from the entities SSSVPG (Sub-Sub-Sub-Vpg), Family, Primary-description-ID, Secondary-description-ID ...(for a complete definition of the schema the reader can refer to appendix .....)

attribute types

The difference between the Statice entities and the Flavor objects is that the former's attributes have a *type* which can be either an *ordinary* type or another Statice entity type. For instance, the part descriptions properties of the *secondary-description-ID* entity are of string type, *has-must-features* are vectors, *belongs-to-primary-description* are of **the secondary-description-id** entity type etc. Statice offers the ability to the knowledge engineer to define his own *physical* and/or *logical* types that he wants to maintain in his application eg. arrays, a list of restricted options etc.

The general conclusion is that the *type* of each attribute is defined at the top level of the database design, as it should happen working with real database systems. Besides, Statice offers greater flexibility in representing the data values of the attributes than does a traditional database system.

There is no need for the user to pre-declare lengths of strings or integer; any length always works. Strings can be in different fonts or sizes, or even in different character sets such as Japanese (a useful characteristic for Rover because of its close relationship with Honda).

Statice pre-defined functions

The definition of a Statice entity type creates *accessor* functions, *reader* or *writer* functions for accessing the value of an entity's attributes, and an *entity constructor function* for creating new instances of the entity. For example, the function *make-secondary-description-ID* is created automatically from the Statice system to construct instances of the **secondary-description-ID** entity, whereas the function *secondary-description-id-part-description* accesses the value of the *part description* attribute of the *secondary-description-ID* entity.

the :initform, :no-nulls, :read only attribute options

The *behaviour* of the attributes of an entity it can be controled more specifically with other facilities available from Statice. For example, the **:initform** tag assigns a *default* mechanism to the data base, **:no-nulls** predefines entities that some of their attributes' values require initialisation in order for them exist and **:read-only** protects the database from inappropriate changes. In the APN catalogue example all part descriptions attributes are unique (**:unique**), valued (**:no-nulls**) and cannot be changed form the user (**:read-only**).

Multiple Inheritance

An entity type can *inherit* from other entity types, just as Flavors can inherit from other Flavors. In the design of both the APN catalogue data base and the Additional Feature Charts database no inheritance has been used. The reason was that the entities within the APN catalogue do not logically inherit each other but instead they are linked together in a tree-like hierarchy. Such a hierarchy, as it has been mentioned, is implemented with bidirectional pointers in the values of the attributes, actually the real Statice objects.

Inheritance could be needed only from softare engineering aspects in the database ie. functionality rather physicality. For example, a logical entity could be defined called **feature-groups-link** which would have as attributes *has-must-features* and *has-maybe-features*. Then all the current entities of the data base could inherit it such an entity as *must* and *maybe (or optional)* feature references do exist in all levels of the hierarchy.

Another way it could be the definition of more restricted logical entities rather than the **feature-groups-link** which would inherited at one level only. eg. between the *family-within-a-SSS-VPG* and *primary-description-ID* or between *primary-description-ID* and *secondary-description-ID,* etc. as entities do tend to resemble more between a single level of the hierarchy structure.

Purely for demonstration reasons, in particular to the Rover

party, the design of the APN database is implemented in the simplicity software level.

inheritance of attributes

Refering to the above example, since the entity **secondary-description-ID** would inherit the entity **feature-groups-link** and **feature-groups-link** has attributes that maintain *must* and *maybe* feature groups then **secondary-description-ID** has attributes which keep *must* and *maybe* feature groups.

In general, entity types inherit all the attributes of their ancestor entities.

inheritance of properties of the attributes.

The property of the attributes *has-must-features* and *has-maybe-features* to always have a value (:no-nulls t) is inheritented from the **secondary-description-ID** entity. In the same way the *uniqueness* of the value of the part-description attribute (:unique t) it will be inheritent from any Statice entity that inherits the **secondary-description-ID** entity.

inheritance of types.

Type membership is affected by inheritance the same way as in Flavors. Every entity of type **secondary-description-ID** is also considered to be of type **feature-groups-link** as it inherits the latter.

inheritance of methods.

The inheritance of the types within Statice implies the inheritance of the methods which are linked to the entity types, as well. As mentioned, very entity of type **secondary-description-ID** is also considered to be of type **feature-groups-link**.

For example, the argument of the accessor function "features-group-link-*has-must-features*" is required to be of type **feature-groups-link**. It is correct to call the function "features-group-link-*has-must-features*" on a **secondary-description-ID** entity, because such an entity is a **feature-groups-link** ,too.

<u>inverse functions and indexes</u>

Inverse functions (or inverse readers) can be defined for the

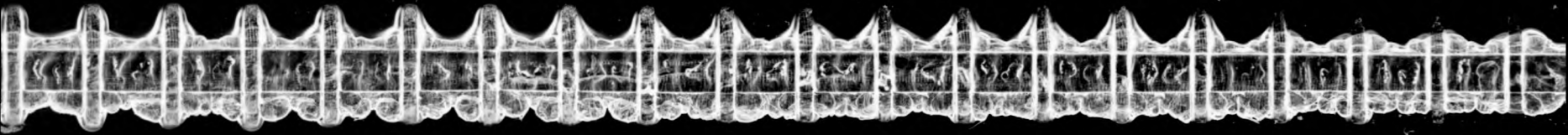
attributes of an entity with the **:inverse** option. These functions access the actual entity of the Statice database where the attribute belongs. For example, the part description attribute of the **secondary-description-ID** entity has as inverse function the *the-part-description-entity-type.* This function is created automatically from the Statice system when it is tagged to the part description attribute. When it functions it accesses the object (entity) whose value of its attribute matches the query, from the database. eg. a query such as "seat-front-complete manual-driver" accesses the Statice entity with such a part-description value. Notice, that each part description attribute is unique in the data base, consequently only one Statice object will be returned after the evaluation of the function. In general every Statice reader function, including the inverse ones, can be characterised as *one-to-one, many-to-one, one-to-many, many-to-many.*

The inverse reader function is one of the additional facilities of Statice that no traditional database systems offer. A reader function, however, may slow down the database if it is not used properly. Considering the inverse reader *the-part-description-entity-type* discussed earlier the way it finds the entity for the part description named "seat-front-complete manual-driver" is by examining each entity of type **secondary-description-ID** (including entities whose type inherits from type **secondary-description-ID**) checking the value of the part-description attribute and seeing whether it equals "seat-front-complete manual-driver". This would be slower and slower as the database grew, taking time linearly

proportional to the number of **secondary-description-ID** entities (a significant problem if considered the almost 15,000 part descriptions within Rover). In fact this is what happens if there isn't an index.

The performance of Statice can be increased by using *indexes*. These are auxiliary structures that reside invisibly in the database. In reality they are compact tables that represent mappings from values of the attributes under consideration to the corresponding entities, sorted alphabetically and generally organised to make searching fast. In database jargon, it's organised as a *B+ tree*. Looking at the **primary-description-ID** entity there is an index which is used to speed up a regular accessor function, rather than an inverse reader function. This index increases the speed of locating the *has-secondary-descriptions* attributes in the data base. In reality, the performance improvement happens in both the reader and the writer functions of this attribute as such functions are used by the system when it builds up the APN catalogue database automatically.

In addition, *multiple indexes* can be used in the design of the data base. That is an index a compact table, as well, (a *B+ tree*) that associates *pairs* of attribute values with pointers to entities.



APPENDIX 3

LISTING OF THE
VAX TAPE DATA
&
THE PARSING
PROGRAM

```
"XF  129 H              N     A     D      E      BA M/7P/88                    P JX033            M32BM32DM3
2F                                                          M32B+T56T=T51A
                                                HND"
"XF  130 H              N     A     D     A       BA M/52P/85    M/7P/88      PPB9307   JX033    M32BM32D
                                                            M32B+T56T
                                                HND"
"XF  131 H              N     A     D     A       BA M/5P/88 S1  M/7P/88      PPJX054   JX033    M32BM32D
  WITHH50PPACK                                              M32B+T56T
                                                JND"
"XF  132 H              N     A     D      E      BA M/7P/88                    P JX033            M32DM32B
  WITHH50PPACK                                              M32B+T56T=T51A
                                                JND"
"XF  135 H              N     A     D      E       A A M/6P/85     M/5P/88      PHB9202   B9553    M20U

                                                HND"
"XF  140 H              N     A     D      E      BA M/23P/85    M/7P/88      PPXF5A    JX033    M32FM32B
                                                            M32B+T51A
                                                HND"
"XF  145 H              N     A     D       F      A A M/45P/86                 P B9453            M20EM20U
  WITHH50KPACK                                              M20E-T56Z
                                                HNM"
"XF  146 H              N     A     D       F      A A M/5P/88                  P B9553            M20EM20U
  WITHH50PPACK                                              M20E-T56Z
                                                JNM"
"XF  150 H              N     A     D       F      BA M/15P/87                  P B9453            M32FM32S
  WITHH50KPACK                                              M32S+T58L=T53V
                                                HNM"
"XF  151 H              N     A     D       F      BA M/5P/88     M/5P/88 S1    PHB9553   JX054    M32BM32DM3
2FM32IM32S    WITHH50PPACK                                  M32B+T51A=T56T M32S-T53Z
                                                JNM"
"XF  155 H               W    A    B      A       A A FSU 84A     M/20P/88     PPFSU 84A JX076    M20UM36Z

                                                HWB"
```

```
"420B95BS       XF  Y
                      M/4/84                    P XF4C              B95

                     00"
"400C16B*******XF       SPECFY WTH C1   GROU & C8 GRUP
 CII7  P7O              ************             P XF6A             +C16****

                    *00"
"410C16B  O     XF  HN D  AH50KH50P         H50KH50P
 R=   P=                M/5P/88 S1                P JX054             C16

                     00"
"420C16BS       XF  W  D  A
                       M/21P/86                 P XF6A              C16

                     00"
"430C16BS       XF  Y  DF A
                       M/5P/88 S1               P JX054             C16

                     00"
"400C16P*******XF       SPECFY WTH C1   GROU & C8 GRUP
 CII7  P7O              ************             P XF6A             +C16****

                    *00"
"410C16PS       XF   G B  A
                       M/21P/86                 P XF6A              C16

                     00"
"420C16PS       XF  DN B  A
                       M/5P/88 S1               P JX054             C16

                     00"
"430C16PS       XF  DW B  A
                       M/5P/88 S1               P JX054             C16

                     00"
"440C16P B      XF  HN B  A
                       M/5P/88 S1               P JX054             C16

                     00"
"450C16P B      XF  HW B  A
                       M/5P/88 S1               P JX054             C16

                     00"
"460C16PS       XF  M  B  A
                       M/21P/86                 P XF6A              C16

                     00"
"400C16T*******XF       SPECFY WTH C1   GROU & C8 GRUP
 CII7  P7O              ************             P XF6A             +C16****

                    *00"
"410C16T   M    XF  HN B  A
                       M/5P/88 S1               P JX054            +C16M32F

                     00"
"420C16T   M    XF  HW B  A
                       M/5P/88 S1               P JX054            +C16M32F

                     00"
"070C16VS       XF  Y  DA A
                       M/5P/88 S1               P JX054             C16

                     00"
"400C16V*******XF       SPECFY WTH C1   GROU & C8 GRUP
 CII7  P7O              ************             P XF6A             +C16****

                    *00"

"400C78A*******XF       SPECFY WTH C6   GROU & C1 GRUP
 CII1  P7O              ************             P XF6A             +C78****

                    *00"
"410C78AS       XF      B  A
                       M/21P/86                 P XF6A              C78
```

```
                        00"
"420C78AS       XF  W  D  A
                          M/21P/86                  P XF6A              C78

                        00"
"430C78A  O     XF  HN D  AH50KH50P         H50KH50P
 R=   P=                  M/5P/88 S1                P JX054             C78

                        00"
"440C78AS       XF  X  D  A
                          M/21P/86                  P XF6A              C78

                        00"
"400C78B*******XF         SPECFY WTH C6   GROU & C1 GRUP
 CII1  P7O                ************              P XF6A             +C78****

                       *00"
"410C78BS       XF  D  D  A
                          M/21P/86                  P XF6A              C78

                        00"
"420C78BS       XF  H  DE A
                          M/21P/86                  P XF6A              C78

                        00"
```

```
"410F95CS       XF  DW
                      M/5P/88 S1              P JX054            F95

                    00"
"400F96C B      XF  D
                      M/5P/88 S1              P JX054            F96

                    00"
"410F96C   M    XF  H     R03IR03AH50KH50P
 ---                  M/5P/88 S1              P JX054            F96

                    00"
"420F96C   M    XF  M     F98JF98KF98Z
 +--                  M/5P/88 S1              P JX054            F96

                    00"
"400F96D B      XF  Y
                      M/5P/88 S1              P JX054            F96

                    00"
"410F96D  O     XF  M ABEBAR05C           R05C
 R    P               M/28P/88                P JX147           +F96M32I
                                  M32I+T55E
                    00"
"400F98A  O     XF  DN  EAAZ96B           Z96B
 R    P               M/21P/86                P XF6A            +F98M20U
                                  M20U+T60A
                    00"
"410F98A  O     XF  DW  EAAZ96B           Z96B
 R    P               M/21P/86                P XF6A            +F98M20U
                                  M20U+T60A
                    00"
"420F98A  O     XF   G  EAAZ96B           Z96B
 R    P               M/21P/86                P XF6A            +F98M20U
                                  M20U+T60A
                    00"
"400F98G   M    XF  DN    Z96BR02R
 --                   M/10/83 S1              P XF4C            +F98M20*M36*M37*M38*M39*

                    00"
"410F98G   M    XF  DW    Z96B
 -                    M/10/83 S1              P XF4A            +F98M20*M36*M37*M38*M39*

                    00"
"420F98G   M    XF  DG    Z96BR01D
 --                   M/4/84                  P XF4C            +F98M20*M36*M37*M38*M39*

                    00"
"440F98I   M    XF  M     R05AR05BR05D
 ---                  M/5P/88 S1              P JX054           +F98M32BM32S
                                  M32B+T54R     M32S+T53V=T58L
                    00"
"441F98I   M    XF  MNADEBB
                      M/28P/88                P JX147           +F98M32I
                                  M32I+T55E
                    00"
"400F98J   M    XF  HN
                      M/5P/88 S1              P JX054           +F98M32*M35I
                                  M32B-T56T
                    00"
"410F98J   M    XF  HW    R03I
 -                    M/5P/88 S1              P JX054           +F98M32*M35I
                                  M32B-T56T
                    00"
"400F98K   M    XF  M     R02JR02KR05CR05D
 ---                  M/5P/88 S1              P JX054           +F98M32BM32DM32IM32S
                                  M32B-T54R-T51A M32S-T53V=T58L M32I-T55E
                    00"
"400F98W B      XF  HN
                      M/10/83 S1              P XF4C             F98

                    00"
"410F98W B      XF  HW
                      M/10/83 S1              P XF4C             F98

                    00"
```

```
"400F98Z    M   XF  M       R05AR05BR05D
 ---                        M/5P/88 S1              P JX054            +F98M32F

                          00"
"400G70BS       XF         B
                            M/34/83                 P XF4C              G70

                          00"
"410G70BS       XF        K
                            M/34/83                 P XF4G              G70

                          00"
"400G70CS       XF        F
                            M/34/83                 P XF4G              G70

                          00"
"410G70C    E   XF        E
                            M/5P/88                 P B9553            +G70M20EM32SM38XM39XM39Y
                                          M20E+T52G       M32S+T53Z
                          00"
"420G70C    E   XF        A
                            M/5P/88                 P B9553            +G70M39YM39Z
                                          M39Y+T55N=T56N
                          00"
"400H37A B      XF
                            M/5P/88 S1              P JX054             H37

                          00"
"400H37B*******XF           B35CB36C
 I,                         M/5P/88 S1              P JX054             H37

                         *00"
"410H37B    M   XF  DN
                            M/5P/88 S1              P JX054            +H37M20EM36*M37*M38*M39*
                                          M20E+T52G=T56G
                          00"
"420H37B    M   XF  DW
                            M/5P/88 S1              P JX054            +H37M20EM36*M37*M38*M39*
                                          M20E+T52G=T56G
                          00"
"430H37B    M   XF  H
                            M/5P/88 S1              P JX054            +H37M20EM32SM35IM36*M37*M38*M39*
                                          M20E+T52G=T56G M32S+T53Z
                          00"
"440H37B    M   XF  M
                            M/5P/88 S1              P JX054            +H37M20EM32*M35IM36*M37*M38*M39*
                                          M20E+T52G=T56G M32S-T53Z
                          00"
"450H37B    M   XF  Y
                            M/5P/88 S1              P JX054            +H37M20EM32*M35IM36*M37*M38*M39*
                                          M20E+T52G=T56G=T56Z          M32I-T60M       M32S-T53Z
                          00"
"460H37B    M   XF  W
                            M/5P/88 S1              P JX054            +H37M20EM32*M36*M37*M38*M39*
                                          M20E+T52G=T56G=T56Z
                          00"
"470H37B    M   XF  X
                            M/5P/88 S1              P JX054            +H37M20EM32*M36*M37*M38*M39*
                                          M20E+T52G=T56G=T56Z          M32S-T53Z
                          00"
"480H37B  O     XF  DN
                            M/5P/88 S1              P JX054            +H37M32*

                          00"
"490H37B  O     XF  DW
                            M/5P/88 S1              P JX054            +H37M32*

                          00"
"500H37B  O     XF  DG
                            M/5P/88 S1              P JX054            +H37M32*

                          00"
"510H37B  O     XF  H
                            M/5P/88 S1              P JX054            +H37M20E
                                          M20E+T56Z
                          00"
```

```
"520H37B  O    XF  HW  E AR03I            R03I
 R    P                M/31P/87 S2 M/31P/87 S4 PPB9550   B9550   -H37M20U

                      00"
"530H37B  O    XF  HN    H50KH50P
 P=                     M/5P/88 S1              P JX054          +H37M32*
                                  M32S-T53Z
                      00"
"540H37B  O    XF  HW
                        M/5P/88 S1              P JX054          +H37M32*
                                  M32S-T53Z
                      00"
"550H37B  O    XF  HG
                        M/5P/88 S1              P JX054          +H37M32*

                      00"
"560H37B  O    XF  M
                        M/5P/88 S1              P JX054          +H37M20E
                                  M20E+T56Z
                      00"
"400H37C*******XF       B34TB35CB36C
 I,,                    M/5P/88 S1              P JX054           H37

                     *00"
"410H37C   M   XF  HG   Z96B
 -                      M/5P/88 S1              P JX054          +H37M20U

                      00"
"420H37C   M   XF  M
                        M/5P/88 S1              P JX054          +H37M20UM20EM32S
                                  M20E+T53R=T53X M32S+T53Z
                      00"
"430H37C   M   XF  Y
                        M/5P/88 S1              P JX054          +H37M32S
                                  M32S+T53Z
                      00"
"440H37C  O    XF  DN   Z96B
 -                      M/5P/88 S1              P JX054          +H37M20UM20EM36*M37*M38*M39*

                      00"
"450H37C  O    XF  DW   Z96B
 -                      M/5P/88 S1              P JX054          +H37M20UM20EM36*M37*M38*M39*

                      00"
"460H37C  O    XF  DG   Z96B
 -                      M/5P/88 S1              P JX054          +H37M20U

                      00"
"470H37C  O    XF  HN   R03A
 -                      M/5P/88 S1              P JX054          +H37M20UM36*M37*M38*M39*M20EM32SM35I
                                  M32S+T53Z
                      00"
"480H37C  O    XF  HNADEAAR03A            R03A
 R    P                M/5P/88                 P JX054          +H37M20U

                      00"
"490H37C  O    XF  HW   R03I
 -                      M/5P/88 S1              P JX054          +H37M20UM36*M37*M38*M39*M20EM32SM35I
                                  M32S+T53Z
                      00"
"500H37C  O    XF  HW  E AR03I            R03I
 R    P                M/31P/87 S2 M/31P/87 S4 PPB9550   B9550   +H37M20U

                      00"
"510H37C  O    XF  HG
                        M/5P/88 S1              P JX054          +H37M20E

                      00"
"520H37C  O    XF  M
                        M/5P/88 S1              P JX054          +H37M36*M37*M38*M39*M20EM35I
                                  M20E+T52G=T56G=T56Z
                      00"
"530H37C  O    XF  Y
                        M/5P/88 S1              P JX054          +H37M36*M37*M38*M39*M20EM35I
                                  M20E+T52G=T56G=T56Z
                      00"
```

```
"540H37C  O    XF  W
                        M/5P/88 S1              P JX054            +H37M36*M37*M38*M39*M20E
                                      M20E+T52G=T56G=T56Z
                       00"
"550H37C  O    XF  X
                        M/5P/88 S1              P JX054            +H37M36*M37*M38*M39*M20E
                                      M20E+T52G=T56G=T56Z
                       00"
"400H37D*******XF       B35CB36EB34S
 I,,                    M/5P/88 S1              P JX054             H37

                      *00"
"410H37D   M   XF  Y
                        M/5P/88 S1              P JX054            +H37M20EM20UM32I
                                      M20E+T53R=T53X M32I+T60M
                       00"
"420H37D   M   XF  W
                        M/5P/88 S1              P JX054            +H37M20EM20U
                                      M20E+T53R=T53X
                       00"
"430H37D   M   XF  X
                        M/5P/88 S1              P JX054            +H37M20EM20UM32S
                                      M20E+T53R=T53X M32S+T53Z
                       00"
"440H37D  O    XF  HN   H50KH50P          R03A
 P=   -                 M/5P/88 S1              P JX054            +H37M20EM20UM35IM36*M37*M38*M39*
                                      M20E+T53R=T53X
                       00"
"450H37D  O    XF  HW   R03I
 -                      M/5P/88 S1              P JX054            +H37M20EM20UM35IM36*M37*M38*M39*
                                      M20E+T53R=T53X
                       00"
"460H37D  O    XF  M
                        M/5P/88 S1              P JX054            +H37M20EM20UM35IM36*M37*M38*M39*
                                      M20E+T53R=T53X
                       00"
"470H37D  O    XF  Y
                        M/5P/88 S1              P JX054            +H37M35IM36*M37*M38*M39*

                       00"
"480H37D  O    XF  W
                        M/5P/88 S1              P JX054            +H37M36*M37*M38*M39*

                       00"
"490H37D  O    XF  X
                        M/5P/88 S1              P JX054            +H37M36*M37*M38*M39*

                       00"
"400H37E*******XF       B34PB35CB36E
 I,,                    M/5P/88 S1              P JX054             H37

                      *00"
"410H37E  O    XF  HN   R03A
 -                      M/5P/88 S1              P JX054            +H37M20UM20E
                                      M20E+T53R=T53X
                       00"
"420H37E  O    XF  HW   R03I
 -                      M/5P/88 S1              P JX054            +H37M20UM20E
                                      M20E+T53R=T53X
                       00"
"430H37E  O    XF  M
                        M/5P/88 S1              P JX054            +H37M20UM20E
                                      M20E+T53R=T53X
                       00"
"440H37E  O    XF  Y
                        M/5P/88 S1              P JX054            +H37M20UM20E
                                      M20E+T53R=T53X
                       00"
"450H37E  O    XF  W
                        M/5P/88 S1              P JX054            +H37M20UM20E
                                      M20E+T53R=T53X
                       00"
"460H37E  O    XF  X
                        M/5P/88 S1              P JX054            +H37M20UM20E
                                      M20E+T53R=T53X
                       00"
```

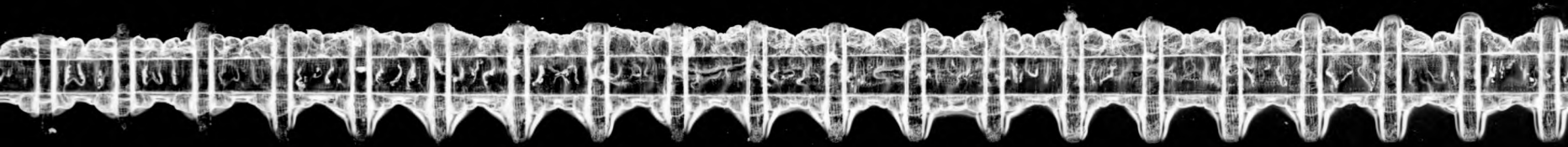
```
"400H48AS       XF  X
                        M/27P/88                P JX125            H48

                       00"
"410H48A  O     XF  HN D   H50P            H50P
 R    P                  M/27P/88                P JX125            H48

                       00"
"420H48A  O     XF  MNADEAAR05A            R05A
 R    P                  M/27P/88                P JX125            H48

                       00"
"400H50F  O     XF  YW DE  H50FB51FB92BD92C
 I,,,                   M/5P/88 S1  M/28P/88    PPJX054   JX147   -H50M32FM32I

                       00"
"401H50F  O     XF  YW DE  H50FB51FB92BD92C
 I,,,                   M/28P/88                P JX147           -H50M32F

                       00"
"410H50F  O     XF  YW DA  H50FB51FB92BD92C
 I,,,                   M/5P/88 S4              P JX054            H50

                       00"
"420H50F   M    XF  YW DF  H50FB51FB92BD92CF34P
 I,,,,                  M/5P/88 S1              P JX054           +H50M32FM32I

                       00"
"400H50K  O     XF  HN D   H50KB06SB12CB15XB45BB51EB90FC16BC35YC40NC41BC42FC71AC78AF25BF26BF66BF76BF98JF98
WI,,,, ,,,, ,,,, ,,,, ,<< M/5P/88 S1              P JX054            H50

                       00"
"400H50P  O     XF  HN D   H50PB06SB12CB15XB45BB51EB90FC16BC35LC41GC42HC71AC78AF25BF26BF66BF76BH48AF98JF98
WI,,,, ,,,, ,,,, ,,,, ,<< M/5P/88 S1              P JX054            H50

                       00"
"400JCOF  O     XF  W
                        M/5P/88                 P B9553            JCO

                       00"
"410JCOF  O     XF  X
                        M/5P/88                 P B9553            JCO

                       00"
"420JCOF  O     XF  H      H50KH50P
 P=                     M/5P/88                 P B9553            JCO

                       00"
"400JFMZ  O     XF  DGABAAAR01S
 +                      M/16P/88                P JX063           +JFMM20U
                                        M20U+T60A
                       00"
"400JFNH  O     XF  H      H50KH50P
 P=                     M/5P/88                 P B9553            JFN

                       00"
"400JHNA  O     XF  W
                        M/5P/88                 P B9553            JHN

                       00"
"410JHNA  O     XF  X
                        M/5P/88                 P B9553            JHN

                       00"
```

PAGES MISSING

1 + 2.

```
        (:mfam3 1)
        (:mfam4 1)
        (:mfam5 1)

        (:mcap1 1)
        (:mcap2 1)
        (:mcap3 1)
        (:mcap4 1)
        (:mcap5 1)
        (:mcap6 1)
        (:mcap7 1)

        (:mperf1 1)
        (:mperf2 1)
        (:mperf3 1)
        (:mperf4 1)
        (:mperf5 1)
        (:mperf6 1)
        (:mperf7 1)

        (:mdrive1 1)
        (:mdrive2 1)

        (:mtrans1 1)
        (:mtrans2 1)

        (:mauthin 12)
        (:mauthout 12)

        (:mrelin 1)
        (:mrelout 1)

        (:mdepin 8)
        (:mdepout 8)

        (:skip-space 72)

;       (:mtergrp1 4)
;       (:mtergrp2 4)
;       (:mtergrp3 4)
;       (:mtergrp4 4)
;       (:mtergrp5 4)
;       (:mtergrp6 4)
;       (:mtergrp7 4)
;       (:mtergrp8 4)
;       (:mtergrp9 4)
;       (:mtergrp10 4)
;       (:mtergrp11 4)
;       (:mtergrp12 4)
;       (:mtergrp13 4)
;       (:mtergrp14 4)
;       (:mtergrp15 4)
;       (:mtergrp16 4)
;       (:mtergrp17 4)
;       (:mtergrp18 4)

;       (:mterexc1 4)
;       (:mterexc2 5)
;       (:mterexc3 5)
;       (:mtq1 1)
;       (:mterexc4 4)
;       (:mterexc5 5)
;       (:mterexc6 5)
;       (:mtq2 1)
;       (:mterexc7 4)
;       (:mterexc8 5)
;       (:mterexc9 5)
;       (:mtq3 1)
;       (:mterexc10 4)
;       (:mterexc11 5)
;       (:mterexc12 5)
;       (:mtq4 1)
;       (:mterexc13 4)
;       (:mterexc14 5)
;       (:mterexc15 5)
;       (:mtq5 1)
```

```
;         (:mterexcl6 4)
;         (:mterexcl7 5)
;         (:mterexcl8 5)
;
;         (:mtq6 1)

          (:all-the-base-features-summary-restrictions   90)  ;BECAUSE THIS WILL BE USED IN THE
                         ; FLAVOR INITIALISATION INSTEAD TO CREATE AN "unbound"  VARIABLE
                         ; "comments "  using (:comments 90) , when no comments EXIST.

          (:msales 3)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun try-base    (filename)
  (with-open-file (stream filename :direction :input)
    (loop repeat 20
          as rec = (read stream)
          as parsed-rec = (parse-record  rec     *new-base-record-format*)


          if   (not (equal (read-comments
                             (second (assoc ':all-the-base-features-summary-restrictions parsed-rec)))
 nil))                         collect
              (take
                (get-the-base-features-summary-comments-restrictions   parsed-rec))
          else collect
              (take   parsed-rec))))




;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(defun  there-are-some-comments  ()
  (not (equal (second (assoc ':all-the-base-features-summary-restrictions parsed-rec))   nil)))




(defun read-comments (comments-string)
  (loop with input-stream = (make-string-input-stream
                              comments-string)

        for (word eof . rest) = (multiple-value-list
                                  (read-delimited-string '(#\space #\return #\line)
                                                           input-stream
                                                           () :eof))
        until eof
        unless (zerop (string-length word))
          collect word
            )
  )


                                         ;


(defun get-the-base-features-summary-comments-restrictions (parsed-record)
  (reset-base-features-summary-comment-restriction-keyword-counter)
  (loop for record-field in parsed-record
        if (eq (car record-field) :all-the-base-features-summary-restrictions)
          append (loop for restriction in (get-base-features-summary-restrictions
                                             (read-comments
                                             (second record-field)))
```



```
                 collect (list (make-new-base-features-summary-comment-restriction-keyword)
                               restriction) into result
                     finally (return  (list (list
                                         ':ALL-THE-BASE-FEATURES-SUMMARY-RESTRICTIONS
                                          result )))  ; '((:rest-of-comments))))
                 )
         else collect record-field))             ;




(defun get-base-features-summary-restrictions (list-of-strings)
  (cond
    ((null list-of-strings) nil)
    ((and
       (= (string-length (car list-of-strings))  4)
       (and
         (alpha-char-p   (aref (car list-of-strings) 0))   ;;;;;DIFFERENT    PATTERN
         (digit-char-p   (aref (car list-of-strings) 1))
         (digit-char-p   (aref (car list-of-strings) 2))
         (alpha-char-p   (aref (car list-of-strings) 3))
         )
       )
     (cons
       (car list-of-strings)
       (get-base-features-summary-restrictions (cdr list-of-strings))
       )
     )
    (t (get-base-features-summary-restrictions (cdr list-of-strings)))
    )
  )




(defun reset-base-features-summary-comment-restriction-keyword-counter ()
  (setq *base-features-summary-comment-restrictions-so-far* 0))


(defun make-new-base-features-summary-comment-restriction-keyword ()
  (intern (format nil "BASE-SUMMARY-COMMENT-RESTRICTIONS--d"
                  (incf *base-features-summary-comment-restrictions-so-far*))
          (find-package 'keyword)))



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;(defun filter (list  function)
;  (cond
;    ((null list)  nil)
;    (t (let ((head  (car list))
;             (tail  (cdr list)))
;         (if (funcall  function   head)
;             (cons
;               head
;               (filter tail  function)))))
;  ))

;;;;;;;;;;;;;;;;;;;;
```

```
(defun class-top (parsed-rec)
  (cond
    ((class parsed-rec))
    (t (list ':class " "))))




(defun class (parsed-rec)
  (loop for i in parsed-rec
        as var = (second (assoc (car i) parsed-rec))
        if (and
             (match '(|:| m c l a s s *) (zl:explode (car i)))
             (not (equal (aref var  0) #\space))
           )
          do
           (return (list ':class var))


))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun body-top (parsed-rec)
  (cond
    ((body parsed-rec))
    (t (list ':body " "))))



(defun body (parsed-rec)
  (loop for i in parsed-rec
        as var = (second (assoc (car i) parsed-rec))
        if (and
             (match '(|:| m b o d y *) (zl:explode (car i)))
             (not (equal (aref var 0) #\space))
           )
          do
           (return (list ':body var))
))

;;;;;;;;;;;;;

(defun family-top (parsed-rec)
  (cond
    ((family parsed-rec))
    (t (list ':family " "))))


(defun family (parsed-rec)
  (loop for i in parsed-rec
        as var = (second (assoc (car i) parsed-rec))
        if (and
             (match '(|:| m f a m *) (zl:explode (car i)))
             (not (equal (aref var  0) #\space))
           )
          do
           (return (list ':family var))
))


;;;;;;;;;;;;;

(defun capacity-top (parsed-rec)
  (cond
    ((capacity parsed-rec))
    (t (list ':capacity " "))))


(defun capacity (parsed-rec)
```

```
  (loop for i in parsed-rec
        as var = (second (assoc (car i) parsed-rec))
        if (and
             (match '(|:| c a p *) (zl:explode (car i)))
             (not (equal (aref var 0) #\space))
           )
          do
          (return (list ':capacity var))
))

;;;;;;;;;;;;;;;

(defun performance-top (parsed-rec)
  (cond
    ((performance parsed-rec))
    (t (list ':performance " "))))


(defun performance (parsed-rec)
  (loop for i in parsed-rec
        as var = (second (assoc (car i) parsed-rec))
        if (and
             (match '(|:| m p e r f *) (zl:explode (car i)))
             (not (equal (aref var 0) #\space))
           )
          do
          (return (list ':performance var))
))


(defvar *clear* '(:mclass1 :mclass2 :mclass3 :mclass4 :mclass5 :mclass6 :mclass7 :mclass8
                  :mclass9 :mclass10 :mclass11 :mclass12  :mclass13
                  :mclass14 :mclass15 :mclass16
                  :mbody1 :mbody2 :mbody2 :mbody3 :mbody4 :mbody5 :mbody6
                  :mfam1 :mfam2 :mfam3 :mfam4 :mfam5
                  :mcap1 :mcap2 :mcap3 :mcap3 :mcap4 :mcap5 :mcap6 :mcap7
                  :mperf1 :mperf2 :mperf3 :mperf4 :mperf5 :mperf6 :mperf7))


(defun clear-the-reduntant-slots (parsed-rec)
  (loop for keyword in *clear*
        collect
          (assoc keyword parsed-rec)))


(defun take (parsed-rec)
  (append
    (list
      (class-top   parsed-rec)
      (body-top   parsed-rec)
      (family-top    parsed-rec)
      (capacity-top    parsed-rec)
      (performance-top   parsed-rec))
    (set-difference
      parsed-rec
      (clear-the-reduntant-slots parsed-rec))
  )
)


(defun read-tape (&optional (file-number 2) to-stream)
  (with-open-stream (tape-stream (tape:make-stream
                                  :direction :input))
    (send tape-stream :rewind)
    ;; Initially Takis was interested in the second data file on the
    ;; tape so that meant skipping four physical files seperated by EOF
    ;; markers.
    (send tape-stream :skip-file  (+  (* 3 (- file-number 1))  1))
    (loop as record = (read-record tape-stream)
          while record do
      (when to-stream
        (print record to-stream)))))
```

```
(defun read-record (stream)
  ;; Skip past any non digit characters.
  (loop as char = (send stream :tyi)
        when (null char) do (return-from read-record nil)
        until (digit-char-p (ascii-to-char char))
        finally
          (send stream :untyi char))
  (let ((record-length-string (make-array 4 :element-type 'integer)))
    (send stream :string-in t record-length-string)
    (let* ((record-length (read-from-string (ascii-to-string record-length-string)))
           (record (make-array (- record-length 4) :element-type 'integer)))
      (send stream :string-in t record)
      (ascii-to-string record))))


(defun read-tape-to-file (filename file-number)
  (with-open-file (file-stream filename :direction :output)
    (read-tape file-number file-stream)))


(defun parse-record (string format)
  (loop as string-left = string then (substring string-left length)
        for (field-name length) in format
        collect (list field-name (substring string-left 0 length))))


(defun match (pattern   input)
  (cond
    ((and (null pattern) (null input))  t)
    ((or  (null pattern) (null input))  nil)

    ((or
         (equal (car pattern) '?)
         (equal (car pattern) (car input)))
                         (match (cdr pattern) (cdr input))
    )



    ((equal (car pattern) '*)
                         (cond
                           ((match (cdr pattern) (cdr input)))
                            ((match pattern (cdr input)))
                            ((match (cdr pattern) input))
                           )
                         ;;;)
    )

  )
)




;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(defflavor base-features-summary-chart-entry
        (all-the-base-features-summary-restrictions
         class
         body
         family
         capacity
         performance
         mmr
         mmc
         mrecno
         mdrive1
         mdrive2
         mtrans1
         mtrans2
         mauthin
```

```
        mauthout
        mrelin
        mrelout
        mdepin
        mdepout
        skip-space
        msales)


       ()
  :writable-instance-variables
  :initable-instance-variables)


(defvar *base-features-summary-chart-entries* nil)



(defmethod (make-instance base-features-summary-chart-entry) (&rest ignore)
  (push self *base-features-summary-chart-entries*))


(defvar *parsed-input-from-base-features-summary*   takis) ;;;;(try-base
                                                    ;;;"ul:>takis>data-xfbasdat-summary.lisp"))


(defvar *base-features-summary-chart-entry-by-class-table*
        (make-hash-table    :test #'string-equal))


(defmethod (add-to-base-features-summary-hash-table base-features-summary-chart-entry) ()
  (let ((entries-with-this-featcode
          (gethash  class
                    *base-features-summary-chart-entry-by-class-table*)))
    (setf (gethash  class
                    *base-features-summary-chart-entry-by-class-table*)
          (cons self entries-with-this-featcode))))




(defun make-base-features-summary-chart-entries ()
  (loop for entry in *parsed-input-from-base-features-summary*
        as new-instance = (make-instance 'base-features-summary-chart-entry)
        do
    (loop for (iv-id value) in entry
          do
      (case iv-id


        (:all-the-base-features-summary-restrictions
          (setf (base-features-summary-chart-entry-all-the-base-features-summary-restrictions
        new-instance) value))

        (:mmr
          (setf (base-features-summary-chart-entry-mmr                    new-instance) value))
        (:mmc
          (setf (base-features-summary-chart-entry-mmc                    new-instance) value))
        (:class
          (setf (base-features-summary-chart-entry-class                 new-instance) value))
        (:body
          (setf (base-features-summary-chart-entry-body                  new-instance) value))
        (:family
          (setf (base-features-summary-chart-entry-family                new-instance) value))
        (:capacity
          (setf (base-features-summary-chart-entry-capacity              new-instance) value))
        (:performance
          (setf (base-features-summary-chart-entry-performance           new-instance) value))
        (:mdrive1
         (setf (base-features-summary-chart-entry-mdrive1                new-instance) value))
        (:mdrive2
         (setf (base-features-summary-chart-entry-mdrive2                new-instance) value))

        (:mtrans1
```

```
          (setf (base-features-summary-chart-entry-mtrans1                  new-instance) value))          (:m
trans2
          (setf (base-features-summary-chart-entry-mtrans2                  new-instance) value))
        (:mauthin
          (setf (base-features-summary-chart-entry-mauthin              new-instance) value))
        (:mauthout
          (setf (base-features-summary-chart-entry-mauthout             new-instance) value))
        (:mrelin
          (setf (base-features-summary-chart-entry-mrelin             new-instance) value))
        (:mrelout
          (setf (base-features-summary-chart-entry-mrelout            new-instance) value))
        (:mdepin
          (setf (base-features-summary-chart-entry-mdepin                   new-instance) value))
        (:mdepout
          (setf (base-features-summary-chart-entry-mdepout                  new-instance) value))
        (:skip-space
          (setf (base-features-summary-chart-entry-skip-space               new-instance) value))
        (:msales
          (setf (base-features-summary-chart-entry-msales                  new-instance) value))
        (:mrecno
          (setf (base-features-summary-chart-entry-mrecno                   new-instance) value))

      )    ;  THE    "  case "
  ))      ;THE  2  LOOPS.

  ;; Now hash on featcode for all entries

  (loop for fc in *base-features-summary-chart-entries* do
    (add-to-base-features-summary-hash-table fc))
)



(defun reset-base-features-summary-chart-db ()
  (setq *base-features-summary-chart-entries* nil)
  (clrhash *base-features-summary-chart-entry-by-featcode-table*)
  (make-base-features-summary-chart-entries))
```

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER; Base: 10 -*-


;;; This program is designed to read a simple fixed format record tape
;;; containing data about additional features on Austin Rover Car
;;; Models. The tape seems to consist of several files. Each file is
;;; prefixed by a 'header file' and is followed by an 'end of file' file
;;; so: |header-file||actual data file||end of file file||header
;;; file||actual data file||end of file file||......
;;;

;;; This function reads the file that Takis was interested in from the
;;; sample tape. It prints the records in that file as string objects to
;;; the to-stream supplied as the optional argument.

(defun read-tape (&optional (file-number 2) to-stream)
  (with-open-stream (tape-stream (tape:make-stream
                                  :direction :input))
    (send tape-stream :rewind)
    ;; Initially Takis was interested in the second data file on the
    ;; tape so that meant skipping four physical files seperated by EOF
    ;; markers.
    (send tape-stream :skip-file  (+  (* 3 (- file-number 1))  1))
    (loop as record = (read-record tape-stream)
          while record do
      (when to-stream
        (print record to-stream)))))


;;; In the file that Takis was interested in, it appeared at first sight
;;; to be one long stream of ascii characters with no easy record
;;; boundary identifiers. After a while we spotted that most records
;;; begin with a four character record length number which is the number
;;; of characters in the record inclusive of the four. This was fine but
;;; there seemed to occasionally be a load of garbage in the middle of
;;; the tape like ^^^^^^^^^^^^^^^^^^^^^^^^. The simple solution to that
;;; was to read past any non digit characters before trying to read a
;;; record length field.

(defun read-record (stream)
  ;; Skip past any non digit characters.
  (loop as char = (send stream :tyi)
        when (null char) do (return-from read-record nil)
        until (digit-char-p (ascii-to-char char))
        finally
          (send stream :untyi char))
  (let ((record-length-string (make-array 4 :element-type 'integer)))
    (send stream :string-in t record-length-string)
    (let* ((record-length (read-from-string (ascii-to-string record-length-string)))
           (record (make-array (- record-length 4) :element-type 'integer)))
      (send stream :string-in t record)
      (ascii-to-string record))))

;;; Read the records off the tape as strings and outputs them to tape.

(defun read-tape-to-file (filename file-number)
  (with-open-file (file-stream filename :direction :output)
    (read-tape file-number file-stream)))

;;; Trivial format specification for structure of a record. Indicates
;;; sequential field lengths in terms of number of characters.

(defvar *record-format*
        '((:recno 3)
          (:featcode 4)

          (:standard 1)
          (:base 1)
          (:optional 1)
          (:marketing 1)
          (:envveng 1)
          (:legal 1)
          (:accessory 1)

          (:mr 2)
          (:mc 2)
```

```
(:class 1)
(:body 1)
(:family 1)
(:capacity 1)
(:performance 1)
(:drive 1)
(:trans 1)



(:c1   4)
(:c2   4)
(:c3   4)
(:c4   4)
(:c5   4)
(:c6   4)
(:c7   4)
(:c8   4)
(:c9   4)
(:c10  4)
(:c11  4)
(:c12  4)
(:c13  4)
(:c14  4)
(:c15  4)
(:c16  4)
(:c17  4)
(:c18  4)
(:c19  4)
(:c20  4)

(:q1   1)
(:q2 1)
(:q3 1)
(:q4 1)
(:q5 1)
(:q6 1)
(:q7 1)
(:q8 1)
(:q9 1)
(:q10 1)
(:q11 1)
(:q12 1)
(:q13 1)
(:q14 1)
(:q15 1)
(:q16 1)
(:q17 1)
(:q18 1)
(:q19 1)
(:q20 1)
(:q21 1)
(:q22 1)
(:q23 1)
(:q24 1)
(:q25 1)

(:auth-ref-in 12)
(:auth-ref-out 12)

(:rel-stage-in 1)
(:rel-stage-out 1)

(:dep-in 8)
(:dep-out 8)

(:terr-def 1)
(:feat-group 3)

(:tergrp1 4)
(:tergrp2 4)
(:tergrp3 4)
(:tergrp4 4)
(:tergrp5 4)
```

```
          (:tergrp6 4)
          (:tergrp7 4)
          (:tergrp8 4)
          (:tergrp9 4)
          (:tergrp10 4)
          (:tergrp11 4)
          (:tergrp12 4)
          (:tergrp13 4)
          (:tergrp14 4)
          (:tergrp15 4)
          (:tergrp16 4)
          (:tergrp17 4)
          (:tergrp18 4)

          (:terrcode_except1 4)
          (:terrcode_except2 5)
          (:terrcode_except3 5)
          (:tq1 1)
          (:terrcode_except4 4)
          (:terrcode_except5 5)
          (:terrcode_except6 5)
          (:tq2 1)
          (:terrcode_except7 4)
          (:terrcode_except8 5)
          (:terrcode_except9 5)
          (:tq3 1)
          (:terrcode_except10 4)
          (:terrcode_except11 5)
          (:terrcode_except12 5)
          (:tq4 1)
          (:terrcode_except13 4)
          (:terrcode_except14 5)
          (:terrcode_except15 5)
          (:tq5 1)
          (:terrcode_except16 4)
          (:terrcode_except17 5)
          (:terrcode_except18 5)

          (:history 1)
          (:errno 2)))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(defvar *proper-record-format*
        '((:recno 3)
          (:featcode 4)

          (:standard 1)
          (:base 1)
          (:optional 1)
          (:marketing  1)
          (:enveng  1)
          (:legal  1)
          (:accessory  1)

          (:mr  2)
          (:mc 2)

          (:class 1)
          (:body  1)
          (:family 1)
          (:capacity  1)
          (:performance  1)
          (:drive  1)
          (:trans 1)

          (:q1    1)
          (:c1 4)

          (:q2  1)
          (:c2 4)

          (:q3  1)
```

```
(:c3  4)

(:q4  1)
(:c4  4)

(:q5  1)   ;;;;;;;;;;
(:q6 1)

(:c5 4)
(:q7 1)

(:c6 4)
(:q8 1)

(:c7 4)
(:q9 1)


(:c8 4)
(:q10 1)

(:q11 1)
(:c9 4)

(:q12 1)
(:c10 4)

(:q13 1)
(:c11 4)

(:q14 1)
(:c12 4)

(:q15 1)      ;;;;;;;;;
(:q16 1)

(:c13 4)
(:q17 1)

(:c14 4)
(:q18 1)

(:c15 4)
(:q19 1)

(:c16 4)
(:q20 1)

(:q21 1)
(:c17 4)

(:q22 1)
(:c18 4)

(:q23 1)
(:c19   4)

(:q24 1)
(:c20 4)
(:q25 1)


(:auth-ref-in  12)
(:auth-ref-out  12)

(:rel-stage-in  1)
(:rel-stage-out  1)

(:dep-in 8)
(:dep-out  8)

(:terr-def 1)
(:feat-group 3)

(:tergrpl 4)
```

```
          (:tergrp2  4)
          (:tergrp3 4)
          (:tergrp4 4)
          (:tergrp5 4)
          (:tergrp6 4)
          (:tergrp7 4)
          (:tergrp8 4)
          (:tergrp9 4)
          (:tergrp10 4)
          (:tergrp11 4)
          (:tergrp12 4)
          (:tergrp13 4)
          (:tergrp14 4)
          (:tergrp15 4)
          (:tergrp16 4)
          (:tergrp17 4)
          (:tergrp18 4)

          (:terrcode_except1 4)
          (:terrcode_except2 5)
          (:terrcode_except3 5)
          (:tq1 1)
          (:terrcode_except4 4)
          (:terrcode_except5 5)
          (:terrcode_except6 5)
          (:tq2 1)
          (:terrcode_except7 4)
          (:terrcode_except8 5)
          (:terrcode_except9 5)
          (:tq3 1)
          (:terrcode_except10 4)
          (:terrcode_except11 5)
          (:terrcode_except12 5)
          (:tq4 1)
          (:terrcode_except13 4)
          (:terrcode_except14 5)
          (:terrcode_except15 5)
          (:tq5 1)
          (:terrcode_except16 4)
          (:terrcode_except17 5)
          (:terrcode_except18 5)

          (:history 1)
          (:errno 2)
))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defvar *the-proper-record-format*
        '(:recno
          :featcode

          :standard
          :base
          :optional
          :marketing
          :enveng
          :legal
          :accessory

          :mr
          :mc 2

          :class
          :body
          :family
          :capacity
          :performance
          :drive
          :trans

          :q1
          :c1

          :q2
```

```
:c2

:q3
:c3

:q4
:c4

:q5      ;;;;;;;;;;
:q6

:c5
:q7

:c6
:q8

:c7
:q9


:c8
:q10

:q11
:c9

:q12
:c10

:q13
:c11

:q14
:c12

:q15       ;;;;;;;;;
:q16

:c13
:q17

:c14
:q18

:c15
:q19

:c16
:q20

:q21
:c17

:q22
:c18

:q23
:c19

:q24
:c20
:q25


:auth-ref-in
:auth-ref-out

:rel-stage-in
:rel-stage-out

:dep-in
:dep-out

:terr-def
```

```
            :feat-group

            :tergrp1
            :tergrp2
            :tergrp3
            :tergrp4
            :tergrp5
            :tergrp6
            :tergrp7
            :tergrp8
            :tergrp9
            :tergrp10
            :tergrp11
            :tergrp12
            :tergrp13
            :tergrp14
            :tergrp15
            :tergrp16
            :tergrp17
            :tergrp18

            :terrcode_except1
            :terrcode_except2
            :terrcode_except3
            :tq1
            :terrcode_except4
            :terrcode_except5
            :terrcode_except6
            :tq2
            :terrcode_except7
            :terrcode_except8
            :terrcode_except9
            :tq3
            :terrcode_except10
            :terrcode_except11
            :terrcode_except12
            :tq4
            :terrcode_except13
            :terrcode_except14
            :terrcode_except15
            :tq5
            :terrcode_except16
            :terrcode_except17
            :terrcode_except18

            :history
            :errno
))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(defvar *new-record-format*
        '((:recno 3)
          (:featcode 4)

          (:standard 1)
          (:base 1)
          (:optional 1)
          (:marketing 1)
          (:enveng 1)
          (:legal 1)
          (:accessory 1)

          (:mr 2)
          (:mc 2)

          (:class 1)
          (:body 1)
          (:family 1)
          (:capacity 1)
          (:performance 1)
          (:drive 1)
          (:trans 1)
```

```
         (:comments 105)


         (:auth-ref-in 12)
         (:auth-ref-out 12)

         (:rel-stage-in 1)
         (:rel-stage-out 1)

         (:dep-in 8)
         (:dep-out 8)

         (:terr-def 1)
         (:feat-group 3)

         (:tergrp1 4)
         (:tergrp2 4)
         (:tergrp3 4)
         (:tergrp4 4)
         (:tergrp5 4)
         (:tergrp6 4)
         (:tergrp7 4)
         (:tergrp8 4)
         (:tergrp9 4)
         (:tergrp10 4)
         (:tergrp11 4)
         (:tergrp12 4)
         (:tergrp13 4)
         (:tergrp14 4)
         (:tergrp15 4)
         (:tergrp16 4)
         (:tergrp17 4)
         (:tergrp18 4)

         (:terrcode_except1 4)
         (:terrcode_except2 5)
         (:terrcode_except3 5)
         (:tq1 1)
         (:terrcode_except4 4)
         (:terrcode_except5 5)
         (:terrcode_except6 5)
         (:tq2 1)
         (:terrcode_except7 4)
         (:terrcode_except8 5)
         (:terrcode_except9 5)
         (:tq3 1)
         (:terrcode_except10 4)
         (:terrcode_except11 5)
         (:terrcode_except12 5)
         (:tq4 1)
         (:terrcode_except13 4)
         (:terrcode_except14 5)
         (:terrcode_except15 5)
         (:tq5 1)
         (:terrcode_except16 4)
         (:terrcode_except17 5)
         (:terrcode_except18 5)

         (:history 1)
         (:errno 2)))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defvar *global-var* nil)



(defvar *comment-restrictions-so-far* 0)
(defvar *combinations-restrictions* 0)
(defvar *territory-restrictions* 0)

(defvar *q* '(:q1 :q2 :q3 :q4 :q5 :q6 :q7 :q8 :q9 :q10 :q11 :q12 :q13 :q14 :q15
```

```
                :q16 :q17 :q18 :q19 :q20 :q21 :q22 :q23 :q24 :q25
 ))
(defvar *c* '(:c1 :c2 :c3 :c4 :c5 :c6 :c7 :c8 :c9 :c10 :c11 :c12 :c13 :c14 :c15 :c16 :c17 :
              :c18 :c19 :c20))

(defvar *except* '(:tergrp1 :tergrp2 :tergrp3 :tergrp4
                   :tergrp5 :tergrp6 :tergrp7 :tergrp8
                   :tergrp9 :tergrp10 :tergrp11
                   :tergrp12
                   :tergrp13 :tergrp14 :tergrp15
                   :tergrp16
                   :tergrp17 :tergrp18))



(defvar *g* '(:tergrp1 :tergrp2 :tergrp3 :tergrp4 :tergrp5 :tergrp6 :tergrp7 :tergrp8 :tergrp9
    :tergrp10 :tergrp11 :tergrp12 :tergrp13 :tergrp14 :tergrp15 :tergrp16 :tergrp17
                  :tergrp18))


(defvar *tq-ls* '(:tq1 :tq2 :tq3 :tq4 :tq5))
(defvar *ter-ls* '(:terrcode_except1 :terrcode_except2 :terrcode_except3 :terrcode_except4
                   :terrcode_except5 :terrcode_except6 :terrcode_except7 :terrcode_except8
                   :terrcode_except9 :terrcode_except10 :terrcode_except11 :terrcode_except12
                   :terrcode_except13 :terrcode_except14 :terrcode_except15 :terrcode_except16
                   :terrcode_except17 :terrcode_except18
                   ))



;*****************************************************************************************


(defun parse-record (string format)
  (loop as string-left = string then (substring string-left length)
        for (field-name length) in format
        collect (list field-name (substring string-left 0 length))))



(defun try (filename)
  (with-open-file (stream filename :direction :input)

    (loop ;;;;;;repeat 10
          as rec = (read stream  nil :eof );                                  ;;string
          until (eq rec :eof)
          as variable = (parse-record  rec *record-format*)                 ;;;;assoc-list
        as result = (return-the-right-order variable *the-proper-record-format*)  ;;;assoc-list
          as again = (back-to-string  result)                                ;;;;string
          as var = (parse-record  again  *proper-record-format*)            ;;;assoc-list


          if (check var)
            collect
             (the-last
              (get-the-territory-restrictions
               (get-the-comments-restrictions
                      (parse-record again *new-record-format*))))
            else
            collect
             (the-last
```

```
            (get-the-territory-restrictions
             (get-the-combinations-restrictions  var)))


     )
   ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;



(defun reset-combinations-restrictions-keyword-counter ()
  (setq *combinations-restrictions* 0))


(defun make-new-combinations-restrictions-keyword ()
  (intern (format nil "COMBINATIONS-RESTRICTIONS-~d" (incf *combinations-restrictions*))
          (find-package 'keyword)))



;


(defun get-the-combinations-restrictions (var1)
  (reset-combinations-restrictions-keyword-counter)
  (append (out-the-keywords var1)
                                                  ;
          (loop for restriction in (combinations-restrictions var1)
                collect (list (make-new-combinations-restrictions-keyword)
                              restriction))))




(defun combinations-restrictions (var1)
  (prog (help result)
   (setq help (find-the-combinations-restrictions
                               *q* *c* t var1))
   LOOP
   (cond
     ((null help) (return (clean-nil  nil result)))
     (t (setq result
             (cons (clean-empty-strings (car help))
                   result))
     ))
   (setq help (cdr help))
   (go LOOP)
 ))


;(defvar var nil)



(defun find-the-combinations-restrictions (q-ls  c-ls first-time-p var1)
  (macrolet ((line-list ()
             '(list (second (assoc (first q-ls) var1))  (second (assoc (first c-ls) var1))
                    (second (assoc (second q-ls) var1)) (second (assoc (second c-ls) var1))
                    (second (assoc (third q-ls) var1))  (second (assoc (third c-ls) var1))
                    (second (assoc (fourth q-ls) var1)) (second (assoc (fourth c-ls) var1))
                    )))
    (cond
```

```
(
          (null q-ls) nil)

((cond
   ((and first-time-p
         (eq (aref (second (assoc (car q-ls) varl)) 0)   #\space))  nil)
   (t (cond
        ((equal (second (assoc (sixth q-ls) varl)) nil) nil);;;;;;;;;;;;;;;
        ((not (eq (aref (second (assoc (sixth q-ls) varl)) 0)   #\space))
         (cons (line-list)
               (find-the-combinations-restrictions
                                (cdr (cdr (cdr (cdr (cdr q-ls)))))
                                (cdr (cdr (cdr (cdr c-ls))))
                                nil
                                varl)
               ))
        (t (cond
             ((eq (aref (second (assoc (fifth q-ls) varl)) 0)   #\space)
              (list (line-list)))

             (t
              (let ((rest-of-result (find-the-combinations-restrictions
                                                  (cdr (cdr (cdr (cdr (cdr q-ls)))))
                                                  (cdr (cdr (cdr (cdr c-ls))))
                                                  nil
                                                  varl
                                                  )))
                                          ;(append
                 (cons
                   (append
                     (line-list)


                     (car rest-of-result))
                   (cdr rest-of-result))
                 )))))))))))
```

```
(defun clean-empty-strings (list-of-strings)
  (cond
    ((null list-of-strings) nil)
    ((not (empty-string (car list-of-strings) )) (clean-empty-strings (cdr list-of-strings)))
    (t (cons
         (car list-of-strings)
         (clean-empty-strings (cdr list-of-strings))))
  )
)
```

```
(defun empty-string (a-string)
  (loop for i from 0 to (- (string-length a-string) 1)
        unless (eq (aref a-string i) #\space)
        collect
       (aref a-string i)
       ))
```

```
(defun out-the-keywords (varl)
  (setq *global-var* nil)
  (set-difference
```

```
    var1
    (apply 'append (list
                     (collect-qualifiers var1)
                     (collect-territories  var1)))
  ))



(defun collect-qualifiers ( var1)
  (setq *global-var* var1)

        (mapcar '(lambda (x) (assoc x *global-var*))
                *q*)

)



(defun collect-territories ( var1)
  (setq *global-var* var1)
        (mapcar '(lambda (x) (assoc x *global-var*))
                *c*)

)



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;



(defun read-comments (comments-string)
  (loop with input-stream = (make-string-input-stream
                              comments-string)

        for (word eof . rest) = (multiple-value-list
                                  (read-delimited-string '(#\space #\return #\line)
                                                         input-stream
                                                         () :eof))
        until eof
        unless (zerop (string-length word))
          collect word
            )
  )



(defun get-the-comments-restrictions (parsed-record)
  (reset-comment-restriction-keyword-counter)
```

```
  (loop for record-field in parsed-record
        if (eq (car record-field) :comments)
          append (loop for restriction in (get-restrictions (read-comments
                                                               (second record-field)))
                   collect (list (make-new-comment-restriction-keyword) restriction) into result
                      finally (return  result )  ; '((:rest-of-comments))))
                 )
        else collect record-field))              ;



;  (reset-combinations-restrictions-keyword-counter)
;  (loop for record-field in parsed-record
;        if (eq (car record-field) :comments)
;          append (loop for restriction in (get-restrictions (read-comments parsed-record))
;                 collect (list (make-new-comment-restriction-keyword) restriction) into result)
;        else collect record-field))



(defun get-restrictions (list-of-strings)
  (cond
    ((null list-of-strings) nil)
    ((and
       (= (string-length (car list-of-strings))  3)
       (and
         (alpha-char-p   (aref (car list-of-strings) 0))
         (digit-char-p   (aref (car list-of-strings) 1))
         (digit-char-p   (aref (car list-of-strings) 2))
         )
       )
     (cons
       (car list-of-strings)
       (get-restrictions (cdr list-of-strings))
       )
     )
    (t  (get-restrictions (cdr list-of-strings)))
    )
  )



(defun reset-comment-restriction-keyword-counter ()
  (setq *comment-restrictions-so-far* 0))

(defun make-new-comment-restriction-keyword ()
  (intern (format nil "COMMENT-RESTRICTIONS--~d" (incf *comment-restrictions-so-far*))
          (find-package 'keyword)))



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;



(defun check (string)
       (cond
         ((string-equal  "C"   (cadr (assoc ':ql string)))
                   t)
```

```
        (t
          nil))
)




(defun back-to-string (a-list-of-key-strings)
  (loop with result = ""
        for pair in a-list-of-key-strings
        do (setq result
                 (concatenate 'string
                              result
                              (second pair)
                              ))
           finally
             (return result)
))


(defun return-the-right-order (list-of-strings string-order)
  (cond
    ((null string-order) nil)
    (t (cons
         (assoc (car  string-order) list-of-strings)
         (return-the-right-order list-of-strings (cdr string-order))
       )))
)



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(defun get-the-territory-restrictions (var)

(remove ':terr-def
  (append

   (set-difference
      var
      (collect-territories-except var))
   (territorial-restrictions var)
   (list  (list ':ALL-THE-GROUP-RESTRICTIONS (list (second (assoc ':terr-def var))
                                                 (collect-groups var))))
  )
           :test #'string-equal  :key #'first ))




;(defun put-groups-together (var)
;
;  (prog (help result)
;   (setq help (collect-groups var))
;
;   LOOP
;   (cond
;     ((null help) (return(apply 'append
;                               (list
;                                 (list (second (assoc ':terr-def var))) result))))
;                               result))
;     (t (setq result
;             (cons (cadr (car help))
;                   result))
;     ))
;   (setq help (cdr help))
;   (go LOOP)
; ))
```

```
(defun collect-territories-except ( var)
  (setq *global-var* var)
  (append

    (mapcar '(lambda (x) (assoc x *global-var*))
            *tq-ls*)

    (mapcar '(lambda (x)
                     (assoc x *global-var*))
            *except*)
    (mapcar '(lambda (x)
                     (assoc x *global-var*))
            *ter-ls*)
    ;;;;;(setq *global-var* nil)   ;;;;;;;;;;;;;;;;;;;;;;88888888888888888888

  )
)




(defun clean-nil (a ls)
  (cond
    ((null ls) nil)
    ((equal a (car ls)) (clean-nil a (cdr ls)))
    (t (cons (car ls)
             (clean-nil a (cdr ls))))))



(defun collect-groups ( var)
  (setq *global-var* var)
  (clean-nil nil
        (mapcar '(lambda (x)
                     (cond ((empty-string (second (assoc x *global-var*)))
                                          (second (assoc x *global-var*))))))
                *g*))
  ;;;;;;;;;;(setq *global-var* nil)
)



(defun collect-groups-help ( var)
  (setq *global-var* var)
       (mapcar '(lambda (x)
                     (cond ((empty-string (second (assoc x var))) (assoc x *global-var*))))
*g*)
   ;;;;;;;;(setq *global-var* nil)
)




;*****************************************************************************************
```

```
(defun reset-territory-restrictions-keyword-counter ()
  (setq *territory-restrictions* 0))


(defun make-new-territory-restrictions-keyword ()
  (intern (format nil "TERRITORY-RESTRICTIONS--d" (incf *territory-restrictions*))
          (find-package 'keyword)))
```

```
;


(defun territorial-restrictions (var)
  (reset-territory-restrictions-keyword-counter)

          (loop for restriction in (the-territory-restrictions var)
                collect (list (make-new-territory-restrictions-keyword)
                              restriction)))


(defun the-territory-restrictions (var)

  (prog (help result)
    (setq help (find-the-territory-restrictions
                                *tq-ls* *ter-ls* t var))
    ;(cond ((equal help nil) (return nil)))
   LOOP
   (cond
     ((null help) (return result))
     (t (setq result
              (cons (clean-empty-strings (car help))
                    result))
     ))
   (setq help (cdr help))
   (go LOOP)
))




(defun find-the-territory-restrictions (tq-ls  ter-ls   first-time-p var)
  (macrolet ((help-list ()
               '(list (second (assoc (first ter-ls) var))
                      (second (assoc (second ter-ls) var))
                      (second (assoc (third ter-ls) var))
                      (second (assoc (first tq-ls) var))
                      )))
    (cond
      ((null ter-ls)
                      nil)
      ((cond
         ((and first-time-p
               (eq (aref (second (assoc (car ter-ls) var)) 0)   #\space))  nil)
         (t (cond
              ;((equal (second (assoc (first tq-ls) var)) nil) nil)
              ((not (eq (aref (second (assoc (first tq-ls) var)) 0)   #\space))
                   (let ((rest-of-result (find-the-territory-restrictions
                                                        (cdr tq-ls)
                                                        (cdr (cdr (cdr ter-ls)))
                                                        nil
                                                        var
                                                        )))
                                                 ;(append
                      (cons
                        (append
                          (help-list)


                          (car rest-of-result))
                        (cdr rest-of-result))
                  )
              )
              (t (cond
                   ((eq (aref (second (assoc (fourth ter-ls) var)) 0)   #\space)
                    (list (help-list)))

                   (t (cons (help-list)
                     (find-the-territory-restrictions
                                    (cdr tq-ls)
                                    (cdr (cdr (cdr ter-ls)))
```

```
                                        nil
                                        var)
                      ))
                     )
                 )
               )
          ))))))




;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun the-last (var)
  (append
    (list
     (list ':ALL-THE-COMBINATIONS-RESTRICTIONS (combinations-restrictions-all var))
     (list ':ALL-THE-COMMENTS-RESTRICTIONS (comments-restrictions-all   var))
     (list ':ALL-THE-TERRITORY-RESTRICTIONS (territory-restrictions-all  var)))

    (set-difference
      var
        (append
          (combinations-restrictions-all var)
          (comments-restrictions-all   var)
          (territory-restrictions-all  var)))
  )
)




(defun match (pattern   input)
  (cond
    ((and (null pattern) (null input))  t)
    ((or  (null pattern) (null input))  nil)

    ((or
         (equal (car pattern) '?)
         (equal (car pattern) (car input)))
                        (match (cdr pattern) (cdr input))
    )



    ((equal (car pattern) '*)
                        (cond
                          ((match (cdr pattern) (cdr input)))
                           ((match pattern (cdr input)))
                           ((match (cdr pattern) input))
                          )
                        ;;;)
    )

  )
)



(defun combinations-restrictions-all (var)
  (cond
    ((null var) nil)
    ((match '(|:| c o m b i n a t i o n s - r e s t r i c t i o n s - ?)
              (zl:explode (caar var)))

       (cons (car var)
```

```
            (combinations-restrictions-all  (cdr var))))
    (t (combinations-restrictions-all (cdr var)))))



(defun comments-restrictions-all (var)
  (cond
    ((null var) nil)
    ((match '(|:| c o m m e n t - r e s t r i c t i o n s - ?)
                (zl:explode (caar var)))

       (cons (car var)
            (comments-restrictions-all  (cdr var))))
    (t (comments-restrictions-all (cdr var))))
)



(defun territory-restrictions-all (var)
  (cond
    ((null var) nil)
    ((match '(|:| t e r r i t o r y - r e s t r i c t i o n s - ?)
                (zl:explode (caar var)))

       (cons (car var)
            (territory-restrictions-all  (cdr var))))
    (t (territory-restrictions-all (cdr var))))
)
```

APPENDIX 4

LISTING OF THE VALIDATION ALGORITHM

```
 ;;;-*-  Mode: lisp;  Package: austin-rover ; syntax: common-lisp-*-
;(defvar *vpg-regions* nil)
;(defvar *part-descriptions* nil)
;(defvar *vpg-SUB-regions* nil)
;
;
;(defflavor vpg-region
;        (;;VPG-name                                  ;NEVER change
;         vpg-code                                 ;NEVER change
;         base-features                            ;NEVER change
;         additional-feature-groups                ;NEVER change
;         HAS-vpg-SUBGROUPS                        ;NEVER change
;         )
;        ()
;        :initable-instance-variables
;        :writable-instance-variables)
;
;
;
;(defmethod (make-instance vpg-region) (&rest ignore)
;  (push self *vpg-regions*))
;
;
;
;(defflavor vpg-SUBgroup
;        (SUBGROUP-CODE-name
;         MUST-additional-features
;         MAYBES-additional-features
;         BELONGS-to-VPG-region
;         has-part-descriptions)
;        ()
;  :initable-instance-variables
;  :writable-instance-variables)
;
;
;
;
;
;(defflavor part-description
;        (description-name                                          ;NEVER change.
;         NEGATIVE-additional-features
;         BELONGS-TO-VPG-SUB-group)
;        ()
;  :initable-instance-variables
;  :readable-instance-variables
;  :writable-instance-variables)
;
;(defmethod (sys:print-self vpg-region) (stream &rest ignore)
;  (format stream "#<vpg-region ~A >" VPG-code))
;
;
;(defmethod (sys:print-self vpg-SUBgroup) (stream &rest ignore)
;  (format stream "#<vpg-SUBGROUP ~A >" SUBGROUP-CODE-name))
;
;
;(defmethod (sys:print-self part-description) (stream &rest ignore)
;  (format stream "#<part-description ~A >" description-name))
;
;
;
;(defmethod (make-instance vpg-region) (&rest ignore)
;  (push self *vpg-regions*))
;
;(defmethod (make-instance vpg-SUBgroup) (&rest ignore)
;  (push self *vpg-SUB-regions* ))
;
;(defmethod (make-instance part-description) (&rest ignore)
;  (push self *part-descriptions*))
;
;
; ;vpg-SUBgroup
;;*****************************************************************************
;;*****************************************************************************
;;*****************************************************************************
;
```

```
;;;;;(defvar *the-pathname-of-the-ROVERs-APN-CATALOGUE* nil)

;;IT IS the  TOP function  :  "given the Click ie. the "part-description"  RETURNS
;;  a list of (the-actual-instance-in-memory-with-that-name,  part-description
;;              the-superior-VPG-instance  it belongs          vpg-regior
;;              the MUST features IF there are any in  MEMORY,for that description.


(defvar *the-pathname-of-the-ROVERs-APN-CATALOGUE*   *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pat
hname*)
;;;;      ie ......  #p"stat-FS:>takis>ROVER-APN")



(defmethod (sys:print-self Secondary-Description-ID) (stream &rest ignore)
  (with-database  (db   *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*)
                  (with-transaction ()
                    (format stream "#<PART-DESCRIPTION:     -A >"  (Secondary-Description-ID-PART-DESCRIPT
ION  self)))))



(defun work  (PART-DESCRIPTION-we-CONCERN)
  (let* ((NEGATIVES (get-negatives-of-that-DESCRIPTION PART-DESCRIPTION-we-CONCERN))
         (musts (get-MUSTS-TOP *the-pathname-of-the-ROVERs-APN-CATALOGUE*  PART-DESCRIPTION-we-CONCERN))
         (maybes (get-Maybes-TOP  *the-pathname-of-the-ROVERs-APN-CATALOGUE* PART-DESCRIPTION-we-CONCERN
)))
    (setq *NEGATIVES*  NEGATIVES)
;    (setq *MUST*  musts)
;     (setq *MAYBES*    maybes)
      (setq *MUST* (remove-duplicates (append musts
                                              (DONT-forget-the-FEATURES-WHICH-REPRESENT-the-SAME-logic-in
-Specification musts))
                                       :test #'string-equal))
      (setq *MAYBES* (remove-duplicates (append maybes
                                              (DONT-forget-the-FEATURES-WHICH-REPRESENT-the-SAME-logic-in
-Specification maybes))
                                       :test #'string-equal))
    ))



;Command: (get-MUSTS-TOP *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname* #<PART-DESCRIPTION:
  PIVOT-SPHERICAL
;  EXT MIRROR ASSY >)
;("NO-SPECIFICATION")
;Command: (get-MUSTS-TOP *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname* #<PART-DESCRIPTION:
  PIVOT-GLASS
;  EXT MIRROR ASSY >)
;NIL
;Command: (get-MUSTS-TOP *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname* [Abort]
;Back to Lisp Top Level in Dynamic Lisp Listener 1.
;
;Command: (get-MaybeS-TOP *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname* #<PART-DESCRIPTION:
   PIVOT-SPHERICAL
;   EXT MIRROR ASSY >)
;("NO-SPECIFICATION")
;Command:
;



;(defvar *FEATURE-GROUPS-WHICH-REPRESENT-THE-SAME-logical-GROUP*   ;;;;;;; FROM the file : "statice-R8.da
t"
;       ' (  (("B40" "B51") "B51"));;;; others if EXIST
;))  ;;;WE need it here in order to   RUN  the program  !!!!!


(defun DONT-forget-the-FEATURES-WHICH-REPRESENT-the-SAME-logic-in-Specification (ls)  ;;;;; *MUST* or *MA
```

```
YBES*
  (let ((result nil))
    (loop for feature-Master-group in  ls
          do
      (loop for i in *FEATURE-GROUPS-WHICH-REPRESENT-THE-SAME-logical-GROUP*
            do
        (cond
          ((string-equal (car (reverse i))  feature-Master-group)
                 (push (car i) result)))))
    (only-atoms result)))



;(defun work  (PART-DESCRIPTION-we-CONCERN)
;  ;(let* (;;;;;;(stream (dw:get-program-pane 'pane-3))
;
;    (let* ((NEGATIVES (get-negatives-of-that-DESCRIPTION PART-DESCRIPTION-we-CONCERN))
;           (VPG-SUBGROUP-INSTANCE (part-description-BELONGS-TO-VPG-SUB-group
;                                                                PART-DESCRIPTION-we-CONCERN))
;           (musts  (vpg-SUBgroup-MUST-additional-features  VPG-SUBGROUP-INSTANCE))
;           (maybes (vpg-SUBgroup-MAYBES-additional-features VPG-SUBGROUP-INSTANCE))
;           )
;      ;;(print musts)
;      ;;;(print maybes)
;      ;(setq *part-description-in-concern*   PART-DESCRIPTION-we-CONCERN)
;     ;(print NEGATIVES)
;      (setq *NEGATIVES*  NEGATIVES)
;
;      (setq *MUST*  musts)
;      (setq *MAYBES*    maybes)
;
;
;      ;;;(update-the-NEGATIVES-of-DESCRIPTION  *part-description-in-concern*)
; ));)

(defmethod     (get-negatives-of-that-DESCRIPTION  Secondary-Description-ID) ()
  (with-database (db *the-pathname-of-the-ROVERs-APN-CATALOGUE*)
    (with-transaction ()
;;;;;;;;;;;;;; We  DON'T need to check if it is UNBOUND because it is BOUND automatically from Statice.
;                (cond
;                  ((not (boundp-in-instance self 'NEGATIVE-additional-features))
;                       (setf (Secondary-Description-ID-NEGATIVE-additional-features  self)  NIL))
;                  (t
(Secondary-Description-ID-NEGATIVE-additional-features   self)
)))


;(defmethod      (get-negatives-of-that-DESCRIPTION  part-description) ()
;                (cond
;                  ((not (boundp-in-instance self 'NEGATIVE-additional-features))
;                       (setf (part-description-NEGATIVE-additional-features  self)  NIL))
;                  (t (part-description-NEGATIVE-additional-features  self)))
;)


;Command: (work #<PART-DESCRIPTION:    SEAT-FRONT COMPLETE   MANUAL-PASSR    >)
;("B51" "B44")
;("B69" "B53" "B51")
;NIL
;Command:
;Command: (setq *NEGATIVES-WHICH-WE-GAVE* nil)
;NIL
;Command: (update-the-NEGATIVES-of-DESCRIPTION #<PART-DESCRIPTION:    SEAT-FRONT COMPLETE   MANUAL-PASSR
   >)
;NIL
;Command: (work #<PART-DESCRIPTION:    SEAT-FRONT COMPLETE   MANUAL-PASSR    >)
;("B51" "B44")
;("B69" "B53" "B51")
;NIL
;Command:

;&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
;The Below function is  WRONG because Statice DOES NOT  accept LISTS.
```

```
(defmethod  (update-the-NEGATIVES-of-DESCRIPTION Secondary-Description-ID) ()
  (with-database (db *the-pathname-of-the-ROVERs-APN-CATALOGUE*)
         (with-transaction ()
           (let* ((temp (get-the-NEGATIVES-OF-THE-entity-type-FROM-STATICE-TOP  self))
                  (total-negatives (append
                                    temp
                                    *NEGATIVES-WHICH-WE-GAVE*))
                  (length-of-vector-to-store-all-negatives (+ 1 (length total-negatives)))
                  (negatives-in-statice-format (let ((result nil))
                                                 (loop for string-feature in total-negatives
                                                       do
                                                   (push (from-feature-string-to-feature-number string-f
eature) result))
                                                 result)))
             (setf (Secondary-Description-ID-NEGATIVE-additional-features  self)
                  (make-array length-of-vector-to-store-all-negatives
                              :element-type 'fixnum
                              :initial-contents  (push length-of-vector-to-store-all-negatives negatives
-in-statice-format)))))))


;from-feature-number-to-feature-string
;from-feature-string-to-feature-number
;(defun  ASSIGN-the-VALUE-VECTOR-of-R8-musts-to-THE-ROVER-CARS----actually-Austin-Rover-entity
;                (pathname THE-ROVER-CARS----actually-Austin-Rover    VECTOR-OF-STRINGS)
;  (let ((help nil))
;    (loop for FEATURE-GROUP-STRING   in VECTOR-OF-STRINGS
;          do
;      (push (given-a-FEATURE-FROUP-STRING-return-the-NUMBER FEATURE-GROUP-STRING) help))
;    (let* ((length-of-array (+ 1 (length help)))
;           (array-elements (push length-of-array help))
;           (VECTOR-value-gone-in-statice
;             (make-array length-of-array :element-type 'fixnum :initial-contents array-elements)))
;      (with-database (db pathname)
;       (with-transaction ()
;         (setf (THE-ROVER-CARS----actually-Austin-Rover-New-ROVER-200-or-ROVER-400--XW-musts
;                 THE-ROVER-CARS----actually-Austin-Rover)
;               VECTOR-value-gone-in-statice))))))


(defun get-the-NEGATIVES-OF-THE-entity-type-FROM-STATICE-TOP (Secondary-Description-ID)
  (let ((result nil) (temp (get-the-NEGATIVES-OF-THE-entity-type-FROM-STATICE Secondary-Description-ID  )
))
    (loop for feature-code-number in temp
          do
      (push (from-feature-number-to-feature-string feature-code-number) result))
result))



(defun get-the-NEGATIVES-OF-THE-entity-type-FROM-STATICE (Secondary-Description-ID)
  (let* ((the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-NEGATIVES
                (Secondary-Description-ID-NEGATIVE-ADDITIONAL-FEATURES  Secondary-Description-ID)))
    (block TAKIS
      (cond ((equal the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-NEGATIVES nil)
               (return-from   get-the-NEGATIVES-OF-THE-entity-type-FROM-STATICE nil)))
      (let ((VECTOR-dimensions (aref the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-N
EGATIVES 0)))
      (loop for i from 1 to (- VECTOR-dimensions 1)
            collect
            (aref the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-NEGATIVES i))))))




;(defmethod  (update-the-NEGATIVES-of-DESCRIPTION part-description) ()
;  (setf (part-description-NEGATIVE-additional-features  self)
;          (append
;            (part-description-NEGATIVE-additional-features  self)
;            *NEGATIVES-WHICH-WE-GAVE*)))
;
;           (loop for i in *NEGATIVES-WHICH-WE-GAVE*
;                 do
```

```
;                 (push  i
;                       (part-description-NEGATIVE-additional-features  self))))



;;;*****************************************************************************************

;(setq   *SYSTEMS-PART-DESCRIPTIONS*   '("seat-rear-complete   split-small  ARTHSA"
;                                       "seat-front-complete         electric-driver  ARTHSA"
;                                       "CUSHN-COMP-SEAT-REAR   BENCH-FOLDING  ARTHSA"
;                                       "takis"))
;
;
;
;(setq *INFERENCE-ENGINE* '(("CRA" (a14 a56))
;                           ("CRB" (b45 b78))
;                           ("HAA" (c34 c56 c89))
;                           ("HAD"  (f67 f78))))
;
;
;;;;;   GIVEN  THE  description from the user IT returns the MUST features IF  any
;
;
;
;
;(defun check-for-the-MUST-FEATURES (part-description-string-from-user)
;  (loop for match-description-WITH-MUSTS   in *INFERENCE-ENGINE*
;       do
;    (cond
;      ((search  (car match-description-WITH-MUSTS)  part-description-string-from-user)
;                            (return
;                              (cadr match-description-WITH-MUSTS))))))
;
;
;;;*****************************************************************************************
;
;
;
;(defmethod (set-MUST-features-if-not-bound  part-description) (list-of-MUST-features)
;  (cond ((boundp-in-instance self 'MUST-additional-features) nil)
;       (t (cond
;             ((null  list-of-MUST-features) nil)
;             (t (setf (part-description-MUST-additional-features  self)
;                                  list-of-MUST-features))
;           )
;        )
;  )
;)
;
;
;
;
;
;
;(defflavor PART-DESCRIPTION
;       (name) ()
;  :initable-instance-variables
;  :writable-instance-variables)

;(defvar *PART-DESCRIPTIONS* nil)
;
;(defun make-some-PART-DESCRIPTIONS ()
;  (setq *PART-DESCRIPTIONS*
;       (loop for name in '("one" "two" "three" "four")
;             collect
;               (make-instance 'PART-DESCRIPTION :description-name name))))


;(defun match-part-descriptions (&optional (stream  *standard-output*))
;  (accept 'part-description  :stream stream))
;

(defun match-PART-DESCRIPTIONS (&optional (stream  *standard-output*))
  (accept 'Secondary-Description-ID  :stream stream  :prompt "STATICE DATA BASE OBJECT :")
```

```
  )

(defvar *Secondary-Description-IDs*
        (let ((result nil))
          (with-database  (db *the-pathname-of-the-ROVERs-APN-CATALOGUE*)
            (with-transaction ()
              (for-each ((sec Secondary-Description-ID))
                (push sec result)))) result))



(define-presentation-type Secondary-Description-ID ()
   :no-deftype t
   :parser ((stream);;;;;;; BECAUSE I have used the CACHED  arrtibute for the Part-description slot
;             (with-database  (db *the-pathname-of-the-ROVERs-APN-CATALOGUE*)
;               (with-transaction ()
                 (dw:complete-from-sequence *Secondary-Description-IDs* stream
                                            :type 'Secondary-Description-ID
                                            :name-key #'Secondary-Description-ID-part-description
                                            :complete-activates t));;; ))
   :printer ((object stream)
;             (with-database  (db *the-pathname-of-the-ROVERs-APN-CATALOGUE*)
;               (with-transaction ()
                 (format stream (Secondary-Description-ID-part-description  object)))) ;;))



;;*******************************************************************************************



;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^
;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^
;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^

;Command: (READ-THE-MUST-FEATURES-from-THE-ROVER-CARS----actually-Austin-Rover *ROVERs-APN-VPG-CATALOGUE-
only-SEATS-and-MIRRORS-pa
;thname* #<PART-DESCRIPTION:    SEAT-FRONT COMPLETE    MANUAL-PASSR     > )
;#<ART-FIXNUM-3 50013533>
;T
;Command: (get-MUSTS *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname* #<PART-DESCRIPTION:     SE
AT-FRONT COMPLETE    MA
;NUAL-PASSR     > )
;(69 76)
;Command: (get-MUSTS-top *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname* #<PART-DESCRIPTION:
  SEAT-FRONT COMPLETE
;  MANUAL-PASSR     > )
;("B51" "B44")
;Command:

;;;;;;The functions "get-MUSTS-TOP" and get-MAYBES-TOP may return ...'(0)   If the Part Description is SP
ECIFIED for all variants
;Command: (let ((result nil) (numbers '(0)))
;    (loop for i in numbers
;          do
;      (push (given-a-number-return-the-Feature-Group i) result))
;result)
;(NIL)
;Command:
;


(defun get-MUSTS-TOP (pathname Secondary-Description-ID-entity)
  (let ((result nil) (numbers (get-MUSTS pathname Secondary-Description-ID-entity)))
    (loop for i in numbers
          do
```

```
      (push (given-a-number-return-the-Feature-Group i) result))
result))


(defun  get-MUSTS (pathname  Secondary-Description-ID-entity)
  (let ((the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-MUSTS
          (READ-THE-MUST-FEATURES-from-THE-ROVER-CARS----actually-Austin-Rover pathname Secondary-Descrip
tion-ID-entity)))
    (BLOCK takis
      (cond ((equal the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-MUSTS nil) ;;;AREN
'T ANY MUSTS  ??
                  (return-from takis nil)))
      (let ((VECTOR-dimensions (aref the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-M
USTS 0)))
        (loop for i from 1 to (- VECTOR-dimensions 1)
              collect
                (aref the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-MUSTS i))))))



;76
;Command: (READ-THE-MUST-FEATURES-from-THE-ROVER-CARS----actually-Austin-Rover *ROVERs-APN-VPG-CATALOGUE-
only-SEATS-and-MIRRORS-pa
;thname* #<PART-DESCRIPTION:    SEAT-FRONT COMPLETE   MANUAL-PASSR    > )
;#<ART-FIXNUM-3 50013533>
;T
;Command:



(defun READ-THE-MUST-FEATURES-from-THE-ROVER-CARS----actually-Austin-Rover (pathname  Secondary-Descripti
on-ID-entity)
  (let ((help (get-THE-ROVER-CARS----actually-Austin-Rover---entity pathname Secondary-Description-ID-ent
ity)))
    ;;;returns-the-actual THE-ROVER-CARS----actually-Austin-Rover statice-object to reference to the
    ;;;;    actual statice object of Secondary-Description-ID-entity as argument to the function.
    (with-database (db pathname)
      (with-transaction ()
        (BLOCK DOES-IT-HAVE-AN-entity-type-LINKED-TO?
          (cond ((equal help nil) (return-from DOES-IT-HAVE-AN-entity-type-LINKED-TO? nil)))
          (case *model*
            (R8 (THE-ROVER-CARS----actually-Austin-Rover-New-ROVER-200-or-ROVER-400--XW-musts   help))
            (metro (THE-ROVER-CARS----actually-Austin-Rover-METRO-musts   help))))))))


;    (MINI-musts    (vector fixnum))
;    (METRO-musts   (vector fixnum))
;    (R6-musts      (vector fixnum))
;    (MAESTRO-musts    (vector fixnum))
;    (MONTEGO-musts    (vector fixnum))
;    (old-ROVER-200--XH-musts     (vector fixnum))
;    (New-ROVER-200-or-ROVER-400--XW-musts    (vector fixnum))
;    (ROVER-800-musts    (vector  fixnum))
;    (R17-musts       (vector fixnum))

;Command: (given-a-FEATURE-FROUP-STRING-return-the-NUMBER "b51")
;76
;Command: (given-a-FEATURE-FROUP-STRING-return-the-NUMBER "b44")
;69
;Command: (given-a-FEATURE-FROUP-STRING-return-the-NUMBER "b69")
;93
;Command: (given-a-FEATURE-FROUP-STRING-return-the-NUMBER "b53")
;78
;Command:

;(defun  get-THE-ROVER-CARS----actually-Austin-Rover---entity  (pathname  Secondary-Description-ID-entity
)
;  (with-database (db pathname)
;    (with-transaction ()
;      (Secondary-Description-ID-REFERENCE-TO-THE-ROVER-VEHICLES Secondary-Description-ID-entity))))


;;;;;;   EXPERIMENTATION !   I manually create a THE-ROVER-CARS----actually-Austin-Rover statice object a
nd I put it manually as
```

```
;;;;;;      the  value of its attribute "REFERENCE-TO-THE-ROVER-VEHICLES" to the statice object
;;;;;;     Secondary-Description-ID-entity    called   :
;;;;;;                  #<PART-DESCRIPTION:    SEAT-FRONT COMPLETE   MANUAL-PASSR    >


;Command: (assign-CAR-to-SECONDARY *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname* #<PART-DESC
RIPTION:    SEAT-FRONT
; COMPLETE   MANUAL-PASSR    > #<Entity-Handle of THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER 62/3 367746220>
)
;#<Entity-Handle of THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER 62/3 367746220>
;Command:
;Command: (get-THE-ROVER-CARS----actually-Austin-Rover---entity *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-
MIRRORS-pathname* #<PART
;-DESCRIPTION:    SEAT-FRONT COMPLETE   MANUAL-PASSR    > )
;#<Entity-Handle of THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER 62/3 367746220>
;T
;Command:
;;*****************************************
;Command: (get-THE-ROVER-CARS----actually-Austin-Rover---entity *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-
MIRRORS-pathname* #<PART
;-DESCRIPTION:    INSTR DRG-FIXING      SEAT FRONT      >)
;NIL
;NIL
;Command:
;;;;;;;;;;     PUSH now a VECTOR as a value to represent the R8-MUSTS in the statice-object
;;;;;    THE-ROVER-CARS----actually-Austin-Rover
;;;
;;;
;Command:
;              (ASSIGN-the-VALUE-VECTOR-of-R8-musts-to-THE-ROVER-CARS----actually-Austin-Rover-entity *ROV
ERs-APN-VPG-CATALOGUE-onl
;y-SEATS-and-MIRRORS-pathname* #<Entity-Handle of THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER 62/3 367746220>
 '("b51" "b44"))
;#<ART-FIXNUM-3 50000535>
;Command: (DESCRIBE #<ART-FIXNUM-3 50000535>)
;#<ART-FIXNUM-3 50000535> is an ART-FIXNUM type array.
;It uses %ARRAY-DISPATCH-FIXNUM; it is 3 elements long.
;#<ART-FIXNUM-3 50000535>
;Command: (aref #<ART-FIXNUM-3 50000535> 0)
;3
;Command: (aref #<ART-FIXNUM-3 50000535> 1)
;69
;Command: (aref #<ART-FIXNUM-3 50000535> 2)
;76
;Command:

(defun  ASSIGN-the-VALUE-VECTOR-of-R8-musts-to-THE-ROVER-CARS----actually-Austin-Rover-entity
                 (pathname THE-ROVER-CARS----actually-Austin-Rover    VECTOR-OF-STRINGS)
  (let ((help nil))
    (loop for FEATURE-GROUP-STRING   in VECTOR-OF-STRINGS
          do
      (push (given-a-FEATURE-FROUP-STRING-return-the-NUMBER FEATURE-GROUP-STRING) help))
    (let* ((length-of-array (+ 1 (length help)))
           (array-elements (push length-of-array help))
           (VECTOR-value-gone-in-statice
             (make-array length-of-array :element-type 'fixnum :initial-contents array-elements)))
      (with-database (db pathname)
        (with-transaction ()
          (setf (THE-ROVER-CARS----actually-Austin-Rover-New-ROVER-200-or-ROVER-400--XW-musts
                  THE-ROVER-CARS----actually-Austin-Rover)
                VECTOR-value-gone-in-statice))))))


;(defun create-the-entity-type-THE-ROVER-CARS----actually-Austin-Rover (pathname)
;  (let ((result nil))
;    (with-database (db pathname)
;      (with-transaction ()
;       (setq result (make-THE-ROVER-CARS----actually-Austin-Rover))))
;result))
;
;(defun assign-CAR-to-SECONDARY (pathname  Secondary-Description-ID-entity     THE-ROVER-CARS----actually
-Austin-Rover)
;  (with-database (db pathname)
;    (with-transaction ()
```

```
;      (setf (Secondary-Description-ID-REFERENCE-TO-THE-ROVER-VEHICLES Secondary-Description-ID-entity)
;            THE-ROVER-CARS----actually-Austin-Rover))))




;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^

;Command: (get-MaybeS-top *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname* #<PART-DESCRIPTION:
   SEAT-FRONT COMPLETE
;   MANUAL-PASSR    > )
;("B69" "B53" "B51")
;Command:


(defun get-MAYBES-TOP (pathname Secondary-Description-ID-entity)
  (let ((result nil) (numbers (get-MAYBES pathname Secondary-Description-ID-entity)))
    (loop for i in numbers
          do
      (push (given-a-number-return-the-Feature-Group i) result))
result))



(defun  get-MAYBES (pathname  Secondary-Description-ID-entity)
  (let ((the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-MAYBES
          (READ-THE-MAYBE-FEATURES-from-THE-ROVER-CARS----actually-Austin-Rover pathname Secondary-Descri
ption-ID-entity)))
    (BLOCK takis
      (cond ((equal the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-MAYBES nil) ;;AREN
'T ANY maybes ??
                  (return-from takis nil)))
      (let ((VECTOR-dimensions (aref the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-M
AYBES 0)))
        (loop for i from 1 to (- VECTOR-dimensions 1)
              collect
                (aref the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-MAYBES i))))))



(defun READ-THE-MAYBE-FEATURES-from-THE-ROVER-CARS----actually-Austin-Rover (pathname  Secondary-Descript
ion-ID-entity)
  (let ((help (get-THE-ROVER-CARS----actually-Austin-Rover---entity pathname Secondary-Description-ID-ent
ity)))
    ;;;returns-the-actual THE-ROVER-CARS----actually-Austin-Rover statice-object to reference to the
    ;;;;    actual statice object of Secondary-Description-ID-entity as argument to the function.
    (with-database (db pathname)
      (with-transaction ()
        (BLOCK  DOES-IT-HAVE-A-statice-entity-type-LINKED-TO?
          (cond ((equal help nil) (return-from DOES-IT-HAVE-A-statice-entity-type-LINKED-TO? nil)))
          (case *model*
            (R8 (THE-ROVER-CARS----actually-Austin-Rover-New-ROVER-200-or-ROVER-400--XW-maybes   help))
            (metro (THE-ROVER-CARS----actually-Austin-Rover-METRO-maybes   help)))))))


;   (MINI-maybes   (vector fixnum))
;   (METRO-maybes  (vector fixnum))
;   (R6-maybes     (vector fixnum))
;   (MAESTRO-maybes   (vector fixnum))
;   (MONTEGO-maybes   (vector fixnum))
;   (old-ROVER-200--XH-maybes    (vector fixnum))
;   (New-ROVER-200-or-ROVER-400--XW-maybes   (vector fixnum))
;   (ROVER-800-maybes   (vector  fixnum))
;   (R17-maybes      (vector fixnum))


;;;;  The below function
;;;; GETS the THE-ROVER-CARS----actually-Austin-Rover statice object  FROM  Secondary-Description-ID  sta
tice object

(defun  get-THE-ROVER-CARS----actually-Austin-Rover---entity  (pathname  Secondary-Description-ID-entity)
```

```
  (with-database (db pathname)
    (with-transaction ()
      (Secondary-Description-ID-REFERENCE-TO-THE-ROVER-VEHICLES Secondary-Description-ID-entity))))


;;;;;;   EXPERIMENTATION !   I manually create a THE-ROVER-CARS----actually-Austin-Rover statice object a
nd I put it manually as
;;;;;;       the  value of its attribute "REFERENCE-TO-THE-ROVER-VEHICLES" to the statice object
;;;;;;      Secondary-Description-ID-entity     called   :
;;;;;;                 #<PART-DESCRIPTION:    SEAT-FRONT COMPLETE   MANUAL-PASSR    >


;Command: (ASSIGN-the-VALUE-VECTOR-of-R8-maybes-to-THE-ROVER-CARS----actually-Austin-Rover-entity  *ROVER
s-APN-VPG-CATALOGUE-only-
;SEATS-and-MIRRORS-pathname*
;#<Entity-Handle of THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER 62/3 367746220> '("b69" "b53" "b51"))
;#<ART-FIXNUM-4 54214062>


(defun  ASSIGN-the-VALUE-VECTOR-of-R8-maybes-to-THE-ROVER-CARS----actually-Austin-Rover-entity
                 (pathname THE-ROVER-CARS----actually-Austin-Rover    VECTOR-OF-STRINGS)
  (let ((help nil))
    (loop for FEATURE-GROUP-STRING   in VECTOR-OF-STRINGS
          do
      (push (given-a-FEATURE-FROUP-STRING-return-the-NUMBER FEATURE-GROUP-STRING) help))
    (let* ((length-of-array (+ 1 (length help)))
           (array-elements (push length-of-array help))
           (VECTOR-value-gone-in-statice
             (make-array length-of-array :element-type 'fixnum :initial-contents array-elements)))
      (with-database (db pathname)
        (with-transaction ()
          (setf (THE-ROVER-CARS----actually-Austin-Rover-New-ROVER-200-or-ROVER-400--XW-maybes
                  THE-ROVER-CARS----actually-Austin-Rover)
                VECTOR-value-gone-in-statice))))))


;(defun create-the-entity-type-THE-ROVER-CARS----actually-Austin-Rover (pathname)
;  (let ((result nil))
;    (with-database (db pathname)
;      (with-transaction ()
;        (setq result (make-THE-ROVER-CARS----actually-Austin-Rover))))
;result))
;
;(defun assign-CAR-to-SECONDARY (pathname  Secondary-Description-ID-entity     THE-ROVER-CARS----actually
-Austin-Rover)
;  (with-database (db pathname)
;    (with-transaction ()
;      (setf (Secondary-Description-ID-REFERENCE-TO-THE-ROVER-VEHICLES Secondary-Description-ID-entity)
;            THE-ROVER-CARS----actually-Austin-Rover))))



;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^




;**************
;;;;;;;*headings*   COMING from the file :      K87.dat



;"A10-MARQUE                                   "
;"A10AAUSTIN                        A          "
;"A10HHONDA                         HONDA      "
;"A10MMORRIS                        MORRIS     "
;"A10RROVER                         ROVER      "
;"A10XAUSTIN ROVER                  AUSROV     "
;"A20-MODEL RANGE (AUSTIN ROVER)               "
```

```
;"A20AADO 73                          A73     "
;"A20BLC8                              LC8     "
;"A20CMAESTRO                          LM10    "
;"A20DACCLAIM                          ACL     "


(defun  each-Feature-Group-has-its-identical-Integer   (ls)   ;;;;"headings"
  (let ((pointer 0) (result nil))
    (loop for i in ls
          do
      (cond
        ((string-equal "-" (subseq i 3 4))
            (push (list (subseq i 0 3) (incf pointer))
                  result))))
(setq   *FEATURE-GROUP-WITH-NUMBER* result)))        ;;;;;;THE result MUST be a  global  variable.


(defun each-Feature-has-its-identical-Integer (ls)
  (let ((result nil) (record-number 0))
    (loop for FEATURE in ls
          do
      (cond ((not (string-equal (subseq  FEATURE 3 4) "-"))
                  (setq record-number (incf record-number))
                  (push (list FEATURE record-number)
                        result))))
    result))


(defvar  *FEATURE-GROUP-WITH-NUMBER*   (each-Feature-Group-has-its-identical-Integer *headings*))
(defvar *EACH-FEATURE-WITH-NUMBER* (each-Feature-has-its-identical-Integer  *headings*))





;Command: (from-feature-number-to-feature-string 329)
;"B35A~~~~~NO AERIAL                        NOARIL  "
;Command: (from-feature-string-to-feature-number "B35A~~~~~NO AERIAL                        NOARIL  ")
;329
;Command:



(defun from-feature-number-to-feature-string (a-number)
  (block katerina
    (loop for pair-of-feature-with-number   in  *EACH-FEATURE-WITH-NUMBER*
          do
      (cond ((equal (second  pair-of-feature-with-number) a-number)
                (return-from  katerina   (string-append (subseq (car pair-of-feature-with-number) 0 4)
                                                        "~~~~~"
                                                        (subseq (car pair-of-feature-with-number) 4 42)))
)))
    nil))


(defun from-feature-string-to-feature-number (a-string) ;;; of the form ::: "B35A~~~~~NO AERIAL
           NOARIL  "
  (block katerina
    (loop for pair-of-feature-with-number   in  *EACH-FEATURE-WITH-NUMBER*
          do
      (cond ((string-equal (first  pair-of-feature-with-number) (string-append
                                                                  (subseq a-string 0 4)
                                                                  (subseq a-string 9 47)))
                (return-from  katerina   (second pair-of-feature-with-number)))))
    nil))
```

```
(defun given-a-number-return-the-Feature-Group (a-number)
  (let ((result nil))
    (cond ((equal a-number 0) (setq result "NO-SPECIFICATION"))
          (t
           (loop for pair in *FEATURE-GROUP-WITH-NUMBER*
                 do
             (cond
               ((equal (second pair) a-number)
                (setq result (first pair)))))))
result))


(defun given-a-FEATURE-FROUP-STRING-return-the-NUMBER (FEATURE-FROUP-STRING) ;;THE OPPOSITE of
                                                      ;;;     the  function "given-a-number-return-the-Feature-
Group"
  (let ((result nil))
    (cond ((string-equal FEATURE-FROUP-STRING "NO-SPECIFICATION") (setq result 0))
          (t
           (loop for pair in *FEATURE-GROUP-WITH-NUMBER*
                 do
             (cond
               ((string-equal (car pair) FEATURE-FROUP-STRING)
                (setq result (second pair)))))))
result))

;Command: (given-a-FEATURE-FROUP-STRING-return-the-NUMBER "b51")
;76
;Command: (given-a-FEATURE-FROUP-STRING-return-the-NUMBER "b44")
;69
;Command: (given-a-FEATURE-FROUP-STRING-return-the-NUMBER "b69")
;93
;Command: (given-a-FEATURE-FROUP-STRING-return-the-NUMBER "b53")
;78
;Command:
```

```
;;;-*-  Mode: lisp;  Package: austin-rover ; syntax: common-lisp-*-


(setq "supergroups"
      '(("M20*" "HOME GROUPS")
        ("M32*" "UNITED KINGDOM  GROUP")
        ("M34*" "REST OF EUROPE  GROUPS")
        ("M35*" "MAJOR GROUPS OUTSIDE EUROPE")
        ("M36*" "ROW  TEMP  CLIMATE  RHD   GROUPS")
        ("M38*" "ROW  HOT   CLIMATE  RHR  GROUPS")))









;("M36Y"
; (("T60R" "ZAMBIA                    " "ZM    ") ("T58C" "SEYCHELLES                 " "SC    ";)
;  ("T56L" "MAURITIUS                 " "MU    ") ("T55R" "KENYA                      " "KE    ";)
;  ("T55J" "JAMAICA                   " "JM    ") ("T54A" "GRENADA                    " "GD    ";)
;  ("T51N" "BRUNEI                    " "BN    ") ("T50Z" "BARBADOS                   " "BB    ";))))
"M39*" "ROW  HOT   CLIMATE LHR  GROUPS")))



(defun supergroups-in ()
  (let ((result nil)  (record nil))
    (loop for supergroup in "supergroups"
          do
      (loop for territory  in "territories"
            do
        (cond
          ((string-equal (substring (car supergroup) 0 3) (substring (car territory) 0 3))
             (cond
               ((member (car supergroup) record :test #'string-equal)
                   (push territory
                         (cadr (assoc (car supergroup)
                                      result
                                      :test #'string-equal))))
               (t (push (list (car supergroup)
                              (list territory )) result)
                  (push (car supergroup) record)
               )
             )
          )
        )
      )
    )
result))

(setq "final-territories" (supergroups-in))



;;;;;WE   NEED IT  IN THE SITUATION WHERE "group-territories" IS   " - ".
;;;;;;   NOTE :   IT can happen one country to belong to more than 1 SUBGROUP, BUT not
;;;;;;   SUPERGROUP.


(defun all-the-COUNTRIES ()
  (let ((result nil))
    (loop for SUPERGROUP in "final-territories"
          do
      (let ((subgroups (car (reverse SUPERGROUP))))
        (loop for subgroup in subgroups
              do
          (let ((countries (car (reverse subgroup))))
            (loop for country in countries
                  do
              (push (car country) result))
          ))))
result))
```

```
(setq *countries* (remove-duplicates (all-the-COUNTRIES) :test #'string-equal))


;*****************************************************************************************
;*****************************************************************************************


;;;;;;; CHECK THE          RESTRICTIONS


;Command: (part-apart-the-list '("D" "W" " " " " " " " ...))
;(("D" "W" " " " " " " " " " " " ") (NIL
;                                   ((:TERRITORY-RESTRICTIONS-1 ("M20E" "+T53R")))
;                                   ("+" ("M20E"))
;                                   )
;
;  )
;

;Command: (part-apart-the-list '("D" "N" " " " " " " " ...))
;(("D" "N" " " " " " " " " " " " ")
; (((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" ";Z96B" "="
 "R00E")))
;  ((:TERRITORY-RESTRICTIONS-1 ("M20U" "-T60A"))) ("-" ("M20U"))))
;Command:
;("H" "W" " " " " " " " " " " " " NIL NIL (" " NIL))
;Command: (part-apart-the-list '("H" "W" " " " " " " " ...))
;(("H" "W" " " " " " " " " " " ") (NIL NIL (" " NIL)))
;Command:


;  THIS comes from  IMPLICIT  !!!!!!!!!!
;
;Command: '("H" "G" " " " " " "E" ...)
;("H" "G" " " " " " "E" "A" "A"
; ((:COMBINATIONS-RESTRICTIONS-1 ("P" "H50K" "=" "H50P")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "H50K" "=" "
H50P")))
; ((:TERRITORY-RESTRICTIONS-1 ("M20E" "+T53R"))) ("+" ("M32*")))
;Command: (part-apart-the-list '("H" "G" " " " " " "E" ...))
;(("H" "G" " " " " " "E" "E" "A")
; (((:COMBINATIONS-RESTRICTIONS-1 ("P" "H50K" "=" "H50P")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "H50K" "="
"H50P")))
;  ((:TERRITORY-RESTRICTIONS-1 ("M20E" "+T53R"))) ("+" ("M32*"))))
;Command:


;*****************************************************************************************
;*****************************************************************************************

;;;;  NOW,  I PUT  some additional information from the data , TO SEE HOW the RESTRICTIONS
;;;;        could come.

;;;ACTUALLY,  returned EXPRESSIONS only   if the value is  NOT   nil. OTHERWISE , expressions
;;;;   i.e.   (:TERRITORY-RESTRICTIONS-1 ("M20E" "+T53R")),
;;;           (:ALL-THE-COMBINATIONS-RESTRICTIONS NIL)    e. t. c.


;((:FEATURE-DESCRIPTION "B39M-----PR DIAGONAL BELTS &CENTRE LAP RLAPBT  ")
; (:ALL-THE-COMBINATIONS-RESTRICTIONS NIL) (:ALL-THE-COMMENTS-RESTRICTIONS NIL)
; (:ALL-THE-TERRITORY-RESTRICTIONS NIL) (:RECNO "400") (:FEATCODE "B39M") (:STANDARD " ")
; (:BASE " ") (:OPTIONAL " ") (:MARKETING " ") (:ENVENG " ") (:LEGAL "L") (:ACCESSORY " ")
; (:MR "XF") (:MC "  ") (:CLASS "D") (:BODY "N") (:FAMILY " ") (:CAPACITY " ")
; (:PERFORMANCE " ") (:DRIVE " ") (:TRANS " ") (:AUTH-REF-IN "M/5P/88 S1  ")
; (:AUTH-REF-OUT "             ") (:REL-STAGE-IN "P") (:REL-STAGE-OUT " ") (:DEP-IN "JX054   ")
; (:DEP-OUT "        ") (:FEAT-GROUP "B39") (:HISTORY " ") (:ERRNO "00")
; (:ALL-THE-GROUP-RESTRICTIONS ("+" ("M20*" "M32B" "M32D" "M32F" "M35I"))))
;
;
;
;((:FEATURE-DESCRIPTION "R03I-----METRO LTD EDTN QUARTER 3 1988 Q3METR  ")
; (:ALL-THE-COMBINATIONS-RESTRICTIONS
```

```
;  ((:COMBINATIONS-RESTRICTIONS-1
;    ("I" "B05K" "," "B25G" "," "B40B" "," "B45B" "B90D" "," "C40K" "," "F25B" "," "F26B"
;     "F64C" "," "H37B" "<" "F96C" "<" "F98J"))))
; (:ALL-THE-COMMENTS-RESTRICTIONS NIL)
; (:ALL-THE-TERRITORY-RESTRICTIONS
;  ((:TERRITORY-RESTRICTIONS-1 ("M32S" "+T58L")) (:TERRITORY-RESTRICTIONS-2 ("M32F" "+T53E"))))
; (:RECNO "420") (:FEATCODE "R03I") (:STANDARD " ") (:BASE " ") (:OPTIONAL "O") (:MARKETING " ")
; (:ENVENG " ") (:LEGAL " ") (:ACCESSORY " ") (:MR "XF") (:MC "  ") (:CLASS "H") (:BODY "W")
; (:FAMILY "A") (:CAPACITY "D") (:PERFORMANCE "E") (:DRIVE "B") (:TRANS "A")
; (:AUTH-REF-IN "M/31P/87    ") (:AUTH-REF-OUT "M/31P/87 S4 ") (:REL-STAGE-IN "P")
; (:REL-STAGE-OUT "P") (:DEP-IN "B9550   ") (:DEP-OUT "B9550   ") (:FEAT-GROUP "R03")
; (:HISTORY " ") (:ERRNO "00") (:ALL-THE-GROUP-RESTRICTIONS ("+" ("M32F" "M32S"))))
;


;((:FEATURE-DESCRIPTION "G70C-----MODINE OILCOOLER-WATER COOLED)OILMOD  ")
; (:ALL-THE-COMBINATIONS-RESTRICTIONS NIL) (:ALL-THE-COMMENTS-RESTRICTIONS NIL)
; (:ALL-THE-TERRITORY-RESTRICTIONS ((:TERRITORY-RESTRICTIONS-1 ("M39Y" "+T55N" "=T56N"))))
; (:RECNO "420") (:FEATCODE "G70C") (:STANDARD " ") (:BASE " ") (:OPTIONAL " ") (:MARKETING " ")
; (:ENVENG "E") (:LEGAL " ") (:ACCESSORY " ") (:MR "XF") (:MC "  ") (:CLASS " ") (:BODY " ")
; (:FAMILY " ") (:CAPACITY " ") (:PERFORMANCE "A") (:DRIVE " ") (:TRANS " ")
; (:AUTH-REF-IN "M/5P/88     ") (:AUTH-REF-OUT "            ") (:REL-STAGE-IN "P")
; (:REL-STAGE-OUT " ") (:DEP-IN "B9553   ") (:DEP-OUT "        ") (:FEAT-GROUP "G70")
; (:HISTORY " ") (:ERRNO "00") (:ALL-THE-GROUP-RESTRICTIONS ("+" ("M39Y" "M39Z"))))

;Command: (create-the-logic-references '("C78B" (("D" # NIL # NIL) ("H" # NIL # NIL))))
;(("H" " " " " " " " " " " " " " (((:COMMENT-RESTRICTIONS-1 "C16") (:COMMENT-RESTRICTIONS-2 "C71")) NIL ("+
" ("*****")))
; ("D" " " " " " " " " " " " " " ((:COMMENT-RESTRICTIONS-1 "C16") (:COMMENT-RESTRICTIONS-2 "C71")) NIL ("+"
 ("*****"))))
;Command:


(defun look-the-restrictions (ls)      ;;;;;IT RETURNS  A LIST  OF  COUNTRIES.

;;;;;;      ie.
;;;;
; (((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "296B" "="
"R00E")))
;  ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U"))))
;
  (let ((result nil)

        (GROUP-RESTRICTIONS (GROUP-RESTRICTIONS-collect ls))
        (BESIDES-THE-TERRITORY-RESTRICTIONS
              (TERRITORY-RESTRICTIONS-collect  ls))
        ;;;;TO  CHECK  When the ODD situation happens : - M32B   ,    M32B - RE .
        (sign-of-GROUP (car (car (reverse ls))))
        (sign-of-RESTRICTIONS (substring (car (reverse (cadr (car (cadr ls))))) 0  1)))
   ;(print sign-of-GROUP)
   ;(print sign-of-RESTRICTIONS)))
   (cond ;;;; ANOTHER ODD CASE :     ("+" ("*****"))
     ((string-equal (car
                     (car (reverse (car (reverse ls))))) "*****")
           (setq result *countries*))
     ((and (string-equal sign-of-GROUP "-") (string-equal sign-of-RESTRICTIONS "-"))
         (setq  result (remove-duplicates
                        (append
                          GROUP-RESTRICTIONS
                          (loop for i in
                                 (cdr (car (reverse (car (cadr ls)))))
                                collect
                                  (substring i 1 5))
                          )
                        :test #'string-equal))
     )
     (t

   (cond

     ((equal BESIDES-THE-TERRITORY-RESTRICTIONS nil)
          (setq result (GROUP-RESTRICTIONS-collect ls)))

     (t  (setq result (set-difference GROUP-RESTRICTIONS
                                      BESIDES-THE-TERRITORY-RESTRICTIONS
```

```
                                    :test #'string-equal)
      ))
    )     ;;;;cond
     )    ;;;;;      ttt
   )
  result))


;*****************************************************************************


;*****************************************************************************


 (defun GROUP-RESTRICTIONS-collect (ls)

   ;;;;;ie
   ;;;;;   THE SECOND OF THE  "part-apart-the-list"    FUNCTION.

; (((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "Z96B" "=" "R00E")))
;  ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U"))))


(let ((result nil)
      (groups (car (reverse ls))))
  ;;;;(print groups)
    (cond   ;;;;;;THE SITUATION where   :  (" " ( )),      SOMETIMES is  only   "nil".
      ((or
         (equal groups nil)
         (and (string-equal (car groups) " ")
              (equal (cadr groups) nil))
       )
                                    (setq result *countries")
      )
      (t (cond ((string-equal (car groups)   "+")
                  (loop for GROUP-OR-SUPERGROUP   in (cadr groups)
                        do
                    (push (find-the-countries  GROUP-OR-SUPERGROUP ) result)))
               ((string-equal (car groups)   "-")
                  (let ((result-help nil))
                    (loop for GROUP-OR-SUPERGROUP   in (cadr groups)
                          do
                      (push (find-the-countries  GROUP-OR-SUPERGROUP ) result-help))
                    (setq result
                          (set-difference
                            *countries*
                            (clear result-help)
                            :test #'string-equal)
                    )
                  )
             )
         ) ;;;cond
     )  ;;;   t
    )
    (clear result)))



;(defun clear (ls)
;  (cond ((null ls) nil)
;        ((atom (car ls)) (cons (car ls)
;                               (clear (cdr ls))))
;        (t (append (clear (car ls))
;                   (clear (cdr ls))))
;))
```

```
(defun find-the-countries (group-or-supergroup)
  (cond ((string-equal (substring (reverse    group-or-supergroup)  0 1) "*")
                 (all-the-COUNTRIES-SUPERGROUP group-or-supergroup))
        (t (all-the-COUNTRIES-GROUP  group-or-supergroup))
  )
)


(defun all-the-COUNTRIES-GROUP (group)
  (let ((result nil))

    (loop for SUPERGROUP in *final-territories*
          do
      (let ((subgroups (car (reverse SUPERGROUP))))
        (let ((reference-of-GROUP (cadr (assoc group subgroups :test #'string-equal))))
          (cond (reference-of-GROUP
                    (loop for country in reference-of-GROUP
                          do
                        (push (car country) result))
                )


          )
            )))
result))


(defun all-the-COUNTRIES-SUPERGROUP   (supergroup)
  (let* ((result nil)
         (reference-of-SUPERGROUP (cadr (assoc supergroup *final-territories*
                                                :test #'string-equal)))
        )
  (loop for subgroup in reference-of-SUPERGROUP
          do
      (let ((countries (car (reverse subgroup))))
        (loop for country in countries
              do
          (push (car country) result))
    ))
result))


;Command: (setq john '(((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W
57A" "=" "Z96
;B" "=" "R00E")))
; ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U" "M35*" "m36*"))))
;(((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "Z96B" "=" "
R00E")))
; ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U" "M35*" "m36*")))
;Command: (group-restrictions-collect john)
;("T50Z" "T51N" "T54A" "T55J" "T55R" "T56L" "T58C" "T60R" "T56C" "T58R" "T50X" "T51G" "T54N" "T54V" "T56E
" "T58G"
; "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T55A" "T60A
")
;Command: (group-restrictions-collect john)
;(("T50Z" "T51N" "T54A" "T55J" "T55R" "T56L" "T58C" "T60R" "T56C" "T58R" "T50X" "T51G" "T54N" "T54V" "T56
E" "T58G")
; ("T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L") ("T55A" "T
60A"))


;Command: (length (remove-duplicates *countries* :test #'string-equal))
;69
;Command: (+ (length (GROUP-RESTRICTIONS-COLLECT ROB)) (length (GROUP-RESTRICTIONS-COLLECT JOHN)))
;69
;Command: rob
;(((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "Z96B" "=" "
R00E")))
; ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("-" ("M20U" "M35*" "m36*")))
```

```
;Command:


;*****************************************************************************************


(defun TERRITORY-RESTRICTIONS-collect (ls)

   ;;;;;ie...
   ;;;;;   THE SECOND OF THE  "part-apart-the-list"    FUNCTION.

; (
;  ((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "Z96B" "="
"R00E")))
;   ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))
;    (:TERRITORY-RESTRICTIONS-2 ("M36Y" "+T55j" "=T54A" "=T50Z")))
;  ("+" ("M20U")))

  (let ((result nil)
        (TERRITORY-RESTRICTIONS  (cadr ls)))
    (cond
      (;;;(or
         (equal TERRITORY-RESTRICTIONS nil))
          ;;;;(equal TERRITORY-RESTRICTIONS '(" " NIL))))  ;;;;IT IS GENERATED from implicits.
      (t
        (loop for i in TERRITORY-RESTRICTIONS
              do

          (push  (find-the-NOT-GO-COUNTRIES-of-EXPRESSION i)  result)))
    )
 (remove-duplicates (clear result) :test #'string-equal))
)


;;;;;   WE THINK   negative   KNOW  !!!!!!!


(defun find-the-NOT-GO-COUNTRIES-of-EXPRESSION (ls)
   ;;;ie
   ;;;;;(:TERRITORY-RESTRICTIONS-2 ("M36Y" "+T55j" "=T54A" "=T50Z"))
   (let*  ((result nil)
           (code (cadr ls))
           (all-countries-of-group-or-SUBGROUP
                 (find-the-countries (car code)))
           (THE-CONTROL-SIGN (substring (cadr code) 0 1))
           (rest-COUNTRIES (cdr code)))

;     (print 'code) (prin1 code)
;     (print 'all-countries-of-group-or-SUBGROUP) (prin1 all-countries-of-group-or-SUBGROUP)
;     (print 'THE-CONTROL-SIGN) (prin1 THE-CONTROL-SIGN)
;     (print 'rest-COUNTRIES) (prin1 rest-COUNTRIES)

   (let ((real-countries
           (loop for country in rest-COUNTRIES
                 collect
                 (substring country 1 5))))

     (cond
       ((string-equal THE-CONTROL-SIGN "+")
              ;;;;THE COUNTRIES WHICH ARE NOT  VALID.
                 (setq result
                       (set-difference all-countries-of-group-or-SUBGROUP
                                       real-countries :test #'string-equal)))
       ((string-equal THE-CONTROL-SIGN "-")
              ;;;;THE COUNTRIES WHICH ARE NOT  VALID.
                 (setq result real-countries))
     )
   )
   result))
```

```
;Command: paul
;((((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "Z96B" "=" "
R00E")))
; ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A")) (:TERRITORY-RESTRICTIONS-2 ("M36Y" "+T55j" "=T54A" "=T50Z
")))
; ("+" ("M20U")))
;Command: (TERRITORY-RESTRICTIONS-collect paul)
;("T51N" "T55R" "T56L" "T58C" "T60R" "T55A")
;Command: (setq peter '(
;  ((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "Z96B" "=" "
R00E")))
;    ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))
;     (:TERRITORY-RESTRICTIONS-2 ("M36Y" "-T55j" "=T54A" "=T50Z")))
;  ("+" ("M20U"))))
;((((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "Z96B" "=" "
R00E")))
; ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A")) (:TERRITORY-RESTRICTIONS-2 ("M36Y" "-T55j" "=T54A" "=T50Z
")))
; ("+" ("M20U")))
;Command: (TERRITORY-RESTRICTIONS-collect peter)
;("T55j" "T54A" "T50Z" "T55A")
;Command:
;Command: (find-the-countries "m36y")
;("T50Z" "T51N" "T54A" "T55J" "T55R" "T56L" "T58C" "T60R")
;Command: (find-the-countries "m20u")
;("T55A" "T60A")
```

```
;Command: (look-the-restrictions peter)
;("T51N" "T55R" "T56L" "T58C" "T60R" "T60A")
;Command: (look-the-restrictions tom)
;("T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J
" "T56V"
; "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E" "T59G" "T51R
" "T51V"
; "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R" "T56L
" "T58C"
; "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T
" "T56U"
; "T58N" "T57J" "T58J" "T60A")
;Command: (find-the-countries "m32b")
;("T51A" "T52J" "T53C" "T54R" "T56T" "T59N" "T60M")
;Command: tom
;((((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "Z96B" "=" "
R00E")))
; ((:TERRITORY-RESTRICTIONS-1 ("M20U" "-T60A"))) ("-" ("M20U")))
;Command: (find-the-countries "m20u")
;("T55A" "T60A")
;Command: (set-difference "countries" (FIND-THE-COUNTRIES "m20u") :test #'string-equal)
;("T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J
" "T56V"
; "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E" "T59G" "T51R
" "T51V"
; "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R" "T56L
" "T58C"
; "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T
" "T56U"
; "T58N" "T57J" "T58J")
;Command: (length (SET-DIFFERENCE *COUNTRIES* (FIND-THE-COUNTRIES "m20u") :TEST #'STRING-EQUAL))
;67
;Command: (length (LOOK-THE-RESTRICTIONS TOM))
;68
;Command: tom
;((((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "Z96B" "=" "
R00E")))
; ((:TERRITORY-RESTRICTIONS-1 ("M20U" "-T60A"))) ("-" ("M20U")))
;Command:
```

;**********

```
;*****************************************************************************************
;*****************************************************************************************


(defun COMBINATIONS-or-COMMENTS-RESTRICTIONS-collect (ls)

  ;;;;;ie...
   ;;;;;   THE SECOND OF THE  "part-apart-the-list"    FUNCTION.

; (
;  ((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "Z96B" "="
"R00E")))
;    ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))
;     (:TERRITORY-RESTRICTIONS-2 ("M36Y" "+T55j" "=T54A" "=T50Z")))
;  ("+" ("M20U")))


(let ((result nil)
      (COMBINATIONS  (car ls)))
  (loop for  combination  in  COMBINATIONS
        do
    (push
      (cadr combination) result))
result))



; Command: boula
;((((:COMMENT-RESTRICTIONS-1 "C16") (:COMMENT-RESTRICTIONS-2 "C71")) NIL ("+" ("*****")))
;Command: (COMBINATIONS-RESTRICTIONS-collect BOULA)
;("C71" "C16")
;Command: (setq georgia '(
;  ((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "Z96B" "="
"R00E")))
;    ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))
;     (:TERRITORY-RESTRICTIONS-2 ("M36Y" "+T55j" "=T54A" "=T50Z")))
; ("+" ("M20U"))))
;((((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "Z96B" "=" "
R00E")))
; ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A")) (:TERRITORY-RESTRICTIONS-2 ("M36Y" "+T55j" "=T54A" "=T50Z
")))
; ("+" ("M20U")))
;Command: (COMBINATIONS-RESTRICTIONS-collect georgia)
;(("R" "W57A" "=" "Z96B" "=" "R00E") ("P" "R00E"))

;*****************************************************************************************
;*****************************************************************************************



(defun translate-to-english-COUNTRIES (ls)
  (let ((result nil))
    (loop for  country   in  ls
          do
      (push (the-country-in-ENGLISH  country) result))
    result))


(defun the-country-in-ENGLISH (country-code)
 (let ((result nil))
  (loop for SUPERGROUP in *final-territories*
        do
    (let ((groups (car (reverse SUPERGROUP))))
    (loop for group in groups
          do
      (let ((countries (car (reverse group))))
        (loop for country-in-CODE  in countries
              do
          (let ((code (car country-in-CODE))
                (ENGLISH (cadr country-in-CODE)))
            (cond ((string-equal code country-code)
                     (setq result (string-trim '(#\space) ENGLISH)))
            )
```

```
))))))
  result))


;*****************************************************************************************


(defvar *records-of-common-ground-through-ALGORITHM*     0)


(defun reset-records-of-common-ground-through-ALGORITHM-keyword-counter ()
  (setq *records-of-common-ground-through-ALGORITHM* 0))


(defun make-new-records-of-common-ground-through-ALGORITHM  ()
  (intern (format nil "RECORD-OF-COMMON-GROUND  ~d"
                (incf   *records-of-common-ground-through-ALGORITHM*))
          (find-package 'keyword)))


;(define-ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT-command
;  (com-animation--- :menu-accelerator t) ()
;   (let ((stream (dw:get-program-pane 'pane-3)))
;     (setq *graphic* stream)
;     (draw1)))

(defvar *graphic* nil)


;;;;;;(defvar *graphic* (tv:make-window 'dw:dynamic-window
;                                  :edges '(100 100 500 500)))

(defun draw1 ()
  (let ((x-and-y-values (multiple-value-list (send *graphic* :read-cursorpos))))
    (loop for j from  (car x-and-y-values)   to  1000 by 15
          do
      (graphics:draw-circle  j (- (cadr x-and-y-values) 15)  30 :stream *graphic* :alu :draw)
      (graphics:draw-circle  j (- (cadr x-and-y-values) 15)  30 :stream *graphic* :alu :erase))))

;       (loop ;;;for i from 0 to 400 by 10
;             for j from 0 to 1000 by 10
;             do
;             (graphics:draw-circle  j 1100 30 :stream *graphic* :alu :draw)
;             (graphics:draw-circle  j 1100 30 :stream *graphic* :alu :erase))
;       (loop ;;;for i from 0 to 400 by 10
;             for j from 0 to 1000 by 10
;             do
;             (graphics:draw-circle  j 1130 30 :stream *graphic* :alu :draw)
;             (graphics:draw-circle  j 1130 30 :stream *graphic* :alu :erase))
;;      (sleep 1)
;       ;;;;(terpri *graphic*) (terpri *graphic*) (terpri *graphic*)
;       (let ((X-Y (multiple-value-list (send *graphic* :read-cursorpos))))
;
;       (loop
;             for j from (+ 10 (car X-Y)) to 1000 by 5
;             do
;             (graphics:draw-circle  j (+ 10 (cadr X-Y)) 25 :stream *graphic* :alu :draw)
;             (graphics:draw-circle  j (+ 10 (cadr X-Y)) 25 :stream *graphic* :alu :erase))

       ;;(send *graphic* :bury)
;)




;;;;;(defvar *result-of-RUNS*  nil)

(defun begin (ls &optional (stream *standard-output*))      ;;;;; list-of-lists
  (let ((result-of-RUNS  0) (result nil))
  ;;(send stream :clear-history)
```

```
      (setq *HOW* nil)
      (print '$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$  stream)
      (terpri stream) (terpri stream) (terpri stream)
      (print (format nil  "THE  VALIDATION ALGORITHM IS  ON ...."))
      (let ((stream (dw:get-program-pane 'pane-3)))
       (setq *graphic* stream)
       (drawl))
      (terpri stream) (terpri stream)
      (reset-records-of-common-ground-through-ALGORITHM-keyword-counter)
      (setq *common-ground-record*  nil)
      (setq *record-of-MENUS-ASKED* nil)
      (initialise-sequence)
      (from-scratch)
      ;;;;;;;;;;(let (;(result nil))
      (loop for list-of-GROUP-comparisons   in   ls
          do
        (let ((the-common-ground-as-is-is-when-you-run-only-2-feature-groups nil))  ;;;; ie (run "b51" "b44
")
                 (run (car list-of-GROUP-comparisons) (cadr list-of-GROUP-comparisons))
                 (setq the-common-ground-as-is-is-when-you-run-only-2-feature-groups *common-ground-record*)
                 (cond ((not (equal the-common-ground-as-is-is-when-you-run-only-2-feature-groups nil))

                          (setq  result-of-RUNS (+ 1 result-of-RUNS)))))

      (setq  *result-of-RUNS*  result-of-RUNS)

      (push
          (list (make-new-records-of-common-ground-through-ALGORITHM)
                 *help*
                 *record-of-PASSED*
                 ;;;;the-common-ground-as-is-is-when-you-run-only-2-feature-groups
                 *common-ground-record*)
          result)
    )

  )
     (setq *HOW* result)
     (print (format nil   "....... END   OF THE  AUDITING  FUNCTION ") stream)
     (print '$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$  stream)
    )
)




(defun from-scratch ()
  (setq *help* nil)
  (setq *record-of-PASSED* nil))
```

```
;;;-*-  Mode: lisp;  Package: austin-rover ; syntax: common-lisp-*-

(defun find-documentation  (trim-level)
  (let ((result nil))
    (loop for i in (cadr (assoc 'class *base-features-in-ENGLISH* ))
          do
      (cond ((string-equal trim-level (substring i 3 4)) (setq result
                                                  (string-append
                                                    trim-level
                                                    "----"
                                                    "MEANS---"
                                                  (string-trim  '(#\space)  (substring i 4 34)))))))
    result))


(defvar *class-levels* nil)
;       (remove-duplicates
;         (loop for i in (do-run)      ;;;;*TRANSMISION-TABLES*
;                   collect (cadr (car i)))
;         :test  #'string-equal))


(defun menu-for-CLASSES (&optional (stream *standard-output*))
  (let ((answers nil))
    (push
      ;;(multiple-value-list
        (dw:accepting-values (stream :own-window t :near-mode '(:point 100 700)
                                     :label
                              (format nil "THE TRIM LEVELS AVAILABLE FOR  -s  ARE:" *model*))
          (values
            (accept '(alist-member :alist ,(more)) :prompt "ENTER THE TRIM YOU WANT TO SEARCH THROUGH : "
                    :stream stream )

          )
        )
      answers)
    answers
  )
)



(defun more ()
  (setq *class-levels*
        (remove-duplicates
          (loop for i in (do-run)      ;;;;*TRANSMISION-TABLES*
                   collect (cadr (car i)))
          :test  #'string-equal))

  (let ((result nil))
    (loop for i in *class-levels*
          do
      (setq result (cons
                     (list i  ':value i ':documentation (find-documentation i))
                     result)))
result))


;*****************************************************************************

(defflavor  base-feature
        () () )


(defflavor  class
        (name
         (legal-statements  nil))
        (base-feature)
  :writable-instance-variables
  :initable-instance-variables)


(defvar  *classes*  nil)
```

```
(defmethod  (sys:print-self   class)   (stream  &rest  ignore)
  (print-node  self  stream))

(defmethod (make-instance  class)   (&rest  ignore)
           (push self *classes*))


(defmethod  (print-node   class) (stream)
  (format  stream  "Class -A"  name))


(defmethod  (node-inferiors   class)  ()
  legal-statements)

;*************************************************************

(defvar *legal-statements* nil)

(defflavor  legal-statement
        (name
         (conceptual-statements nil))  ;;;; TO HELP the PRINTING...

        (base-feature)
  :writable-instance-variables
  :initable-instance-variables)









(defmethod (make-instance  legal-statement)   (&rest  ignore)
           (push self *legal-statements*))

(defmethod  (print-node   legal-statement) (stream)
  (format  stream  "VALID-STATEMENT: -A"  name))

(defmethod  (node-inferiors   legal-statement)  ()
  conceptual-statements)


;*************************************************************




;(defun graph-top ()
;  (let ((answer nil))
;    (setq answer
;          (dw:menu-choose '(("Yes" . T) ("No" . nil)) :prompt "Do You Want To CONTINUE SEARCHING  FOR A
SPECIFIC CLASS ?"))
;    (cond (answer (graph))
;          (t nil))))



(defun graph (stream)
  (setq *classes* nil)
  (setq *legal-statements* nil)

  (let* ((help (car (menu-for-CLASSES stream)))
         (VALID-STATEMENTS (find-the-strings-legal-statements help))
         (instance (make-instance 'class :name help)))  ;;;;I  NEED  instance

    (loop for i  in VALID-STATEMENTS
          do
      (make-instance 'legal-statement :name i))
    (restore-data)
    (create-the-REAL-legal-STATEMENTS
      (find-the-INSTANCE help)
    )
    (graph-tree (find-the-INSTANCE help) stream)
```

```
))


(defun  restore-data  ()
  (loop for c in  *classes*
        do
    (when  (not (boundp-in-instance  c 'legal-statements))
      (setf (class-legal-statements  c)   nil)
  ))
  (loop for i  in  *LEGAL-STATEMENTS*
        do
    (when  (not (boundp-in-instance  i 'conceptual-statements))
      (setf (legal-statement-conceptual-statements  i)   nil)
  )))



(defmethod  (create-the-REAL-legal-STATEMENTS class) ()
  (loop for i in *LEGAL-STATEMENTS*
        do
    (push i
          (CLASS-legal-statements  self))))



(defun  graph-tree  (class &optional   (stream  *standard-output*))
            (fresh-line  stream)
  (format-graph-from-root   class
                            #'print-node   ;;princ
                            #'node-inferiors
                            :orientation :horizontal
                            :dont-draw-duplicates t
                            :stream  stream))




;*************************************************************


(defun find-the-strings-legal-statements (a-single-char-string)   ;; ie.  "Z"
  (let ((result nil) (result-TOP nil) (takis nil) (katerina nil))
    (loop for i in  (do-run)            ;;;;;*TRANSMISION-tables*
          do
      (cond ((string-equal (cadr (assoc ':class  i ))  a-single-char-string)
                (push i result))))
    (setq  result-TOP
          (remove-duplicates result :test #'equal))

    (loop for k in result-TOP
          do
      (push (list
              (loop for j in (cadr (assoc 'transmision *base-features-in-ENGLISH*))
                            do
                        (cond ((string-equal (substring j 3 4) (cadr (seventh k)))
                                  (return (string-right-trim '(#\space) (substring j 4 34))))))
              "~"
              (cadr (seventh k)) "TRANSMISION"
              (loop for j in (cadr (assoc 'drive *base-features-in-ENGLISH*))
                            do
                        (cond ((string-equal (substring j 3 4) (cadr (sixth k)))
                                  (return (string-right-trim '(#\space) (substring j 4 34))))))
              "~" (cadr (sixth k)) "DRIVE:"
              (loop for j in (cadr (assoc 'capacity *base-features-in-ENGLISH*))
                            do
                        (cond ((string-equal (substring j 3 4) (cadr (fifth k)))
                                  (return (string-right-trim '(#\space) (substring j 4 34))))))
              "~" (cadr (fifth k)) "CAPACITY:"
              (loop for j in (cadr (assoc 'performance *base-features-in-ENGLISH*)) ;;;; THERE is not exi
st the capacity "H"
```

```
                                ;;;;;; In the *base-features-in-ENGLISH* ie the BASE FEATURE CHART file is n
ot updated.!!!!!
                                do
                            (cond ((string-equal (substring j 3 4) (cadr (fourth k)))
                                      (return (string-right-trim '(#\space) (substring j 4 34))))))
                "-" (cadr (fourth k)) "PERFORMANCE:"
                (loop for j in (cadr (assoc 'family *base-features-in-ENGLISH*))
                                do
                            (cond ((string-equal (substring j 3 4) (cadr (third k)))
                             (return (string-right-trim '(#\space) (substring j 4 34))))))
                "-" (cadr (third k)) "FAMILY:"
                (loop for j in (cadr (assoc 'body *base-features-in-ENGLISH*))
                                do
                            (cond ((string-equal (substring j 3 4) (cadr (second k)))
                                      (return (string-right-trim '(#\space) (substring j 4 34))))))
                "-" (cadr (second k))  "BODY:")
              takis))

    (let ((help-result nil))
      (loop for i in takis do
        (loop for j in i do
          (setq help-result  (string-append j help-result)))
        (push
              (substring help-result 0 (- (length help-result) 3))   ;;;TO GET RID-OF  nil.
                 katerina)
        (setq help-result nil)
      ))
    katerina
))


(defun find-the-INSTANCE   (a-string-header)
            (let ((result nil))
              (loop for i in *classes* do
                (cond ((string-equal a-string-header (class-name i))
                              (setq result i))))
result))


;*****************************************************************************
(defun finish-it-IN-PIMS (ls  stream)
  (let ((help   (PIMS-CODE ls)))
    (loop for pim-final-with-base-and-countries  in help
          do
      ;;;;(print pim-final-with-base-and-countries  stream)
      (print '*****************************************************************************
             stream)
      (let ((PIMS-STATEMENTS (correct (car pim-final-with-base-and-countries)))
            (TERRITORIES (cdr  pim-final-with-base-and-countries))
            )
        (print (format nil "PIMS-STATEMENTS") stream)
        (print PIMS-STATEMENTS  stream)


        (loop for  COMMON-GROUND-before  in  TERRITORIES
              do
          (let* ((COMMON-GROUND (car COMMON-GROUND-before))
```

```
                (base-statement-in-PIMS (car COMMON-GROUND)) ;;;; (translate-base-to-english

                (countries (translate-to-english-COUNTRIES  (cadr COMMON-GROUND)))
                (restrictions-in-PIMS (caddr COMMON-GROUND)))
            ;;;;;(TRANSLATE-TO-English-COMBINATIONS-RESTRICTIONS
            ;;;;;(caddr COMMON-GROUND))))

            (print (format nil "BASE-FEATURES") stream)
            (print base-statement-in-PIMS stream)
            (print (format nil "COUNTRIES") stream)
            (print countries stream)
            (print (format nil "COMBINATIONS-RESTRICTIONS") stream)
                   (print restrictions-in-PIMS stream)
            (print '^^^^^^^^^^^^^^^^^^^^^^^^OR^^^^^^^^^^^^^^^^^^^^^^^^  stream))))
    ))
)


(defun finish-it-ONLY-FOR-ONE-FEATURE-GROUP-SPECIFICATION (ls  stream)          ;;;;;;;;;"COMMON-GROUND-R
ECORD"
  (let ((help   (PIMS-CODE ls)))
    (loop for pim-final-with-base-and-countries  in help
          do
      ;;;;(print pim-final-with-base-and-countries  stream)
      (print '*****************************************************************************
stream)
      (let ((PIMS-STATEMENTS (cond
                                ((string-equal (car (car pim-final-with-base-and-countries)) "plus")
                                       (list (string-append "+" (second (car pim-final-with-base-and-count
ries)))))
                                (t  (list (list (string-append "+" (second (car pim-final-with-base-and-co
untries))))
                                           (list (string-append "-" (second (car pim-final-with-base-and-co
untries))))))))
            (TERRITORIES (cdr  pim-final-with-base-and-countries))
           )
        (print (format nil "PIMS-STATEMENTS") stream)
        (print PIMS-STATEMENTS stream)

        (loop for  COMMON-GROUND-before  in  TERRITORIES
              do
          (let* ((COMMON-GROUND (car COMMON-GROUND-before))
                 (base-statement (translate-base-to-english  (car COMMON-GROUND)))
                 (countries (translate-to-english-COUNTRIES  (cadr COMMON-GROUND)))
                 (restrictions (TRANSLATE-TO-English-COMBINATIONS-RESTRICTIONS
                                     (caddr COMMON-GROUND))))

            (print (format nil "BASE-FEATURES") stream)
            (print base-statement stream)
            (print (format nil "COUNTRIES") stream)
            (print countries stream)
            (print (format nil "COMBINATIONS-RESTRICTIONS") stream)
                   (print restrictions stream)
            (print '^^^^^^^^^^^^^^^^^^^^^^^^OR^^^^^^^^^^^^^^^^^^^^^^^^ stream))))
    )
)
)


(defun finish-it (ls  stream)           ;;;;;;;;*COMMON-GROUND-RECORD*
  (let ((help   (PIMS-CODE ls)))
    (loop for pim-final-with-base-and-countries  in help
          do
      ;;;;(print pim-final-with-base-and-countries  stream)
      (print '*****************************************************************************
stream)
      (let ((PIMS-STATEMENTS (correct (car pim-final-with-base-and-countries)))
            (TERRITORIES (cdr  pim-final-with-base-and-countries))
           )
        (print (format nil "PIMS-STATEMENTS") stream)
        (print PIMS-STATEMENTS stream)

        (loop for  COMMON-GROUND-before  in  TERRITORIES
              do
          (let* ((COMMON-GROUND (car COMMON-GROUND-before))
                 (base-statement (translate-base-to-english  (car COMMON-GROUND)))
```

```
                (countries (translate-to-english-COUNTRIES  (cadr COMMON-GROUND)))
                (restrictions (TRANSLATE-TO-English-COMBINATIONS-RESTRICTIONS
                                    (caddr COMMON-GROUND))))

           (print (format nil "BASE-FEATURES") stream)
           (print base-statement stream)
           (print (format nil "COUNTRIES") stream)
           (print countries stream)
           (print (format nil "COMBINATIONS-RESTRICTIONS") stream)
                 (print restrictions stream)
           (print '^^^^^^^^^^^^^^^^^^^^OR^^^^^^^^^^^^^^^^^^^^ stream))))
    )
)
)



;*****************************************************************************************
;*****************************************************************************************

(defun PIMS-CODE (final-list) ;;;;;          *COMMON-GROUND-RECORD*

  (let ((result nil)  (record nil));;; (help-variable final-list))
    (loop for code-in-english    in   final-list
          do
      (let ((plus-or-minus (car code-in-english))
            (main-feature (cadr code-in-english))
            (other-features (caddr code-in-english)))
        (cond ((not (member (string-append main-feature other-features) record
                            :test #'string-equal))
               (push
               (find-all-common-PIMS-statements
                 plus-or-minus main-feature other-features   final-list)
               result)
               (push (string-append main-feature other-features)
                     record))
       )))
result))




(defun find-all-common-PIMS-statements (plus-or-minus main-feat-string  other-feat-string   ls)
  (let ((result (list (list plus-or-minus main-feat-string other-feat-string))))
    (loop for PIM-statement    in ls
          do
      (cond
        ((and (string-equal (cadr PIM-statement) main-feat-string)
              (string-equal (caddr PIM-statement) other-feat-string))
         (push (cadddr PIM-statement) result))))
(reverse result)))


;*****************************************************************************************
;*****************************************************************************************
;*****************************************************************************************


(defun translate-base-to-english  (ls)
        ;;; ie    ("H" "N" " " " " " " " " " ")

  (let* ((class (first ls))
          (body (second ls))
          (family (third ls))
          (capacity (fourth ls))
          (performance (fifth ls))
          (drive (sixth ls))
          (transmision (seventh ls))
     (class-english
```

```
            (base-in-english class  (cadr (assoc 'class *base-features-in-ENGLISH* ))))
          (body-english
            (base-in-english body  (cadr (assoc 'body *base-features-in-ENGLISH* ))))
          (family-english
            (base-in-english family  (cadr (assoc 'family *base-features-in-ENGLISH* ))))
          (capacity-english
            (base-in-english capacity (cadr (assoc 'capacity *base-features-in-ENGLISH* ))))
          (performance-english
            (base-in-english performance  (cadr (assoc 'performance *base-features-in-ENGLISH* ))))
          (drive-english
            (base-in-english drive  (cadr (assoc 'drive *base-features-in-ENGLISH* ))))
          (transmision-english
            (base-in-english transmision  (cadr (assoc 'transmision *base-features-in-ENGLISH* )))))

      (list (cond
            (class-english  (list 'CLASS** class-english))
            (t (list (format nil 'CLASS**)  'NO-RESTRICTED)))
          (cond
            (body-english (list 'BODY** body-english))
            (t (list 'BODY**  'NO-RESTRICTED)))
          (cond
            (family-english (list 'FAMILY** family-english))
            (t (list 'FAMILY**  'NO-RESTRICTED)))
          (cond
            (capacity-english (list 'CAPACITY** capacity-english))
            (t (list 'CAPACITY**  'NO-RESTRICTED)))
          (cond
            (performance-english (list 'PERFORMANCE** performance-english))
            (t (list 'PERFORMANCE**  'NO-RESTRICTED)))
          (cond
            (drive-english (list 'DRIVE** drive-english))
            (t (list 'DRIVE**  'NO-RESTRICTED)))
          (cond
            (transmision-english (list 'TRANSMISION** transmision-english))
            (t (list 'TRANSMISION**  'NO-RESTRICTED)))
      )
))



(defun base-in-english (one-char-string     ls)
  (loop for i in ls
        do
    (cond
      ((string-equal (substring i 3 4) one-char-string) (return (string-right-trim '(#\space)
                                                                 (substring i 4 31)))))
))

;*****************************************************************************************
;*****************************************************************************************



(defun   correct (ls)
    (let* (
          (help (apoint-each ls))


          (plus-statement (append (list (car help))
                                  (get-plus  (cadr help))))
          (minus-state (get-minus (cadr help))))
      ;(print plus-statement)
      ;(print minus-state)

      (cond ((equal minus-state nil) plus-statement)
            (t
             (recur  minus-state plus-statement))
      )
    )
)
```

```
(defun get-minus (ls)
  (let ((result-minus nil))

    (loop for i in ls
          do
      (cond ((string-equal (car i) "plus-and-minus") (push (cadr i) result-minus))))
    result-minus))


(defun apoint-each (ls)
          ;;;;;;("PLUS-AND-MINUS,PLUS,PLUS" "B15X" "B02B,B11C,B12C")
(let ((result nil))
  (let ((signs (car ls))
        (features (car (reverse ls)))
        (main-feature (cadr ls)))
    (push  main-feature result)
    (push
      (together signs features) result)
)(reverse result)))



(defun together (a-string  another-string)
  (let ((result nil)
        (result1
          (read-pims (string-append a-string ",takis")))
        (result2
          (read-pims (string-append another-string ",takis")))
       )
    (loop for i in result1 for j in result2
          do
          (push (list i j) result))
result))


 (defun read-pims (comments-string)
  (loop with input-stream = (make-string-input-stream
                              comments-string)

        for (word eof . rest) = (multiple-value-list   ;;;;;;; BECAUSE  THE function DOES NOT
                                                       ;;;;;;  work  correctly.
                                  (read-delimited-string '(#\,);;;;;;space #\return #\line)
                                                          input-stream
                                                          () !eof))
        until eof
        unless (zerop (string-length word))
          collect word
            )
  )





(defun get-plus (ls)
  (let ((result-plus nil))
    (loop for i in ls
          do
      (cond ((string-equal (car i) "plus") (push (cadr i) result-plus))))

      (let ((statement nil))
        (loop for i in result-plus
            do
                (push i statement) (push "+" statement))
      statement))
)



;(defun  recur-TOP   (ls statement)
```

```
;  (let ((result nil) (result-final nil))
;    (loop for pim-statement  in  (recur ls statement)
;         do
;      (push (list (length pim-statement)    pim-statement)  result))
;    (let* ((numbers (clear
;                     (list 'max
;                         (loop for length-and-statement  in  result
;                               collect
;                                 (car length-and-statement)))))
;         (maximum (eval numbers))
;         )
;
;      (loop for length-and-statement in result
;           do
;       (cond ((equal (car length-and-statement) maximum)
;                (push (cadr length-and-statement) result-final))))
;    )
;result-final))



(defun  recur (ls statement)
  (let ((result  nil) (result-first nil) (result-help nil) (result-top nil) (takis nil))
         (loop for minus-feature in ls
               do
           (push "+" result-first)
           (push minus-feature result-first))
;;;;;; WE NEED TO PUT  list OF statement TO TREAT STATEMENT AS ONE plus statement
         (setq result-help (append (list statement)  result-first))
         ;(print result-help)

         (let* ((result-second (remove "+" result-help))
                (combinations (combination-for-one (car result-second)
                                                   result-second)))
           ;(print result-second)
           ;(print combinations)
;;;;BECAUSE IT DOES'NT  WORK FOR ONLY  one.
           (cond ((equal (length combinations) 1)
                     (setq result-top (plus-minus-of-TWO  (car combinations))))
                 (t
           (loop for combination in  combinations
                 do
             (push (plus-minus-of-TWO  combination) result))

           (setq result-top  (sotiris result))
                 )
           )
        );;;; BECAUSE IT COMMES WITH DIFFERENT format,  DEPENDENT on that IF THE sotiris
         ;;;; FUNCTION HAS BEEN invoked  OR  NOT.
         (cond ((equal (length result-top) 1)
                  (loop for i in result-top
                        do
                    (loop for PIMS-STATEMENT in i
                          do
                          (push
                            (clear PIMS-STATEMENT) takis)))
               )
               (t
                (loop for PIMS-STATEMENT in result-top
                      do
                          (push
                            (clear PIMS-STATEMENT)  takis))
               )
        )
takis
))



(defvar *pims* nil)
```

```
(defun sotiris (ls)
  (setq *pims* nil)
  (now ls)
  *pims*
  )


(defun now (ls)  ;ie
;    ((("B12C" "+" "B11C") ("B12C" "-" "B11C")) (("B12C" "+" "B02B") ("B12C" "-" "B02B"))
;        (("B12C" "+" "cr56") ("B12C" "-" "cr56")))

(cond
  ((null (cdr ls)) nil)
  (t ;
     (let ((result nil))

       (loop for i in (car ls)
             do
         (loop for j in (cadr ls)
               do
           (setq result (cons (append i (cdr j)) result))))
       (setq *pims* (list result))
       (setq result nil)
      )


     (now (append *pims* (cddr ls)))

  )
))



(defun plus-minus-of-TWO        (ls)  ;;    (one-string  another-string)
  (list (list  (car ls)"+" (cadr ls))
        (list  (car ls) "-" (cadr ls)))
)

;*****************************************************************************************
;*****************************************************************************************

;
;(drive       (
; "A50ARIGHT HAND      STEERING        RHS      RHS"
;"A50BLEFT HAND       STEERING        LHS      LHS"
;        ))



(defun  TRANSLATE-TO-English-COMBINATIONS-RESTRICTIONS  (ls)
       ;;;(("R" "R05A" "=" "R05B") ("P" "R05A" "=" "R05B"))
  (let ((result nil))
    (cond ((equal ls nil) (setq result   "GNIHTEMOS-HTIW-DEGAKCAP-RO-DETCIRTSER-TON-SI-TI"))

          (t
           (loop for restriction  in   ls
                 do
             (push (English-COMBINATIONS-RESTRICTIONS restriction) result))
          )
    )
(list (reverse result))))  ;;;;; In oredr to GRAPH IT



(defun English-COMBINATIONS-RESTRICTIONS (ls)
  (let ((result nil))
    (loop for i in ls
          do
      (push (ENGLISH-word i) result))
(reverse result)))
```

```
(defun ENGLISH-word (a-string)
  (let ((result nil))
    (cond
      ((string-equal a-string "R") (setq result "RESTRICTED-TO-PACKAGE "))
      ((string-equal a-string "P") (setq result "PACKAGED-WITH "))
      ((string-equal a-string "=") (setq result "O R"))
      ((string-equal a-string "+") (setq result "W I T H"))
      ((string-equal a-string ",") (setq result "A N D "))
      ((string-equal a-string "<") (setq result "AND  NOT"))
      ((string-equal a-string "/") (setq result "IF  NOT"))
      ((string-equal a-string "?") (setq result "I F"))
      (t (let ((help-result
           (loop for line-entry in   (find-the-ADDITIONALS-of-model)
                          do
                       (cond
                         ((string-equal (substring
                                          (cadr (assoc ':feature-description line-entry))
                                          0 4)
                                        a-string)
                          (return
                            (cadr (assoc ':FEATURE-DESCRIPTION line-entry)))
                         )
                       ))
              ))));;;;BECAUSE FOR strange REASONS !!!!!, again !!!! the FEATURE-DESCRIPTION
                ;;;;;  OF THE "RESTRICTED" may not included in the ADDOTIONAL FEATURES.
           (cond ((equal help-result nil)
                  (setq result
                        (loop for group-entry in   *the-GROUP-headers*
                          do
                       (cond
                         ((string-equal (substring group-entry 0 3)

                                        (substring a-string 0 3))
                          (return
                            (string-right-trim '(#\space) (substring  group-entry 0 34)))
                         )
                       ))
                  ))
                 (t (setq result help-result))
           )
          )  ;;;let
        )
      )
    result))
```

```
;*****************************************************************************************************
***************
;;;;;;;;;; Assuming we have ONLY  one  FEATURE GROUP for the SPECIFICATION OF THE Part Description
;;;;;;;;;   ie......
;Command: *grouped*
;(("B51"
;  (("B51F"
;    (("B51F" "Z" (#<Entity-Handle of FEATURE-CHART-ENTRY 83/1 400163706>) NIL)
;     ("B51F" "Y" (#<Entity-Handle of FEATURE-CHART-ENTRY 83/2 400163520>) NIL)))
;   ("B51L"
;    (("B51L" "Z" (#<Entity-Handle of FEATURE-CHART-ENTRY 85/1 400162756> #<Entity-Handle of FEATURE-CHAR
T-ENTRY 85/2 400162570>)
;      NIL)
;     ("B51L" "Y" (#<Entity-Handle of FEATURE-CHART-ENTRY 86/1 400162402>) NIL)))
;   ("B51R"
;    (("B51R" "X" (#<Entity-Handle of FEATURE-CHART-ENTRY 86/2 400162224> #<Entity-Handle of FEATURE-CHAR
T-ENTRY 87/1 400162036>)
;      NIL)
;     ("B51R" "W" (#<Entity-Handle of FEATURE-CHART-ENTRY 87/2 400161666>) NIL))))))
;;;;; I have Changed the INTERSECTION in the FUNCTION:   "all-trims-common-with-RESTRICTIONS". HENCE:
;;Command: (all-trims-common-with-RESTRICTIONS '("B51F" (("B51F" "Z" # NIL) ("B51F" "Y" # NIL))) nil)
;((NIL NIL)       ;;;;  The reference from the Pseudo nil  Feature I manually  put  !!!
; ("B51F"
```

```
;  (("Y" (#<Entity-Handle of FEATURE-CHART-ENTRY 83/2 400163520>)) ("Z" (#<Entity-Handle of FEATURE-CHART
-ENTRY 83/1 400163706>)))
;  ))
;Command:

(defun  the-Specification-of-a-part-with-only-ONE-FEATURE-GROUP ()
  (let ((result nil))
    (cond ((equal (length *grouped*) 1)
           (setq result
                 (remove-duplicates
                   (specify-for-each-specific-feature-with-a-pseudo-NIL-ALL (car *grouped*))))))
    result))


(defun specify-for-each-specific-feature-with-a-pseudo-NIL-ALL (ls)
  (let ((result nil))
    (loop for feature-reference in (cadr ls)
          do
      (push (specify-for-each-specific-feature-with-a-pseudo-NIL feature-reference) result))
    result))



;Command: (specify-for-each-specific-feature-with-a-pseudo-NIL '("B51F" (("B51F" "Z" # NIL) ("B51F" "Y" #
 NIL))))
;("PLUS" "B51F" ""
; ((("Y" " " " " " " " " " " " " ")
;   ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T
53G" "T54C" "T56J"
;    "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E" "T
59G" "T51R" "T51V"
;    "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R" "T
56L" "T58C" "T60R"
;    "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T" "T
56U" "T58N" "T57J"
;    "T58J")
;   (("+" "B53B")))
;  (("Z" " " " " " " " " " " " " ")
;   ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T
53G" "T54C" "T56J"
;    "T56V" "T57X" "T59L" "T59V" "T53N" "T50R" "T53C" "T57E" "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T
50V" "T52R" "T55T"
;    "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T
50Z" "T51G" "T52G"
;    "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;   (("+" "B53B")))))
;Command:



(defun specify-for-each-specific-feature-with-a-pseudo-NIL (a-feature-reference)
;  a-feature-reference   =   ("B51F"
;                            (("B51F" "Z" (#<Entity-Handle of FEATURE-CHART-ENTRY 83/1 400163706>) NIL)
;                             ("B51F" "Y" (#<Entity-Handle of FEATURE-CHART-ENTRY 83/2 400163520>) NIL)))
  (let ((help (cadr (all-trims-common-with-RESTRICTIONS a-feature-reference nil))) (result nil)
        (PLUS-OR-MINUS  (give-the-PLUS-OR-MINUS-afterwards  a-feature-reference)))
;;;;;;  help    ======
;;'((NIL NIL)       ;;;;  The reference from the Pseudo nil  Feature I manually  put  !!!
; ("B51F"
;  (("Y" (#<Entity-Handle of FEATURE-CHART-ENTRY 83/2 400163520>)) ("Z" (#<Entity-Handle of FEATURE-CHART
-ENTRY 83/1 400163706>)))
;  ))
    ;;;(print help)
    (let ((the-reference-of-MASTER-Feature
            (create-the-logic-references help)))
      (loop for MASTER-POSSIBILITY in the-reference-of-MASTER-Feature
            do
        (let* ((split-of-Master (part-apart-the-list MASTER-POSSIBILITY))
               (Master-Base-Features (car split-of-Master))
               (Master-Restrictions (cadr split-of-Master))
               (GROUPS-AVAILABLE-FOR-Master (GROUP-RESTRICTIONS-collect
                                             Master-Restrictions))
               (TERRITORIES-AVAILABLE-FOR-Master (TERRITORY-RESTRICTIONS-collect
                                                  Master-Restrictions))
               (COMBINATIONS-or-COMMENTS-of-Master
                 (COMBINATIONS-or-COMMENTS-RESTRICTIONS-collect Master-Restrictions))
```

```
            (final-the-COUNTRIES-AVAILABLE-for-MASTER
               (set-difference
                       GROUPS-AVAILABLE-FOR-Master TERRITORIES-AVAILABLE-FOR-Master
                       :test #'string-equal))
            (BASE-FEATURES-to-be-compared
              (reverse        ;;BECAUSE IT comes in OPPOSITE direction
                (NO-opposite  Master-Base-Features '(" " " " " " " " " " " " " ")))) ;;;; a pseudo BAS
E FEATURES
                       ;;;; in order to keep the same format with the algorithm

            )
;          (print Master-Base-Features )(print GROUPS-AVAILABLE-FOR-Master) (print TERRITORIES-AVAILABLE-F
OR-Master)
          (print COMBINATIONS-or-COMMENTS-of-Master) (print final-the-COUNTRIES-AVAILABLE-for-MASTER)
;          (push  (list
                BASE-FEATURES-to-be-compared final-the-COUNTRIES-AVAILABLE-for-MASTER COMBINATIONS-or-
COMMENTS-of-Master)
              result))))
    (list ;;;;;; reference ***** and *******    ...In order to keep the same format with the old *COMMO
N-GROUND-RECORD*
      PLUS-OR-MINUS
      (car a-feature-reference)
      ""
      (remove-duplicates result))))


;Command: *record-of-MENUS-ASKED*
;(("B69D-----HEATED DRIVERS & PASS SEAT    TWOHOT  " "PLUS-AND-MINUS"))
;Command:
;

(defun give-the-PLUS-OR-MINUS-afterwards (a-feature-reference)
  (PLUS-or-MINUS a-feature-reference nil)
  (cond ((> (length (cadr (car *grouped*))) 1) "PLUS")
                   (t (BLOCK  plus-or-minus
                          (loop for answer in *record-of-MENUS-ASKED*
                                do
                            (cond ((string-equal  (car a-feature-reference) (substring (car answer) 0 4
))
                                      (return-from  plus-or-minus (cadr answer)))))
                          "plus"))
))


;Command: *COMMON-GROUND-RECORD*      *****
;(("PLUS-AND-MINUS" "B51R" "B69D"
;  ((("X" "W" " " " " " " "B" " ") ("T52J" "T53N" "T50R" "T57E" "T59E" "T59G") (("+" "B53A")))
;   (("X" "N" " " " " " " "B" " ") ("T52J" "T53N" "T50R" "T57E" "T59E" "T59G") (("+" "B53A")))))
; ("PLUS-AND-MINUS" "B51L" "B69D" ((("Y" " " " " " " "D" "B" " ") ("T52J" "T53N" "T50R" "T57E" "T59E" "T5
9G") (("+" "B53B")))))
; ("PLUS-AND-MINUS" "B51F" "B69D" ((("Y" " " " " " " "D" "B" " ") ("T52J" "T53N" "T50R" "T57E" "T59E" "T5
9G") (("+" "B53B"))))))
;Command:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; The  below FROM the file :   "statice-the-program-algorithm"  the functions "goal" or "goal-else"
.                 ;;;;;;;;;;
;(let ((help (PLUS-or-MINUS   Master-feature  Feature-from-group)))
;(list    ;;;;;;  *******
;              help
;              (car Master-feature)
;              (car Feature-from-group)
;
;              found-common
```

```
;;;-*-  Mode: lisp;  Package: austin-rover ; syntax: common-lisp-*-


;;;;;in file :     * statice-the-program-the-menus.lisp >takis>statice ATC-0:                                    (30
)

(defun    give-the-priority-for-the-SEARCH-of-ALGORITHM  ()
  (let ((result nil) (result-musts nil) (result-maybes nil) (help (numbering-the-feat-GROUPS-top)))
    (loop for feature-with-priority-number    in   help
          do
      (loop for must in *musts-after*
            do
        (cond ((string-equal must (car feature-with-priority-number))
                  (push feature-with-priority-number result-musts)))

      ))
    (cond ((equal (cadr (car help)) 'pseudo)
              (let ((katerina nil))
                (loop for i in help
                      do
                  (push (car i) katerina))
                (setq result (reverse (combination-for-one (car (reverse katerina)) (reverse katerina))))
))
          (t
    (loop for feature-with-priority-number    in   help
          do
      (loop for maybe in *maybes-after*
            do
        (cond ((string-equal maybe (car feature-with-priority-number))
                  (push feature-with-priority-number result-maybes)))

      ))

    (setq result (loop for feature-with-priority-number in
                            (append (sort result-musts #'> :key #'second)
                                    (sort result-maybes #'> :key #'second))
                        collect (car feature-with-priority-number)))
    ;;;;WE DON'T NEED to  put out the GROUP with length  0 BECAUSE it WILL not APPEAR at
    ;;;;;     THE  *subdescription-tables* ANYWAY.
    (reverse
      (combination-for-one (car result)  result))
    )  ;;;t
) ;;;cond
))


;;;;;;;;; in  file:    * STATICE-the-program-algorithm.lisp >takis>statice ATC-0:
 (10)


(defun numbering-the-feat-GROUPS-top  ()
  (let ((result nil))
    (let ((stream (dw:get-program-pane 'pane-3)))
      (cond ((menu-query-if-he-wants-his-own-sequence)
                (let ((new (find-the-english-of-the-feature-groups-yes)))
                  (let ((takis nil))
                    (loop for i in  (QUERY-for-the-SEQUENCE-from-ENGINEER-MENU new stream)
                          do
                      (push (list (subseq i 0 3) 'pseudo) takis))
                    (setq result takis))))
            (t
             (loop for (description-key . subdescription-table) in *SUBDESCRIPTION-TABLES*
                   do
               (push (list description-key (length (keys subdescription-table)))
                     result))
            )))
  result))


(defun numbering-the-feat-GROUPS  ()
  (let ((result nil))
    (loop for (description-key . subdescription-table) in *SUBDESCRIPTION-TABLES*
          do
      (push (list description-key (length (keys subdescription-table)))
            result))
  result)
```

```
)


(defun QUERY-for-the-SEQUENCE-from-ENGINEER-MENU  (feature-descriptions   &optional (stream *s
tandard-output*))

  ;(let ((stream (dw:get-program-pane 'pane-3)))
        (let ((answers nil))
    (push
      (dw:accepting-values (stream :own-window t
                                   :near-mode '(:point 100 700)
                                   :label "GIVE THE FEATURE GROUPS IN THE SEQUENCE YOU THINK IS APPRO
PRIATE :")
        (accept (clear (append (list  'subset)  feature-descriptions))
          :prompt "CLICK IN ORDER YOU PREFER THE FEATURE GROUPS :"  :stream  stream))       ;
      answers)
     (car answers)))



(defun find-the-english-of-the-feature-groups-yes ()
  (let* ((feature-groups-no-english (append "MUSTS-AFTER" *maybes-after*))
         (feature-groups-in-english (collect-from-FEAT-GROUP-INDEX-REPORT--the-VALID
                                     feature-groups-no-english *model*)))
    (loop for i in feature-groups-in-english
              collect (cadr i))))



(defun menu-query-if-he-wants-his-own-sequence ()  ;;;;;; YOU MUST NOT put different character format to
this menu ie. BOLD , italic etc.
  ;;;;;;;; only  nil.nil.nil
  (values
    (dw:menu-choose '(("Yes" . T) ("No" . nil)) :prompt (zl:format nil "DO YOU WANT TO ENTER YOUR OWN SEQ
UENCE OF FEATURE GROUPS  ? ") :near-mode '(:point 100 700))))

;(defun another-translation-to-English-the-feature-group-takis (feat-string)
;  (let ((help (find-the-enlish-of-the-feature-groups-yes)))
;    (BLOCK takis
;      (loop for i in help
;          do
;       (cond ((string-equal feat-string  (car i))
;                (return-from takis (cadr i)))))
;      nil)))
```

```
;;;-*- Mode: lisp;  Package: austin-rover ; syntax: common-lisp-*-


;;;(setq  *sequence* ' ((:body ()) (:family ()) (:capacity ()) (:performance ()) (:drive ()) (:transmisio
n ())))

(setq  *the-GROUP-headers* '(


"A10-MARQUE                          AXXXXXXXXXXXX XX   00521 Nov 88*"
"A20-MODEL RANGE (AUSTIN ROVER)      AXXXXXXXXXXXX XX   01021 Nov 88*"
"A22-MODEL RANGE (ROVER-TRIUMPH)     A                  01521 Nov 88*"
"A30-BODY VARIANT - PASSENGER CARS BXXXXXXXXXXXX XX   00521 Nov 88*"
"A40-TRANSMISSION                    TXXXXXXXXXXXX XX   00521 Nov 88*"
"A50-STEERING HAND                   QXXXXXXXXXXXX XX   00521 Nov 88*"
"A60-VEHICLE CLASS CARS AUST/MORRISAX                   02021 Nov 88*"
"A63-VEHICLE CLASS (LM10 ONWARDS)  A XXXXXXXXXXX XX   02521 Nov 88*"
"A70-ENGINE CAPACITY-AUSTIN MORRIS LXXXXXXXXXXXX XX   01521 Nov 88*"
"A71-ENGINE CAPACITY-ROVER-TRIUMPH L                    02021 Nov 88*"
"A80-ENGINE SERIES                   LXXXXXXXXXXXX XX   005 2 May 89*"
"A99-PERFORMANCE                     LXXXXXXXXXXXX XX   02521 Nov 88*"
"B01-ASHTRAY TYPE                    F          X X X   04521 Nov 88*"
"B02-BACKLIGHTS/REAR WINDOWS         C X                01521 Nov 88*"
"B03-PARTITIONS                      B     X            03021 Nov 88*"
"B04-THEME PACKAGES                  B     XX XX X      04021 Nov 88*"
"B05-BUMPER CONDITION                DXX   XXX     XX   03021 Nov 88*"
"B06-BUMPERS                         DXXXX    XX X      03521 Nov 88*"
"B10-CROSSMEMBER CARPETS             E  X               03006 SEP 89 "
"B11-REAR QUARTER LIGHTS             CXXX   X           02021 Nov 88*"
"B12-EXTERIOR MIRRORS FINISH         C XXX X XXXXX      03521 Nov 88*"
"B14-EXT MIRROR GLASS CONDITIONS     C     X   X X      03021 Nov 88*"
"B15-EXTERIOR MIRRORS                CXXXX XXXXXX  X    02521 Nov 88*"
"B19-FLOOR COVERINGS-CAR INTERIORS EXXX  XXX     X     02021 Nov 88*"
"B20-FLOOR COVERING-RR COMPARTMENT E                    02521 Nov 88*"
"B21-FLOOR TYPES                     B     X            02521 Nov 88*"
"B22-HANDBRAKE GAITERS               F  X               07021 Nov 88*"
"B23-GLAZING TYPE                    CXXXX XXXXX X X    01021 Nov 88*"
"B24-HEADLININGS                     E  X               00521 Nov 88*"
"B25-FRONT HEAD RESTRAINTS           HXXX  XXXXX   X    07521 Nov 88*"
"B26-CONSOL                          F                  07521 Nov 88*"
"B27-INSTRUMENT BINNACLES AND PACKSY  X  XXXX    X     00521 Nov 88*"
"B29-FRONT LASHING  EQUIPMENT        KX                 00521 Nov 88*"
"B30-SCREEN HEATING                  J             X    01021 Nov 88*"
"B31-ADDITIONAL FLOOR COVERING       E     X   X X X    03521 Nov 88*"
"B32-CARPET PROTECTION COVERS        E XXX X  X         04021 Nov 88*"
"B33-MUDFLAPS                        D XXX X XXX X X    13521 Nov 88*"
"B34-RADIO -TYPES                    XXXXXXXXXXXXX X    01021 Nov 88*"
"B35-RADIO -AERIALS                  XXXX XX  X         01521 Nov 88*"
"B36-RADIO -SPEAKERS                 XXXXXXXXXXXXX X    02021 Nov 88*"
"B37-SEAT ADJUSTMENT-FRONT           HX                 01521 Nov 88*"
"B38-SEAT BELTS -FRONT               F          X X     10521 Nov 88*"
"B39-SEAT BELTS -REAR                FXXXXXXXXXX X X    11021 Nov 88*"
"B40-SEAT FACE MATERIAL              HXX XXXX XXXX      04521 Nov 88*"
"B41-SEAT MATERIALFINISH             H        X         05521 Nov 88*"
"B43-SEAT CROSS MEMBERS              H      X           04021 Nov 88*"
"B44-SEATS -RECLINING                HXXX  XXXX    X    01021 Nov 88*"
"B45-SEAT TYPE -FRONT                H XXXX X  X  X      00521 Nov 88*"  ;;;;; IT is a mistake  !!!!
"B47-DOOR INSERTS                    E   X   XXX X XX   01521 Nov 88*"
"B48-DOOR CASINGS                    EX X   X           01021 Nov 88*"
"B49-REAR SEAT ARMRESTS              H   X  XXX    X    06521 Nov 88*"
"B50-RADIO HOUSINGS                  XX                 04521 Nov 88*"
"B51-SEAT FACE MATERIAL              HXXX   XXXX X XX   05021 Nov 88*"
"B52-SUN VISORS                      FX X        X X    01021 Nov 88*"
"B53-SEAT SIDE ROLLS                 H      X           06021 Nov 88*"
"B54-SPEAKER CONFIGURATION           X         X        03012 Apr 89*"
"B55-TOWING EQUIPMENT                KX    X            01021 Nov 88*"
"B56-UNDERFRAMES                     K         X X      01521 Nov 88*"
"B57-GLAZING                         C X                00521 Nov 88*"
"B58-BODY SHELL VARIATIONS           B      X           01021 Nov 88*"
"B60-BODY STRENGTHENING              B        X X X     01521 Nov 88*"
"B61-REAR SEATS                      H XX  X  XXX       02521 Nov 88*"
"B64-WHEEL ARCH FLARES               D         X        06521 Nov 88*"
"B65-ROOF REINFORCEMENT              B     X  X         02021 Nov 88*"
"B67-WHEEL ARCH MATS                 E                  05021 Nov 88*"
"B68-SEAT OPERATION                  X         XXX      06521 Nov 88*"
"B69-SEAT HEATING                    H      XX XXX      02021 Nov 88*"
```

```
"B73-AIR BAG                          F          X X      11521 Nov 88*"
"B76-SEAT BELT SENSORS                Y       X XX X X    12021 Nov 88*"
"B77-SILL EXTENSIONS                  D  X           X    05521 Nov 88*"
"B78-DOOR SIGNS                       D                   20021 Nov 88*"
"B80-REAR APPLIQUES                   D          X        05021 Nov 88*"
"B81-PLINTHS REAR NUMBER PLATES       D         X         13021 Nov 88*"
"B82-SEATS -OCCASIONAL                H         X         03021 Nov 88*"
"B83-NO PLATE BRACKETS/BASES          D          X X      12521 Nov 88*"
"B86-SPARE WHEEL COVER                E X                 05521 Nov 88*"
"B90-BODY SIDE MOULDINGS              D XXX XXXXX X XX    07021 Nov 88*"
"B92-FACIA INSERTS                    F XX   XX   X  X    02021 Nov 88*"
"B93-FACIA MATS                       F  X        X       02521 Nov 88*"
"B94-DOOR BINS                        FX                  06021 Nov 88*"
"B95-PARCEL SHELVES                   E XX                06021 Nov 88*"
"B96-TUNNEL CONSOLE                   F  X  X  XX   X     08021 Nov 88*"
"B97-TUNNEL CONSOLE INSERTS           F          X X X    08521 Nov 88*"
"B98-INTERIOR SIDE RACKING            F     X             12521 Nov 88*"
"C02-ANTI-ROLL BARS                   R       X           02021 Nov 88*"
"C04-BRAKE SPECIFICATION              S                   02021 Nov 88*"
"C07-BRAKE SERVO                      S                   02521 Nov 88*"
"C08-BRAKE TYPES                      S  X   XXX          00521 Nov 88*"
"C09-BRAKE SYSTEMS                    S      XXXXXX X     01021 Nov 88*"
"C11-BRAKE PAD TYPES                  S           X       015           "
"C14-EXHAUST TAIL PIPE TRIM           UX X   XXX          09521 Nov 88*"
"C15-MA GEARBOX FINAL DRIVE RATIOS    T  X   X            03521 Nov 88*"
"C16-FINAL DRIVE RATIOS               T X X X  XXXX X     04021 Nov 88*"
"C17-FRESH AIR VENTS                  F                   03521 Nov 88*"
"C18-HEATERS                          JX                  00521 Nov 88*"
"C20-PETROL CAPS                      U                   01521 Nov 88*"
"C29-STEERING COLUMNS-LOCKING         Q          X X      02021 Nov 88*"
"C30-STEERING TYPES                   Q     XXXX    X     01021 Nov 88*"
"C31-FRONT&REAR SUSPENSION/SPRINGS    R  X  XXXXX X X     00521 Nov 88*"
"C34-TYRE GRADES                      R     X             06521 Nov 88*"
"C35-TYRE PLY -SIZE                   R XXXXXXXXXXX X     07021 Nov 88*"
"C36-TYRE PROFILES                    R      X            075 5 May 89*"
"C38-TYRE MANUFACTURES                RXXXXXXXXXXXX X     08521 Nov 88*"
"C39-WHEEL FINISH                     R X                 02520 Apr 89*"
"C40-WHEEL TRIMS                      RXXXXXXXXXX   XX    05521 Nov 88*"
"C41-WHEEL TYPE                       RXXXX XXXXXXX XX    03021 Nov 88*"
"C42-RIM SIZE                         R XXXXXXXXXXX XX    03521 Nov 88*"
"C44-ROAD WHEEL MANUFACTURERS         R XX   XX           04521 Nov 88*"
"C45-STOWAGE TRAYS                    F       X           09021 Nov 88*"
"C53-SPARE WHEEL TYPE                 R          X X      05021 Nov 88*"
"C56-WHEEL SIZE                       R                   04021 Nov 88*"
"C65-STEERING COLUMN TYPES            Q      XX     X     01521 Nov 88*"
"C68-COOLING SYSTEM                   P        X  X       01021 Nov 88*"
"C70-FUEL TANK                        U         X X       01021 Nov 88*"
"C71-GEARBOX SPEC SPEED/RATIO         T XXX XX XXXX X     02521 Nov 88*"
"C72-GEAR BOX CONDITION               T                   02021 Nov 88*"
"C74-ECONOMY GEARBOX                  T                   03021 Nov 88*"
"C75-GEARBOX TYPE                     T  XX XX XXXX X     01021 Nov 88*"
"C76-TRANSMISSION INTERLOCK           T          X X      01522 Mar 89*"
"C78-IDLER GEAR RATIOS                T X                 04521 Nov 88*"
"C82-SUSPENSION TYPES -REAR           R        XX X       01021 Nov 88*"
"C83-SUSPENSION TYPES-UNIT SALES      R                   01521 Nov 88*"
"C88-AIR CONDITIONING                 J   X  XXXX X X     01521 Nov 88*"
"C89-COMPRESSOR SAFETY VALVE          J           X       020           "
"C93-TYRE LABELS/INSTRUCTION          R       X   X       09021 Nov 88*"
"C94-TYRE SPEED RATING                R      XXXX X       08021 Nov 88*"
"C98-PETROL TANKS                     U                   00521 Nov 88*"
"D01-FOG/SPOT LAMPS                   W      XX   X       070 6 May 89*"
"D03-BATTERY CONDITION                Y                   10021 Nov 88*"
"D04-BATTERY PLATAGES                 Y                   10521 Nov 88*"
"D05-CIGAR LIGHTERS                   F XX  XXXXX X X     04021 Nov 88*"
"D06-TIME CLOCKS                      Y              X    04021 Nov 88*"
"D08-ECONOMETER                       Y                   03021 Nov 88*"
"D09-FLASHER REPEATERS/RE             W                   04021 Nov 88*"
"D10-BRITISH BATTERY MANUF STANDARDYXXXX XXXXX X          11521 Nov 88*"
"D11-FLASHER UNIT TYPE                W                   09521 Nov 88*"
"D14-ALTERNATORS                      N X   X  XX X       01021 Nov 88*"
"D16-HEADLAMP CIRCUIT FUSE            WX                  09021 Nov 88*"
"D17-HEADLAMP CLEANERS                W      XXXX X X     03021 Nov 88*"
"D19-HEADLAMP FLASHER                 W                   03521 Nov 88*"
"D20-SECURITY SYSTEMS                 Y         XXX X     08021 Nov 88*"
"D21-HORNS                            Y      XX     X     09521 Nov 88*"
"D26-INTERIOR ILLUMINATION            W         X X       10021 Nov 88*"
```

```
"D27-INTERIOR LAMP SYSTEM -REAR     W XX  X      X X    10521 Nov 88*"
"D30-GLOVEBOX LAMPS                 W  X                12021 Nov 88*"
"D31-ROOF SIGN WIRING               Y     X  X          12521 Nov 88*"
"D32-REARGUARD LAMPS                W X                 06521 Nov 88*"
"D33-REVERSING LAMPS                W X                 08021 Nov 88*"
"D34-PUDDLE LAMPS                   W       X   X       11521 Nov 88*"
"D35-CD AUTOCHANGER                 X         X X       06012 Apr 89*"
"D36-SIDEMARKERS                    W           X       045          *"
"D37-SPEEDOMETER MPH / KPH          Y           X       085          *"
"D38-STARTER TYPE                   N    X              00521 Nov 88*"
"D39-SWITCH CONDITION               Y                   04521 Nov 88*"
"D40-TACHOMETER                     YX                  02021 Nov 88*"
"D43-WIPER POSITION                 D          X        01521 Nov 88*"
"D46-WINDSCREEN WIPER - SPEED       D                   01021 Nov 88*"
"D47-REAR WINDOW WASH/WIPE FITMENT  D XX  X X           02021 Nov 88*"
"D48-HEADLAMP DIP CONDITIONS        WXXXXXXXXXX X X     02021 Nov 88*"
"D49-BATTERY AMP/HOUR RATINGS       Y                   11021 Nov 88*"
"D52-SPEEDOMETER TYPE               Y     X  XX         01521 Nov 88*"
"D55-TWEETERS                       X   X    XXX        02521 Nov 88*"
"D57-VEHICLE COMMUNICATIONS         X                   05521 Nov 88*"
"D62-AMPLIFIERS                     X         X   X     03521 Nov 88*"
"D63-AERIAL FREQUENCY TUNE          X         X X       05021 Nov 88*"
"D64-ASHTRAY ILLUMINATION           Y      XX     X     05521 Nov 88*"
"D65-CIGAR ILLUMINATION             Y      XX           06021 Nov 88*"
"D66-IGNITION ILLUMINATION          Y      X      X     06521 Nov 88*"
"D67-ON BOARD DIAGNOSTICS           Y           X       035          *"
"D68-HEADLAMP MODE                  W                   01521 Nov 88*"
"D70-RADIO CONTROL                  X           X X     04021 Nov 88*"
"D73-LAZY LOCKING                   C           X       050           "
"D76-OIL WARNING LIGHTS             Y         X         07021 Nov 88*"
"D78-COURTESY LAMPS                 W      X      X     11021 Nov 88*"
"D84-UNDER BONNET LAMPS             W         X         12521 Nov 88*"
"D85-FACIA ILLUMINATION             Y      XXX    X     05021 Nov 88*"
"D89-HEADLAMP LEVELING SYSTEM       W  X   XX   X X     02521 Nov 88*"
"D90-HEADLAMP SYSTEMS               W X                 00521 Nov 88*"
"D91-SIDE LIGHT CONDITIONS          W       X X X       05021 Nov 88*"
"D92-WINDOW LIFTS                   F XXXXXXXXX X X     05021 Nov 88*"
"D93-HEADLAMP INHIBITION            W         X X       08521 Nov 88*"
"D95-REARGUARD LAMP INHIBITION      W         X X       075 5 Jan 89*"
"D97-LIGHTING HARNESS               WX                  13021 Nov 88*"
"E01-BRAKE HORSE POWER              L                   04521 Nov 88*"
"E02-FUEL FEED TYPE                 U                   02521 Nov 88*"
"E03-COMPRESSION     RATIO          L                   03021 Nov 88*"
"E05-EXHAUST EMISSION CONTROLS      UXX      X          07521 Nov 88*"
"E06-EXHAUST RECIRCULATION SYSTEMS  U      X  XXX       08021 Nov 88*"
"E09-FUEL EVAPORATIVE LOSS CONTROL  U  X   X XXXX       04021 Nov 88*"
"E12-VALVE CONFIGURATION            L  X       X        04021 Nov 88*"
"E13-ENGINE ASPIRATION              L X      X          05021 Nov 88*"
"E15-PETROL PIPES                   U X                 03521 Nov 88*"
"E16-FUEL ECU TUNES                 N         XXX       02521 Nov 88*"
"E17-EMISSION AIR ENRICHMENT        U      X  X         06521 Nov 88*"
"E18-FUEL PUMP TYPE                 U    X   X          03021 Nov 88*"
"E27-UNLEADED FUEL POWER UNIT       L     X  X          06521 Nov 88*"
"E30-EXHAUST SYSTEM                 U X                 07021 Nov 88*"
"E33-TEMPERATURE SENSORS            U         XXX       08521 Nov 88*"
"E37-AIR CLEANERS                   P            X      00521 Nov 88*"
"E41-EXHAUST CATALYST               UXXX   XX XXX X     09021 Nov 88*"
"E42-CONTACT BREAKER SYSTEMS        N X                 01521 Nov 88*"
"E43-FUEL LOSS RESTRICTORS          UX                  04521 Nov 88*"
"E54-ENGINE CONTROL SYSTEMS         M     X  X          00521 Nov 88*"
"E55-STATIONARY ENG CONTROL SYSTEMSM        X X         02021 Nov 88*"
"E65-FUEL FILLER NECK               UXXX      X X       05521 Nov 88*"
"E66-FINISHED ENGINE CONDITIONS     L                   05521 Nov 88*"
"E99-EMISSION PACKAGES              UXXX   XX           06021 Nov 88*"
"F01-DOOR FRAME FINISHERS           D             X     11021 Nov 88*"
"F02-DOOR HANDLES                   CXXX  XXXX    X     04521 Nov 88*"
"F04-DOOR LOCK SYSTEMS              C XX  XXXXX X X     04021 Nov 88*"
"F06-REAR DOOR/BOOT RELEASES        F             X     10021 Nov 88*"
"F07-PETROL CAP FINISH              U  X                02031 AUG 89*"
"F12-DOOR PULLS                     F                   05521 Nov 88*"
"F13-GRAB HANDLES                   F  X  X             06521 Nov 88*"
"F15-ACCIDENT WARNING DEVICES       K         XX        02521 Nov 88*"
"F17-LOAD SPACE SIDE COVERS         E                   04521 Nov 88*"
"F19-REAR PARCEL SHELF              E                   06521 Nov 88*"
"F21-LOAD SPACE COVERS              E        X          07021 Nov 88*"
"F22-REAR SCREEN BLINDS             E        XX X X     07521 Nov 88*"
```

```
"F25-STEERING WHEEL FINISH          Q X X XX XX X XX   03021 Nov 88*"
"F26-STEERING WHEEL TYPE            Q XX  XXXX    XX   02521 Nov 88*"
"F29-REAR BADGES - R8               D      X           16015 Mar 89*"
"F30-CAPACITY BADGES                D XX               18521 Nov 88*"
"F31-REAR BADGES                    D     X   XX       15521 Nov 88*"
"F32-REAR CLASS BADGES              D X   X   XX       16521 Nov 88*"
"F33-BONNET BADGES                  D        X         19021 Nov 88*"
"F34-REAR CLASS BADGES              DXX X X            18021 Nov 88*"
"F35-MARQUE BADGES                  D X   X  XXXX      14521 Nov 88*"
"F36-STEERING WHEEL BADGES          Q X                03521 Nov 88*"
"F39-MARQUE SHIELDS                 D          XXX     15021 Nov 88*"
"F40-REAR BADGES (CONT)             D        X         17021 Nov 88*"
"F42-MISCELLANEOUS SYMBOLS          D          X X     19521 Nov 88*"
"F43-BONNET AND BOOT FINISHERS      D                  11521 Nov 88*"
"F47-SUN ROOFS                      F XXX XXXXXXX X    00521 Nov 88*"
"F48-REAR HEADRESTS                 H      XXXX X X    08021 Nov 88*"
"F52-HONDA BADGES                   D       X          17522 Feb 89*"
"F53-HONDA DECALS                   D       X          09522 Feb 89*"
"F56-CHILD SEATS                    H         X        03521 Nov 88*"
"F60-AIR SPOILERS FRONT             D  XX  XXXX X X    04021 Nov 88*"
"F62-ROOF OPTIONS                   D        X         02521 Nov 88*"
"F64-RADIATOR GRILLES(FINISH)       D X                10521 Nov 88*"
"F66-AIR SPOILERS -REAR             D XXX  XXXX X XX   04521 Nov 88*"
"F76-GLOVE BOXES                    F XX               03021 Nov 88*"
"F78-COIN BOXES                     F                  09521 Nov 88*"
"F84-WAIST RAILS                    D                  07521 Nov 88*"
"F91-WHEEL DECALS                   R                  06013 Jan 89*"
"F93-LABELS                         K   X      XXX     03521 Nov 88*"
"F94-E MARK LABEL HEADLAMPS         B          X X     03521 Nov 88*"
"F95-COACHLINES/DECALS              D X          X     08021 Nov 88*"
"F96-COACHLINES/DECALS              DXX                08521 Nov 88*"
"F97-COLOURED DECAL PACKAGES        D X                10021 Nov 88*"
"F98-COACHLINES/DECALS              DXXX  X  X         09021 Nov 88*"
"G04-SOUND INSULATION VARIOUS       F        X X       13021 Nov 88*"
"G23-CHOKE CONTROLS                 M  X XXXXX         01021 Nov 88*"
"G28-SUMP PROTECTION                K          X X     02021 Nov 88*"
"G70-ENGINE OIL COOLER              P X                01521 Nov 88*"
"H02-STOP/TAIL LIGHT                W      X  X X      06021 Nov 88*"
"H19-SIDE LIGHTS    FUNCTION        W                  05521 Nov 88*"
"H21-SPEED WARNING DEVICES          Y        XX        07521 Nov 88*"
"H37-I.C.E. PACKAGES                XXXX               00521 Nov 88*"
"H41-CRUISE CONTROL                 M          XXX     01521 Nov 88*"
"H45-LEGISLATION                    A                  03021 Nov 88*"
"H48-BODY EXTENSIONS                D  X   X      X    06021 Nov 88*"
"H49-ELECTRONIC OPTION PACKS        Z          X       04521 Nov 88*"
"H50-TRIM PACKAGES                  Z X   X  XX        04021 Nov 88*"
"H51-WIPER SYSTEMS                  D  XX XXXX    X    00521 Nov 88*"
"H52-ROVER 800 89MY HONDA SUPPLY    Z          X       09521 Nov 88*"
"H56-TRIP COMPUTERS                 Y          X X X   02521 Nov 88*"
"H59-LANGUAGE                       A      XX XXX      03521 Nov 88*"
"H61-INSTRUMENT PACK DISPLAY        Y                  01021 Nov 88*"
"H66-BODY PROTECTION COVERS         D          X       14021 Nov 88*"
"H67-BONNET PROTECTION              D        X         12021 Nov 88*"
"H71-HANDBRAKE GRIPS                S  X               030 6 Feb 89*"
"H80-LICENCE HOLDER                 F        X         01521 Nov 88*"
"H92-INFORMATION PLATE FIXINGS      K                  03021 Nov 88*"
"H96-FIRST AID EQUIPMENT            F        X         12021 Nov 88*"
"JCN-PAINT (RED/MAROON)             B X                05021 Nov 88*"
"JCO-PAINT (RED)                    B X   X            05521 Nov 88*"
"JFM-PAINT (YELLOW)                 B X   X            06021 Nov 88*"
"JFN-PAINT (YELLOW)                 B X                06521 Nov 88*"
"JHN-PAINT (GREEN)                  B X                07021 Nov 88*"
"JJN-PAINT (BLUE)                   B X   X            07521 Nov 88*"
"JJP-PAINT (BLUE)                   B X                08021 Nov 88*"
"JLM-PAINT (BLUE)                   B X                08521 Nov 88*"
"JMM-PAINT (SILVER/ALUMINIUM)       B X   X            09021 Nov 88*"
"JNM-PAINT (WHITE/CREAM/IVORY)      BXX X X  X         09521 Nov 88*"
"JPM-PAINT (BLACK)                  B X                10021 Nov 88*"
"JRS-PAINT (MULTICOLOUR)            B     X            10521 Nov 88*"
"JRW-PAINT (MULTICOLOUR)            BX                 11021 Nov 88*"
"M---TERRITORY CODES                Z        X         07521 Nov 88*"
"R00-SPECIAL CUSTOMERS              ZXX X X            02021 Nov 88*"
"R01-SPECIAL CUSTOMERS CONTINUED    Z X   X  X         02521 Nov 88*"
"R02-LIMITED EDITIONS               Z X        X       00521 Nov 88*"
"R03-LIMITED EDITIONS CONTINUED     ZXX X    X         01021 Nov 88*"
"R05-LIMITED EDITIONS - CONTINUED   Z X   X  X         01521 Nov 88*"
```

```
"R10-INDUSTRIAL SALES UNITS          Z             X     06021 Nov 88*"
"W18-REAR SEAT ARMREST TYPES         H                   07021 Nov 88*"
"W27-ENGINE SERIES                   L                   01021 Nov 88*"
"W33-PAINT CATEGORIES                B                   11521 Nov 88*"
"W57-FLEET                           ZXX   X  X          03021 Nov 88*"
"X05-NUMBER PLATES                   D      X            205          "
"X06-FUEL TYPE                       UXX      XXXXX      05021 Nov 88*"
"X07-ENGINE NUMBER                   LXXXX X  X          06021 Nov 88*"
"X19-PAINT TYPE                      B        X          04521 Nov 88*"
"X32-SALES SPECIFICATION             Z         X         06521 Nov 88*"
"X33-MARKET REQUIREMENTS             Z         X X       07021 Nov 88*"
"X34-CLIMATES                        PXXX  XXXXXX XX     02021 Nov 88*"
"X37-ENGINE TUNE BANDS               N        XXXX       02021 Nov 88*"
"X38-ENGINE COMPRESSION              LXX   X  X   X      03521 Nov 88*"
"X43-HEADLAMPS                       WXXXXXXXXXXXX X     01021 Nov 88*"
"X97-PACKS - MISC                    Z     X  X          05521 Nov 88*"
"Z09-MISCELLANEOUS                   ZXX X XX XX         08521 Nov 88*"
"Z11-CATALYST CONVERSION KIT         Z X X X  X          05011 May 89*"
"Z75-SPECIAL CONDITIONS              M                   02521 Nov 88*"
"Z86-SECURITY EQUIPMENT              YXX X X  XX         09021 Nov 88*"
"Z90-R8 D1 VEHICLES SRINGS REQUIREDZ      X              10021 Nov 88*"
"Z93-VIN TERRITORIES                 Z        XXX X      08021 Nov 88*"
"Z96-POLICE VEHICLE                  ZXXX  X  XX X       03521 Nov 88*"
"Z99-MISC OBJECT                     Z     X XXX X       09021 Nov 88*"

))



;;;;;*****************************************************************************************
; ;;;;*****************************************************************************************
;;;;;*****************************************************************************************



;;ASSUME  we have the  "MUST"    and      "MAYBES".
;;;; (defvar "model" nil)
 ;;(defvar  "must"      '("b15"  "b12"))
 ;;(defvar  "maybes"         '("b11" "b02" "b45"))


(defvar *models* '(("METRO" :value metro)
                   ("R8" :value r8)
                   ("MAESTRO" :value ms)
                   ("MINI" :value XH)))

;;(defvar "part-description-in-concern"  nil)
(defvar "MUSTS-AFTER" nil)
(defvar "MAYBES-AFTER" nil)
(defvar "the-ENGLISH-STATEMENTS-all-AFTER" nil)
(defvar   *RECORD-OF-SUBFEATURES-before* nil)
(defvar *result-of-RUNS*     0)


;(defvar  "negatives"
;'(  "B45B-----SPORTS STYLE SEAT              SPSEAT  "))


 ;;;"B12D-----BLACK FINISH(LARGE HEAD)      BIGBLK  "))


;;"B11C-----REAR QTR LIGHT HINGED         HINRQ   "


(defvar "data-we-concern"  nil)


(defvar "NEGATIVES-WHICH-WE-GAVE"  nil)
(defvar *BAFCs-PATHNAME* nil)

;;;;;(defvar "sequence" ' ((:body ()) (:family ()) (:capacity ()) (:performance ()) (:drive ()) (:transmi
sion ())))
```

```
;;;;;;(initialise-sequence)


;*****************************************************************************************


(defun STATICE-CLEAR-ALL-objects-FROM-KNOWLEDGE-BASE ()
  (setq *sequence* ' ((:body ()) (:family ()) (:capacity ()) (:performance ()) (:drive ()) (:transmision
()))
   (initialise-sequence)
   ;;;(setq *part-description-in-concern*   nil)
   (setq *negatives*  nil)
;;;;;;;;;;;;;;;;;;;;THE only  change   ;;;;;(setq *data-we-concern* nil)

    ;;;;;;;;(setq *MODEL* nil);;;;;;BECAUSE I moved the function: (menu-models) into the file foo.lisp in
 the function:
       ;;;;;;;;;  define-ENGINEERING-SPEC-------AUDITING-VALIDATION-command before the function: " work:
   (setq *MUSTS-AFTER* nil)
   (setq *MAYBES-AFTER*  nil)
   (setq *the-ENGLISH-STATEMENTS-all-AFTER* nil)
   (setq *NEGATIVES-WHICH-WE-GAVE* nil)
   (setq   *RECORD-OF-SUBFEATURES-before* nil)
   (setq  *result-of-RUNS*    nil)
;;;*****************************************

   (setq *common-ground-record* nil)
   (setq *implicit-trim-levels*  nil)
   (setq *array* nil)
   (setq *grouped*  nil)
)



(defun  STATICE-UPDATE-THE-*data-we-concern*-ACTUALLY-the-*feature-chart-entries*  (NEGATIVES) ;;ie
     ;;;;;;;;;;;;NEGATIVES   ====   (STATICE-the-valid-groups-now-ARE-TOP  PART-DESCRIPTION-we-CONCERN)
  (loop for   STATICE-ENTITY   in   *feature-chart-entries*
        do
    (cond ((member  (Feature-chart-entry-FEATURE-DESCRIPTION  STATICE-ENTITY)  NEGATIVES :test #'string-e
qual)

              (delete STATICE-ENTITY *feature-chart-entries*)))))



(defun STATICE-do-the-begining (&optional (stream *standard-output*)  PART-DESCRIPTION-we-CONCERN ) ;;; B
AFCs-PATHNAME)

  ;;;;;(setq *BAFCs-PATHNAME* nil)      ;;;;;;; WE donn't use it BUT if we use in the future, here is the
 position
                                        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;in which it must be INITIALIZED.
  (STATICE-CLEAR-ALL-objects-FROM-KNOWLEDGE-BASE)
  (setq *feature-chart-entries*  nil)
  ;;;;;;;(reset-feature-chart-db)   ;;;;BECAUSE now we don't CONCERN with *data-we-concern* now,   BUT..
*feature-chart-entries*
  ;;;(send stream :clear-history)

  ;*****************************************************************
(let ((help (STATICE-the-valid-groups-now-ARE-TOP PART-DESCRIPTION-we-CONCERN)))  ;;;; BAFCs-PATHNAME)))
  (STATICE-UPDATE-THE-*data-we-concern*-ACTUALLY-the-*feature-chart-entries*   help)
    (STATICE-reset-feature-chart-db)
    (STATICE-after-reset)


    (setq *implicit-trim-levels* (give-the-implicit-trim-levels-TOP))

    (setq *array* (create-the-array))

    (setq  *grouped* (group-together-GROUPS  (group-together-SUBGROUPS *array*)))

    (setq *territories* (do-it-new))

    (setq *final-territories* (supergroups-in))

    (setq *countries* (remove-duplicates (all-the-countries) :test #'string-equal))
```

```
(begin (give-the-priority-for-the-SEARCH-of-ALGORITHM))
))


;(defun do-the-begining (&optional (stream *standard-output*)  PART-DESCRIPTION-we-CONCERN  )
;
;
;  (CLEAR-ALL-objects-FROM-KNOWLEDGE-BASE)
;
;  ;;;(send stream :clear-history)
;  (let* ((result nil) (all-negatives (the-valid-groups-now-ARE-TOP  PART-DESCRIPTION-we-CONCERN))
;              ;; first to RUN the
;              ;;;;   function... (the-valid-groups-now-ARE-TOP ),to setq *data-we-concern*
;        (data-we-concern  *data-we-concern*))
;    (loop for negative  in  all-negatives
;         do
;      (loop for entry-line in data-we-concern
;           do
;       (cond ((string-equal  negative  (cadr (assoc ':FEATURE-DESCRIPTION entry-line)))
;               (push entry-line result)))))
;    (setq *data-we-concern*
;         (set-difference *data-we-concern* result :test #'equal))
;  ;**********************************************************************
;
;    (reset-feature-chart-db)
;
;    (after-reset)
;
;
;    (setq *implicit-trim-levels* (give-the-implicit-trim-levels-TOP))
;
;    (setq *array* (create-the-array))
;
;    (setq  *grouped* (group-together-GROUPS  (group-together-SUBGROUPS *array*)))
;
;    (setq *territories* (do-it-new))
;
;    (setq *final-territories* (supergroups-in))
;
;    (setq *countries* (remove-duplicates (all-the-countries) :test #'string-equal))
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;
;;     (cond ((values (menu-KATERINA))
;;            (let ((help (reverse
;;                         (loop for i in
;;                               (KATERINA-features-menu  *the-ENGLISH-STATEMENTS-all-AFTER* )
;;                               collect
;;                               (substring i 0 3)))))
;;              (begin
;;
;;(defun menu-KATERINA ()
;;  (dw:menu-choose '(("Yes" . T) ("No" . nil)) :prompt (zl:format nil "DO YOU WANT TO consult THE ROUTE
of the ALGORITHM ? .....the DEFAULT it is CALCULATED from THE SYSTEM..."))
;;)
;;
;;
;;
;;(defun KATERINA-features-menu (ls   &optional (stream *standard-output*))
;;  (let ((answers nil) (variable (loop for FEATURE in ls
;;                                      collect (cadr FEATURE))))
;;
;;
;;     (push
;;       (dw:accepting-values (stream :own-window t
;;                            :label "FEATURES WHICH ARE AVAILABLE FOR THE Part Description: ")
;;       (accept (clear (list 'subset variable))
;;               :prompt "ENTER IN THE REQUIRED sequence"  :stream  stream)
;;       )
;;       answers)
;;     (reverse (car answers)))
;;)
;;
;
```

```
;
;
;
;
;
;
;    (begin (give-the-priority-for-the-SEARCH-of-ALGORITHM))
;
;
;
;))


(defun    give-the-priority-for-the-SEARCH-of-ALGORITHM  ()
  (let ((result nil) (result-musts nil) (result-maybes nil) (help (numbering-the-feat-GROUPS)))
    (loop for feature-with-priority-number    in   help
          do
      (loop for must in *musts-after*
            do
        (cond ((string-equal must (car feature-with-priority-number))
                  (push feature-with-priority-number result-musts)))

      ))
    (loop for feature-with-priority-number    in   help
          do
      (loop for maybe in *maybes-after*
            do
        (cond ((string-equal maybe (car feature-with-priority-number))
                  (push feature-with-priority-number result-maybes)))

      ))

    (setq result (loop for feature-with-priority-number in
                           (append (sort result-musts #'> :key #'second)
                                   (sort result-maybes #'> :key #'second))
                       collect (car feature-with-priority-number)))
    ;;;;WE DON'T NEED to  put out the GROUP with length  0 BECAUSE it WILL not APPEAR at
    ;;;;;     THE  *subdescription-tables* ANYWAY.
    (reverse
      (combination-for-one (car result)  result))
))


;*****************************************************************************************
;Command: (collect-from-FEAT-GROUP-INDEX-REPORT--the-VALID    '("b15" "b12" "b02" "b45") 'metro)
;(("b45" "B45-SEAT TYPE -FRONT                H ") ("b02" "B02-BACKLIGHTS/REAR WINDOWS        C ")
; ("b12" "B12-EXTERIOR MIRRORS FINISH        C ") ("b15" "B15-EXTERIOR MIRRORS               CX"))
;Command:


(defun   the-begining ( PART-DESCRIPTION-we-CONCERN stream)
  (with-database (db *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*)
```

```
    (with-transaction ()
  (print '%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% stre
am)
  (print (format nil  "THE   MUST ADDITIONAL FEATURES KNOWN IN THE Knowledge Base FOR THE  PART DESCRIPT
ION ~A ARE:" (SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION  PART-DESCRIPTION-we-CONCERN)))
  (nested-formatting-2-MUSTS (loop for i in (the-begining-MUSTS)
                              collect (list i)) stream)



;  (loop for i in (the-begining-MUSTS)
;        do
;        (print i))
   (let ((takis-MAYBES (loop for j in (the-begining-MAYBES)
                 collect  j)))
     (cond (takis-MAYBES
            (print (format  nil "THE   MAYBES  ADDITIONAL FEATURES KNOWN IN THE  Knowledge Base    FOR
THE PART DESCRIPTION ~A  ARE:" (SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION  PART-DESCRIPTION-we-CONCERN)))
           (nested-formatting-2-MAYBES (loop for i in (the-begining-MAYBES)
                              collect (list i)) stream))
;           (loop for j in (the-begining-MAYBES)
;                  do
;             (print j)))
          (t (print (format nil "NOT   MAYBES  ADDITIONAL FEATURES ARE KNOWN IN THE   Knowledge Base  F
OR THE PART  DESCRIPTION: ~S."  (SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION  PART-DESCRIPTION-we-CONCERN))
)))))))



(defun the-begining-MUSTS ()
  (let ((result nil))
    (loop for i in *must*
          do
      (loop for j in  *the-GROUP-headers*
            do
        (cond ((string-equal (substring j 0 3) i)
                   (push (substring j 0 34) result)))))
result))



(defun the-begining-maybes ()
  (let ((result nil))
    (loop for i in *maybes*
          do
      (loop for j in  *the-GROUP-headers*
            do
        (cond ((string-equal (substring j 0 3) i)
                   (push (substring j 0 34) result)))))
result))




;;*************************************************************************************

(defun STATICE-the-valid-groups-now-ARE-TOP (PART-DESCRIPTION-we-CONCERN) ;;;;;;    BAFCs-PATHNAME)

  (let ((BAFCs-PATHNAME
          (The-DATA-BASE-pathname-of-the-BASE-and-ADDITIONAL-features-RESTRICTIONS-of-the-model-we-concer
n  *MODEL*)))

    (STATICE-the-valid-groups-now-ARE *model* PART-DESCRIPTION-we-CONCERN
           (The-DATA-BASE-pathname-of-the-BASE-and-ADDITIONAL-features-RESTRICTIONS-of-the-model-we-conc
ern *model*))))
                                        ;;;;;BAFCs-PATHNAME)))


;Command: (return-the-name-of-ALL-FEATURE-GROUPS (collect-all-feature-groups-from-the-BAFCs 'r8))
;("B04-THEME PACKAGES                         " "B05-BUMPER CONDITION                     "
; "B11-QUARTER LIGHTS - REAR                  " "B15-EXTERIOR MIRRORS                     "
```

```
; "B19-FLOOR COVERINGS-CAR INTERIORS          " "B23-GLAZING TYPE                          "
; "B25-FRONT HEADRESTRAINSTS                  " "B27-INSTRUMENT BINNACLES AND PACKS        "
; "B33-MUDFLAPS                               " "B34-RADIO -TYPES                          "
; "B36-RADIO -SPEAKERS                        " "B39-SEAT BELTS  -  REAR                   "
; "B40-SEAT FACE MATERIAL                     " "B44-SEATS-RECLINING                       "
; "B45-SEAT TYPE-FRONT                        " "B47-DOOR INSERTS                          "
; "B48-DOOR CASING STYLE                      " "B49-REAR ARMRESTS                         "
; "B51-SEAT MATERIALS(1989 M/Y ONWARD         " "B53-TONNEAU COVER                         "
; "B58-BODY SHELL VARIATIONS                  " "B61-REAR SEATS                            "
; "B69-SEAT HEATING                           " "B76-SEAT BELT SENSORS                     "
; "B90-BODY SIDE MOULDINGS                    " "C02-ANTI-ROLL BARS                        "
; "C08-BRAKE TYPES                            " "C09-BRAKE SYSTEMS                         "
; "C14-EXHAUST TAIL PIPE TRIM                 " "C15-FINAL DRIVE RATIOS GROUP 2            "
; "C30-STEERING TYPES                         " "C31-FRONT&REAR SUSPENSION/SPRINGS         "
; "C35-TYRE PLY-SIZE                          " "C38-TYRE MANUFACTURES                     "
; "C40-WHEEL TRIMS                            " "C41-WHEEL TYPE                            "
; "C42-RIM SIZE                               " "C44-ROAD WHEEL MANUFACTURERS              "
; "C65-STEERING COLUMN TYPES                  " "C71-GEARBOX SPEC SPEED/RATIO              "
; "C75-GEARBOX TYPE                           " "C88-AIR CONDITIONING                      "
; "C94-TYRE SPEED RATING                      " "D05-CIGAR LIGHTERS                        "
; "D10-BRITISH BATTERY MANUF STANDARD         " "D17-HEADLAMP CLEANERS                     "
; "D21-HORN TYPE                              " "D37-SPEEDOMETER MPH / KPH                 "
; "D47-REAR WINDOW WASH/WIPE FITMENT          " "D48-HEADLAMP DIP CONDITIONS               "
; "D78-COURTESY LAMPS                         " "D85-FACIA ILLUMINATION                    "
; "D89-HEADLAMP LEVELLING SYSTEM              " "D91-SIDE LIGHT CONDITIONS                 "
; "D92-WINDOW LIFTS                           " "E09-FUEL EVAPORATIVE LOSS CONTROL         "
; "E12-VALVE CONFIGURATION                    " "E41-EXHAUST CATALYST                      "
; "E99-EMISSION PACKAGES                      " "F02-DOOR HANDLES                          "
; "F04-DOOR LOCK SYSTEMS                      " "F25-STEERING WHEEL FINISH                 "
; "F26-STEERING WHEEL TYPE                    " "F36-STEERING WHEEL BADGES                 "
; "F47-SUN ROOFS                              " "F48-REAR HEADRESTS                        "
; "F60-AIR SPOILERS FRONT                     " "F66-AIR SPOILERS REAR                     "
; "G23-CHOKE CONTROLS                         " "H02-STOP/TAIL LIGHT                       "
; "H45-LEGISLATION                            " "H48-BODY EXTENSIONS                       "
; "H50-OPTION PACKS                           " "H51-WIPER SYSTEMS                         "
; "H59-LANGUAGE                               " "X06-FUEL                                  "
; "X34-CLIMATES                               " "X43-HEADLAMPS                             "
; "Z09-MISCELLANEOUS                          " "Z90-R8 D1 VEHICLES SRINGS REQUIRED        ")
;Command:

(defun return-the-name-of-ALL-FEATURE-GROUPS (ls)
  ;;;(collect-all-feature-groups-from-the-BAFCs  MODEL)
  (let ((result nil))
    (loop for i in ls
          do
      (push (return-the-name-of-each-FEATURE-GROUP i) result))
    (remove nil result)))


(defun return-the-name-of-each-FEATURE-GROUP  (string)
  (block FEATURE_GROUP_NAME
    (loop for i in *headings*
          do
      (cond ((and (string-equal (aref i  3) #\-)
                  (string-equal (substring i 0 3) string))
             (return-from FEATURE_GROUP_NAME i))))
    nil))


;Command: (collect-all-feature-groups-from-the-BAFCs 'r8)
;("Z90" "Z09" "X43" "X34" "X06" "H59" "H51" "H50" "H48" "H45" "H02" "G23" "F66" "F60" "F48" "F47" "F36" "
F26" "F25" "F04" "F02"
; "E99" "E41" "E12" "E09" "D92" "D91" "D89" "D85" "D78" "D48" "D47" "D37" "D21" "D17" "D10" "D05" "C94" "
C88" "C75" "C71" "C65"
; "C44" "C42" "C41" "C40" "C38" "C35" "C31" "C30" "C15" "C14" "C09" "C08" "C02" "B90" "B76" "B69" "B61" "
B58" "B53" "B51" "B49"
; "B48" "B47" "B45" "B44" "B40" "B39" "B36" "B34" "B33" "B27" "B25" "B23" "B19" "B15" "B11" "B05" "B04")
;Command:


(defun collect-all-feature-groups-from-the-BAFCs (model)
  (let ((pathname (The-DATA-BASE-pathname-of-the-BASE-and-ADDITIONAL-features-RESTRICTIONS-of-the-model-w
e-concern model))
        (temporary nil))
```

```
    (with-database (db pathname)
      (with-transaction ()
        (for-each ((FCE Feature-chart-entry))
          (push (substring (Feature-chart-entry-feature-description  FCE) 0 3) temporary))))
    (remove-duplicates temporary :test #'string-equal)))



(defun The-DATA-BASE-pathname-of-the-BASE-and-ADDITIONAL-features-RESTRICTIONS-of-the-model-we-concern (m
odel)
  (case model
    (R8      "stat-FS:>takis>R8-data-only-the-ADDITIONALS")
    (metro   "stat-FS:>takis>METRO-data-only-the-ADDITIONALS" ;;;;;"Another  PATHNAME"
)))



(defun menu-models (&optional (stream *standard-output*))
  (let ((stream (dw:get-program-pane 'pane-3)))
  (let ((answers nil))
    (push
      ;;(multiple-value-list
        (dw:accepting-values (stream :own-window t
                                     :near-mode '(:point 100 700)
                                     :label "THE AUSTIN ROVER'S  MODELS AVAILABLE. ")
          (values
            (accept '(alist-member :alist ,*models*) :prompt "ENTER THE MODEL YOU WANT TO SPECIFY THE par
t FOR"
                    :stream stream )

          )
        )
      answers)
    answers
  )
))



(defun STATICE-the-valid-groups-now-ARE  (MODEL   PART-DESCRIPTION-we-CONCERN   THE-PATHNAME-OF-THE-BAFCs
-OF-THE-MODEL)
      ;;collect-from-FEAT-GROUP-INDEX-REPORT--the-VALID
  (let ((stream (dw:get-program-pane 'pane-3)))
  (terpri stream)
  ;;(terpri)
  (with-database (db *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*)
    (with-transaction ()

  (the-begining PART-DESCRIPTION-we-CONCERN stream)
  (let ((result nil) (help (collect-from-FEAT-GROUP-INDEX-REPORT--the-VALID
                            (append *must* *maybes*)   MODEL)))

    (let ((musts-valid (cond
                        ((equal *must* '("NO-SPECIFICATION")) *must*)
                        (t
                         (intersection *must*
                                       (loop for i in help
                                             collect (car i))
                                       :test #'string-equal))))
          (maybes-valid (cond
                         ((equal *maybes* '("NO-SPECIFICATION")) *maybes*)
                         (t
                          (intersection *maybes*
                                       (loop for i in help
                                             collect (car i))
```

```
                                        :test #'string-equal))))
       )
      (let ((musts-valid-ENGLISH (cond
                                       ((and (equal *must* '("NO-SPECIFICATION")) (equal *maybes* '("NO-SPEC
IFICATION")))
                                         (list "THE-PART-DESCRIPTION-HAS-BEEN-SPECIFIED-FOR-ALL-CARS-OF-THI
S-MODEL"))
                                       (t (cond ((and (equal *must* nil) (equal *maybes* nil))
                                                 (list "NO-INFORMATION-IS-KNOWN-FROM-THE-SYSTEM-FOR-THIS-PAR
T-DESCRIPTION"))
                                                (t
                                                 (let ((res nil))
                                                   (loop for i  in musts-valid
                                                         do
                                                     (push (return-the-ENGLISH i help) res)) res))))))

            (maybes-valid-ENGLISH (cond
                                       ((and (equal *must* '("NO-SPECIFICATION")) (equal *maybes* '("NO-SPE
CIFICATION")))
                                        (list "THE-PART-DESCRIPTION-HAS-BEEN-SPECIFIED-FOR-ALL-CARS-OF-THE-
MODEL"))
                                       (t (cond ((and (equal *must* nil) (equal *maybes* nil))
                                                 (list "NO-INFORMATION-IS-KNOWN-FROM-THE-SYSTEM-FOR-THIS-PAR
T-DESCRIPTION"))
                                                (t
                                                 (let ((res nil))
                                                   (loop for i  in maybes-valid
                                                         do
                                                     (push (return-the-ENGLISH i help) res)) res))))))
       )
        (print '%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% stream) (terpri stream)
        (print
          (format nil  "SEARCHING THE    Additional Feature Group Index Report   DATA, FOR ~A  .......
. . . . " MODEL )  stream)
        (print (format nil "...........................................................................
............................................") stream)
        ;;;(terpri stream)(terpri stream)
        (let ((mus (loop for i in musts-valid-ENGLISH
                         collect i))
              (may (loop for j in maybes-valid-ENGLISH
                         collect j)))

          (cond ((string-equal (car musts-valid-ENGLISH)
                               "THE-PART-DESCRIPTION-HAS-BEEN-SPECIFIED-FOR-ALL-CARS-OF-THE-MODEL")

                         (print
                          (format nil  "THE PART DESCRIPTION HAS BEEN SPECIFIED FOR ALL CARS OF THE  ~A
MODEL IN THE PAST !!!" model) stream))
                (t (cond ((string-equal (car musts-valid-ENGLISH)
                                         "NO-INFORMATION-IS-KNOWN-FROM-THE-SYSTEM-FOR-THIS-PART-DESCRIPTIO
N")
                           (print
                           (format nil  "THERE IS NOT KNOWN INFORMATION FOR  PART DESCRIPTION ~A FOR THE
~A MODEL"
                                         (SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION  PART-DESCRIPTION-we-CONCERN
) model) stream)
                           )
                          (t
          (cond ((equal (length mus) (length *MUST*))
                 (print (format nil "...ALL THE      MUST     ADDITIONAL  FEATURES  KNOWN IN Knowlede Base
 for ~A ,  ARE  OFFERED  FOR THE MODEL ~A ie." (SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION  PART-DESCRIPT
ION-we-CONCERN)          MODEL) stream)
                                                          ;
;                        (loop for i in musts-valid-ENGLISH
;                              do
;                          (print i stream))
               (nested-formatting-2-MUSTS (loop for i in musts-valid-ENGLISH
                                                collect (list i)) stream)
              )
              (t (print (format nil "...FROM THE  MUST  ADDITIONAL FEATURES KNOWN IN THE Knowlede Base
for  ~A   ,  ONLY THE ... " (SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION  PART-DESCRIPTION-we-CONCERN)) str
eam)
;
;                (loop for i in musts-valid-ENGLISH
;                      do
;                (print i stream))
```

```
                (nested-formatting-2-MUSTS (loop for i in musts-valid-ENGLISH
                                                  collect (list i)) stream)
                (print (format nil "... ARE OFFERED FOR  ~A" MODEL) stream))
          )))))
          (cond ((and (equal (length may) (length *MAYBES*))
                      (not (equal *MAYBES* nil)))
               (print (format nil "and ALL THE  MAYBES ADDITIONAL  FEATURES  KNOWN IN Knowlede Base  f
or ~A  and  OFFERED  FOR  ~A ie."(SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION  PART-DESCRIPTION-we-CONCERN)
 MODEL) stream)
;              (loop for i in maybes-valid-ENGLISH
;                    do
;                (print i stream))
               (nested-formatting-2-MAYBES (loop for i in maybes-valid-ENGLISH
                                                  collect (list i)) stream)
              )
              (t (cond (maybes-valid
                        (print (format nil "and FROM THE   MAYBES  ADDITIONAL FEATURES KNOWN IN THE Kno
wlede Base for ~A ,  ONLY THE ... " (SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION  PART-DESCRIPTION-we-CONCE
RN)) stream)
;                        (loop for i in maybes-valid-ENGLISH
;                              do
;                          (print i stream))
                        (nested-formatting-2-MAYBES (loop for i in maybes-valid-ENGLISH
                                                           collect (list i)) stream)
                        (prinl (format nil "... ARE OFFERED FOR  ~A" MODEL) stream))
                       (t (print (format nil "OF COURSE, NO   MAYBES   ADDITIONAL FEATURES EXIST FOR ~A
 ." (SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION  PART-DESCRIPTION-we-CONCERN)) stream))))
        )

(print '$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ stream
)
       ;(setq *MUSTS-AFTER* musts-valid)
(print-the-information-from-the-reasoning-through-the-tape *ROVERs-APN-VPG-CATALOGUE-only-SEA
TS-and-MIRRORS-pathname*
                                              MODEL              PART-DESCRIPTION-we-CONCERN stream
)
(cond ((and (equal *must* nil) (equal *maybes* nil))   ;;;;;THERE IS NOT KNOWN INFORMATION FOR THE Part D
escription
     (print (format nil "THERE IS NOT KNOWN INFORMATION FOR THE Part Description ~A  FOR THE MODEL ~A "
(SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION  PART-DESCRIPTION-we-CONCERN) MODEL) stream)
     (terpri stream)
     (let ((feature-groups-checked-from-Add-FEAt-GROUP-INdex-report
             (print-the-information-known-for-the-other-models
       ;;;;; FROM THE FILE : "check-for-the-other-models-if-you-dont-know-the-one-in-concern.lisp >takis
>statice"
       *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname* PART-DESCRIPTION-we-CONCERN str
eam)))
     (terpri stream)
      (print (format nil "NOW THE SYSTEM SEARCHES THE BASE and ADDITIONAL FEATURES OF THE MODEL FOR ALL
THE AVAILABLE FEATURE GROUPS") stream)
                    (terpri stream)
      (let* ((all-feature-group-codes (collect-all-feature-groups-from-the-BAFCs MODEL))
             (FEATURE-GROUP-NAMES (set-difference        ;;;;(set-difference
                                    (return-the-name-of-ALL-FEATURE-GROUPS all-feature-group-codes)
                                    feature-groups-checked-from-Add-FEAt-GROUP-INdex-report :test #'stri
ng-equal)))
                  (let* (
                  (*MUSTS-AFTER*faulse (QUERY-for-the-MUSTS-from-ENGINEER-MENU FEATURE-GROUP
-NAMES))
                  (*MAYBES-AFTER*faulse (QUERY-for-the-MAYBES-from-ENGINEER-MENU
                                          (set-difference FEATURE-GROUP-NAMES *MUSTS-AFTER*fauls
e :test #'string-equal))))

              (setq *MUSTS-AFTER* (loop for i in *MUSTS-AFTER*faulse
                                        collect
                                          (substring i 0 3)))

             (setq *MAYBES-AFTER*  (loop for i in *MAYBES-AFTER*faulse
                                        collect
                                          (substring i 0 3)))
              (let ((help-new-other nil))
                   (loop for i in (collect-from-FEAT-GROUP-INDEX-REPORT--the-VALID
                                     (append musts-valid maybes-valid) MODEL)
                         do
                      (push (second i)  help-new-other))
         (let ((must-duplicated (which-feature-groups-belong-to-the-same-record-TOP *MUSTS-AFTER*))
```

```
                (maybes-duplicated (which-feature-groups-belong-to-the-same-record-TOP *MAYBES-AFTER*))
)
            (cond ((or (equal must-duplicated nil)
                       (not (some #'(lambda (list) (> (length list) 1)) must-duplicated))))  ;;;; NO dup
licate LOGICAL features
                                          ;;;;in the Specification
                  (t (let* ((the-features-MUSTS-no-duplicate
                   (query-the-Engineer-to-avoid-to-specify-the-part-for-similar-FEATURE-GROUPS must-dupl
icated help-new-other))
                            (valid-musts-from-ENGINEER (car the-features-MUSTS-no-duplicate))
                            (no-valid-musts-from-ENGINEER (second the-features-MUSTS-no-duplicate)))
                       (setq *MUSTS-AFTER*
                             (remove-duplicates (append
                                                 valid-musts-from-ENGINEER
                                                 (set-difference *MUSTS-AFTER* no-valid-musts-from-ENGI
NEER :test #'string-equal))
                                                  :test #'string-equal)))))
            (cond ((or (equal maybes-duplicated nil)
                       (not (some #'(lambda (list) (> (length list) 1)) maybes-duplicated))))  ;;;; NO d
uplicate LOGICAL features
                                          ;;;;in the Specification
                  (t (let* ((the-features-MAYBES-no-duplicate
                   (query-the-Engineer-to-avoid-to-specify-the-part-for-similar-FEATURE-GROUPS maybes-du
plicated help-new-other))
                            (valid-maybes-from-ENGINEER (car the-features-MAYBES-no-duplicate))
                            (no-valid-maybes-from-ENGINEER (second the-features-MAYBES-no-duplicate)))
                       (setq *MAYBES-AFTER*
                             (remove-duplicates (append
                                                 valid-maybes-from-ENGINEER
                                                (set-difference *MAYBES-AFTER* no-valid-maybes-from-ENG
INEER
                                                                :test #'string-equal))
                                                :test #'string-equal)))))))

                )
          )   ;;;;;;; let without the CHECKED groups
)    ;;;;;;;;let with the CHECKED groups
  )      ;;;;;;; NOT KNOWN  INFORMATION
      (t
  (cond ((menu-QUERY-for-AGGREMENT);;;;;;;; NOW, WE HAVE TO QUERY the  *MUSTS-AFTER* as well.
 .ie
                                  ;;;;;;;; IF the ENGINEER agrees.
          (let ((help-new-other nil))
                        (loop for i in (collect-from-FEAT-GROUP-INDEX-REPORT--the-VALID
                                        (append musts-valid maybes-valid) MODEL)
                              do
                            (push (second i)  help-new-other))
           (let ((must-duplicated (which-feature-groups-belong-to-the-same-record-TOP musts-valid))
                 (maybes-duplicated (which-feature-groups-belong-to-the-same-record-TOP maybes-valid)))
             (cond ((or (equal must-duplicated nil)
                        (not (some #'(lambda (list) (> (length list) 1)) must-duplicated))))  ;;;; NO dup
licate LOGICAL features
                                          ;;;;in the Specification
                  (t (let* ((the-features-MUSTS-no-duplicate
                   (query-the-Engineer-to-avoid-to-specify-the-part-for-similar-FEATURE-GROUPS must-dupl
icated help-new-other))
                            (valid-musts-from-ENGINEER (car the-features-MUSTS-no-duplicate))
                            (no-valid-musts-from-ENGINEER (second the-features-MUSTS-no-duplicate)))
                       (setq musts-valid
                             (remove-duplicates (append
                                                 valid-musts-from-ENGINEER
                                                 (set-difference musts-valid no-valid-musts-from-ENGINE
ER :test #'string-equal))
                                                  :test #'string-equal)))))
             (cond ((or (equal maybes-duplicated nil)
                        (not (some #'(lambda (list) (> (length list) 1)) maybes-duplicated))))  ;;;; NO d
uplicate LOGICAL features
                                          ;;;;in the Specification
                  (t (let* ((the-features-MAYBES-no-duplicate
                   (query-the-Engineer-to-avoid-to-specify-the-part-for-similar-FEATURE-GROUPS maybes-du
plicated help-new-other))
                            (valid-maybes-from-ENGINEER (car the-features-MAYBES-no-duplicate))
                            (no-valid-maybes-from-ENGINEER (second the-features-MAYBES-no-duplicate)))
                       (setq maybes-valid
                             (remove-duplicates (append
                                                 valid-maybes-from-ENGINEER
```

```
                                                    (set-difference maybes-valid no-valid-maybes-from-ENGIN
EER
                                                                    :test #'string-equal))
                                                  :test #'string-equal)))))))
          (setq *MUSTS-AFTER*  musts-valid)
          (cond (maybes-valid
                 (setq *MAYBES-AFTER* (loop for i in
                                       (maybes-add-features-menu maybes-valid-ENGLISH);;;;;;;;;BE
FORE this we have to
                                          ;;;;; QUERY the Engineer if agrees with the MUSTS and M
AYBES

                                    collect
                                    (substring i 0 3)))
                 )))

        (t (cond ((and (equal *must* '("NO-SPECIFICATION")) (equal *maybes* '("NO-SPECIFICATION")))
                  (print (format nil "NOW THE SYSTEM SEARCHES THE BASE and ADDITIONAL FEATURES OF THE M
ODEL FOR ALL THE AVAILABLE FEATURE GROUPS") stream)
                  (terpri stream)
                  (let* ((all-feature-group-codes (collect-all-feature-groups-from-the-BAFCs MODEL))
                         (FEATURE-GROUP-NAMES (return-the-name-of-ALL-FEATURE-GROUPS all-feature-group-
codes)))
                    (let* (
                    (*MUSTS-AFTER*faulse (QUERY-for-the-MUSTS-from-ENGINEER-MENU FEATURE-GROUP
-NAMES))
                    (*MAYBES-AFTER*faulse (QUERY-for-the-MAYBES-from-ENGINEER-MENU
                                           (set-difference FEATURE-GROUP-NAMES *MUSTS-AFTER*fauls
e :test #'string-equal))))

                 (setq *MUSTS-AFTER* (loop for i in *MUSTS-AFTER*faulse
                                      collect
                                          (substring i 0 3)))

                (setq *MAYBES-AFTER*  (loop for i in *MAYBES-AFTER*faulse
                                      collect
                                          (substring i 0 3)))
                 (let ((help-new-other nil))
                        (loop for i in (collect-from-FEAT-GROUP-INDEX-REPORT--the-VALID
                                        (append musts-valid maybes-valid) MODEL)
                              do
                          (push (second i)  help-new-other))
          (let ((must-duplicated (which-feature-groups-belong-to-the-same-record-TOP *MUSTS-AFTER*))
                (maybes-duplicated (which-feature-groups-belong-to-the-same-record-TOP *MAYBES-AFTER*))
)
            (cond ((or (equal must-duplicated nil)
                       (not (some #'(lambda (list) (> (length list) 1)) must-duplicated))))  ;;;; NO dup
licate LOGICAL features
                                      ;;;;in the Specification
                  (t (let* ((the-features-MUSTS-no-duplicate
                   (query-the-Engineer-to-avoid-to-specify-the-part-for-similar-FEATURE-GROUPS must-dupl
icated help-new-other))
                          (valid-musts-from-ENGINEER (car the-features-MUSTS-no-duplicate))
                          (no-valid-musts-from-ENGINEER (second the-features-MUSTS-no-duplicate)))
                    (setq *MUSTS-AFTER*
                          (remove-duplicates (append
                                              valid-musts-from-ENGINEER
                                              (set-difference *MUSTS-AFTER* no-valid-musts-from-ENGI
NEER :test #'string-equal))
                                               :test #'string-equal)))))
            (cond ((or (equal maybes-duplicated nil)
                       (not (some #'(lambda (list) (> (length list) 1)) maybes-duplicated))))  ;;;; NO d
uplicate LOGICAL features
                                      ;;;;in the Specification
                  (t (let* ((the-features-MAYBES-no-duplicate
                   (query-the-Engineer-to-avoid-to-specify-the-part-for-similar-FEATURE-GROUPS maybes-du
plicated help-new-other))
                          (valid-maybes-from-ENGINEER (car the-features-MAYBES-no-duplicate))
                          (no-valid-maybes-from-ENGINEER (second the-features-MAYBES-no-duplicate)))
                    (setq *MAYBES-AFTER*
                          (remove-duplicates (append
                                              valid-maybes-from-ENGINEER
                                              (set-difference *MAYBES-AFTER* no-valid-maybes-from-ENG
INEER
                                                              :test #'string-equal))
                                              :test #'string-equal)))))))
```

```
                )
        ))  ;;;let   and cond
                (t (cond (
                 (menu-QUERY-IF-HE-WANTS-TO-GIVE-FEATURES-GROUPS-from-SCRATCH)
                    (print (format nil "NOW THE SYSTEM SEARCHES THE BASE and ADDITIONAL FEATURES OF THE
MODEL FOR ALL THE AVAILABLE FEATURE GROUPS") stream)
                  (terpri stream)
                   (let* ((all-feature-group-codes (collect-all-feature-groups-from-the-BAFCs MODEL))
                          (FEATURE-GROUP-NAMES (return-the-name-of-ALL-FEATURE-GROUPS all-feature-group-
codes)))
                     (let* (
                     (*MUSTS-AFTER*faulse (QUERY-for-the-MUSTS-from-ENGINEER-MENU FEATURE-GROUP
-NAMES))
                     (*MAYBES-AFTER*faulse (QUERY-for-the-MAYBES-from-ENGINEER-MENU
                                           (set-difference FEATURE-GROUP-NAMES *MUSTS-AFTER*fauls
e :test #'string-equal))))

                 (setq *MUSTS-AFTER* (loop for i in *MUSTS-AFTER*faulse
                                        collect
                                            (substring i 0 3)))

                (setq *MAYBES-AFTER*  (loop for i in *MAYBES-AFTER*faulse
                                        collect
                                            (substring i 0 3)))
                 (let ((help-new-other nil))
                          (loop for i in (collect-from-FEAT-GROUP-INDEX-REPORT--the-VALID
                                            (append musts-valid maybes-valid) MODEL)
                                do
                            (push (second i)  help-new-other))
          (let ((must-duplicated (which-feature-groups-belong-to-the-same-record-TOP *MUSTS-AFTER*))
                (maybes-duplicated (which-feature-groups-belong-to-the-same-record-TOP *MAYBES-AFTER*))
)
            (cond ((or (equal must-duplicated nil)
                       (not (some #'(lambda (list) (> (length list) 1)) must-duplicated))))  ;;;; NO dup
licate LOGICAL features
                                        ;;;;in the Specification
                    (t (let* ((the-features-MUSTS-no-duplicate
                     (query-the-Engineer-to-avoid-to-specify-the-part-for-similar-FEATURE-GROUPS must-dupl
icated help-new-other))
                            (valid-musts-from-ENGINEER (car the-features-MUSTS-no-duplicate))
                            (no-valid-musts-from-ENGINEER (second the-features-MUSTS-no-duplicate)))
                        (setq *MUSTS-AFTER*
                             (remove-duplicates (append
                                                 valid-musts-from-ENGINEER
                                                 (set-difference *MUSTS-AFTER* no-valid-musts-from-ENGI
NEER :test #'string-equal))
                                                  :test #'string-equal)))))
            (cond ((or (equal maybes-duplicated nil)
                       (not (some #'(lambda (list) (> (length list) 1)) maybes-duplicated))))  ;;;; NO d
uplicate LOGICAL features
                                        ;;;;in the Specification
                    (t (let* ((the-features-MAYBES-no-duplicate
                     (query-the-Engineer-to-avoid-to-specify-the-part-for-similar-FEATURE-GROUPS maybes-du
plicated help-new-other))
                            (valid-maybes-from-ENGINEER (car the-features-MAYBES-no-duplicate))
                            (no-valid-maybes-from-ENGINEER (second the-features-MAYBES-no-duplicate)))
                        (setq *MAYBES-AFTER*
                             (remove-duplicates (append
                                                  valid-maybes-from-ENGINEER
                                                 (set-difference *MAYBES-AFTER* no-valid-maybes-from-ENG
INEER
                                                                 :test #'string-equal))
                                                :test #'string-equal)))))))

            )
        )
                )  ;;;; ) ,,,,,(menu-QUERY-IF-HE-WANTS-TO-GIVE-FEATURES-GROUPS-from-SCRATCH)
      (t
       (let ((help-new nil))
                          (loop for i in (collect-from-FEAT-GROUP-INDEX-REPORT--the-VALID
                                            (append musts-valid maybes-valid) MODEL)
                                do
                            (push (second i)  help-new))
              (let* (
                     (*MUSTS-AFTER*faulse (QUERY-for-the-MUSTS-from-ENGINEER-MENU help-new))
```

```
                    (*MAYBES-AFTER*faulse (QUERY-for-the-MAYBES-from-ENGINEER-MENU
                                          (set-difference help-new *MUSTS-AFTER*faulse :test #
'string-equal))))

                (setq *MUSTS-AFTER* (loop for i in *MUSTS-AFTER*faulse
                                          collect
                                              (substring i 0 3)))

               (setq *MAYBES-AFTER*  (loop for i in *MAYBES-AFTER*faulse
                                          collect
                                              (substring i 0 3)))
                (let ((help-new-other nil))
                        (loop for i in (collect-from-FEAT-GROUP-INDEX-REPORT--the-VALID
                                          (append musts-valid maybes-valid) MODEL)
                                do
                              (push (second i)  help-new-other))
          (let ((must-duplicated (which-feature-groups-belong-to-the-same-record-TOP *MUSTS-AFTER*))
                (maybes-duplicated (which-feature-groups-belong-to-the-same-record-TOP *MAYBES-AFTER*))
)
            (cond ((or (equal must-duplicated nil)
                       (not (some #'(lambda (list) (> (length list) 1)) must-duplicated))))  ;;;; NO dup
licate LOGICAL features
                                        ;;;;in the Specification
                  (t (let* ((the-features-MUSTS-no-duplicate
                   (query-the-Engineer-to-avoid-to-specify-the-part-for-similar-FEATURE-GROUPS must-dupl
icated help-new-other))
                              (valid-musts-from-ENGINEER (car the-features-MUSTS-no-duplicate))
                              (no-valid-musts-from-ENGINEER (second the-features-MUSTS-no-duplicate)))
                       (setq *MUSTS-AFTER*
                             (remove-duplicates (append
                                                 valid-musts-from-ENGINEER
                                                 (set-difference *MUSTS-AFTER* no-valid-musts-from-ENGI
NEER :test #'string-equal))
                                                  :test #'string-equal)))))
            (cond ((or (equal maybes-duplicated nil)
                       (not (some #'(lambda (list) (> (length list) 1)) maybes-duplicated))))  ;;;; NO d
uplicate LOGICAL features
                                        ;;;;in the Specification
                  (t (let* ((the-features-MAYBES-no-duplicate
                   (query-the-Engineer-to-avoid-to-specify-the-part-for-similar-FEATURE-GROUPS maybes-du
plicated help-new-other))
                              (valid-maybes-from-ENGINEER (car the-features-MAYBES-no-duplicate))
                              (no-valid-maybes-from-ENGINEER (second the-features-MAYBES-no-duplicate)))
                       (setq *MAYBES-AFTER*
                             (remove-duplicates (append
                                                  valid-maybes-from-ENGINEER
                                                 (set-difference *MAYBES-AFTER* no-valid-maybes-from-ENG
INEER
                                                                  :test #'string-equal))
                                                 :test #'string-equal)))))))

                )))))))

))  ;;;cond;;;;;;;;    (manu-QUERY-for-AGGREMENT)
))
     )  ;;;;;;;   LET

;        (terpri stream) (terpri stream) ;;;(terpri)
;
;        (setq *data-we-concern* (collect-the-groups-we-concern-TOP
;                                      (append *MUSTS-AFTER* *MAYBES-AFTER*)))


(setq  *feature-chart-entries*
 (create-the-STATICE-*feature-chart-entries*  (append *MUSTS-AFTER* *MAYBES-AFTER*  )   THE-PATHNAME-OF-F
HE-BAFCs-OF-THE-MODEL ))


;         (setq  *RECORD-OF-SUBFEATURES-before*
;                (remove-duplicates
;                  (loop for entry in *data-we-concern*
;                        collect
;                        (cadr (assoc ':FEATURE-DESCRIPTION   entry)))
;                  :test #'string-equal
;                )
;         )
```

```
          (setq *RECORD-OF-SUBFEATURES-before*
                (remove-duplicates
                  (loop for STATICE-ENTITY in *feature-chart-entries*
                        collect
                        (Feature-chart-entry-FEATURE-DESCRIPTION STATICE-ENTITY))
                :test #'string-equal))

        (setq *the-ENGLISH-STATEMENTS-all-AFTER*
              (collect-from-FEAT-GROUP-INDEX-REPORT--the-VALID
                  (append *MUSTS-AFTER* *MAYBES-AFTER*)  *model*))
      (print '%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
stream)
        (cond (*MAYBES-AFTER*
                (print (format nil "THE NEGATIVE ADDITIONAL FEATURES FOR ~A THAT THE SYSTEM ALREADY KNOWS
 FROM EXPERIENCE IN THE PAST, ACCORDING TO  THE valid  ADD-FEAT-GROUPS  WHICH ARE available for   ~A ,  A
ND YOU propose, ARE:" (SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION  PART-DESCRIPTION-we-CONCERN) MODEL) str
eam)
              )
              (t (print (format nil "THE NEGATIVE ADDITIONAL FEATURES FOR ~A THAT THE SYSTEM ALREADY KNOW
S FROM EXPERIENCE IN THE PAST, ACCORDING TO  THE valid  ADD-FEAT-GROUPS  WHICH ARE available for  ~A  ARE
:"  (SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION  PART-DESCRIPTION-we-CONCERN)  MODEL) stream)
              )
        )
        (let ((the-negatives-which-the-system-KNOWS (the-negatives-the-system-KNOWS
                                                     PART-DESCRIPTION-we-CONCERN  )))
          (cond (the-negatives-which-the-system-KNOWS
                 (nested-formatting-2-NEGATIVES (loop for i in the-negatives-which-the-system-KNOWS
                                             collect (list i)) stream))
                (t (print (format nil"..............................................................
..................................") stream)
                (print (format nil "NO NEGATIVE ADDITIONAL FEATURES KNOWN FOR THE PART DESCRIPTI
ON") stream)
                (terpri stream) (terpri stream)))

        ;;(print the-negatives-which-the-system-KNOWS stream) ;;;;;;;  HERE WE MUST BE CAREFUL HOW
TO EXPRESS IN ENGLISH THE
;          NEGATIVE_FEATURES AS THEY ARE  STORED IN STATICE AS vectors.
;          WE HAVE TO ACCESS VECTORS of numbers, TRANSLATE THE numbers TO ENGLISH LANGUAGE and
 Backwards WHEN WE INSERT THE
;          NEGATIVES to TRANSLATE THE ENGLISH to a VECTOR OF NUMBERS !!!!!!!!!!!
          (print (format nil "PLEASE ENTER SOME MORE negatives INTO THE MEMORY OF  ~A FROM THE EXISTING
 FEAT-GROUPS" (SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION  PART-DESCRIPTION-we-CONCERN) stream))
          (terpri stream) (terpri stream)

          (let ((takis-negatives
                      (remove 'nil
                              (clear
                                 (NEGATIVES-features-menu (STATICE-prompt-for-NEGATIVES)
                                              the-negatives-which-the-system-KNOWS))
                      )))

            (setq result (append
                      the-negatives-which-the-system-KNOWS

                      takis-negatives
                      )
            )
            (setq *NEGATIVES-WHICH-WE-GAVE* takis-negatives)
            ;(print *NEGATIVES-WHICH-WE-GAVE*)
           (update-the-NEGATIVES-of-DESCRIPTION PART-DESCRIPTION-we-CONCERN  )
        ;; (print (part-description-NEGATIVE-additional-features  PART-DESCRIPTION-we-CONCERN))

        )
       )
        (print '%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% stream)
     )
) result)))))


(defun QUERY-for-the-MAYBES-from-ENGINEER-MENU  (feature-descriptions   &optional (stream *st
andard-output*))
  (let ((answers nil))
    (push
      (dw:accepting-values (stream :own-window t
```

```
                                        :near-mode '(:point 100 700)
                                        :label " MAYBES  FEATURES YOU WILL USE FOR THE SPECIFICATIO
N THE OF PART DESCRIPTION  ? ")
        (accept (clear (append (list  'subset)  feature-descriptions))
          :prompt "Enter the MAYBES  FEATURE GROUPS IN YOUR OPINION : "  :stream  stream))   ;
       answers)
      (car answers)))



(defun QUERY-for-the-MUSTS-from-ENGINEER-MENU  (feature-descriptions   &optional (stream *sta
ndard-output*))
  (let ((answers nil))
    (push
      (dw:accepting-values (stream :own-window t
                                        :near-mode '(:point 100 700)
                                        :label " MUST  FEATURES YOU THINK REPRESENT THE CORRECT SPE
CIFICATION FOR THE   PART DESCRIPTION  ? ")
        (accept (clear (append (list  'subset)  feature-descriptions))
          :prompt "Enter the MUST  FEATURE GROUPS IN YOUR OPINION :"  :stream  stream))     ;
       answers)
      (car answers)))



(defun menu-QUERY-for-AGGREMENT ()  ;;;;;; YOU MUST NOT put different character format to this menu ie. B
OLD , italic etc.
  ;;;;;;;; only  nil.nil.nil
  (values
    (dw:menu-choose '(("Yes" . T) ("No" . nil)) :prompt (zl:format nil "DO YOU AGREE WITH THE ANALYSIS OF
 THE FEATURES DONE FROM THE SYSTEM  ?") :near-mode '(:point 100 700))))


(defun menu-QUERY-IF-HE-WANTS-TO-GIVE-FEATURES-GROUPS-from-SCRATCH ()  ;;;;;; YOU MUST NOT put different
character format to this menu ie. BOLD , italic etc.
  ;;;;;;;; only  nil.nil.nil
  (values
    (dw:menu-choose '(("Yes" . T) ("No" . nil)) :prompt (zl:format nil "DO YOU WANT TO ENTER YOUR OWN FEA
TURE GROUPS FROM SCRATCH  ? ") :near-mode '(:point 100 700))))


(defun query-the-Engineer-to-avoid-to-specify-the-part-for-similar-FEATURE-GROUPS (ls help-ne
w-other)
      ;;;;;;;;;    (which-feature-groups-belong-to-the-same-record-TOP (append '("B40" "B51")
'("f**" )))
     ;;;;;;;;;;;;    ls      ======    (("B40" "B51") ("f**"))
  (let ((help nil) (result nil) (result-final nil) (valids nil) (no-valids nil))
    (loop for i in ls
          do
      (cond ((> (length i) 1) (push i help))))
    (loop for i in help
          do
           (push
             (another-translation-to-English-the-feature-group  i help-new-other) result))
    (loop for i in result
          do
      (push (QUERY-for-the-DUPLICATED i) result-final))
   ;;;;;;result-final))  ;;;;((NIL NIL) ("B51-SEAT FACE MATERIAL              " ("B40-SEAT FAC
E MATERIAL                  ")))
    (loop for valid-and-no-valids in result-final
          do
      (push (car valid-and-no-valids) valids)
      (push (cadr valid-and-no-valids) no-valids))
    (list (loop for i in valids
                collect
                  (substring i 0 3))
          (loop for i in (only-atoms no-valids)
                collect
                  (substring i 0 3)))))


(defun QUERY-for-the-DUPLICATED ( list &optional (stream *standard-output*))
 ;(let ((stream (dw:get-program-pane 'pane-3)))
```

```
  (let ((answers nil))
    (push
      ;;(multiple-value-list
        (dw:accepting-values (stream :own-window t
                                     :near-mode '(:point 100 700)
                                     :label
                                     "THE BELOW FEATURE GROUPS REPRESENT THE SAME LOGICAL SPECIFICATION F
OR A PART WITHIN ROVER.")
          (values
            (accept '(alist-member :alist ,list) :prompt "PLEASE CHOOSE THE ONE YOU ARE GOING TO SPE
CIFY WITH."
                    :stream stream )

          )
        )
      answers)
    (list (car answers) (set-difference list answers :test #'string-equal))
  )
)




(defun another-translation-to-English-the-feature-group (i help-new-other)
;("B40-SEAT FACE MATERIAL              " "B51-SEAT FACE MATERIAL              " "B69-SEAT HEATING
                    "
; "B44-SEATS -RECLINING                " "B53-SEAT SIDE ROLLS                 " "B45-SEAT TYPE -F
RONT                ")
  (let ((result nil))
    (loop for feature-group-code in i
          do
      (loop for feature-group-in-English in help-new-other
            do
       (cond ((string-equal feature-group-code (substring feature-group-in-English 0 3))
                  (push feature-group-in-English result)))))
result))

(defun  number-the-common-feature-groups ()
  (let ((result nil) (record-number 0))
    (loop for i in *FEATURE-GROUPS-WHICH-REPRESENT-THE-SAME-logical-GROUP*
          do
          (setq record-number (+ 1 record-number))
          (push (list i record-number)
                result))
result))


;Command: (which-feature-groups-belong-to-the-same-record-TOP (append '("B40" "B51") '("f**")
))
;(("B40" "B51") ("f**"))
;Command:


(defun which-feature-groups-belong-to-the-same-record-TOP  (ls)      ;;;musts-valid or maybes-
valid
  (let ((help (which-feature-groups-belong-to-the-same-record ls)) (result nil))
    (let ((numbers (remove-duplicates (loop for i in help
                                            collect (cadr i)))))
      (loop for  record-number in numbers
            do
       (push  (get-the-record-of-FG-with-this-number record-number help) result)))
result))


;(defvar *FEATURE-GROUPS-WHICH-REPRESENT-THE-SAME-logical-GROUP*   ;;;;;;; FROM the file : "s
tatice-R8.dat"
;        '(  (("B40" "B51") "B51") (("D##" "D&&" "F**") "F**") ;;;; others if EXIST
;))
;Command: (number-the-common-feature-groups )
;((((("D##" "D&&" "F**") "F**") 2) ((("B40" "B51") "B51") 1))
;Command: (which-feature-groups-belong-to-the-same-record  '("B40" "B51"))
;(("B51" 1) ("B40" 1))
;Command: (which-feature-groups-belong-to-the-same-record  (append '("B40" "B51") '("f**")))
;(("f**" 2) ("B51" 1) ("B40" 1))
```

```
;Command:


(defun which-feature-groups-belong-to-the-same-record (ls)     ;;;musts-valid or maybes-valid
  (let ((temporary (number-the-common-feature-groups )) (result nil))
    (loop for i in ls
          do
      (loop for record-of-common-feature-groups   in temporary
            do
        (let ((list-of-common-FEATURE-GROUPS-to-be-CHECKED (caar record-of-common-feature-grou
ps)))
          (cond ((member i list-of-common-FEATURE-GROUPS-to-be-CHECKED  :test #'string-equal)
                 (push (list i (car (reverse record-of-common-feature-groups))) result))))))
    result))



(defun get-the-record-of-FG-with-this-number (number help)
  (let ((result nil))
       (loop for record-of-FG-with-number in help
             do
         (cond ((equal (second record-of-FG-with-number) number)
                 (push (first record-of-FG-with-number) result))))
       result))



(defun    print-the-information-from-the-reasoning-through-the-tape (APN-pathname  MODEL PART
-DESCRIPTION-we-CONCERN
                                                                     &optional (stream *standard-outp
ut*))
  (with-database (db APN-pathname)
    (with-transaction ()
      (let ((flag-of-volume-of-data-from-TAPE
             (Secondary-Description-ID-Flag-for-information-of-FEATURES-gathered-from-the-tap
e PART-DESCRIPTION-we-CONCERN)))
       (cond ((string-equal flag-of-volume-of-data-from-TAPE "QUITE-POSSIBLE")
                 (print (format nil "THE INFORMATION WE HAVE FROM THE TAPE FOR THE PART DES
CRIPTION: ~A CAN BE SAID THAT IS ENOUGH and THE Analysis OF THE features FROM THE System MAY
REPRESENT THE Reality !!!!!"
                                (secondary-Description-Id-Part-Description  PART-DESCRIPTION-w
e-CONCERN)) stream))
             (t (print (format nil "THE INFORMATION WE HAVE FROM THE TAPE FOR THE PART DESCRI
PTION: ~A IS  ONLY INDICATIVE AS NOT ENOUGH SPECIFICATION DATA WERE PROVIDED in the TAPE and
THE Analysis OF THE features FROM THE System MAY  NOT REPRESENT THE Reality !!!!!"
                               (secondary-Description-Id-Part-Description  PART-DESCRIPTION-we-C
ONCERN)) stream)))

       (let* ((all-feature-groups-with-comments
              (Secondary-Description-ID-In-English-the-detailed-information-of-HOW-the-MUST-a
nd-MAYBES-FEATURES-derived
               PART-DESCRIPTION-we-CONCERN))
             (each-feature-group-with-comments
              (get-words-from-string-* all-feature-groups-with-comments)))  ;;;;It's a list

         (loop for i in each-feature-group-with-comments
               do
           (let ((separarion-of-comment-with-code (get-words-from-string-.. i)))
             (let ((feature-group-comment (car separarion-of-comment-with-code))
                   (feature-group-codes (get-words-from-string-~ (second separarion-of-comment
-with-code))))
             (print feature-group-comment stream)
             (loop for FEAT-GROUP-CODE in feature-group-codes
                   do
```

```
(defun translation-to-English-the-feature-group (string ls)
  (BLOCK takis
    (loop for i in ls
        do
      (cond ((string-equal (car i)  string) (return-from takis (second i)))))
    nil))



;(defun create-the-STATICE-*feature-chart-entries*  (list-of-strings-of-FEATURE-GROUPS  the-BAFCs-PATHNAM
E)
;  (let ((result nil))
;    (loop for  FEATURE-GROUP   in   list-of-strings-of-FEATURE-GROUPS
;        do
;      (push  (collect-FEATURE-GROUPS-from-statice FEATURE-GROUP the-BAFCs-PATHNAME) result));;***
;    (clear result)))   ;;;;  THE OLD way of collecting the Entities from STATICE without the use of the
;inverse index
                          ;;;;  ie. the inverse function :  entity-of-the-FEATURE-GROUP



(defun collect-FEATURE-GROUPS-from-statice   (a-string    BAFCs-PATHNAME)
  (let ((result nil))
  (with-database  (db         BAFCs-PATHNAME)                ;;;;;;;*additional-features-R8-example-PATHNAME
*)
                 (with-transaction ()
                   (for-each ((ADD Feature-chart-entry)
                              (:where (string-equal (Feature-chart-entry-feat-group ADD)  a-string)))
                     (push ADD result))))
result))

;;$$$
```

```
47220>)
;Command: (USING-THE-INVERSE-capability-of-STATICE-collect-FEATURE-GROUPS-from-statice  "b51" "stat-FS:>t
akis>R8-data-onl
;y-the-ADDITIONALS")
;(#<Entity-Handle of FEATURE-CHART-ENTRY 83/1 374036210> #<Entity-Handle of FEATURE-CHART-ENTRY 83/2 3677
42702>
;  #<Entity-Handle of FEATURE-CHART-ENTRY 84/1 367753002> #<Entity-Handle of FEATURE-CHART-ENTRY 84/2 367
763742>
;  #<Entity-Handle of FEATURE-CHART-ENTRY 85/1 367773522> #<Entity-Handle of FEATURE-CHART-ENTRY 85/2 374
004360>
;  #<Entity-Handle of FEATURE-CHART-ENTRY 86/1 374014140> #<Entity-Handle of FEATURE-CHART-ENTRY 86/2 374
024560>
;  #<Entity-Handle of FEATURE-CHART-ENTRY 87/1 374034340> #<Entity-Handle of FEATURE-CHART-ENTRY 87/2 367
741032>)
;Command: (collect-FEATURE-GROUPS-from-statice "b51" "stat-FS:>takis>R8-data-only-the-ADDITIONALS")
;(#<Entity-Handle of FEATURE-CHART-ENTRY 83/1 374036210> #<Entity-Handle of FEATURE-CHART-ENTRY 83/2 3677
42702>
; #<Entity-Handle of FEATURE-CHART-ENTRY 84/1 367753002> #<Entity-Handle of FEATURE-CHART-ENTRY 84/2 3677
63742>
; #<Entity-Handle of FEATURE-CHART-ENTRY 85/1 367773522> #<Entity-Handle of FEATURE-CHART-ENTRY 85/2 3740
04360>
; #<Entity-Handle of FEATURE-CHART-ENTRY 86/1 374014140> #<Entity-Handle of FEATURE-CHART-ENTRY 86/2 3740
24560>
; #<Entity-Handle of FEATURE-CHART-ENTRY 87/1 374034340> #<Entity-Handle of FEATURE-CHART-ENTRY 87/2 3677
41032>)
;Command:


;;$$$
```

```
      (let ((code (car  group-in-ENGLISH))
            (english (cadr group-in-ENGLISH)))
        (push (list english (list nil)) result)
        (loop for STATICE-ENTITY   in            *feature-chart-entries*
              do
          (cond ((string-equal (Feature-chart-entry-feat-group  STATICE-ENTITY)
                               code)
                 (cond ((not (member (Feature-chart-entry-FEATURE-DESCRIPTION STATICE-ENTITY)

                                     (cadr (assoc english

                                                  result
                                                  :test #'string-equal
                                                  ))
                                     :test #'string-equal
                                )
                 )
                 (push (Feature-chart-entry-FEATURE-DESCRIPTION  STATICE-ENTITY)
                       (cadr (assoc english

                                     result
                                     :test #'string-equal
                                     )))
                       ))
              )
        )
))
result))
```

```
(defun NEGATIVES-features-menu (ls   negatives  &optional (stream *standard-output*))
  (let ((answers nil))
    (loop for i in ls
          do
      (let ((variable
                                     (set-difference (cadr i)
                                                     negatives
                                                     :test #'string-equal))
          )
    (push
      (dw:accepting-values (stream :own-window t
                                   :near-mode '(:point 100 700)
                                   :label "FEATURES WHICH DON'T MATCH WITH THE Part Description ")
        (accept (clear (list 'subset variable))
                :prompt (car i)  :stream  stream))       ;
      answers)
    )
)
   answers))
```

```
(defun  the-negatives-the-system-KNOWS    (PART-DESCRIPTION-we-CONCERN  )
  (get-the-NEGATIVES-OF-THE-entity-type-FROM-STATICE-TOP   PART-DESCRIPTION-we-CONCERN))
```

```
;       do
;    (let ((header (substring negative 0 3)))
;      (loop for j in *the-ENGLISH-STATEMENTS-all-AFTER*
;           do
;           (let ((header-of-group  (car j)))
;             (cond ((string-equal header header-of-group)
;                     (push negative result))))))
;   )
;result))
;
;


;Command: *the-ENGLISH-STATEMENTS-all-AFTER*
;(("B69" "B69-SEAT HEATING                    ") ("B51" "B51-SEAT FACE MATERIAL              ")
; ("B44" "B44-SEATS -RECLINING                "))
;Command:


;Command: (prompt-for-NEGATIVES)
;(("B44-SEATS -RECLINING                 " ("B44R-----PAIR REC FT SEATS LUMB&HT ADJ LUMBER  " NIL))
; ("B51-SEAT FACE MATERIAL               "
;  ("B51R-----LIGHTENING VELVET               LITVEL  " "B51L-----MORSE                          MORSE   "
;   "B51F-----AUTOLUX LEATHER                 AUTLET  " NIL))
; ("B69-SEAT HEATING                     " ("B69D-----HEATED DRIVERS & PASS SEAT    TWOHOT  " NIL)))
;;;;;;  RETURNS THE no-negative features  in this FORMAT. which are used in the
;;;;;;     "the-negatives-the-system-KNOWS"  as the first argument.    The second is the negatives from t
he FLAVORS.


(defun    prompt-for-NEGATIVES ()   ;;;;;;THE name is DECEITFUL.  It actually returns all Feature-Groups
with their features
                                    ;;;;;;;     EXISTENT  in   *data-we-concern*
  (let ((result nil))
    (loop for group-in-ENGLISH   in  *the-ENGLISH-STATEMENTS-all-AFTER*
          do
      (let ((code (car  group-in-ENGLISH))
            (english (cadr group-in-ENGLISH)))
        (push (list english (list nil)) result)
        (print result)
        (loop for line-of-input  in   *data-we-concern*
              do
          (cond ((string-equal (substring (cadr (assoc ':FEATURE-DESCRIPTION  line-of-input))
                                          0 3)
                               code)
                 (cond ((not (member (cadr (assoc ':FEATURE-DESCRIPTION  line-of-input))
                                     (cadr (assoc english

                                    result
                                    :test #'string-equal
                                    ))
                                     :test #'string-equal))
                 (push (cadr (assoc ':FEATURE-DESCRIPTION  line-of-input))
                       (cadr (assoc english

                                    result
                                    :test #'string-equal
                                    )))
                       ))
                 )
          )
))
result))
```

```
(defun maybes-add-features-menu (the-maybes   &optional (stream *standard-output*))
  (let ((answers nil))
    (push
      (dw:accepting-values (stream :own-window t
                                   :near-mode '(:point 100 700)
                                   :label "MAYBES **** ADDITIONAL Features REQUIRE Specification  ? ")
        (accept (clear (append (list  'subset)  the-maybes))
          :prompt "Enter the MAYBES ADDITIONAL FEATURES required
for the PINS Specification of that description"  :stream  stream))      ;
      answers)
     (car answers)))



(defun return-the-ENGLISH (a-string     list-of-lists-of-strings)
  (loop for i in list-of-lists-of-strings
        do
    (cond ((string-equal (car i) a-string) (return (cadr i))))))


;*****************************************************************************************

(defun      collect-from-FEAT-GROUP-INDEX-REPORT--the-VALID   (ls  MODEL)
       ;;;;ie.. (append   *must*   *maybes*),     'r8
  (let ((result nil)
        (number-indicates-MODEL (find-the-model-number    MODEL)))

    (loop for group   in   ls        ;;;; *the-GROUP-headers*
          do
      (let ((group-line  (loop for i in *the-GROUP-headers*
                                 do
                              (cond ((string-equal group (substring i 0 3))
                                       (return i))))))
      (cond (group-line
             (cond
               ((DOES-the-GROUPS-go-TO-this-MODEL?  number-indicates-MODEL   group-line)
                  (push (list group
                              (substring group-line 0 34)) result)))))
   ))result))



(defun DOES-the-GROUPS-go-TO-this-MODEL? (number    group-line)
  (string-equal (substring group-line (- number 1)
                                       number)
                "X"))


(defun  find-the-model-number  (MODEL)
  (case MODEL
    (metro 37)
    (r8 42)
    (Rover800 ????)))

;;;;;*****************************************************************************************

(defun collect-the-groups-we-concern-TOP (list-of-groups)
  (let ((result nil))
    (loop for group in (collect-the-groups-we-concern list-of-groups)
          do
      (loop for SUBGROUP in group
            do
        (push SUBGROUP result)))
result))



(defun  collect-the-groups-we-concern   (list-of-groups)
  ;;;;;; i. e...   ("b15" "b12" "b11" "b02")
  (let ((result nil))
```

```
    (loop for group in list-of-groups
          do
      (push (collect-subgroups-related   group) result)
    )
result))


(defun  collect-subgroups-related (group)
  (let ((result   nil))
    (loop  for line-of-DATA  in   (find-the-ADDITIONALS-of-modal)          ;;;;;;;;;*METRO*
           do
      (cond
        ((string-equal   group   (substring (cadr (assoc ':FEATURE-DESCRIPTION line-of-DATA))
                                            0 3))
             (push line-of-DATA
                   result))))
result))
```

```
;;;-*-  Mode: lisp;  Package: austin-rover ; syntax: common-lisp-*-



(defvar *base-features-metro* nil)
(defvar *base-features-R8* nil)
(defvar *additional-features-metro* nil)
(defvar *additional-features-R8* nil)

(defun find-the-base-of-model  ()
  (case *MODEL*
    (metro *base-features-metro*)
    (r8 *base-features-R8*))
)


(defun find-the-ADDITIONALS-of-model  ()
  (case *MODEL*
    (metro *additional-features-metro*)
    (r8 *additional-features-R8*))
)



 ;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER; Base: 10 -*-


;(defflavor feature-chart-entry
;        (FEATURE-DESCRIPTION
;         all-the-combinations-restrictions
;         all-the-comments-restrictions
;         all-the-territory-restrictions
;         recno
;         featcode
;         standard
;         base
;         optional
;         marketing
;         enveng
;         legal
;         accessory
;         mr
;         mc
;         class
;         body
;         family
;         capacity
;         performance
;         drive
;         trans
;         auth-ref-in
;         auth-ref-out
;         rel-stage-in
;         rel-stage-out
;         dep-in
;         dep-out
;         feat-group
;         history
;         errno
;         all-the-group-restrictions)
;
;        ()
;  :writable-instance-variables
;  :initable-instance-variables)


;*****************************************************************************
;;;;(defvar *feature-chart-entries* nil)
(defvar *Feature-chart-entry-by-FeatureDescription-table*
        (make-hash-table :test #'string-equal))
(defvar  *subdescription-tables* nil)
(defvar *CLASS-tables* nil)
```

```
(defun STATICE-reset-feature-chart-db ()
  ;;;;;(setq *feature-chart-entries* nil)  IS NOT now this  a JOB of this FUNCTION
  (clrhash *Feature-chart-entry-by-FeatureDescription-table*)

   (setq *subdescription-tables* nil)
   (setq *class-tables* nil))


;(defun reset-feature-chart-db ()
;   (setq *feature-chart-entries* nil)
;   (clrhash *Feature-chart-entry-by-FeatureDescription-table*)
;
;    (setq *subdescription-tables* nil)
;    (setq *class-tables* nil))

(defun STATICE-after-reset ()
   (STATICE-make-feature-chart-entries)
   (create-the-subDescription-tables)
   (create-the-class-tables)
)


(defun STATICE-make-feature-chart-entries ()
  ;;;;;; check-if-the-entry-is-not-deleted
  (let ((help nil))
    (loop for fc in *feature-chart-entries*
          do
      (let ((DEPARTURE-OUT-OF-THE-FEATURE (Feature-Chart-ENTRY-dep-out fc)))  ;;a  string  of length 8
        (cond ((IS-IT-EMPTY-the-STRING?  DEPARTURE-OUT-OF-THE-FEATURE)
                    (push fc help)))))
    (setq *feature-chart-entries* help))

  (loop for fc in *feature-chart-entries*
        do
            (add-to-hash-table fc)))



(defun IS-IT-EMPTY-the-STRING? (a-string)
  (block flag
    (loop for dimension-matrix  from 0 to (- (length a-string) 1)
          do
      (cond ((not (string-equal (aref a-string dimension-matrix) #\space))
                  (return-from flag nil))))
    T))


;(defun after-reset ()
;   (make-feature-chart-entries)
;   (create-the-subDescription-tables)
;   (create-the-class-tables)
;)


;(defun keys (h)
;  (let ((l  nil))
;    (maphash #'(lambda (key  value)
;                 (push  key l))   h)
;    l))


;*****************************************************************************************


;;;;;(defvar *feature-chart-entries* nil)
```

```
;;;;;;;(defvar *parsed-input-from-tape* (try "ul:>takis>despo.lisp"))


;(defvar *Feature-chart-entry-by-FeatureDescription-table*
;       (make-hash-table :test #'string-equal))



;*****************************************************************************************


;(defun make-feature-chart-entries ()
;  (loop for entry in  *data-we-concern*  ;;;;*mirrors-data*     ;;;;*parsed-input-from-tape*
;       as new-instance = (make-instance 'Feature-chart-entry)
;       do
;    (loop for (iv-id value) in entry
;         do
;      (case iv-id
;
;
;
;
;       (:FEATURE-DESCRIPTION
;         (setf (Feature-chart-entry-FEATURE-DESCRIPTION   new-instance) value))
;       (:all-the-combinations-restrictions
;         (setf (Feature-chart-entry-all-the-combinations-restrictions   new-instance) value))
;       (:all-the-comments-restrictions
;         (setf (Feature-chart-entry-all-the-comments-restrictions       new-instance) value))
;       (:all-the-territory-restrictions
;         (setf (Feature-chart-entry-all-the-territory-restrictions      new-instance) value))
;       (:recno
;         (setf (Feature-chart-entry-recno                       new-instance) value))
;       (:featcode
;         (setf (Feature-chart-entry-featcode                    new-instance) value))
;       (:standard
;         (setf (Feature-chart-entry-standard                    new-instance) value))
;       (:base
;         (setf (Feature-chart-entry-base                        new-instance) value))
;       (:optional
;         (setf (Feature-chart-entry-optional                    new-instance) value))
;       (:marketing
;         (setf (Feature-chart-entry-marketing                    new-instance) value))
;       (:enveng
;         (setf (Feature-chart-entry-enveng                 new-instance) value))
;       (:legal
;         (setf (Feature-chart-entry-legal                  new-instance) value))
;       (:accessory
;         (setf (Feature-chart-entry-accessory              new-instance) value))
;       (:mr
;         (setf (Feature-chart-entry-mr                     new-instance) value))
;       (:mc
;         (setf (Feature-chart-entry-mc                     new-instance) value))
;       (:class
;         (setf (Feature-chart-entry-class                  new-instance) value))
;       (:body
;         (setf (Feature-chart-entry-body                   new-instance) value))
;       (:family
;         (setf (Feature-chart-entry-family                 new-instance) value))
;       (:capacity
;         (setf (Feature-chart-entry-capacity               new-instance) value))
;       (:performance
;         (setf (Feature-chart-entry-performance            new-instance) value))
;       (:drive
;         (setf (Feature-chart-entry-drive                  new-instance) value))
;       (:trans
;         (setf (Feature-chart-entry-trans                  new-instance) value))
;       (:auth-ref-in
;         (setf (Feature-chart-entry-auth-ref-in            new-instance) value))
;       (:auth-ref-out
;         (setf (Feature-chart-entry-auth-ref-out           new-instance) value))
;       (:rel-stage-in
;         (setf (Feature-chart-entry-rel-stage-in           new-instance) value))
;       (:rel-stage-out
;         (setf (Feature-chart-entry-rel-stage-out          new-instance) value))
```

```
;       (:dep-in
;         (setf (Feature-chart-entry-dep-in                 new-instance) value))
;       (:dep-out
;         (setf (Feature-chart-entry-dep-out                new-instance) value))
;       (:feat-group
;         (setf (Feature-chart-entry-feat-group             new-instance) value))
;       (:history
;         (setf (Feature-chart-entry-history                new-instance) value))
;       (:errno
;         (setf (Feature-chart-entry-errno                  new-instance) value))
;       (:all-the-group-restrictions
;         (setf (Feature-chart-entry-all-the-group-restrictions
;                                                                 new-instance) value))
;      )   ;  THE   "  case "
;   ))     ;THE  2  LOOPS.
;
;  ;; Now hash on featcode for all entries
;
;  (loop for fc in *feature-chart-entries* do
;    (add-to-hash-table fc))
;)



;(defmethod (make-instance Feature-chart-entry) (&rest ignore)
;  (push self *feature-chart-entries*))



(defmethod (add-to-hash-table Feature-chart-entry) ()
  (let ((entries-with-this-FeatureDescription
          (gethash (substring (FEATURE-CHART-ENTRY-Feature-Description self) 0 3)
                   *Feature-chart-entry-by-FeatureDescription-table*)))
    (setf (gethash (substring (FEATURE-CHART-ENTRY-Feature-Description self) 0 3)
                   *Feature-chart-entry-by-FeatureDescription-table*)
          (cons self entries-with-this-FeatureDescription))))


;*****************************************************************************************


;;(defvar  *subdescription-tables* nil)


(defun create-the-subDescription-tables ()
  (loop for  description-key   in (keys  *Feature-chart-entry-by-FeatureDescription-table*)
        do
    (let  ((table  (make-hash-table    :test #'string-equal  )))
      (create-hash-table-SUBDESCRIPTIONS (gethash
                                          description-key
                                          *Feature-chart-entry-by-FeatureDescription-table*)
                                          table)
      (push (cons description-key table) *subdescription-tables*))
   )
)


;; Now hash on featcode for all entries

(defun create-hash-table-SUBDESCRIPTIONS   (a-Description-region   table)
  (loop for fc in    a-Description-region
        do
    (add-to-hash-table-SUBDESCRIPTIONS   fc   table)))



(defmethod (add-to-hash-table-SUBDESCRIPTIONS
                                  Feature-chart-entry) (table)
  (let ((entries-with-this-featcode
          (gethash  (FEATURE-CHART-ENTRY-Feature-Description self)  table)))
```

```
    (setf (gethash  (FEATURE-CHART-ENTRY-Feature-Description self)   table)
                          (cons self entries-with-this-featcode))))


;*****************************************************************************************


;;;;;;;;(defvar *CLASS-tables* nil)


(defun create-the-CLASS-tables ()
  (loop for (description-key . subdescription-table) in  *SUBDESCRIPTION-TABLES*
      do

    (loop for  subdescription-key   in (keys subdescription-table)
                                        do
      (let  ((table  (make-hash-table    :test #'string-equal  :size 5)))
        (create-hash-table-CLASSES (gethash
                                     subdescription-key
                                     subdescription-table)
                                   table)
        (push (cons (cons description-key subdescription-key) table) *CLASS-tables*))
    )
  )
)



;; Now hash on featcode for all entries

(defun create-hash-table-CLASSES   (a-subdescription-region  table)
  (loop for fc in    a-subdescription-region
        do
    (add-to-hash-table-CLASSES   fc   table)))



(defmethod (add-to-hash-table-CLASSES
                                Feature-chart-entry) (table)
  (let ((entries-with-this-featcode
          (gethash  (FEATURE-CHART-ENTRY-CLASS  self)  table)))

    (setf (gethash  (FEATURE-CHART-ENTRY-CLASS  self)   table)
                          (cons self entries-with-this-featcode))))



;*****************************************************************************************


;*****************************************************************************************
;;;;IT RETURNS TERRITORIES and GROUPS with THEIR countries



(defun do-it-new ()
 (let ((result-TOP nil))
 (let ((result nil))
  (loop for entry in *data-for-territories*
        do

      (let  (
```

```
            (territory-group (substring entry 34 38)))
        (cond ((find territory-group result :key #'car :test #'string-equal)
                 (let ((found (assoc territory-group result :test #'string-equal)))
                      (setq result
                            (append
                              (remove territory-group result :test #'string-equal
                                                              :key #'car)
                              (list (list territory-group
                                          (push entry
                                                (cadr found))))
                            )
                      )
                      ;(print 'result-is)(prinl result)
                 )
              )
              (t
               (push (list territory-group (list
                                                entry
                                          ))  result))
        )
      ))
   (loop for terr-code-WITH-description in result

         do
     (push (list
            (car terr-code-WITH-description)
            (what-returns-from (cadr terr-code-WITH-description)))
                        result-TOP)
  ))
 result-TOP))


(defun what-returns-from (list-of-strings)
  (loop for terr-code-and-description  in list-of-strings
        collect
          (do-function
            (substring terr-code-and-description 0 4)
            (substring terr-code-and-description 4 30)
            (substring terr-code-and-description 56 62))))


(defun do-function (a-string  another-one   one-more)
  (list a-string another-one  one-more))



;(defun take-the-table (the-header)
;  (loop for i in  *subdescription-tables*
;        do
;     (cond
;       ((string-equal the-header  (car i))
;        (return  (cdr i))))))


(defvar *territories*  nil)

(setq *territories* (do-it-new))

;*****************************************************************************************
```

```
;*****************************************************************************************


;*****************************************************************************************

;;;;Command: (give-the-implicit-TRIM-levels)
;;;;(("B15J" ((:BODY "G"))) ("B02B" NIL) ("B12D" ((:BODY "G"))))
;;;;Command: (give-the-implicit-TRIM-levels)
;;;;(("B15J" ((:BODY "G"))) ("B12D" ((:BODY "G"))))


;;;Command: (give-the-implicit-TRIM-levels-top)
;;;(("B12D" ((:BODY "G")) ("D" "E")) ("B15J" ((:BODY "G")) ("D" "E")))
;;;Command:









(defun give-the-implicit-TRIM-levels-TOP   ()
 (let ((result nil))
  (loop for subdescription-with-implicit-restrictions in (give-the-implicit-TRIM-levels)
        do
    (push
      (append   (reverse (cdr (reverse subdescription-with-implicit-restrictions)))
                (return-TRIM-LEVELS-TOP  (car (reverse subdescription-with-implicit-restrictions)))

                )

      result))
 (append result))
)



(defun give-the-implicit-TRIM-levels ()
  (let ((result nil))
    (loop for ((description-key . subdescription-key)  . class-table  ) in  *CLASS-TABLES*
          do

      (loop for  class-key   in (keys class-table)
            do
        (cond
          ((string-equal class-key " ")

           (loop for entry in (gethash class-key class-table)
                 do
             ;(cond
;              ((and              ;;;;;TO AVOID THE SITUATION "- - - - - - -"
;                 (string-equal
;                   (Feature-chart-entry-class entry) " ")
;                 (string-equal
;                   (Feature-chart-entry-body entry) " ")
;                 (string-equal
;                   (Feature-chart-entry-family entry) " ")
;                 (string-equal
;                   (Feature-chart-entry-capacity  entry) " ")
;                 (string-equal
;                   (Feature-chart-entry-performance entry) " ")
;                 (string-equal
;                   (Feature-chart-entry-drive entry) " ")
;                 (string-equal
;                   (Feature-chart-entry-trans entry) " ")
;               ))
             ;(t
            (push
                  (list
```

```
                    (Feature-chart-entry-featcode entry)

                    ;WE KNOW THAT EITHER situation CAN HAPPEN

                    (or (Feature-chart-entry-all-the-combinations-restrictions entry)
                        (Feature-chart-entry-all-the-comments-restrictions entry))

                    ;(Feature-chart-entry-all-the-combinations-restrictions entry)
                    ;(Feature-chart-entry-all-the-comments-restrictions entry)


                    (Feature-chart-entry-all-the-territory-restrictions entry)
                    (Feature-chart-entry-all-the-group-restrictions entry)


                    (remove 'nil
                    (list
                      (cond ((not (string-equal
                                  (Feature-chart-entry-class entry) " "))
                             (list ':class (Feature-chart-entry-class entry))))
                      (cond ((not (string-equal
                                  (Feature-chart-entry-body entry) " "))
                             (list ':body (Feature-chart-entry-body entry))))
                      (cond ((not (string-equal
                                  (Feature-chart-entry-family entry) " "))
                             (list ':family (Feature-chart-entry-family entry))))
                      (cond ((not (string-equal
                                  (Feature-chart-entry-capacity  entry) " "))
                             (list ':capacity (Feature-chart-entry-capacity entry))))
                      (cond ((not (string-equal
                                  (Feature-chart-entry-performance entry) " "))
                             (list ':performance (Feature-chart-entry-performance entry))))
                      (cond ((not (string-equal
                                  (Feature-chart-entry-drive entry) " "))
                             (list ':drive (Feature-chart-entry-drive entry))))
                      (cond ((not (string-equal
                                  (Feature-chart-entry-trans entry) " "))
                             (list ':transmision (Feature-chart-entry-trans entry))))

                    ))
                  )
                  result))

        ))))
  result)
)


;*****************************************************************************************


;;;;;IT     WORKS OUT   all the TRIM levels which are implied  from that COMBINATION of
;;;;;       BASE-FEATURES, (consulting the Base Features Chart) ie. '( "h" "x" "y")
;;;;;       and The CAUSE for such a search ie.::::
;;;;;                      ((:family "a") (:capacity "d") (:performance "e"))
;;;;;
;;;;;
;;;;;  IT returns a list
;;;;;  Command: (return-TRIM-LEVELS-top '((:family "a") (:capacity "d") (:performance "f")))
;;;;;(((:FAMILY "a") (:CAPACITY "d") (:PERFORMANCE "f")) ("H" "W" "Y" "Z"))
;;;;;Command:



(defun return-TRIM-LEVELS-TOP (ls)
  (list ls
        (remove-duplicates
          (loop for i in (return-TRIM-LEVELS  ls)
                collect (cadr (assoc ':class i)))   :test #'string-equal)

  )
)
```

```
(defun return-TRIM-LEVELS (ls)  ;;      ((:family "a") (:capacity "d") (:performance "e"))
 (let ((result nil))
  (loop for base-features-DETAILED-list in (do-run)
        do
    (cond
      ((all-the-indicators-true  ls  base-features-DETAILED-list)
             (push base-features-DETAILED-list result))))
  result)
)


(defun all-the-indicators-true (ls  base-features-DETAILED-list)
  (let ((flag t))
    (loop for indicator-with-value in ls
          do
      (cond
        ((not (string-equal
                (cadr (assoc  (car indicator-with-value)
                              base-features-DETAILED-list))
                (cadr indicator-with-value)))
           (setq flag nil)
        )
      ))
    flag)
)



(defun do-run ()
  (let ((result nil))
    (loop for i in (find-the-base-of-MODEL)
          do
      (push
        (list
           (assoc ':class i)
           (assoc ':body i)
           (assoc ':family i)
           (assoc ':capacity i)
           (assoc ':performance i)
           (assoc ':drive i)
           (assoc ':transmision i))
        result))
    result))

;*****************************************************************************************

:Command: (numbering-the-feat-GROUPS)
;(("B12" 3) ("C16" 1) ("B02" 1) ("B15" 5) ("B45" 1) ("B11" 2))




(setq *IMPLICIT-TRIM-LEVELS* (give-the-implicit-TRIM-levels-TOP))
                  ;I HAVE TO RUN FIRST THIS  :*IMPLICIT-TRIM-LEVELS*
                  ;;;;;THAN THE "get-the-implicit-TRIMS"  FUNCTION.









(setq *IMPLICIT-TRIM-LEVELS* (give-the-implicit-TRIM-levels-TOP))



;          (setq *combinations* (get-the-combinations-of-all-GROUPS "b15"))
```

```
;((((("B12C" "M" (#<FEATURE-CHART-ENTRY 262261677>) NIL) ("B15X" "M" (#<FEATURE-CHART-ENTRY 262257567>) NI
L))
;  (("B12A" "M" (#<FEATURE-CHART-ENTRY 262261432>) NIL) ("B15X" "M" (#<FEATURE-CHART-ENTRY 262257567>) NI
L))
;  (("C16V" "m" (#<FEATURE-CHART-ENTRY 262261226>) NIL) ("B15X" "M" (#<FEATURE-CHART-ENTRY 262257567>) NI
L))
;  (("B45B" "M" (#<FEATURE-CHART-ENTRY 262252332> #<FEATURE-CHART-ENTRY 262252265>) NIL)
;   ("B15X" "M" (#<FEATURE-CHART-ENTRY 262257567>) NIL))
;  (("B11B" "M" (#<FEATURE-CHART-ENTRY 262256651>) NIL) ("B15X" "M" (#<FEATURE-CHART-ENTRY 262257567>) NI
L))
;  (("B11C" "M" (#<FEATURE-CHART-ENTRY 262256024>) NIL) ("B15X" "M" (#<FEATURE-CHART-ENTRY 262257567>) NI
L)))
; ((("B12A" "Y" (#<FEATURE-CHART-ENTRY 262261473>) NIL) ("B15X" "Y" (#<FEATURE-CHART-ENTRY 262257526>) NI
L))
;  (("C16V" "Y" (#<FEATURE-CHART-ENTRY 262261330>) NIL) ("B15X" "Y" (#<FEATURE-CHART-ENTRY 262257526>) NI
L))
;  (("B45B" "Y" (#<FEATURE-CHART-ENTRY 262252442>) NIL) ("B15X" "Y" (#<FEATURE-CHART-ENTRY 262257526>) NI
L))
;  (("B11C" "Y" (#<FEATURE-CHART-ENTRY 262257116>) NIL) ("B15X" "Y" (#<FEATURE-CHART-ENTRY 262257526>) NI
L)))
; (("Z96B" " " NIL
;    (((((:COMBINATIONS-RESTRICTIONS-1
;        ("I" "Z96B" "," "D10P" "," "D32B" "," "E30B" "F98A" "<" "B34J" "<" "B35C" "<" "B36C" "F98G")))
;      ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U"))
;      ((:BODY "G") (:PERFORMANCE "E") (:DRIVE "A") (:TRANSMISION "A")) ("D" "H"))))
;   ("B15X" "H" (#<FEATURE-CHART-ENTRY 262257465>) NIL))
;  (("B12D" " " NIL ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))) ("B15X" "H" (#<FEATURE-CHART-ENTRY 262
257465>) NIL)
;   )
;  (("B12C" "H" (#<FEATURE-CHART-ENTRY 262261636>) NIL) ("B15X" "H" (#<FEATURE-CHART-ENTRY 262257465>) NI
L))
;  (("B12A" "H" (#<FEATURE-CHART-ENTRY 262261267> #<FEATURE-CHART-ENTRY 262261371>) NIL)
;   ("B15X" "H" (#<FEATURE-CHART-ENTRY 262257465>) NIL))
;  (("C16V" "H" (#<FEATURE-CHART-ENTRY 262261016>) NIL) ("B15X" "H" (#<FEATURE-CHART-ENTRY 262257465>) NI
L))
;  (("B45B" "H" (#<FEATURE-CHART-ENTRY 262250661> #<FEATURE-CHART-ENTRY 262252376>) NIL)
;   ("B15X" "H" (#<FEATURE-CHART-ENTRY 262257465>) NIL))
;  (("B11B" "H" (#<FEATURE-CHART-ENTRY 262256547>) NIL) ("B15X" "H" (#<FEATURE-CHART-ENTRY 262257465>) NI
L))
;  (("B11C" "H" (#<FEATURE-CHART-ENTRY 262256126> #<FEATURE-CHART-ENTRY 262256065>) NIL)
;   ("B15X" "H" (#<FEATURE-CHART-ENTRY 262257465>) NIL)))
; ((("B12C" "X" (#<FEATURE-CHART-ENTRY 262261534>) NIL) ("B15X" "X" (#<FEATURE-CHART-ENTRY 262257424>) NI
L))
;  (("C16V" "x" (#<FEATURE-CHART-ENTRY 262261122>) NIL) ("B15X" "X" (#<FEATURE-CHART-ENTRY 262257424>) NI
L))
;  (("B45B" "X" (#<FEATURE-CHART-ENTRY 262252507>) NIL) ("B15X" "X" (#<FEATURE-CHART-ENTRY 262257424>) NI
L))
;  (("B11C" "X" (#<FEATURE-CHART-ENTRY 262255712>) NIL) ("B15X" "X" (#<FEATURE-CHART-ENTRY 262257424>) NI
L)))
; ((("B12C" "W" (#<FEATURE-CHART-ENTRY 262261575>) NIL) ("B15X" "W" (#<FEATURE-CHART-ENTRY 262257363>) NI
L))
;  (("B45B" "W" (#<FEATURE-CHART-ENTRY 262250104>) NIL) ("B15X" "W" (#<FEATURE-CHART-ENTRY 262257363>) NI
L))
;  (("B11C" "W" (#<FEATURE-CHART-ENTRY 262255763>) NIL) ("B15X" "W" (#<FEATURE-CHART-ENTRY 262257363>) NI
L)))
; (("Z96B" " " NIL
;    (((((:COMBINATIONS-RESTRICTIONS-1
;        ("I" "Z96B" "," "D10P" "," "D32B" "," "E30B" "F98A" "<" "B34J" "<" "B35C" "<" "B36C" "F98G")))
;      ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U"))
;      ((:BODY "G") (:PERFORMANCE "E") (:DRIVE "A") (:TRANSMISION "A")) ("D" "H"))))
;   ("B15K" "D" (#<FEATURE-CHART-ENTRY 262257220> #<FEATURE-CHART-ENTRY 262257261>) NIL))
;  (("B12D" " " NIL ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))) ("B15K" "D" (#<FEATURE-CHART-ENTRY 262257220> #<FEATURE-CHART-ENTRY 262257261>) NIL))
;  (("B12A" "D" (#<FEATURE-CHART-ENTRY 262261057> #<FEATURE-CHART-ENTRY 262261163>) NIL)
;   ("B15K" "D" (#<FEATURE-CHART-ENTRY 262257220> #<FEATURE-CHART-ENTRY 262257261>) NIL))
;  (("B11B" "D" (#<FEATURE-CHART-ENTRY 262256404>) NIL)
;   ("B15K" "D" (#<FEATURE-CHART-ENTRY 262257220> #<FEATURE-CHART-ENTRY 262257261>) NIL)))
; (((("Z96B" " " NIL
;    (((((:COMBINATIONS-RESTRICTIONS-1
;        ("I" "Z96B" "," "D10P" "," "D32B" "," "E30B" "F98A" "<" "B34J" "<" "B35C" "<" "B36C" "F98G")))
;      ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U"))
;      ((:BODY "G") (:PERFORMANCE "E") (:DRIVE "A") (:TRANSMISION "A")) ("D" "H"))))
;   ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("B12D" " " NIL ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))
;   ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
```

```
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("B12C" "H" (#<FEATURE-CHART-ENTRY 262261636>) NIL)
;   ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("B12A" "D" (#<FEATURE-CHART-ENTRY 262261057> #<FEATURE-CHART-ENTRY 262261163>) NIL)
;   ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("B12A" "H" (#<FEATURE-CHART-ENTRY 262261267> #<FEATURE-CHART-ENTRY 262261371>) NIL)
;   ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("C16V" "H" (#<FEATURE-CHART-ENTRY 262261016>) NIL)
;   ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("B45B" "H" (#<FEATURE-CHART-ENTRY 262250661> #<FEATURE-CHART-ENTRY 262252376>) NIL)
;   ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("B11B" "D" (#<FEATURE-CHART-ENTRY 262256404>) NIL)
;   ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("B11B" "H" (#<FEATURE-CHART-ENTRY 262256547>) NIL)
;   ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("B11C" "H" (#<FEATURE-CHART-ENTRY 262256126> #<FEATURE-CHART-ENTRY 262256065>) NIL)
;   ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))))
; (((Z96B" " " NIL
;    ((((:COMBINATIONS-RESTRICTIONS-1
;        ("I" "Z96B" "," "D10P" "," "D32B" "," "E30B" "F98A" "<" "B34J" "<" "B35C" "<" "B36C" "F98G")))
;      ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U"))
;      ((:BODY "G") (:PERFORMANCE "E") (:DRIVE "A") (:TRANSMISION "A")) ("D" "H"))))
;   ("B15J" "D"
;    (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;     #<FEATURE-CHART-ENTRY 262257014>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("B12D" " " NIL ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))
;   ("B15J" "D"
;    (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;     #<FEATURE-CHART-ENTRY 262257014>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("B12C" "H" (#<FEATURE-CHART-ENTRY 262261636>) NIL)
;   ("B15J" "D"
;    (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;     #<FEATURE-CHART-ENTRY 262257014>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("B12A" "D" (#<FEATURE-CHART-ENTRY 262261057> #<FEATURE-CHART-ENTRY 262261163>) NIL)
;   ("B15J" "D"
;    (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;     #<FEATURE-CHART-ENTRY 262257014>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("B12A" "H" (#<FEATURE-CHART-ENTRY 262261267> #<FEATURE-CHART-ENTRY 262261371>) NIL)
;   ("B15J" "D"
;    (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;     #<FEATURE-CHART-ENTRY 262257014>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("C16V" "H" (#<FEATURE-CHART-ENTRY 262261016>) NIL)
;   ("B15J" "D"
;    (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;     #<FEATURE-CHART-ENTRY 262257014>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("B45B" "H" (#<FEATURE-CHART-ENTRY 262250661> #<FEATURE-CHART-ENTRY 262252376>) NIL)
;   ("B15J" "D"
;    (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;     #<FEATURE-CHART-ENTRY 262257014>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("B11B" "D" (#<FEATURE-CHART-ENTRY 262256404>) NIL)
;   ("B15J" "D"
;    (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;     #<FEATURE-CHART-ENTRY 262257014>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("B11B" "H" (#<FEATURE-CHART-ENTRY 262256547>) NIL)
;   ("B15J" "D"
;    (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;     #<FEATURE-CHART-ENTRY 262257014>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H")))))
;  (("B11C" "H" (#<FEATURE-CHART-ENTRY 262256126> #<FEATURE-CHART-ENTRY 262256065>) NIL)
;   ("B15J" "D"
```

```
;     (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;      #<FEATURE-CHART-ENTRY 262257014>)
;     ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))))
; (("Z96B" " " NIL
;     ((((:COMBINATIONS-RESTRICTIONS-1
;         ("I" "Z96B" "," "D10P" "," "D32B" "," "E30B" "F98A" "<" "B34J" "<" "B35C" "<" "B36C" "F98G")))
;       ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U"))
;       ((:BODY "G") (:PERFORMANCE "E") (:DRIVE "A") (:TRANSMISION "A")) ("D" "H"))))
;    ("B15Y" "D" (#<FEATURE-CHART-ENTRY 262257630> #<FEATURE-CHART-ENTRY 262257671>) NIL))
;   (("B12D" " " NIL ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))
;    ("B15Y" "D" (#<FEATURE-CHART-ENTRY 262257630> #<FEATURE-CHART-ENTRY 262257671>) NIL))
;   (("B12A" "D" (#<FEATURE-CHART-ENTRY 262261057> #<FEATURE-CHART-ENTRY 262261163>) NIL)
;    ("B15Y" "D" (#<FEATURE-CHART-ENTRY 262257630> #<FEATURE-CHART-ENTRY 262257671>) NIL))
;   (("B11B" "D" (#<FEATURE-CHART-ENTRY 262256404>) NIL)
;    ("B15Y" "D" (#<FEATURE-CHART-ENTRY 262257630> #<FEATURE-CHART-ENTRY 262257671>) NIL)))
; (("B12C" "M" (#<FEATURE-CHART-ENTRY 262261677>) NIL) ("B15P" "M" (#<FEATURE-CHART-ENTRY 262257322>) NI
L))
;   (("B12A" "M" (#<FEATURE-CHART-ENTRY 262261432>) NIL) ("B15P" "M" (#<FEATURE-CHART-ENTRY 262257322>) NI
L))
;   (("C16V" "m" (#<FEATURE-CHART-ENTRY 262261226>) NIL) ("B15P" "M" (#<FEATURE-CHART-ENTRY 262257322>) NI
L))
;   (("B45B" "M" (#<FEATURE-CHART-ENTRY 262252332> #<FEATURE-CHART-ENTRY 2[Abort]
;Back to Lisp Top Level in Dynamic Lisp Listener 1.
;
;Command:

;;;*****************************************************************************************

;;;;;;;FIRST  create  the *IMPLICIT-TRIM-LEVELS*   THEN "create-the-ARRAY"....



(defun create-the-ARRAY  ()
  (let ((result nil))
    (loop for ((description-key . subdescription-key)  . class-table  ) in  *CLASS-TABLES*
          do

      (loop for  class-key   in (keys class-table)
            with number-of-keys = (length (keys class-table))
            do

            ;;;WE  DON'T  WANT THE empty KEYS, BECAUSE THAT WILL BE COVERED
            ;;;   FROM THE "(implicit-record)"  FUNCTION  ie.  *IMPLICIT-TRIM-LEVELS*

        (cond ((string-equal " " class-key)
               (cond   ;;;;;WE DON'T WANT TO MISS A SITUATION WHERE ONLY A line OF AN
                       ;;;;;   EMPTY class IS APPEARED.  ie "b12d"
                 ((equal number-of-keys 1)
                     (push (list
                             description-key
                             (substring subdescription-key 0 4)
                             class-key
                             nil
                             (implicit-record (substring subdescription-key 0 4))
                           ) result))))
              (t
               (push (list
                       description-key
                       (substring subdescription-key 0 4)
                       class-key
                       (gethash class-key class-table)
                       (implicit-record (substring subdescription-key 0 4))

                       )
              result)))
      ))
   result))


(defun implicit-record (subdescription-key)
  (let ((result nil))
       (loop for implicit-field in *implicit-trim-levels*
             do
         (cond
           ((string-equal subdescription-key (car implicit-field))
```

```
                (push (cdr implicit-field) result))))
        result))


;;;;;(defvar  *array* (create-the-ARRAY))

;;*****************************************************************************************

;;;;;;;ASSUME  WE HAVE a global variable *negatives*  :
;;;;;;;
;;;;;;;
;;;;;;;  Command: *negatives*
;;;;;;; ("B11C-----REAR QTR LIGHT HINGED          HINRQ    " "B15J-----TWIN HAND SET DOOR MIRRORS;;;;;;;
 2HSMIR  ")
;;;;;;;Command:



(defun OUT-NEGATIVES-FROM-ARRAY ()
  (let ((result *array*))
    (loop for all-negative-key  in *negatives*
          do
      (setq result
            (remove (substring  all-negative-key  0 4)  result  :test #'string-equal :key #'second)))
    result))


;*****************************************************************************************



;;;THEN WE HAVE A FUNCTION WHICH  defines WHICH IS THE  Master FEATURE  TO START WITH .
;;;;    ie.  1)  ARE  THERE    "MUST" features?
;;;;;                 get THE ONE WITH THE MOST   VARIANTS.
;;;;         2)  ONLY  one ?  GET THAT
;;;;         3)  ELSE  number the AVAILABLE    ONES .


;*****************************************************************************************

;;;;;LET'S   WE DECIDED THE   M A S T E R   FEATURE  is:  "b15"
;;;;;;
;;;;;


;;;;;;    A F T E R   T H E   N E G A T I V E S     O   U   T.


;;;*****************************************************************************************
;
;;COMMAND:
;;              (setq *array* (create-the-ARRAY))
;(("B11" "B11C" "X" (#<FEATURE-CHART-ENTRY 364005125>) NIL) ("B11" "B11C" "W" (#<FEATURE-CHART-ENTRY 3640
05233>) NIL)
; ("B11" "B11C" "Y" (#<FEATURE-CHART-ENTRY 364005555>) NIL) ("B11" "B11C" "M" (#<FEATURE-CHART-ENTRY 3640
05445>) NIL)
; ("B11" "B11C" "H" (#<FEATURE-CHART-ENTRY 364004756> #<FEATURE-CHART-ENTRY 364005017>) NIL)
; ("B11" "B11B" "M" (#<FEATURE-CHART-ENTRY 364005402>) NIL) ("B11" "B11B" "H" (#<FEATURE-CHART-ENTRY 3640
04713>) NIL)
; ("B11" "B11B" "D" (#<FEATURE-CHART-ENTRY 364004506>) NIL)
; ("B45" "B45B" "M" (#<FEATURE-CHART-ENTRY 364005276> #<FEATURE-CHART-ENTRY 364005341>) NIL)
; ("B45" "B45B" "H" (#<FEATURE-CHART-ENTRY 364004607> #<FEATURE-CHART-ENTRY 364004652>) NIL)
; ("B45" "B45B" "Y" (#<FEATURE-CHART-ENTRY 364005512>) NIL) ("B45" "B45B" "X" (#<FEATURE-CHART-ENTRY 3640
05062>) NIL)
; ("B45" "B45B" "W" (#<FEATURE-CHART-ENTRY 364005170>) NIL) ("B15" "B15X" "M" (#<FEATURE-CHART-ENTRY 3640
00002>) NIL)
; ("B15" "B15X" "Y" (#<FEATURE-CHART-ENTRY 364000047>) NIL) ("B15" "B15X" "H" (#<FEATURE-CHART-ENTRY 3640
```

```
00112>) NIL)
; ("B15" "B15X" "X" (#<FEATURE-CHART-ENTRY 364000155>) NIL) ("B15" "B15X" "W" (#<FEATURE-CHART-ENTRY 3640
00220>) NIL)
; ("B15" "B15K" "D" (#<FEATURE-CHART-ENTRY 364000263> #<FEATURE-CHART-ENTRY 364000326>) NIL)
; ("B15" "B15J" "H" (#<FEATURE-CHART-ENTRY 364000367> #<FEATURE-CHART-ENTRY 364000432>)
;  ((NIL NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))
; ("B15" "B15J" "D"
;  (#<FEATURE-CHART-ENTRY 364000473> #<FEATURE-CHART-ENTRY 364000536> #<FEATURE-CHART-ENTRY 364000642>
;   #<FEATURE-CHART-ENTRY 364001014>)
;  ((NIL NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))
; ("B15" "B15Y" "D" (#<FEATURE-CHART-ENTRY 364000577> #<FEATURE-CHART-ENTRY 364000703>) NIL)
; ("B15" "B15P" "M" (#<FEATURE-CHART-ENTRY 364000744>) NIL) ("B02" "B02B" " " NIL NIL)
; ("C16" "C16V" "Y" (#<FEATURE-CHART-ENTRY 364001055>) NIL) ("C16" "C16V" "m" (#<FEATURE-CHART-ENTRY 3640
02013>) NIL)
; ("C16" "C16V" "x" (#<FEATURE-CHART-ENTRY 364002061>) NIL) ("C16" "C16V" "H" (#<FEATURE-CHART-ENTRY 3640
02127>) NIL)
; ("B12" "B12A" "Y" (#<FEATURE-CHART-ENTRY 364002174>) NIL) ("B12" "B12A" "M" (#<FEATURE-CHART-ENTRY 3640
02237>) NIL)
; ("B12" "B12A" "H" (#<FEATURE-CHART-ENTRY 364002302> #<FEATURE-CHART-ENTRY 364002345>) NIL)
; ("B12" "B12A" "D" (#<FEATURE-CHART-ENTRY 364002406> #<FEATURE-CHART-ENTRY 364002451>) NIL)
; ("B12" "B12C" "M" (#<FEATURE-CHART-ENTRY 364002512>) NIL) ("B12" "B12C" "H" (#<FEATURE-CHART-ENTRY 3640
02555>) NIL)
; ("B12" "B12C" "W" (#<FEATURE-CHART-ENTRY 364002620>) NIL) ("B12" "B12C" "X" (#<FEATURE-CHART-ENTRY 3640
02663>) NIL)
; ("B12" "B12D" " " NIL ((NIL NIL NIL (" " NIL) ((:BODY "G")) ("D" "M"))))
; ("Z96" "Z96B" " " NIL
;  ((((:COMBINATIONS-RESTRICTIONS-1
;      ("I" "Z96B" "," "D10P" "," "D32B" "," "E30B" "F98A" "<" "B34J" "<" "B35C" "<" "B36C" "F98G")))
;    NIL ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U"))
;    ((:BODY "G") (:PERFORMANCE "E") (:DRIVE "A") (:TRANSMISION "A")) ("D" "H")))))


;*****************************************************************************************
;; I WILL CONCERN ONLY WITH ONE list-restrictions IN ORDER TO FIND if THERE EXIST COMMON GROUND
;;;     OF SPECIFICATION.
;;;;  NOTE :
;;;;         THIS COMBINATION OF RESTRICTIONS in common ground INCLUDES *implicit-trims ,as wel;   1:
;
;   1) NIL
;   2) ((NIL NIL NIL (" " NIL) ((:BODY "G")) (NIL))
;
;
;(("B12A" "D" (#<FEATURE-CHART-ENTRY 262157174> #<FEATURE-CHART-ENTRY 262157133>) NIL)
; ("B15J" "D"
;  (#<FEATURE-CHART-ENTRY 262154453> #<FEATURE-CHART-ENTRY 262156111> #<FEATURE-CHART-ENTRY 262156360>
;   #<FEATURE-CHART-ENTRY 262156421>)
;  ((NIL NIL NIL (" " NIL) ((:BODY "G")) (NIL)))))
;
;
;
;
;
;
;;;;;   MORE  SPECIFICALLY,  I WILL CONCERN  FOR THE  common  ground of EACH of the
;;;    restrictions FIRST.
;;;;   MORE SPECIFICALLY, first with the COMMON GROUND of the first part of each one
;;;     i e.
;;;;
;;;;   (#<FEATURE-CHART-ENTRY 262157174> #<FEATURE-CHART-ENTRY 262157133>)
;;;
;;;     AND
;;;
;;;    (#<FEATURE-CHART-ENTRY 262154453> #<FEATURE-CHART-ENTRY 262156111> #<FEATURE-CHART-ENTRY 262156360
>
;   #<FEATURE-CHART-ENTRY 262156421>)
;;;
;
;



;(defun common-ground-of-FEATURE-CHART-ENTRIES-of-the-same-feature
;            (list-of-FEATURE-CHART-ENTRIES)
```

```
(defvar *GROUPS* '("b02" "b15" "b12" "c16" "Z96" "b11" "b45"))



(defun group-together-GROUPS (ls)     ;;;;;(group-together-SUBGROUPS *subgroup*)
   (prog  (result subgroups record)
          (setq subgroups ls)

     LOOP

        (cond
          ((null subgroups) (return result))
          ((member (substring (caar subgroups) 0 3)
                   result
                   :test #'string-equal :key #'car)
                        (push

                              (car subgroups) (cadr (assoc (substring (caar subgroups) 0 3)
                                                           result
                                                     :test #'string-equal)


                                                 )))
          (t (cond
               ((member (substring (caar subgroups) 0 3)
                        record :test #'string-equal))
               (t (push (list (substring (caar subgroups) 0 3) (list (car subgroups)))  result)
                  (push (substring (caar subgroups) 0 3) record)
                )
              )
          )

        )


        (setq subgroups (cdr subgroups))
        (go LOOP)
  )
)



;(defun group-together-SUBGROUPS (ls)
;   (prog  (result array record)
;          (setq array ls)
;     LOOP
;
;        (cond
;          ((null array) (return result))
;          ((member (second (car array)) result  :test #'string-equal :key #'car)
;                        (push
;
;                              (cdr (car array)) (cadr (assoc (second (car array))
;                                                             result
;                                                       :test #'string-equal)
;
;
;
;                                                 )))
;          (t (cond
;               ((member (second (car array))
;                        record :test #'string-equal))
;               (t (push (list (second (car array)) (list (cdr (car array)))) result)
;                  (push (second (car array)) record)
;                )
;              )
;          )
;
;        )
;        (setq array (cdr array))
;        (go LOOP)
;  )
;)
```

```
;Command: (group-together-SUBGROUPS *array*)
;(("Z96B"
;  (("Z96B" " " NIL
;    ((((:COMBINATIONS-RESTRICTIONS-1
;        ("I" "Z96B" "," "D10P" "," "D32B" "," "E30B" "F98A" "<" "B34J" "<" "B35C" "<" "B36C" "F98G")))
;      ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U"))
;      ((:BODY "G") (:PERFORMANCE "E") (:DRIVE "A") (:TRANSMISION "A")) ("D" "H"))))))
; ("B12D" (("B12D" " " NIL ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))))
; ("B12C"
;  (("B12C" "X" (#<FEATURE-CHART-ENTRY 262261534>) NIL) ("B12C" "W" (#<FEATURE-CHART-ENTRY 262261575>) NI
L)
;   ("B12C" "H" (#<FEATURE-CHART-ENTRY 262261636>) NIL) ("B12C" "M" (#<FEATURE-CHART-ENTRY 262261677>) NI
L)))
; ("B12A"
;  (("B12A" "D" (#<FEATURE-CHART-ENTRY 262261057> #<FEATURE-CHART-ENTRY 262261163>) NIL)
;   ("B12A" "H" (#<FEATURE-CHART-ENTRY 262261267> #<FEATURE-CHART-ENTRY 262261371>) NIL)
;   ("B12A" "M" (#<FEATURE-CHART-ENTRY 262261432>) NIL) ("B12A" "Y" (#<FEATURE-CHART-ENTRY 262261473>) NI
L)))
; ("C16V"
;  (("C16V" "H" (#<FEATURE-CHART-ENTRY 262261016>) NIL) ("C16V" "x" (#<FEATURE-CHART-ENTRY 262261122>) NI
L)
;   ("C16V" "m" (#<FEATURE-CHART-ENTRY 262261226>) NIL) ("C16V" "Y" (#<FEATURE-CHART-ENTRY 262261330>) NI
L)))
; ("B02B" (("B02B" " " NIL NIL))) ("B15P" (("B15P" "M" (#<FEATURE-CHART-ENTRY 262257322>) NIL)))
; ("B15Y" (("B15Y" "D" (#<FEATURE-CHART-ENTRY 262257630> #<FEATURE-CHART-ENTRY 262257671>) NIL)))
; ("B15J"
;  (("B15J" "D"
;    (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;     #<FEATURE-CHART-ENTRY 262257014>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))
;   ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;    ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))))
; ("B15K" (("B15K" "D" (#<FEATURE-CHART-ENTRY 262257220> #<FEATURE-CHART-ENTRY 262257261>) NIL)))
; ("B15X"
;  (("B15X" "W" (#<FEATURE-CHART-ENTRY 262257363>) NIL) ("B15X" "X" (#<FEATURE-CHART-ENTRY 262257424>) NI
L)
;   ("B15X" "H" (#<FEATURE-CHART-ENTRY 262257465>) NIL) ("B15X" "Y" (#<FEATURE-CHART-ENTRY 262257526>) NI
L)
;   ("B15X" "M" (#<FEATURE-CHART-ENTRY 262257567>) NIL)))
; ("B45B"
;  (("B45B" "W" (#<FEATURE-CHART-ENTRY 262250104>) NIL) ("B45B" "X" (#<FEATURE-CHART-ENTRY 262252507>) NI
L)
;   ("B45B" "Y" (#<FEATURE-CHART-ENTRY 262252442>) NIL)
;   ("B45B" "H" (#<FEATURE-CHART-ENTRY 262250661> #<FEATURE-CHART-ENTRY 262252376>) NIL)
;   ("B45B" "M" (#<FEATURE-CHART-ENTRY 262252332> #<FEATURE-CHART-ENTRY 262252265>) NIL)))
; ("B11B"
;  (("B11B" "D" (#<FEATURE-CHART-ENTRY 262256404>) NIL) ("B11B" "H" (#<FEATURE-CHART-ENTRY 262256547>) NI
L)
;   ("B11B" "M" (#<FEATURE-CHART-ENTRY 262256651>) NIL)))
; ("B11C"
;  (("B11C" "H" (#<FEATURE-CHART-ENTRY 262256126> #<FEATURE-CHART-ENTRY 262256065>) NIL)
;   ("B11C" "M" (#<FEATURE-CHART-ENTRY 262256024>) NIL) ("B11C" "Y" (#<FEATURE-CHART-ENTRY 262257116>) NI
L)
;   ("B11C" "W" (#<FEATURE-CHART-ENTRY 262255763>) NIL) ("B11C" "X" (#<FEATURE-CHART-ENTRY 262255712>) NI
L)))   (setq *grouped* '
;
;
:Command: (group-together-GROUPS  (group-together-SUBGROUPS *array*))
;(("B11"
;  (("B11C"
;    (("B11C" "H" (#<FEATURE-CHART-ENTRY 262256126> #<FEATURE-CHART-ENTRY 262256065>) NIL)
;     ("B11C" "M" (#<FEATURE-CHART-ENTRY 262256024>) NIL) ("B11C" "Y" (#<FEATURE-CHART-ENTRY 262257116>)
NIL)
;     ("B11C" "W" (#<FEATURE-CHART-ENTRY 262255763>) NIL) ("B11C" "X" (#<FEATURE-CHART-ENTRY 262255712>)
NIL)))
;   ("B11B"
;    (("B11B" "D" (#<FEATURE-CHART-ENTRY 262256404>) NIL) ("B11B" "H" (#<FEATURE-CHART-ENTRY 262256547>)
NIL)
;     ("B11B" "M" (#<FEATURE-CHART-ENTRY 262256651>) NIL)))))
; ("B45"
;  (("B45B"
;    (("B45B" "W" (#<FEATURE-CHART-ENTRY 262250104>) NIL) ("B45B" "X" (#<FEATURE-CHART-ENTRY 262252507>)
NIL)
;     ("B45B" "Y" (#<FEATURE-CHART-ENTRY 262252442>) NIL)
;     ("B45B" "H" (#<FEATURE-CHART-ENTRY 262250661> #<FEATURE-CHART-ENTRY 262252376>) NIL)
;     ("B45B" "M" (#<FEATURE-CHART-ENTRY 262252332> #<FEATURE-CHART-ENTRY 262252265>) NIL)))))
```

```
; ("B15"
;  (("B15X"
;    (("B15X" "W" (#<FEATURE-CHART-ENTRY 262257363>) NIL) ("B15X" "X" (#<FEATURE-CHART-ENTRY 262257424>)
NIL)
;     ("B15X" "H" (#<FEATURE-CHART-ENTRY 262257465>) NIL) ("B15X" "Y" (#<FEATURE-CHART-ENTRY 262257526>)
NIL)
;     ("B15X" "M" (#<FEATURE-CHART-ENTRY 262257567>) NIL)))
;   ("B15K" (("B15K" "D" (#<FEATURE-CHART-ENTRY 262257220> #<FEATURE-CHART-ENTRY 262257261>) NIL)))
;   ("B15J"
;    (("B15J" "D"
;      (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753
>
;       #<FEATURE-CHART-ENTRY 262257014>)
;      ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))
;     ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;      ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))))
;   ("B15Y" (("B15Y" "D" (#<FEATURE-CHART-ENTRY 262257630> #<FEATURE-CHART-ENTRY 262257671>) NIL)))
;   ("B15P" (("B15P" "M" (#<FEATURE-CHART-ENTRY 262257322>) NIL)))))
; ("B02" (("B02B" (("B02B" " " NIL NIL)))))
; ("C16"
;  (("C16V"
;    (("C16V" "H" (#<FEATURE-CHART-ENTRY 262261016>) NIL) ("C16V" "x" (#<FEATURE-CHART-ENTRY 262261122>)
NIL)
;     ("C16V" "m" (#<FEATURE-CHART-ENTRY 262261226>) NIL) ("C16V" "Y" (#<FEATURE-CHART-ENTRY 262261330>)
NIL)))))
; ("B12"
;  (("B12A"
;    (("B12A" "D" (#<FEATURE-CHART-ENTRY 262261057> #<FEATURE-CHART-ENTRY 262261163>) NIL)
;     ("B12A" "H" (#<FEATURE-CHART-ENTRY 262261267> #<FEATURE-CHART-ENTRY 262261371>) NIL)
;     ("B12A" "M" (#<FEATURE-CHART-ENTRY 262261432>) NIL) ("B12A" "Y" (#<FEATURE-CHART-ENTRY 262261473>)
NIL)))
;   ("B12C"
;    (("B12C" "X" (#<FEATURE-CHART-ENTRY 262261534>) NIL) ("B12C" "W" (#<FEATURE-CHART-ENTRY 262261575>)
NIL)
;     ("B12C" "H" (#<FEATURE-CHART-ENTRY 262261636>) NIL) ("B12C" "M" (#<FEATURE-CHART-ENTRY 262261677>)
NIL)))
;   ("B12D" (("B12D" " " NIL ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))))))
; ("Z96"
;  (("Z96B"
;    (("Z96B" " " NIL
;      ((((:COMBINATIONS-RESTRICTIONS-1
;          ("I" "Z96B" "," "D10P" "," "D32B" "," "E30B" "F98A" "<" "B34J" "<" "B35C" "<" "B36C" "F98G")))
;        ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U"))
;        ((:BODY "G") (:PERFORMANCE "E") (:DRIVE "A") (:TRANSMISION "A")) ("D" "H"))))))))))
;Command: '("B15" (("B15X" #) ("B15K" #) ("B15J" #) ("B15Y" #) ("B15P" #)))
;("B15"
; (("B15X"
;   (("B15X" "W" (#<FEATURE-CHART-ENTRY 262257363>) NIL) ("B15X" "X" (#<FEATURE-CHART-ENTRY 262257424>) N
IL)
;    ("B15X" "H" (#<FEATURE-CHART-ENTRY 262257465>) NIL) ("B15X" "Y" (#<FEATURE-CHART-ENTRY 262257526>) N
IL)
;    ("B15X" "M" (#<FEATURE-CHART-ENTRY 262257567>) NIL)))
;  ("B15K" (("B15K" "D" (#<FEATURE-CHART-ENTRY 262257220> #<FEATURE-CHART-ENTRY 262257261>) NIL)))
;  ("B15J"
;   (("B15J" "D"
;     (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;      #<FEATURE-CHART-ENTRY 262257014>)
;     ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))
;    ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;     ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))))
;  ("B15Y" (("B15Y" "D" (#<FEATURE-CHART-ENTRY 262257630> #<FEATURE-CHART-ENTRY 262257671>) NIL)))
;  ("B15P" (("B15P" "M" (#<FEATURE-CHART-ENTRY 262257322>) NIL)))))
;Command:
;
```

```
      ;;;ie.
      ;;;;
;("B15J"
;;    (("B15J" "D"
```

```
;;        (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 26225675
3>
;;         #<FEATURE-CHART-ENTRY 262257014>)
;;        ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))
;;       ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;;        ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))))
;;;
;;;
;;;        AGAINST
;;;
;;;
;;;
;("B12"
;;   (("B12A"
;;      (("B12A" "D" (#<FEATURE-CHART-ENTRY 262261057> #<FEATURE-CHART-ENTRY 262261163>) NIL)
;;       ("B12A" "H" (#<FEATURE-CHART-ENTRY 262261267> #<FEATURE-CHART-ENTRY 262261371>) NIL)
;;       ("B12A" "M" (#<FEATURE-CHART-ENTRY 262261432>) NIL) ("B12A" "Y" (#<FEATURE-CHART-ENTRY 262261473>)
 NIL)))
;;    ("B12C"
;;     (("B12C" "X" (#<FEATURE-CHART-ENTRY 262261534>) NIL) ("B12C" "W" (#<FEATURE-CHART-ENTRY 262261575>)
 NIL)
;;       ("B12C" "H" (#<FEATURE-CHART-ENTRY 262261636>) NIL) ("B12C" "M" (#<FEATURE-CHART-ENTRY 262261677>)
 NIL)))
;;    ("B12D" (("B12D" " " NIL ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))))))
```

```
:Command: (setq boula '("B12A" (("B12A" "D" # NIL) ("B12A" "H" # NIL) ("B12A" "M" # NIL) ("B12A" "Y" # NI
L))))
;("B12A"
; (("B12A" "D" (#<FEATURE-CHART-ENTRY 262261057> #<FEATURE-CHART-ENTRY 262261163>) NIL)
;  ("B12A" "H" (#<FEATURE-CHART-ENTRY 262261267> #<FEATURE-CHART-ENTRY 262261371>) NIL)
;  ("B12A" "M" (#<FEATURE-CHART-ENTRY 262261432>) NIL) ("B12A" "Y" (#<FEATURE-CHART-ENTRY 262261473>) NIL
)))
:Command: mas
;("B15J"
; (("B15J" "D"
;   (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;    #<FEATURE-CHART-ENTRY 262257014>)
;   ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))
;  ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;   ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))))
:Command: (all-trims-common-with-RESTRICTIONS mas boula)
;(("B12A"
;  (("H" (#<FEATURE-CHART-ENTRY 262261267> #<FEATURE-CHART-ENTRY 262261371>))
;   ("D" (#<FEATURE-CHART-ENTRY 262261057> #<FEATURE-CHART-ENTRY 262261163>)))))
; ("B15J"
;  (("H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>))
;   ("D"
;    (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;     #<FEATURE-CHART-ENTRY 262257014>))
;   ("D" NIL NIL (" " NIL) ((:BODY "G"))) ("H" NIL NIL (" " NIL) ((:BODY "G"))))))
:Command:
```

```
(defun  all-trims-common-with-RESTRICTIONS (Master-feature   Feature-from-group)
  (let* ((TOP-RESULT nil)
         (trims-of-Master-feature (all-trims-of-FEATURE Master-feature))
         (trims-of-Feature-from-group (all-trims-of-FEATURE Feature-from-group))
         (intersection-of-those-FEATURES
           (cond ((equal Master-feature nil) trims-of-Feature-from-group)
                 ((equal Feature-from-group nil) trims-of-Master-feature)
                 (t
```

```
                  (intersection trims-of-Master-feature trims-of-Feature-from-group :test #'string-equal)
)
        )))


    (cond
        (intersection-of-those-FEATURES
           (let (
                 (common-trims-RESTRICTED-of-this-MASTER-FEATURE

                     (append
                       (create-the-explicit-statements Master-feature intersection-of-those-FEATURES)
                       (create-the-implicit-statements  Master-feature intersection-of-those-FEATURES)
    ))

                 (common-trims-RESTRICTED-of-this-SUBFEATURE
                   (append
                       (create-the-explicit-statements Feature-from-group intersection-of-those-FEATURES)
                       (create-the-implicit-statements  Feature-from-group intersection-of-those-FEATURES
)
                 ))
          )


             (push (list (car Master-feature)
                         common-trims-RESTRICTED-of-this-MASTER-FEATURE) TOP-RESULT)
             (push (list (car Feature-from-group)
                         common-trims-RESTRICTED-of-this-SUBFEATURE) TOP-RESULT)
        ))))let
      )))))intersection
    )  )))))the  top COND

))))(print intersection-of-those-FEATURES)
TOP-RESULT

))




(defun all-trims-of-FEATURE (a-Feature ) )))) i.e
      )))))("B15J"
;;     (("B15J" "D"
;;       (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 26225675
3>
;;        #<FEATURE-CHART-ENTRY 262257014>)
;;       ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "E"))))
;;      ("B15J" "E" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;;       ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "E"))))))
;;;

 (let* ((TRIM-LEVELS-of-THIS-Feature-from-ENTRIES
           (loop for ms-feature in (car (reverse a-Feature))
                 collect (second ms-feature))
         )
         (TRIM-LEVELS-of-this-Feature-from-implicit
           (let ((vague-trims nil))
             (loop for ms-feature in (car (reverse a-Feature))
                   do
               (loop for values in (car (reverse ms-feature))
                     do
                       (push
                         (car (reverse values)) vague-trims)))
                 (clear vague-trims))
         )

         (ALL-the-TRIMS-of-that-FEATURE
           (remove-duplicates (append TRIM-LEVELS-of-this-Feature-from-ENTRIES
                                      TRIM-LEVELS-of-this-Feature-from-implicit)
                              :test #'string-equal)
         )
) (remove " " ALL-the-TRIMS-of-that-FEATURE  :test #'string-equal)))
```

```
;Command: mas
;("B15J"
; (("B15J" "D"
;   (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;    #<FEATURE-CHART-ENTRY 262257014>)
;   ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))
;  ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;   ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))))
;Command: (create-the-explicit-statements mas '("D" "M"))
;(("D"
;  (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;   #<FEATURE-CHART-ENTRY 262257014>)))
;Command: (create-the-implicit-statements mas '("D" "M"))
;(("D" NIL NIL (" " NIL) ((:BODY "G"))))
;Command:
```

```
(defun create-the-explicit-statements (a-feature intersection)
  (let ((result nil))
    (loop for ms-feature in (car (reverse a-Feature))
          do
      (cond ((member (second ms-feature) intersection :test #'string-equal)
                 (push (list  (second ms-feature) (third ms-feature)) result)))
    )result))
```

```
(defun  create-the-implicit-statements (a-feature  intersection)
 (let ((result
  (multiply
    (let* ((vague-restrictions nil))
         (loop for ms-feature in (car (reverse a-Feature))
               do
           (loop for values in (car (reverse ms-feature))
                 do

             (push values vague-restrictions)))
         vague-restrictions))
  )
      (result-top nil)
     )
  (loop for implicit-statement in result
        do
    (cond
      ((member (car implicit-statement) intersection :test #'string-equal)
          (push implicit-statement result-top))))
  result-top))
```

```
(defun multiply (ls)
 (let ((result nil))
  (loop for i in ls
        do
    (let ((implicit-trims (car (reverse i))))
      (loop for implicit-trim in implicit-trims
            do
        (cond ((not (member (append (list implicit-trim)
                                    (remove  implicit-trims i))
                            result  :test #'equal))
               (push
                 (append (list implicit-trim)
                         (remove  implicit-trims i)) result))))))
  result))
```

```
;*****************************************************************************************

;Command: (all-trims-common-with-RESTRICTIONS mas boula)
;(("B12A"
;  (("H" (#<FEATURE-CHART-ENTRY 262261267> #<FEATURE-CHART-ENTRY 262261371>))
;   ("D" (#<FEATURE-CHART-ENTRY 262261057> #<FEATURE-CHART-ENTRY 262261163>))))
; ("B15J"
;  (("H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>))
;   ("D"
;    (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;     #<FEATURE-CHART-ENTRY 262257014>))
;   ("D" NIL NIL (" " NIL) ((:BODY "G"))) ("H" NIL NIL (" " NIL) ((:BODY "G"))))))
;Command:





(defun group-together-SUBGROUPS (ls)
   (prog  (result array record)
          (setq array ls)
     LOOP

        (cond
          ((null array) (return result))
          ((member (second (car array)) result  :test #'string-equal :key #'car)
                        (push

                              (cdr (car array)) (cadr (assoc (second (car array))
                                                              result
                                                         :test #'string-equal)


                                                 )))
          (t (cond
               ((member (second (car array))
                        record :test #'string-equal))
               (t (push (list (second (car array)) (list (cdr (car array)))) result)
                  (push (second (car array)) record)
               )
             )
          )

        )
        (setq array (cdr array))
        (go LOOP)
  )
)



;Command: *grouped*
;(("B11"
;  (("B11C"
;    (("B11C" "H" (#<FEATURE-CHART-ENTRY 262256126> #<FEATURE-CHART-ENTRY 262256065>) NIL)
;     ("B11C" "M" (#<FEATURE-CHART-ENTRY 262256024>) NIL) ("B11C" "Y" (#<FEATURE-CHART-ENTRY 262257116>)
NIL)
;     ("B11C" "W" (#<FEATURE-CHART-ENTRY 262255763>) NIL) ("B11C" "X" (#<FEATURE-CHART-ENTRY 262255712>)
NIL)))
;   ("B11B"
;    (("B11B" "D" (#<FEATURE-CHART-ENTRY 262256404>) NIL) ("B11B" "H" (#<FEATURE-CHART-ENTRY 262256547>)
NIL)
;     ("B11B" "M" (#<FEATURE-CHART-ENTRY 262256651>) NIL)))))
; ("B45"
;  (("B45B"
;    (("B45B" "W" (#<FEATURE-CHART-ENTRY 262250104>) NIL) ("B45B" "X" (#<FEATURE-CHART-ENTRY 262252507>)
NIL)
;     ("B45B" "Y" (#<FEATURE-CHART-ENTRY 262252442>) NIL)
;     ("B45B" "H" (#<FEATURE-CHART-ENTRY 262250661> #<FEATURE-CHART-ENTRY 262252376>) NIL)
;     ("B45B" "M" (#<FEATURE-CHART-ENTRY 262252332> #<FEATURE-CHART-ENTRY 262252265>) NIL)))))
; ("B15"
;  (("B15X"
```

```
;     (("B15X" "W" (#<FEATURE-CHART-ENTRY 262257363>) NIL) ("B15X" "X" (#<FEATURE-CHART-ENTRY 262257424>)
NIL)
;      ("B15X" "H" (#<FEATURE-CHART-ENTRY 262257465>) NIL) ("B15X" "Y" (#<FEATURE-CHART-ENTRY 262257526>)
NIL)
;      ("B15X" "M" (#<FEATURE-CHART-ENTRY 262257567>) NIL)))
;    ("B15K" (("B15K" "D" (#<FEATURE-CHART-ENTRY 262257220> #<FEATURE-CHART-ENTRY 262257261>) NIL)))
;    ("B15J"
;     (("B15J" "D"
;       (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753
>
;        #<FEATURE-CHART-ENTRY 262257014>)
;       ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))
;      ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;       ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))))
;    ("B15Y" (("B15Y" "D" (#<FEATURE-CHART-ENTRY 262257630> #<FEATURE-CHART-ENTRY 262257671>) NIL)))
;    ("B15P" (("B15P" "M" (#<FEATURE-CHART-ENTRY 262257322>) NIL)))))
; ("B02" (("B02B" (("B02B" " " NIL NIL)))))
; ("C16"
;  (("C16V"
;     (("C16V" "H" (#<FEATURE-CHART-ENTRY 262261016>) NIL) ("C16V" "x" (#<FEATURE-CHART-ENTRY 262261122>)
NIL)
;      ("C16V" "m" (#<FEATURE-CHART-ENTRY 262261226>) NIL) ("C16V" "Y" (#<FEATURE-CHART-ENTRY 262261330>)
NIL)))))
; ("B12"
;  (("B12A"
;     (("B12A" "D" (#<FEATURE-CHART-ENTRY 262261057> #<FEATURE-CHART-ENTRY 262261163>) NIL)
;      ("B12A" "H" (#<FEATURE-CHART-ENTRY 262261267> #<FEATURE-CHART-ENTRY 262261371>) NIL)
;      ("B12A" "M" (#<FEATURE-CHART-ENTRY 262261432>) NIL) ("B12A" "Y" (#<FEATURE-CHART-ENTRY 262261473>)
NIL)))
;    ("B12C"
;     (("B12C" "X" (#<FEATURE-CHART-ENTRY 262261534>) NIL) ("B12C" "W" (#<FEATURE-CHART-ENTRY 262261575>)
NIL)
;      ("B12C" "H" (#<FEATURE-CHART-ENTRY 262261636>) NIL) ("B12C" "M" (#<FEATURE-CHART-ENTRY 262261677>)
NIL)))
;    ("B12D" (("B12D" " " NIL ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))))))
; ("Z96"
;  (("Z96B"
;     (("Z96B" " " NIL
;       ((((:COMBINATIONS-RESTRICTIONS-1
;           ("I" "Z96B" "," "D10P" "," "D32B" "," "E30B" "F98A" "<" "B34J" "<" "B35C" "<" "B36C" "F98G")))
;         ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U"))
;         ((:BODY "G") (:PERFORMANCE "E") (:DRIVE "A") (:TRANSMISION "A")) ("D" "H"))))))))
;Command:
;("B15Y" (("B15Y" "D" (#<FEATURE-CHART-ENTRY 262257630> #<FEATURE-CHART-ENTRY 262257671>) NIL)))
;Command: (all-trims-common-with-RESTRICTIONS '("B15Y" (("B15Y" "D" # NIL))) boula)
;(("B12A" (("D" (#<FEATURE-CHART-ENTRY 262261057> #<FEATURE-CHART-ENTRY 262261163>))))
; ("B15Y" (("D" (#<FEATURE-CHART-ENTRY 262257630> #<FEATURE-CHART-ENTRY 262257671>))))))
;Command: (all-trims-common-with-RESTRICTIONS '("Z96B" (("Z96B" " " NIL #))) mas)
;(("B15J"
;  (("D"
;     ((NIL NIL (" " NIL) ((:BODY "G")))
;      (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;       #<FEATURE-CHART-ENTRY 262257014>)))
;    ("H" ((NIL NIL (" " NIL) ((:BODY "G"))) (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 26225
7157>)))))
; ("Z96B"
;  (("H"
;     (((:COMBINATIONS-RESTRICTIONS-1
;        ("I" "Z96B" "," "D10P" "," "D32B" "," "E30B" "F98A" "<" "B34J" "<" "B35C" "<" "B36C" "F98G")))))
;    ("D"
;     (((:COMBINATIONS-RESTRICTIONS-1
;        ("I" "Z96B" "," "D10P" "," "D32B" "," "E30B" "F98A" "<" "B34J" "<" "B35C" "<" "B36C" "F98G")))))
))
;Command:



;*****************************************************************************************


;Command: (loop for i in *grouped* do (let ((j (car (reverse i)))) (loop for k in j do (print (all-tri
ms-common-wit
```

```
;b-RESTRICTIONS mas k)))))
;
;
;(("B11C" (("H" (#<FEATURE-CHART-ENTRY 262256126> #<FEATURE-CHART-ENTRY 262256065>))))
; ("B15J"
;  (("H" ((NIL NIL (" " NIL) ((:BODY "G"))) (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 26225
7157>))))))
;(("B11B" (("H" (#<FEATURE-CHART-ENTRY 262256547>)) ("D" (#<FEATURE-CHART-ENTRY 262256404>))))
; ("B15J"
;  (("D"
;    ((NIL NIL (" " NIL) ((:BODY "G")))
;     (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;      #<FEATURE-CHART-ENTRY 262257014>)))
;   ("H" ((NIL NIL (" " NIL) ((:BODY "G"))) (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 26225
7157>))))))
;(("B45B" (("H" (#<FEATURE-CHART-ENTRY 262250661> #<FEATURE-CHART-ENTRY 262252376>))))
; ("B15J"
;  (("H" ((NIL NIL (" " NIL) ((:BODY "G"))) (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 26225
7157>))))))
;(("B15X" (("H" (#<FEATURE-CHART-ENTRY 262257465>))))
; ("B15J"
;  (("H" ((NIL NIL (" " NIL) ((:BODY "G"))) (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 26225
7157>))))))
;(("B15K" (("D" (#<FEATURE-CHART-ENTRY 262257220> #<FEATURE-CHART-ENTRY 262257261>))))
; ("B15J"
;  (("D"
;    ((NIL NIL (" " NIL) ((:BODY "G")))
;     (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;      #<FEATURE-CHART-ENTRY 262257014>))))))
;(("B15J"
;  (("D"
;    ((NIL NIL (" " NIL) ((:BODY "G")))
;     (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;      #<FEATURE-CHART-ENTRY 262257014>)))
;   ("H" ((NIL NIL (" " NIL) ((:BODY "G"))) (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 26225
7157>)))))
; ("B15J"
;  (("D"
;    ((NIL NIL (" " NIL) ((:BODY "G")))
;     (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;      #<FEATURE-CHART-ENTRY 262257014>)))
;   ("H" ((NIL NIL (" " NIL) ((:BODY "G"))) (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 26225
7157>))))))
;(("B15Y" (("D" (#<FEATURE-CHART-ENTRY 262257630> #<FEATURE-CHART-ENTRY 262257671>))))
; ("B15J"
;  (("D"
;    ((NIL NIL (" " NIL) ((:BODY "G")))
;     (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;      #<FEATURE-CHART-ENTRY 262257014>))))))
;NIL
;(("B02B" (("H" (NIL)) ("D" (NIL))))
; ("B15J"
;  (("D"
;    ((NIL NIL (" " NIL) ((:BODY "G")))
;     (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;      #<FEATURE-CHART-ENTRY 262257014>)))
;   ("H" ((NIL NIL (" " NIL) ((:BODY "G"))) (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 26225
7157>))))))
;(("C16V" (("H" (#<FEATURE-CHART-ENTRY 262261016>))))
; ("B15J"
;  (("H" ((NIL NIL (" " NIL) ((:BODY "G"))) (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 26225
7157>))))))
;(("B12A"
;  (("H" (#<FEATURE-CHART-ENTRY 262261267> #<FEATURE-CHART-ENTRY 262261371>))
;   ("D" (#<FEATURE-CHART-ENTRY 262261057> #<FEATURE-CHART-ENTRY 262261163>))))
; ("B15J"
;  (("D"
;    ((NIL NIL (" " NIL) ((:BODY "G")))
;     (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;      #<FEATURE-CHART-ENTRY 262257014>)))
;   ("H" ((NIL NIL (" " NIL) ((:BODY "G"))) (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 26225
7157>))))))
;(("B12C" (("H" (#<FEATURE-CHART-ENTRY 262261636>))))
; ("B15J"
;  (("H" ((NIL NIL (" " NIL) ((:BODY "G"))) (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 26225
7157>))))))
```

```
;(("B12D" (("H" (NIL)) ("D" (NIL))))
; ("B15J"
;  (("D"
;    ((NIL NIL (" " NIL) ((:BODY "G")))
;     (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;      #<FEATURE-CHART-ENTRY 262257014>)))
;   ("H" ((NIL NIL (" " NIL) ((:BODY "G"))) (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 26225
7157>))))))
;(("Z96B"
;  (("H"
;    (((:COMBINATIONS-RESTRICTIONS-1
;       ("I" "Z96B" "," "D10P" "," "D32B" "," "E30B" "F98A" "<" "B34J" "<" "B35C" "<" "B36C" "F98G")))))
;   ("D"
;    (((:COMBINATIONS-RESTRICTIONS-1
;       ("I" "Z96B" "," "D10P" "," "D32B" "," "E30B" "F98A" "<" "B34J" "<" "B35C" "<" "B36C" "F98G"))))))
)
; ("B15J"
;  (("D"
;    ((NIL NIL (" " NIL) ((:BODY "G")))
;     (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 262256753>
;      #<FEATURE-CHART-ENTRY 262257014>)))
;   ("H" ((NIL NIL (" " NIL) ((:BODY "G"))) (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 26225
7157>))))))
;NIL
;Command:
;
```

```
(defun create-the-logic-references (reference-of-ALL-FEATURE)
```

```
;("B15J"
;  (("H" (#<FEATURE-CHART-ENTRY 263730253> #<FEATURE-CHART-ENTRY 263731255>))
;   ("D"
;    (#<FEATURE-CHART-ENTRY 263730212> #<FEATURE-CHART-ENTRY 263731112> #<FEATURE-CHART-ENTRY 263731153>
;     #<FEATURE-CHART-ENTRY 263731214>))
;   ("D" NIL NIL (" " NIL) ((:BODY "G"))) ("H" NIL NIL (" " NIL) ((:BODY "G"))))))
;Command;
;
;
;       O  R :
;
;
; ("B02B"
;  (("D"
;    ((:COMBINATIONS-RESTRICTIONS-1 ("P" "H50K" "=" "H50P")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "H50K" "=
" "H50P")))
;    ((:TERRITORY-RESTRICTIONS-1 ("M20E" "+T53R"))) (" " NIL) NIL)
;   ("H"
;    ((:COMBINATIONS-RESTRICTIONS-1 ("P" "H50K" "=" "H50P")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "H50K" "=
" "H50P")))
;    ((:TERRITORY-RESTRICTIONS-1 ("M20E" "+T53R"))) (" " NIL) NIL)))
;
;  OR
;;
;
;Command:(all-trims-common-with-RESTRICTIONS mas new-sub))
;(("B02B"
;  (("D"
;    ((:COMBINATIONS-RESTRICTIONS-1 ("P" "H50K" "=" "H50P")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "H50K" "=
" "H50P")))
;    ((:TERRITORY-RESTRICTIONS-1 ("M20E" "+T53R"))) ("+" ("M32*")) NIL)
;   ("H"
;    ((:COMBINATIONS-RESTRICTIONS-1 ("P" "H50K" "=" "H50P")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "H50K" "=
" "H50P")))
;    ((:TERRITORY-RESTRICTIONS-1 ("M20E" "+T53R"))) ("+" ("M32*")) NIL)))
; ("B15J"
;  (("H" (#<FEATURE-CHART-ENTRY 263730253> #<FEATURE-CHART-ENTRY 263731255>))
;   ("D"
```

```
;     (#<FEATURE-CHART-ENTRY 263730212> #<FEATURE-CHART-ENTRY 263731112> #<FEATURE-CHART-ENTRY 263731153>
;      #<FEATURE-CHART-ENTRY 263731214>))
;    ("D" NIL NIL (" " NIL) ((:BODY "G"))) ("H" NIL NIL (" " NIL) ((:BODY "G"))))))
;Command:


  (let ((result nil) (flag nil)
        (TRIMS-and-RESTRICTIONS (car (reverse reference-of-ALL-FEATURE))))

      (loop  for    restrictions-OR-restriction   in   TRIMS-and-RESTRICTIONS
             do
        (let
             ((trim-in-search (car restrictions-OR-restriction))
              ;;;WE DON'T  KNOW IF WE HAVE only one restriction of an IMPLICIT
              ;;;OR  a sequence of RESTRICTIONS of an EXPLICIT... i e.  entries
              (reference-of-RESTRICTIONS-or-RESTRICTION
                                 (cadr  restrictions-OR-restriction))
             )
          (cond
             ((every #'instancep reference-of-RESTRICTIONS-or-RESTRICTION)
                 (cond
                      ((not (equal nil reference-of-RESTRICTIONS-or-RESTRICTION))
                           (push
                             (analyze-all-meanings reference-of-RESTRICTIONS-or-RESTRICTION
                                                   trim-in-search t)
                             result)
                           (setq flag t)
                      )
                      (t (push
                           (analyze-all-meanings (cdr RESTRICTIONS-or-RESTRICTION)
                                                 trim-in-search nil)
                               result)
                      )
                 )
             )
             (t (push
                  (analyze-all-meanings (cdr RESTRICTIONS-or-RESTRICTION)
                                                 trim-in-search nil)
                               result)
             )
          )
;         (print restrictions-OR-restriction)  (print reference-of-RESTRICTIONS-or-RESTRICTION); (print (
cdr RESTRICTIONS-or-RESTRICTION))
;         (print '*****************************************************************)
      )));;;WE MAKE only ONE list OF the RESULT

     (one-list-only-clear   result)
))


(defun analyze-all-meanings (list-of-restrictions   trim  flag)
  (let ((result nil))
   (cond (flag
          (loop for entry in list-of-restrictions
                do
          (push   (what-really-means  ENTRY) result))
          )
         (t  (push (do-the-ANALYSIS-for-IMPLICITS list-of-restrictions trim )
                                result)
      ))
    result
))
```

```
(defun what-really-means (entry)
  (list
    (Feature-chart-entry-class entry)
    (Feature-chart-entry-body entry)
    (Feature-chart-entry-family entry)
    (Feature-chart-entry-capacity entry)
    (Feature-chart-entry-performance entry)
    (Feature-chart-entry-drive entry)
    (Feature-chart-entry-trans entry)


    (Feature-chart-entry-all-the-combinations-restrictions entry)
    (Feature-chart-entry-all-the-territory-restrictions entry)
    (Feature-chart-entry-all-the-group-restrictions entry)
  )
)



(defun do-the-ANALYSIS-for-IMPLICITS (list-of-RESTRICTIONS  trim-in-search)

;;ie
;     ( ((:COMBINATIONS-RESTRICTIONS-1 ("P" "H50K" "=" "H50P")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "H50K"
 "=" "H50P")))
;;    ((:TERRITORY-RESTRICTIONS-1 ("M20E" "+T53R"))) (" " NIL) NIL)))
;
;        O   R
;
;
;        ;;;;;i e.... (NIL NIL (" " NIL) ((:BODY "G")))  ,   "H"
;
;    ((:COMBINATIONS-RESTRICTIONS-1 ("P" "H50K" "=" "H50P")) (:COMBINATIONS-RESTRICTIONS-2 ("R"; "H50K" "
=" "H50P")))
;    ((:TERRITORY-RESTRICTIONS-1 ("M20E" "+T53R"))) ("+" ("M32*")) NIL)


  (let ((result nil)
        (result0 (car (reverse list-of-RESTRICTIONS))))
    (let* (
          (result1
               (base-features-analysis result0  trim-in-search))
          ;;;; (result2 (reverse result1))
          (result3 (car list-of-RESTRICTIONS))   ;;;;;; COMBINATIONS
          (result4 (cadr list-of-RESTRICTIONS))  ;;;;;;   COUNTRIES ----  restricted
          (result5 (caddr list-of-RESTRICTIONS))) ;;;;;   GROUPR ------restricted

      ;(print result0)
      ;(print result1)
      ;;;;;;;;;;;;(print result2)
     ;;;;;;;;;;;(print result3)

     ;;;;;;(print result4) (print result5)
      (setq result
        (append result1  (list result3) (list result4)    (list result5)))
        ) result))




(defvar *sequence* nil)



(defun initialise-sequence ()
  (loop for i in *sequence*
        collect
        (list (car i)
              (setf (cadr i) nil)))
)
```

```
;;;;;;;;   SOMETHING WRONG   with THE    COMPILATION.

(defun base-features-analysis (list-of-lists-of-keywords-and-values   trim-in-search)
    ;;;;;ie
   ;;;;;; ((:BODY "G"))

(initialise-sequence)
  (let* ((result nil)
         (help *sequence*)
         )

           (loop for feature-list in list-of-lists-of-keywords-and-values
                 do
             (push (cadr feature-list)
                   (cadr (assoc (car feature-list) help :test #'equal))))

           (loop for i in help  do
             (cond
               ((equal (cadr i) nil) (push " " result))
               (t (push (car (cadr i)) result)))
             finally (setq help nil))


            (append  (list trim-in-search) (reverse  result))
))




(defun one-list-only-clear (ls)
  (cond ((null ls) nil)
        ((list (car ls))
           (cond ((atom (caar ls))
                  (cons (car ls)
                                (one-list-only-clear (cdr ls))))
                 (t
                  (append (one-list-only-clear (car ls))
                   (one-list-only-clear (cdr ls))))
           )
        )
))

;*****************************************************************************************
;*****************************************************************************************
;*****************************************************************************************


(defun get-the-implicit-TRIMS (header)

    (loop for i in *IMPLICIT-TRIM-LEVELS*
          do
      (cond ((string-equal header (substring  (car i)  0 3))
             (return (car (reverse i)))))
))

(defun vague-levels (header)
 (let  ((result nil) (table (take-the-table header )))
  (loop for key in (keys table)
        do
    (loop for entry in (gethash key table)
        do
    (push  (car (reverse
                  (return-TRIM-LEVELS-top

                    (remove 'nil
                    (list
                      (cond ((not (string-equal
                                    (Feature-chart-entry-class entry)  " "))
                             (list ':class (Feature-chart-entry-class entry))))
                      (cond ((not (string-equal
                                    (Feature-chart-entry-body entry)  " "))
```

```
                                (list ':body (Feature-chart-entry-body entry))))
                        (cond ((not (string-equal
                                     (Feature-chart-entry-family entry) " "))
                                (list ':family (Feature-chart-entry-family entry))))
                        (cond ((not (string-equal
                                     (Feature-chart-entry-capacity  entry) " "))
                                (list ':capacity (Feature-chart-entry-capacity entry))))
                        (cond ((not (string-equal
                                     (Feature-chart-entry-performance entry) " "))
                                (list ':performance (Feature-chart-entry-performance entry))))
                        (cond ((not (string-equal
                                     (Feature-chart-entry-drive entry) " "))
                                (list ':drive (Feature-chart-entry-drive entry))))
                        (cond ((not (string-equal
                                     (Feature-chart-entry-trans entry) " "))
                                (list ':transmision (Feature-chart-entry-trans entry))))

                    )))))  result)
   )) (remove-duplicates (clear result) :test #'string-equal)
))



(defun clear (ls)
  (cond ((null ls) nil)
        ((atom (car ls)) (cons (car ls)
                               (clear (cdr ls))))
        (t (append (clear (car ls))
                   (clear (cdr ls))))
))



;(defun give-the-PLUS-or-MINUS (ls)          ; (numbering-the-feat-GROUPS)
;  (let ((result nil))
;    (loop for list-of-GROUP-and-number-of-its-keys  in   ls
;         do
;      (cond ((>= (cadr list-of-GROUP-and-number-of-its-keys)  2)
;                (push (append list-of-GROUP-and-number-of-its-keys
;                              (list '+))
;                      result)
;            )
;           (t (let* (  ;;;;;;BECAUSE WE DON'T WANT TO MISS THE IMPLICIT TRIMS.
;
;                      (vague-trim-levels (remove 'nil
;                              (vague-levels
;                                      (car list-of-GROUP-and-number-of-its-keys))))
;
;                      (trim-levels-before
;                              (remove " "
;                                      (remove-duplicates
;                                    (loop for entry in
;
;                                            (gethash (car list-of-GROUP-and-number-of-its-keys)
;                                            "Feature-chart-entry-by-FeatureDescription-table")
;                                        collect
;                                          (Feature-chart-entry-CLASS entry))
;                                    :test #'string-equal)
;                                    :test #'string-equal)
;
;                      )
;                     (trim-levels-after (remove " "
;                                             (remove-duplicates
;                                          (append
;                                            (get-the-implicit-TRIMS
;                                              (car list-of-GROUP-and-number-of-its-keys))
;                                            trim-levels-before
;                                            vague-trim-levels) ;;;;APPEND
;                                    :test #'string-equal)
;                                             :test #'string-equal)
;
;                     )
;
;
```

```
;                    (standard-availabilities
;                              (loop for entry in
;                                         (gethash (car list-of-GROUP-and-number-of-its-keys)
;                                           *Feature-chart-entry-by-FeatureDescription-table*)
;                                       collect
;                                           (Feature-chart-entry-standard  entry)))
;                    (key (car (keys
;                          (take-the-table (car list-of-GROUP-and-number-of-its-keys)))))
;                   )
;
;                     ;;;IF (- - - - - - -) and "s" ->  IT's   +
;                     ;;;; ELSE give The DECISION to the  ENGINEER !
;
;                   ;;;;;WE WANT AS WELL TO AVOID A SITUATION LIKE:
;                   ;;;;;   (- - - - - - -)
;                   ;;;;;   (- g f - - - -)
;                   ;;;;;   (- - - - y - -)
;
;                ;(print 'the-key-is) (prin1 key)
;                ;(print 'the-group-is)(prin1 (car list-of-GROUP-and-number-of-its-keys))
;                ;(print 'the-trim-levels-before-are)(prin1 trim-levels-before)
;                ;(print 'the-trim-levels-after-are) (prin1 trim-levels-after)
;                ;(print 'the-vague-trim-levels-are) (prin1 vague-trim-levels)
;                ;(terpri) (terpri) (terpri) (terpri)
;
;                (cond ((equal  trim-levels-before nil)
;                         (cond ((equal (length trim-levels-after)
;                                       (length *all-the-trim-levels*))
;
;
;;(accept '(alist-member :alist ,*models*) :prompt "Model"
;;                   :stream stream)
;
;
;                              (cond ((all-are-standard standard-availabilities)
;                                     (push (append list-of-GROUP-and-number-of-its-keys
;                                                   (list '+)) result))
;                                  (t (cond ((menu key *standard-output*)
;                                            (dw:menu-choose '(("Yes" . T) ("No" . nil)) :prompt (forma
t nil "When the Car  DOES NOT get the -A  DOES it get something else?" key
;;) ))
;
;                                  (push (append list-of-GROUP-and-number-of-its-keys ;
;                                    (list '-)) result))
;                                       (t (push (append list-of-GROUP-and-number-of-its-keys
;                                    (list '+)) result)))
;                                 ) ;;;t
;                             ))  ;;;;;;;cond  AND "("
;                           (t (push (append list-of-GROUP-and-number-of-its-keys
;                                            (list '-)) result)
;                            )  ;;;t
;                        ))
;
;
;                    (t
;                         ;;;IF IT COVERS ALL trim levels AND standard  ie. "s" ->  +
;                         ;;;; else IF IT COVERS ALL trim levels AND NOT all standard
;                         ;;;;;  give The DECISION to the  ENGINEER !
;
;
;
;                    (cond ((equal (length *all-the-TRIM-LEVELS*)
;                                  (length trim-levels-after))
;
;                           (cond ((not (all-are-standard standard-availabilities))
;
;
;
;                                  (cond ((menu key *standard-output*)
;;                             (values (dw:menu-choose '(("Yes" . T) ("No" . nil)) :prompt (zl:format nil
"When the Car  DOES NOT get the  -A  DOES it get something else?" key
;;     ) ))
;
;
;                                   (push (append list-of-GROUP-and-number-of-its-keys ;
;                                     (list '-)) result))
;                                        (t (push (append list-of-GROUP-and-number-of-its-keys
```

```
;                                        (list '+)) result)))
;                                     )
;                                    (t (push (append list-of-GROUP-and-number-of-its-keys
;                                        (list '+)) result)
;                                    )
;                                 )
;                              )
;
;                              ;;;;;;  ELSE WILL BE +   AND  -
;
;                              (t (push (append list-of-GROUP-and-number-of-its-keys
;                                        (list '-)) result)
;                               ) ;;;t
;                          ))  ;;;COND   ;;;t
;                    )
;                    )      ;;;;LET
;               );;;;;;;  t
;          )
;       )
;            result)  ;;;   LET
;)
```

```
(defun menu (the-key-string   &optional (stream *standard-output*))
  (values
    (dw:accepting-values (stream :own-window t :label  "ENGINEER'S    ANSWER")
      (accept '(alist-member :alist (("yes". t) ("No" . nil)))
                              :prompt (format stream "When the Car  DOES NOT get the -A  DOES it get somethi
ng else?" the-key-string)
                              :default 't     ;;;;THE default SHOULD BE the actual VALUES.
                              :stream stream)))
)
```

```
(defun all-are-standard (ls-of-standard-availabilities)
  (let ((flag t))
    (loop for standard-value  in ls-of-standard-availabilities
          do
      (cond
        ((or
           (not (string-equal
                  standard-value  "s"))
           (not (string-equal  standard-value ""))
          )
                            (setq  flag nil)
        )
      ))
    flag)
)
```

```
(defun take-the-table (the-header)
  (loop for i in  *subdescription-tables*
        do
    (cond
      ((string-equal the-header  (car i))
       (return  (cdr i))))))
```

```
;*****************************************************************************************

(defun numbering-the-feat-GROUPS  ()
  (let ((result nil))
    (loop for (description-key . subdescription-table) in *SUBDESCRIPTION-TABLES*
          do
      (push (list description-key (length (keys subdescription-table)))
            result))
  result)
)



;Command: (the-TEST-function '("B15J" (("B15J" "D" # #) ("B15J" "H" # #))) '("B12A" (("B12A" "D" # NIL) (
"B12A" "H" #
;NIL) ("B12A" "M" # NIL) ("B12A" "Y" # NIL))))
;
;((("H" "N" " " " " " " " " " ")
;  ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T5
3G" "T54C"
;   "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T5
9E" "T59G"
;   "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T5
4A" "T55R"
;   "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T5
3A" "T54H"
;   "T54T" "T56U" "T58N" "T57J" "T58J")
;  NIL)
; (("H" "W" " " " " " " " " " " ")
;  ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T5
3G" "T54C"
;   "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T5
9E" "T59G"
;   "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T5
4A" "T55R"
;   "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T5
3A" "T54H"
;   "T54T" "T56U" "T58N" "T57J" "T58J")
;  NIL)
; (("D" "N" " " " " " " " " " " " ") ("T60A") (("R" "W57A" "=" "Z96B" "=" "R00E") ("P" "R00E")))
; (("D" "W" " " " " " " " " " " ") ("T60A") (("R" "W57A" "=" "Z96B"))) (("D" "N" " " " " " " " " " ") ("T53
R") NIL)
; (("D" "W" " " " " " " " " " ") ("T53R") NIL))
;
;*****************************************************************************************
;*****************************************************************************************
;*****************************************************************************************
;*****************************************************************************************
;*****************************************************************************************


(defvar *COMMON-GROUND-RECORD* nil)
(defvar *help* nil)

;(defvar *sequence* ' ((:body ()) (:family ()) (:capacity ()) (:performance ()) (:drive ()) (:t;ransmisio
n ())))
;(initialise-sequence)


(defun run   (MASTER-FEATURE  SUBFEATURE)    ;;;;;;  "b15"  "b12"
   (let ((record-help  *COMMON-GROUND-RECORD*))
   (cond ((equal record-help nil) (start MASTER-FEATURE SUBFEATURE))
         (t
          (let* (


                 (help-variable
                   (one-list-only-clear (start-after MASTER-FEATURE SUBFEATURE)))

                 (help-variable-AFTER
                   (remove "nothing" help-variable :test #'string-equal :key #'car))
                 (NOTHINGS-and-NO_NOTHINGS
                          (let ((final-one  nil)
```

```
                                    (final-two  nil))
                                      (loop for i in help-variable
                                            do
                                        (cond ((string-equal (car i) "NOTHING")
                                                 (push (cadr i) final-one))
                                              (t (push (cadr i) final-two))
                                        )
                                      )
                                      (list (remove-duplicates final-one :test #'string-equal)
                                            (remove-duplicates final-two :test #'string-equal))
                                    )
                  )
                  (NOTHINGS (car NOTHINGS-and-NO_NOTHINGS))
                  (NO_NOTHINGS (cadr NOTHINGS-and-NO_NOTHINGS))
                )


            ;(print NOTHINGS)
            ;(print NO_NOTHINGS)
            ;(print help-variable-AFTER)

        (setq
                 *COMMON-GROUND-RECORD*
               (EXPAND-THE-PIMS-STATEMENT help-variable-AFTER NOTHINGS  NO_NOTHINGS))
        (let ((NOT-PASSED (check-which-not-PASSED)))  ;;;; BECAUSE here the variable
                                        ;;;;;  *record-of-PASSED* get its value.
                                        ;;;;; AFTER the function (analyze-again) run --->
                                        ;;;;;; AFTER the function EXPAND-THE-PIMS-STATEMENT.
          ;;;;;(print NOT-PASSED)

          (backup  NOT-PASSED record-help)
        ))
      )  ;;;;; t
   ))
)


;           ;(setq *COMMON-GROUND-RECORD* result)
;;          (cond
;;            (result (setq *COMMON-GROUND-RECORD* result))
;;            (t (format nil "THE feature ~s DOES NOT MATCH ITS RESTRICTIONS WITH THE ALREADY valid REQUI
REMENTS OF THE FEATURES SUBFEATURE")
;;                (setq *COMMON-GROUND-RECORD* record-help  )))
;
;)        )))))result
;))



;*****************************************************************************************

(defun check-which-NOT-passed ()
  (let ((record nil))
    (loop for i in (remove-duplicates *record-of-PASSED* :test #'equal)
          do
      (let ((main-feature (car i))
            (other-features (cadr i)))
        (loop for j in (remove-duplicates *help* :test #'equal)
              do
          (let
            ((main-feature-other (car j))
             (other-features-other  (cadr j)))
            ;(print main-feature) (print other-features) (print '&&&&&)(print main-feature-other)
            ;(print other-features-other) (print '******)
            (cond
              ((string-equal main-feature main-feature-other)
                (cond
                  ((search   other-features-other  other-features  :test #'string-equal)
                     (cond ((not (member  j  record :test #'equal))
                              (push j record))
                     )
                  )
                )
              ))))))
```

```
  (set-difference *help* record :test #'equal))
 )




 (defun   backup (list-of-no-pasted  common-ground-record-BEFORE)  ;;;;;; chech-which-NOT-passed
;;;;;;Command: (check-which-NOT-passed )
;;;;;;(("B15Y" "B12A") ("B15J" "B12D") ("B15K" "B12A"))
  (let ((result nil))
   (loop for i in  list-of-no-pasted
         do
     (let ((main-feature (car i))
           (other-feature (cadr i)))
       (loop for j in common-ground-record-BEFORE
             do
         (let ((main-feature-other (cadr j))
               (other-combination (caddr j)))
           (cond
             ((and (string-equal main-feature main-feature-other)
                   (string-equal other-feature other-combination)
                   (not (string-equal (car j) "nothing"))
              )
              (push j result)))
        ))))
;;;;;  PUT the  lost ones........    the MISSING  information
   (setq *common-ground-record*
         (remove-duplicates
         (append result *common-ground-record* )  :test #'equal))
;;;TO  LOOP  Again  and  Again.     WE WANT TO PUT IN THE *help* the new ones, as WELL.

   (setq *help* (append  list-of-no-pasted   *record-of-PASSED*))
   (setq *record-of-PASSED* nil)

   )
 )


;*****************************************************************************************

(defun EXPAND-THE-PIMS-STATEMENT (common-ground  NOTHINGS   NO_NOTHINGS)


;  Command: (run "b15" "b02")    ============   common-ground
;(("PLUS-AND-MINUS" "B15P" "B02B"
;  (((("M" "G" " " " " " " " " " " " ")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;    (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))))
; ("NOTHING" "B15Y" "B02B" "NOTHING")
; ("PLUS-AND-MINUS" "B15J" "B02B"
;  (((("D" "G" " " " " " " " " " " " ")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;    (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))
;   ((("H" "G" " " " " " " " " " " " ")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;    (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))))
; ("NOTHING" "B15K" "B02B" "NOTHING")
; ("PLUS-AND-MINUS" "B15X" "B02B"
;  (((("M" "G" " " " " " " " " " " " "A")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;    (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P") ("R" "R05A" "=" "R05B") ("P" "R05A" "=" "R05B")))
;   ((("Y" "G" " " " " " " " " " " " ")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
```

```
;     (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))
;    (("X" "G" " " " " " " " " " " " ")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;     (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))
;    (("W" "G" " " " " " " " " " " " ")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;     (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P"))))))
;;
;
;
;("B15K" "B15Y")      ==========  NOTHINGS
;("B15X" "B15J" "B15P")   =============   NO_NOTHINGS
;



  (let* ((result nil) (result-top nil)
         (temporary-result  *COMMON-GROUND-RECORD*)
        )
         (loop for MASTER-FEATURE-header    in   NO_NOTHINGS
               do
            (let ((help (record-GROUPS MASTER-FEATURE-header NOTHINGS temporary-result)))
;;;;;;IT actually returns all entries in previous *COMMON-GROUND-RECORD* which are "b15j",
;;;;;;     NOt included in NOTHINGs,  NOT are  nils.

               (cond ((not (equal help nil))
                      (push (list
                              (record-GROUPS MASTER-FEATURE-header nil common-ground)
                              help
                      )
                            result)
                    )
            )
         )
        )

 (setq result-top (analyze-again result))
;;;;result-top
  )
)


(defun  record-GROUPS   (header     NOTHINGs   list-of-STATEMENTS)
  (let ((result nil))
       (loop for PIMS-STATEMENT   in   list-of-STATEMENTS
             do
         (cond ((string-equal (cadr PIMS-STATEMENT) header)
                 (cond ((not (string-equal (car PIMS-STATEMENT) "nothing"))
                         (cond ((or
                                 (not
                                 (member
                                   (cadr PIMS-STATEMENT) NOTHINGs :test #'string-equal))
                                 (not (katerina (cadr PIMS-STATEMENT) NOTHINGs))
                               )
                               (push PIMS-STATEMENT result))
                       )
                     )
                 )
         )
      ))
       result
  )
)
```

```
;;;;;  THe function  "katerina"  get as its arguments the
;;;;    a-string   =      "bo2b,b15f,z96r"
;;;;;   list-of-strings=NOTHINGS    '("b02b" "b35c" "z67y")
;;;;;;   IT IS NEEDED when we run the program the    SECOND   time.


(defun katerina   (a-string    list-of-strings=NOTHINGS)
  (let ((flag nil))
    (loop for i in    list-of-strings=NOTHINGS
          do
      (cond ((search i a-string :test #'string-equal)(setq flag t)  (return)))
    )
    flag
))

;;;;;; THE function "analyze-again"   gets as argument the result of
;;;;;    "(EXPAND-THE-PIMS-STATEMENT common-ground  NOTHINGS   NO_NOTHINGS)"
;;;;;         coming as result from the function "(run "b15""b02")     WHERE
;;;;;         the global variable     *COMMON-GROUND-RECORD*   has already a value ::::
;;;;;;;        ie .......   the result of (run "b15"     "b12")
;;;;;
;;;;;
;;;;;   THE   job   of the function "analyze-again" is to change the value of the
;;;;;    global variable     *COMMON-GROUND-RECORD*  to the NEW  valid  COMBINATIONS.
;;;;;
;;;;;
;;;;;
;;;;;  Any list inside the "(EXPAND-THE-PIMS-STATEMENT common-ground  NOTHINGS   NO_NOTHINGS)"
;;;;;  has a list like  "DESPOINA"...or ......ANNA
;;;;;
;;;;;
;;;;;
;;;;;




;
;((("PLUS-AND-MINUS" "B15P" "B02B"
;   (((("M" "G" " " " " " " " " " " ")
;      ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L"
"T59V"
;       "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;      (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))))))
; (("PLUS" "B15P" "B12A"
;   (((("M" " " " " " " " " " " " " " ")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
"T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
"T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
"T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
"T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      NIL)))
;  ("PLUS" "B15P" "B12C"
;   (((("M" " " " " " " " " "E" "E" "A")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
"T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
"T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
"T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
"T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      (("R" "R05A" "=" "R05B") ("P" "R05A" "=" "R05B")))))))



(defvar *record-of-PASSED*  nil)

(defun analyze-again   (ls)
  (let ((result  nil))
```

```
(loop for try-to-match-STATEMENTs    in   ls
     do
  (let* ((COMBINATION-subfeatures-we-try-to-match (car try-to-match-STATEMENTs))
         (takis-found-wrong (car (cdddr (car ;(car
                                                COMBINATION-subfeatures-we-try-to-match))))
         (already-valid-COMBINATIONS (cadr try-to-match-STATEMENTs))
         (plus-or-minus (caar COMBINATION-subfeatures-we-try-to-match))
         (main-feature (cadr (car COMBINATION-subfeatures-we-try-to-match)))
         (other-feature (caddr (car COMBINATION-subfeatures-we-try-to-match)))
        )

    (loop for COMBINATION-subfeature---we-try-to-match  in takis-found-wrong

          do
      (let (

             (BASE-FEATURES-of-feature-in-TRIAL
                   (car COMBINATION-subfeature---we-try-to-match))

             (COUNTRIES-of-feature-in-TRIAL
                     (cadr COMBINATION-subfeature---we-try-to-match))

             (COMBINATIONS-or-COMMENTS-of-feature-in-TRIAL
               (caddr COMBINATION-subfeature---we-try-to-match))

           )
    (loop for one-more   in   already-valid-COMBINATIONS
          do
      (let ((plus-or-minus-other (car one-more))
            (other-feature-next (caddr one-more))
            (rest-of-PIMS-specifications-other
                (car (cdddr one-more)))
           )
        ;(print one-more)
        ;(print already-valid-COMBINATIONS)
        ;(print '*************************************************************)

     (loop for already-valid-COMBINATION in   rest-of-PIMS-specifications-other
           do

        (let (

              (BASE-FEATURES-of-feature-EXIST
                (car already-valid-COMBINATION))
              (COUNTRIES-of-feature-EXIST (cadr already-valid-COMBINATION))
              (COMBINATIONS-or-COMMENTS-of-feature-EXIST
                (caddr already-valid-COMBINATION))
             )
;          (print one-more) (print '***********)
;          (print rest-of-PIMS-specifications-other)
;          (print BASE-FEATURES-of-feature-EXIST)
;          (print COUNTRIES-of-feature-EXIST)
;          (print COMBINATIONS-or-COMMENTS-of-feature-EXIST)


        (let ((base-feature-to-be-compared
                (reverse          ;;BECAUSE IT comes in OPPOSITE direction
                  (NO-opposite BASE-FEATURES-of-feature-in-TRIAL
                                                BASE-FEATURES-of-feature-EXIST)))
              (common-countries (intersection COUNTRIES-of-feature-in-TRIAL
                                              COUNTRIES-of-feature-EXIST
                                              :test #'string-equal))
              (COMMENTS-or-RESTRICTIONS-to-be-compared
                    (which-COVERS-which-TOP
                                  COMBINATIONS-or-COMMENTS-of-feature-in-trial
                                  COMBINATIONS-or-COMMENTS-of-feature-EXIST ))
              )
              (cond
                ((and base-feature-to-be-compared common-countries)

                    (cond
                      ((not (member (list
                                     (string-append plus-or-minus
                                                    ","
                                                    plus-or-minus-other)
                                     main-feature
```

```
                                        (string-append other-feature
                                                       ","
                                                       other-feature-next)
                                        (list (list
                                        base-feature-to-be-compared
                                        common-countries
                                        COMMENTS-or-RESTRICTIONS-to-be-compared))
                                       )
                                      result
                                      :test #'equal))
                              (push
                                (list
                                  (string-append plus-or-minus
                                                 ","
                                                 plus-or-minus-other)
                                  main-feature
                                  (string-append other-feature
                                                 ","
                                                 other-feature-next)
                                  (list
                                  (list  ;;;BECAUSE WE WANT TO GIVE THE SAME format WITH
                                         ;;;;*COMMON-GROUND-RECORD*   AS IT WAS before.
                                  base-feature-to-be-compared
                                  common-countries
                                  COMMENTS-or-RESTRICTIONS-to-be-compared)))
                                result)
                              (push (list main-feature
                                          (string-append plus-or-minus
                                                         ","
                                                         plus-or-minus-other)
                                          (string-append other-feature
                                                         ","
                                                         other-feature-next)
                                     )
                                     *record-of-PASSED*)
                            )
                        )    ;;;;;;;;   cond
                      )
                      ;;;;(t (push already-valid-COMBINATION  *help*))
                    )     ;;;;;;;; cond
                )))
))))))

      result
))

  ;*****************************************************************************************
```

```
;;;;;THE IMPORTANT  point here is that a single    "COMBINATION-subfeature---we-try-to-match"
;;;;;  generates  more than one statements to be PUSHed.    this HAPPENS because IT IS AT
;;;;   the top of the   LOOP in the FUNCTION (analyze-again).
;;;;; IN order to achieve the same format with the *COMMON-GROUND-RECORD* we have to do
;;;;  (list (list (..........
;;;; SEE  below  ,     "takis"  variable create 4 different lists.
```

```
;Command: '("PLUS" "B15P" "B12C" ((# # #)))  ============     TAKIS
;("PLUS" "B15P" "B12C"
; (((#"M" " " " " " " "E" " " "A")
;   ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T
53G" "T54C"
;    "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T
59E" "T59G"
;    "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T
54A" "T55R"
;    "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T
```

```
53A" "T54H"
;     "T54T" "T56U" "T58N" "T57J" "T58J")
;    (("R" "R05A" "=" "R05B") ("P" "R05A" "=" "R05B")))))
;Command: (setq *COMMON-GROUND-RECORD* (list '("PLUS" "B15P" "B12C" ((# # #)))))
;(("PLUS" "B15P" "B12C"
;   (("M" " " " " " " " "E" " " "A")
;     ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "
T53G" "T54C"
;      "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "
T59E" "T59G"
;      "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "
T54A" "T55R"
;      "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "
T53A" "T54H"
;      "T54T" "T56U" "T58N" "T57J" "T58J")
;     (("R" "R05A" "=" "R05B") ("P" "R05A" "=" "R05B")))))
;Command: (analyze-again (list despoina anna))
;(("PLUS-AND-MINUS,PLUS" "B15X" "B02B,B12C"
;   (("M" "G" " " " " " "E" "E" "A")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;     (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P") ("R" "R05A" "=" "R05B") ("P" "R05A" "=" "R05B")))))
; ("PLUS-AND-MINUS,PLUS" "B15X" "B02B,B12A"
;   (("M" "G" " " " " " " " " " "A")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;     (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))))
; ("PLUS-AND-MINUS,PLUS" "B15P" "B02B,B12C"
;   (("M" "G" " " " " " "E" "E" "A")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;     (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P") ("R" "R05A" "=" "R05B") ("P" "R05A" "=" "R05B")))))
; ("PLUS-AND-MINUS,PLUS" "B15P" "B02B,B12A"
;   (("M" "G" " " " " " " " " " " " ")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;     (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))))
;
;
;
;
;
;Command: setiris     IS  "(run  "b15"  "b02")
;;;;
;;;
;;;
;(("PLUS-AND-MINUS" "B15P" "B02B"
;   (("M" "G" " " " " " " " " " " " ")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;     (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))))
; ("NOTHING" "B15Y" "B02B" "NOTHING")
; ("PLUS-AND-MINUS" "B15J" "B02B"
;   (("D" "G" " " " " " " " " " " " ")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;     (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))
;    (("H" "G" " " " " " " " " " " " ")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;     (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))))
; ("NOTHING" "B15K" "B02B" "NOTHING")
; ("PLUS-AND-MINUS" "B15X" "B02B"
;   (("M" "G" " " " " " " " " " "A")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
```

```
;     (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P") ("R" "R05A" "=" "R05B") ("P" "R05A" "=" "R05B")))
;    (("Y" "G" " " " " " " " " " " ")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;     (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))
;    (("X" "G" " " " " " " " " " " ")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;     (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))
;    (("W" "G" " " " " " " " " " " ")
;     ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L" "
T59V" "T51X"
;      "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;     (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P"))))))
;
;
;
;
;Command: (EXPAND-THE-PIMS-STATEMENT  sotiris '("B15K" "B15Y") '("B15X" "B15J" "B15P"))
;                                                "negatives"  from  "(run "b15"  "B02")"
;                                                "NO-negatives" from "(run "b15"  "B02" )
;
;(
;  (
;  (("PLUS-AND-MINUS" "B15P" "B02B"
;    (((M" "G" " " " " " " " " " " ")
;      ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L"
 "T59V"
;       "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;      (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P"))))))
;
; (("PLUS" "B15P" "B12A"
;    (((M" " " " " " " " " " " " ")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
 "T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
 "T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
 "T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      NIL)))
;   ("PLUS" "B15P" "B12C"
;    (((M" " " " " " " " "E" "E" "A")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
 "T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
 "T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
 "T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      (("R" "R05A" "=" "R05B") ("P" "R05A" "=" "R05B")))))
; )
;
;
;
; (
;
;(("PLUS-AND-MINUS" "B15J" "B02B"
;    (((D" "G" " " " " " " " " " " ")
;      ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L"
 "T59V"
;       "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;      (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))
;    (((H" "G" " " " " " " " " " " ")
;      ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L"
 "T59V"
;       "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;      (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P"))))))
;
;  (("PLUS" "B15J" "B12A"
```

```
;    (((("H" "N" " " " " " " " " " " ")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
 "T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
 "T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
 "T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      NIL)
;     (("H" "W" " " " " " " " " " " ")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
 "T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
 "T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
 "T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      NIL)
;     (("D" "N" " " " " " " " " " " ") ("T60A") (("R" "W57A" "=" "Z96B" "=" "R00E") ("P" "R00E")))
;     (("D" "W" " " " " " " " " " " ") ("T60A") (("R" "W57A" "=" "Z96B"))) (("D" "N" " " " " " " " " " ") (
"T53R") NIL)
;     (("D" "W" " " " " " " " " " " ") ("T53R") NIL)))
;   ("PLUS" "B15J" "B12C"
;    (((("H" "N" " " " " " " " " " " ")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
 "T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
 "T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
 "T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))))
;   ("PLUS" "B15J" "B12D"
;    (((("D" "G" " " " " " " " " " " ")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
 "T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
 "T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
 "T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      NIL)
;     (("H" "G" " " " " " " " " " " ")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
 "T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
 "T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
 "T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      NIL)))))
; (((("PLUS-AND-MINUS" "B15X" "B02B"
;    ((("M" "G" " " " " " " " " " " "A")
;      ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L"
 "T59V"
;       "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;      (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P") ("R" "R05A" "=" "R05B") ("P" "R05A" "=" "R05B")))
;     (("Y" "G" " " " " " " " " " " ")
;      ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L"
 "T59V"
;       "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;      (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))
;     (("X" "G" " " " " " " " " " " ")
;      ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L"
 "T59V"
;       "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
```

```
;       (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))
;      (("W" "G" " " " " " " " " " " ")
;       ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L"
 "T59V"
;        "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;       (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P"))))))
;  (("PLUS" "B15X" "B12A"
;    (((("M" " " " " " " " " " " " " "A")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G"
;        "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
 "T57E"
;        "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
 "T55L"
;        "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
 "T58G"
;        "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;       (("R" "R05A" "=" "R05B") ("P" "R05A" "=" "R05B")))
;      (("X" " " " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G"
;        "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
 "T57E"
;        "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
 "T55L"
;        "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
 "T58G"
;        "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;       NIL)
;      (("H" "N" " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G"
;        "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
 "T57E"
;        "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
 "T55L"
;        "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
 "T58G"
;        "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;       (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))))
;   ("PLUS" "B15X" "B12C"
;    (((("M" " " " " " " " " "E" "E" "A")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G"
;        "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
 "T57E"
;        "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
 "T55L"
;        "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
 "T58G"
;        "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;       NIL)
;      (("H" "N" " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G"
;        "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
 "T57E"
;        "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
 "T55L"
;        "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
 "T58G"
;        "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;       NIL)
;      (("X" " " " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G"
;        "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
 "T57E"
;        "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
 "T55L"
;        "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
 "T58G"
;        "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;       NIL)
;      (("W" " " " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
```

```
 "T53G"
;        "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
 "T57E"
;        "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
 "T55L"
;        "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
 "T58G"
;        "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;       NIL)))))))
;
;
;Command: (setq despoina '((("PLUS-AND-MINUS" "B15P" "B02B" #)) (("PLUS" "B15P" "B12A" #) ("PLUS" "B15P"
"B12C" #))))
;IT IS THE CAR of the (EXPAND-THE-PIMS-STATEMENTS .........
;
;       I    WILL   CONCERN   ONLY   with  the variables     despoina ,   ANNA
;
;*****************************************************************************************
;
;((("PLUS-AND-MINUS" "B15P" "B02B"
;    (("M" "G" " " " " " " " " " " " ")
;      ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L"
"T59V"
;       "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;      (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))))))
; (("PLUS" "B15P" "B12A"
;    (("M" " " " " " " " " " " " " ")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
"T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
"T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
"T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
"T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      NIL)))
;  ("PLUS" "B15P" "B12C"
;    (("M" " " " " " " " " "E" "E" "A")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
"T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
"T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
"T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
"T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      (("R" "R05A" "=" "R05B") ("P" "R05A" "=" "R05B")))))))


;*****************************************************************************************

;Command: (setq ANNA '(((#) (# #)))
;((("PLUS-AND-MINUS" "B15X" "B02B"
;    (("M" "G" " " " " " " " " " " " " "A")
;      ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L"
"T59V"
;       "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;      (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P") ("R" "R05A" "=" "R05B") ("P" "R05A" "=" "R05B")))
;     (("Y" "G" " " " " " " " " " " " " ")
;      ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L"
"T59V"
;       "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;      (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))
;     (("X" "G" " " " " " " " " " " " ")
;      ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L"
"T59V"
;       "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;      (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))
;     (("W" "G" " " " " " " " " " " " ")
;      ("T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T53G" "T54C" "T56J" "T56V" "T57X" "T59L"
"T59V"
;       "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T53C")
;      (("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))))))
; (("PLUS" "B15X" "B12A"
```

```
;   (((("M" " " " " " " " " " " " " "A")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
"T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
"T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
"T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
"T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      (("R" "R05A" "=" "R05B") ("P" "R05A" "=" "R05B")))
;     (("Y" " " " " " " " " " " " " " " ")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
"T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
"T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
"T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
"T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      NIL)
;     (("H" "N" " " " " " " " " " " " ")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
"T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
"T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
"T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
"T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      ("R" "H50K" "=" "H50P") ("P" "H50K" "=" "H50P")))))
;  ("PLUS" "B15X" "B12C"
;   (((("M" " " " " " " " "E" "E" "A")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
"T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
"T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
"T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
"T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      NIL)
;     (("H" "N" " " " " " " " " " " " ")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
"T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
"T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
"T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
"T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      NIL)
;     (("X" " " " " " " " " " " " " " ")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
"T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
"T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
"T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
"T58G"
;       "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;      NIL)
;     (("W" " " " " " " " " " " " " " ")
;      ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
"T53G"
;       "T54C" "T56J" "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C"
"T57E"
;       "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C"
"T55L"
;       "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E"
"T58G"
```

```
;      "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;     NIL)))))
;Command:
;


;*****************************************************************************************
;*****************************************************************************************


(defun start (MASTER-FEATURE  SUBFEATURE)     ;;;;; ie  "B15"    "B12"
  (let
    (;;;;;;(result nil)
     (MASTER-FEATURE-REFERENCE (cadr (assoc MASTER-FEATURE *grouped* :test #'string-equal)))
     (SUBFEATURE-REFERENCE (cadr (assoc SUBFEATURE *grouped* :test #'string-equal)))
    )

    (loop for  MASTER-SUBFEATURE  in MASTER-FEATURE-REFERENCE
          do
      (loop for subfeature-of-SUBFEATURES in   SUBFEATURE-REFERENCE
          do

        (goal MASTER-SUBFEATURE subfeature-of-SUBFEATURES)
     ))
    ;;;;;result
))



(defun start-after (MASTER-FEATURE  SUBFEATURE)      ;;;;; ie  "B15"    "B12"
  (let ((result nil)
        (MASTER-FEATURE-REFERENCE (cadr (assoc MASTER-FEATURE *grouped* :test #'string-equal)))
        (SUBFEATURE-REFERENCE (cadr (assoc SUBFEATURE *grouped* :test #'string-equal)))
       )
    ;;;(print MASTER-FEATURE-REFERENCE) (print SUBFEATURE-REFERENCE)

   (loop for  MASTER-SUBFEATURE  in MASTER-FEATURE-REFERENCE
          do
      (loop for subfeature-of-SUBFEATURES in   SUBFEATURE-REFERENCE
          do
       (push
        (goal-ELSE MASTER-SUBFEATURE subfeature-of-SUBFEATURES) result)
     ))
result))




(defun goal-ELSE (Master-feature   Feature-from-group)
  (let ((result nil)
        (found-common (the-TEST-function Master-feature   Feature-from-group)))
    (cond (found-common

           (push  (list
                    (PLUS-or-MINUS   Master-feature  Feature-from-group)
                    (car Master-feature)
                    (car Feature-from-group)
                    found-common
                    ;;(translate-to-english-COUNTRIES found-common)
                   ) result))


          (t  (push (list "NOTHING" (car Master-feature) (car Feature-from-group) "NOTHING")
                    result)
          )
    )
result
))
```

```
(defun goal (Master-feature   Feature-from-group)
  (let (;;;(result nil)
        (found-common (the-TEST-function Master-feature   Feature-from-group)))
    (cond (found-common
           (let ((help (PLUS-or-MINUS   Master-feature  Feature-from-group)))
           (push  (list
                    help
                    (car Master-feature)
                    (car Feature-from-group)

                    found-common
                   )

                 *COMMON-GROUND-RECORD*)
           (push (list (car Master-feature)
                       (car Feature-from-group))
                       ;;;;;help)
                 *help*)
                       )
          )
          (t   (push (list "NOTHING" (car Master-feature) (car Feature-from-group) "NOTHING")
                 *COMMON-GROUND-RECORD*)
          )
    )
))



(defun PLUS-or-MINUS  (Master-feature  Feature-from-group);;; common-ground)
          ;;;; ie. .......
      ;;;;;("B15J"
;;    (("B15J" "D"
;;      (#<FEATURE-CHART-ENTRY 262256610> #<FEATURE-CHART-ENTRY 262256712> #<FEATURE-CHART-ENTRY 26225675
3>
;;       #<FEATURE-CHART-ENTRY 262257014>)
;;      ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))
;;     ("B15J" "H" (#<FEATURE-CHART-ENTRY 262257055> #<FEATURE-CHART-ENTRY 262257157>)
;;      ((NIL NIL (" " NIL) ((:BODY "G")) ("D" "H"))))))
;;;   ......................("b12A" ( ...........


  (let*  ((result nil)
          (record (numbering-the-feat-GROUPS))
          (a-string (car Master-feature))
          (another-string (car Feature-from-group))
          (Master-FEATCODE nil)
          (SUBFEATURE-featcode nil))


;       Command: (numbering-the-feat-GROUPS)
;       (("Z96" 1) ("B12" 3) ("C16" 1) ("B02" 1) ("B15" 5) ("B45" 1) ("C78" 1) ("B11" 2))

    (loop for i in record
          do
      (cond ((string-equal (substring a-string 0 3) (car i))
             (setq Master-FEATCODE (cadr i)))))
    (loop for i in record
          do
      (cond ((string-equal (substring another-string 0 3) (car i))
               (setq SUBFEATURE-featcode (cadr i)))))


;*************************************************************
(cond ((equal (length *grouped*) 1)
        (cond
            ;;;;CHECK THE POSSIBILITY OF ONLY ONE FEATURE GROUP Specification   !!!!
           ((equal (length (cadr (car *grouped*))) 1)
        (let ((result-other nil)  (help (cadr (car (cadr (car *GROUPED*))))) )
          ;;; I use "(LET....."  in order to find the    FEATURE DESCRIPTION
          (setq result-other
                (block feature-description
                  (loop for list in help
                        do
                    (loop for i in (third list)
                          do
                      (cond ((instancep i) (return-from feature-description
```

```
                                                (feature-chart-entry-FEATURE-DESCRIPTION i)))))
                 nil))
          (cond  ((values (menu-takis result-other)) (setq result "PLUS-and-MINUS"))
                 (t (setq result "PLUS")))

          (push (list  result-other
                       result) *record-of-MENUS-ASKED*))

       ) ;;;;
          (t (setq result "PLUS"))))
       ((and (> Master-FEATCODE 1)
             (> SUBFEATURE-featcode 1)) (setq result "PLUS"))
         ;;;WE put THE BELOW CONDITION because THE  "Feature-from-group" can BE  "+" at the
         ;;;  BEGINNING   ,but
         ;;;;  THE NEGATIVES we GAVE made it   " +  and - ".
         ((or
            (and (> Master-FEATCODE 1)
               (member (substring another-string 0 3)
                       *NEGATIVES-WHICH-WE-GAVE*
                       :test #'string-equal :key #'(lambda (i) (substring i 0 3))))
             ;;;;WE PUT THIS AS WELL because WHAT happen IF WE HAVE PUT A negative AND THEN
             ;;;;WE RUN THE PROGRAM for a SECOND time ?
             ;;;;IT   WILL THINK  THAT IS   -  .
              (and (> Master-FEATCODE 1)
                   (> (length
                        (let ((takis nil))
                         (loop for subfeature in *RECORD-OF-SUBFEATURES-before*

                               do
                           (cond ((string-equal (substring subfeature 0 3)
                                                (substring another-string 0 3))
                                  (setq takis (cons  subfeature takis))))) takis))
                      1)
             )

           )

                  (setq result "PLUS"))

         (t (setq result
              (check-the-PIMS-plus-or-MINUS Master-feature  Feature-from-group);; common-ground)
             )))
;*************************************************************
;    (cond ((and (> Master-FEATCODE 1)
;             (> SUBFEATURE-featcode 1)) (setq result "PLUS"))
;         ;;;WE put THE BELOW CONDITION because THE  "Feature-from-group" can BE  "+" at the
;         ;;;  BEGINNING   ,but
;         ;;;;  THE NEGATIVES we GAVE made it   " +  and - ".
;         ((and (> Master-FEATCODE 1)
;               (member (substring another-string 0 3)
;                       *NEGATIVES-WHICH-WE-GAVE*
;                       :test #'string-equal :key #'(lambda (i) (substring i 0 3))))
;                  (setq result "PLUS"))
;         (t (setq result
;              (check-the-PIMS-plus-or-MINUS Master-feature  Feature-from-group);; common-ground)
;             )))




    ))




(defun check-the-PIMS-plus-or-MINUS (Master-feature  Feature-from-group);; common-ground)
 ;;;;; we know : a-string   is the MASTER and it HAS more than 2.
 ;;;;;          the  another-string is the SUBFEATURE and it HAS  1
 (let ((result nil)
;       (common-trims (remove-duplicates
;                       (loop for i in common-ground
;                             collect
;                               (caar i))
;                       :test #'string-equal))
```

```
        (MASTER-feature-TRIMS (all-trims-of-FEATURE Master-feature))
        (SUBfeature-TRIMS (all-trims-of-FEATURE Feature-from-group)))
    ;;;;;;DOES    M*  wins  I*  ?
    (cond
      ((> (length MASTER-feature-TRIMS) (length SUBfeature-TRIMS))
          (setq result  "PLUS-AND-MINUS"))
      ;;;;;;;;; DOES  I*   wins   M*
      (t (let ((standard-availabilities
                 (loop for entry in
                           (gethash (substring (car Feature-from-group) 0 3)
                                    *Feature-chart-entry-by-FeatureDescription-table*)
                               collect
                                 (Feature-chart-entry-standard  entry))))
           (cond ((not (all-are-standard standard-availabilities))

                  (let ((key  (car (keys (take-the-table
                                           (substring (car Feature-from-group) 0 3)))))
                        )
                    (cond ((member key *record-of-MENUS-ASKED*
                                   :test #'string-equal :key #'first)
                           (setq result (cadr (assoc key
                                                     *record-of-MENUS-ASKED*
                                                     :test #'string-equal))))
                          (t
                     (cond (;;;;(menu key *standard-output*)
                            (values (menu-takis key))

              (setq result  "PLUS-AND-MINUS")
              (push (list  key
                           result) *record-of-MENUS-ASKED*)
                            )
                            (t (setq result "PLUS")
                               (push (list  key
                                            result)
                                     *record-of-MENUS-ASKED*))
                     )  ;;;;;cond
                          )   ;;;; t
                    )  ;;;;;;   COND
                   )   ;;;;let
              )
              (t
               (setq result "PLUS"))
          )  ;;;; COND
        )   ;;;;let
      )       ;;;;;;     t
    )
      ;;;;;;;   cond
  result)
)

(defvar *record-of-MENUS-ASKED* nil)

(defun menu-takis (key)
  (dw:menu-choose '(("Yes" . T) ("No" . nil)) :prompt (zl:format nil "When the Car  DOES NOT get the -A D
OES it get something else?" key
    ) :near-mode '(:point 100 700) )
)

;*****************************************************************************************
;*****************************************************************************************
;*****************************************************************************************



(defun the-TEST-function (Master-feature   Feature-from-group)
  (let ((result (all-trims-common-with-RESTRICTIONS Master-feature Feature-from-group)))
              (cond (result
                      ;;;;;;PUT FIRST THE MASTER feature
                     (cond ((string-equal (caar result) (car Master-feature))
                                          (TEST  result))
                           (t (TEST (reverse result))))))))

(defvar *record* nil)
```

```
(defun TEST   (list-of-two-features-to-be-compared)

;  (all-trims-common-with-RESTRICTIONS mas sub-other)
;(("B12A"
;  (("E" (#<FEATURE-CHART-ENTRY 263730663> #<FEATURE-CHART-ENTRY 263731420>))
;   ("D" (#<FEATURE-CHART-ENTRY 263730622> #<FEATURE-CHART-ENTRY 263731357>))))
; ("B15J"
;  (("E" (#<FEATURE-CHART-ENTRY 263730253> #<FEATURE-CHART-ENTRY 263731255>))
;   ("D"
;    (#<FEATURE-CHART-ENTRY 263730212> #<FEATURE-CHART-ENTRY 263731112> #<FEATURE-CHART-ENTRY 263731153>
;     #<FEATURE-CHART-ENTRY 263731214>)]
;   ("D" NIL NIL (" " NIL) ((:BODY "G"))) ("E" NIL NIL (" " NIL) ((:BODY "G"))))))
;Command
  (let ((record nil)) ;(takis (caar list-of-two-features-to-be-compared))
                      ;(katerina (car (cadr list-of-two-features-to-be-compared))))

    (let ((the-reference-of-MASTER-Feature
            (create-the-logic-references (car list-of-two-features-to-be-compared)))
          (the-reference-of-SubFeature
            (create-the-logic-references (cadr list-of-two-features-to-be-compared)))
        )
      (loop for MASTER-POSSIBILITY in the-reference-of-MASTER-Feature
            do
        (let* ((split-of-Master (part-apart-the-list MASTER-POSSIBILITY))
               (Master-Base-Features (car split-of-Master))
               (Master-Restrictions (cadr split-of-Master))
               (GROUPS-AVAILABLE-FOR-Master (GROUP-RESTRICTIONS-collect
                                             Master-Restrictions))
               (TERRITORIES-AVAILABLE-FOR-Master (TERRITORY-RESTRICTIONS-collect
                                                  Master-Restrictions))
               (COMBINATIONS-or-COMMENTS-of-Master
                 (COMBINATIONS-or-COMMENTS-RESTRICTIONS-collect Master-Restrictions))
               (final-the-COUNTRIES-AVAILABLE-for-MASTER
                  (set-difference
                           GROUPS-AVAILABLE-FOR-Master TERRITORIES-AVAILABLE-FOR-Master
                           :test #'string-equal))
               )
          ;(print GROUPS-AVAILABLE-FOR-Master) (print TERRITORIES-AVAILABLE-FOR-Master)
          ;(print COMBINATIONS-or-COMMENTS-of-Master)

          (loop for SUBFEATURE-POSSIBILITY in the-reference-of-SubFeature
                do
            (let* ((split-of-SUBFEATURE (part-apart-the-list SUBFEATURE-POSSIBILITY))
                   (Subfeature-Base-Features (car split-of-SUBFEATURE))
                   (Subfeature-Restrictions (cadr split-of-SUBFEATURE))
                   (GROUPS-AVAILABLE-FOR-Subfeature (GROUP-RESTRICTIONS-collect
                                                     Subfeature-Restrictions))
                   (TERRITORIES-AVAILABLE-FOR-Subfeature (TERRITORY-RESTRICTIONS-collect
                                                          Subfeature-Restrictions))
                   (COMBINATIONS-or-COMMENTS-of-Subfeature
                     (COMBINATIONS-or-COMMENTS-RESTRICTIONS-collect
                                      Subfeature-Restrictions))
                   (final-the-COUNTRIES-AVAILABLE-for-SUBFEATURE
                     (set-difference
                         GROUPS-AVAILABLE-FOR-Subfeature TERRITORIES-AVAILABLE-FOR-Subfeature
                         :test #'string-equal))
                 )
              ;(print GROUPS-AVAILABLE-FOR-Subfeature)
              ;(print TERRITORIES-AVAILABLE-FOR-Subfeature)
              ;(print final-the-COUNTRIES-AVAILABLE-for-SUBFEATURE)
              (let ((BASE-FEATURES-to-be-compared
                       (reverse       ;;BECAUSE IT comes in OPPOSITE direction
                         (NO-opposite  Master-Base-Features Subfeature-Base-Features)))
                    (RESTRICTIONS-to-be-compared
                       (intersection final-the-COUNTRIES-AVAILABLE-for-MASTER
                                     final-the-COUNTRIES-AVAILABLE-for-SUBFEATURE
                                     :test #'string-equal))
                    (COMMENTS-or-RESTRICTIONS-to-be-compared
                       (which-COVERS-which-TOP
                                       COMBINATIONS-or-COMMENTS-of-Subfeature
                                       COMBINATIONS-or-COMMENTS-of-Master ))

                    )
              ;(print BASE-FEATURES-to-be-compared)
              ;(print RESTRICTIONS-to-be-compared)
              ;(print COMMENTS-or-RESTRICTIONS-to-be-compared)
```

```
                ;(print '******************************)

                (cond
                  ((and BASE-FEATURES-to-be-compared   RESTRICTIONS-to-be-compared)
                    (cond
                       ((not  (member (list BASE-FEATURES-to-be-compared
                                            RESTRICTIONS-to-be-compared
                                            COMMENTS-or-RESTRICTIONS-to-be-compared)
                                      record :test #'equal))
                        (push
                              (list BASE-FEATURES-to-be-compared
                                    RESTRICTIONS-to-be-compared
                                    COMMENTS-or-RESTRICTIONS-to-be-compared)
                                                                  record)

                        ))
                    )
                );;;
              );;;    let
            )))))
record ))


;*****************************************************************************************
;*****************************************************************************************


(defun which-COVERS-which-TOP (ls1 ls2)
  (let ((result nil))
    ;;;;   TO AVOID the       COMMENTS
    (cond ((and (atom (car ls1))
                (not (equal (car ls1)  nil)))
           (setq result (append (list ls1)
                                (cond ((and (atom (car ls2))
                                            (not (equal (car ls2) nil))) (list ls2))
                                      (t ls1)))))
          ((and (atom (car ls2))
                (not (equal (car ls2) nil))) (setq result (append (list ls2)
                                                  (cond ((and
                                                            (atom (car ls1))
                                                            (not (equal (car ls1) nil)))
                                                               (list ls1))
                                                        (t ls2)))))
          (t

   (cond ((equal ls1 nil)
            (cond
              ((not (equal ls2 nil)) (setq result ls2))))
         ((equal ls2 nil)
            (cond
              ((not (equal ls1 nil)) (setq result ls1))))
         ((equal ls1 ls2) (setq result ls1)  )
                                ;;;;; BECAUSE the function  "which-COVERS-which" returns nil
                                ;;;;; if both of them are EQUAL...as   neither of them
                                ;;;;;;   COVERS   the other.

         (t
  (let ((record (append ls1 ls2)))
    (setq result
    (set-difference record (which-COVERS-which ls1 ls2) :test #'equal))
    ))
    )
      );;;;;;;;  tttttt
  )  ;;;;; cond
   result
))




;(defun which-COVERS-which (ls1 ls2)
;  ;;;ie.
;  ;;;ls1=     (("R" "W57A" "=" "Z96B"))
;  ;;;;ls2 =      (("R" "W57A" "=" "Z96B" "=" "R00E") ("P" "R00E"))
```

```
;   (let* ((result nil)
;        (combination1 (squeeze-to-string-TOP ls1))
;        (combination2 (squeeze-to-string-TOP ls2)))
;     (loop for i in combination1
;          do
;       (push i result))
;     (loop for j in combination2
;          do
;       (push j result))
;     (the-most-wide-COMBINATION
;                              (CREATE-COMBINATIONS result))
;   )
;)


;(defun the-most-wide-COMBINATION (ls)
;;   ie.
;;Command: (which-COVERS-which door window)
;;(((((("P" "R00E") "R00EP") (("R" "W57A" "=" "Z96B") "Z96B=W57AR"))
;;  ((("P" "R00E") "R00EP") (("R" "W57A" "=" "Z96B" "=" "R00E") "R00E=Z96B=W57AR")))
;; ((((("R" "W57A" "=" "Z96B" "=" "R00E") "R00E=Z96B=W57AR") (("R" "W57A" "=" "Z96B") "Z96B=W57AR"))))
;
;  (let ((record nil))
;    (loop for k in ls
;         do
;      (loop for i in k    ;;;;; BECAUSE the COMBINATIONS come IN  PAIRS.
;           do
;
;      (let ((first (car (reverse (car i))))
;            (second (car (reverse (cadr i)))))
;       ;;;(print first) (print second)
;       (cond
;         ((search first second)
;                     (push (caar i) record))
;         (t (cond
;              ((search second first)
;                     (push (car (cadr i)) record)))
;        )
;       )
;       )))
;record))


;(defun CREATE-COMBINATIONS  (ls)
;  (cond
;    ((null   (cdr ls)) nil)
;    (t (cons (combination-for-one (car ls) ls)
;             (CREATE-COMBINATIONS (cdr ls))
;       )
;    )
;  )
;)


;(defun combination-for-one (atom  ls)
;  (let ((result nil))
;
;    (loop for i in (cdr ls)
;         do
;       (push (list atom i) result))
;result))


;(defun squeeze-to-string-TOP (ls)
;  (loop for comment-or-restriction   in ls
;       collect
;       (squeeze-to-string   comment-or-restriction)))
```

```
;
;
;
;(defun squeeze-to-string (ls1)
;  (let ((result  (car ls1)))
;
;    (loop for word in (cdr ls1)
;         do
;      (setq result (string-append word result)))
;    (list ls1 result)
;))

;*****************************************************************************************
;&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
;&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
;*****************************************************************************************


;Command: (part-apart-the-list '("D" "N" " " " " " " " ...))
;(("D" "N" " " " " " " " " " " ")
; (((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "Z96B" "="
"R00E")))
;  ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U"))))
;Command: (part-apart-the-list '("D" "G" "M" " " " " " ...))
;(("D" "G" "M" " " " " " " " "c") (NIL NIL (" " NIL)))
;Command:


;(defun part-apart-the-list (ls)    ;;;; ie.....
;         ;;;;;;;;          ("H" "G" "M" " " " " " " " "c" NIL NIL (" " NIL))
;         ;;;;;;;;    OR
;         ;;;;;;;;
;;;;;;;;     ("D" "N" " " " " " " " " " " " "
;;;;     ((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" ";;;;;   "=
" "Z96B" "=" "R00E")))
;
;  (let* (
;         (var-1m (first ls)) (var-2m (second ls)) (var-3m (third ls)) (var-4m (fourth ls))
;         (var-5m (fifth ls)) (var-6m (sixth ls))(var-7m (seventh ls))
;         (base-features (list var-1m var-2m var-3m var-4m var-5m var-6m var-7m))
;         (restrictions (cdr (cdr (cdr (cdr (cdr (cdr (cdr ls))))))))
;        )
;    ;(print var-1m) (print var-2m) (print var-3m) (print var-4m) (print var-5m) (print var-6m)
;    ;(print var-7m)
;    (list base-features restrictions))
;)


;;;;;;;"(NO-opposite )   RETURNS   nil     OR      a   Base-Features   list.

;(defun NO-opposite (ls1 ls2)
;      (cond ((check-base-statements-TOP  ls1 ls2)
;             (cond
;              ;;;;;IN THE CASE where '(" " " " " ") '(" " " " " ")
;              ((and (= (length (count-the-BASE-FEATURES  ls1)) 0)
;                    (= (length (count-the-BASE-FEATURES  ls2)) 0))
;                   "NO-BASE-FEATURE-RESTRICTIONS")
;             (t (change-the-empty-values ls1 ls2))
;            )
;          )
```

```
;          (t nil)
;     )
;)



;(defun    change-the-empty-values (ls1 ls2)
;  (let ((result nil))
;    (loop for i in ls1 for j in ls2
;          do
;      ;;;;WE KNOW  THAT THEY   match.
;      (cond ((string-equal i " ") (push j result))
;            (t (push i result)))
;    )
;    result))


;(defun check-base-statements-TOP (ls1 ls2)   ;;;;;;RETURNS  t  OR    nil.
;  (prog (flag ls1-help ls2-help)
;       (setq flag t)
;       (setq ls1-help ls1)
;       (setq ls2-help ls2)
;       LOOP
;          (cond ((or
;                    (null ls1-help) (null ls2-help)) (return flag))
;                 (t
;                  (cond
;                    ((check-base-statements (car ls1-help) (car ls2-help))
;                             (setq ls1-help (cdr ls1-help))
;                             (setq ls2-help (cdr ls2-help))
;                             (go LOOP)
;                    )
;                    (t (return (setq flag nil)))
;                  )
;                 )
;          )
;   )
;)
;
;
;
;
;
;(defun check-base-statements (atom1-string  atom2-string)
;  (cond
;    ((string-equal atom1-string atom2-string))
;    (t (or
;         (string-equal atom1-string " ")
;         (string-equal atom2-string " ")))))



;(defun count-the-BASE-FEATURES (ls)
;  (cond
;    ((null ls) nil)
;    ((string-equal (car ls) " ") (count-the-BASE-FEATURES (cdr ls)))
;    (t (cons (car ls)
;             (count-the-BASE-FEATURES (cdr ls))))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;*******************************************************************************************
;*******************************************************************************************


;;;;;;; CHECK THE         RESTRICTIONS


;Command: (part-apart-the-list '("D" "W" " " " " " " ...))
;(("D" "W" " " " " " " " " " " ") (NIL
;                                   ((:TERRITORY-RESTRICTIONS-1 ("M20E" "+T53R")))
```

```
;                                   ("+" ("M20E"))
;                                   )
;
;  )
;

;Command: (part-apart-the-list '("D" "N" " " " " " " ...))
;(("D" "N" " " " " " " " " " " ")
; (((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "Z96B" "="
"R00E")))
;  ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U"))))
;Command:
;("H" "W" " " " " " " " " " " " NIL NIL (" " NIL))
;Command: (part-apart-the-list '("H" "W" " " " " " " ...))
;(("H" "W" " " " " " " " " " " ") (NIL NIL (" " NIL)))
;Command:


;  THIS comes from  IMPLICIT  !!!!!!!!!!
;
;Command: '("H" "G" " " " " "E" ...)
;("H" "G" " " " " "E" "A" "A"
; ((:COMBINATIONS-RESTRICTIONS-1 ("P" "H50K" "=" "H50P")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "H50K" "=" "
H50P")))
; ((:TERRITORY-RESTRICTIONS-1 ("M20E" "+T53R"))) ("+" ("M32*")))
;Command: (part-apart-the-list '("H" "G" " " " " "E" ...))
;(("H" "G" " " " " "E" "E" "A")
; (((:COMBINATIONS-RESTRICTIONS-1 ("P" "H50K" "=" "H50P")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "H50K" "=" "
H50P")))
;  ((:TERRITORY-RESTRICTIONS-1 ("M20E" "+T53R"))) ("+" ("M32*"))))
;Command:


;(defun CREATE-COMBINATIONS  (ls)
;  (cond
;    ((null   (cdr ls)) nil)
;    (t (cons (combination-for-one (car ls) ls)
;             (CREATE-COMBINATIONS (cdr ls))
;        )
;    )
;  )
;)


(defun which-COVERS-which (ls1 ls2)
  ;;;ie.
  ;;;ls1=     (("R" "W57A" "=" "Z96B"))
  ;;;;ls2 =      (("R" "W57A" "=" "Z96B" "=" "R00E") ("P" "R00E"))
   (let* ((result nil)
          (combination1 (squeeze-to-string-TOP ls1))
          (combination2 (squeeze-to-string-TOP ls2)))
    (loop for i in combination1
          do
      (push i result))
    (loop for j in combination2
          do
      (push j result))
    (the-most-wide-COMBINATION
```

```
                                (CREATE-COMBINATIONS result))
    )
)




(defun the-most-wide-COMBINATION (ls)
;   ie.
;Command: (which-COVERS-which door window)
;(((((("P" "R00E") "R00EP") (("R" "W57A" "=" "Z96B") "Z96B=W57AR"))
;  ((("P" "R00E") "R00EP") (("R" "W57A" "=" "Z96B" "=" "R00E") "R00E=Z96B=W57AR")))
; ((((("R" "W57A" "=" "Z96B" "=" "R00E") "R00E=Z96B=W57AR") (("R" "W57A" "=" "Z96B") "Z96B=W57AR"))))

  (let ((record nil))
    (loop for k in ls
          do
      (loop for i in k    ;;;;; BECAUSE the COMBINATIONS come IN  PAIRS.
            do

      (let ((first (car (reverse (car i))))
            (second (car (reverse (cadr i)))))
        ;;;(print first) (print second)
        (cond
          ((search first second)
                      (push (caar i) record))
          (t (cond
               ((search second first)
                      (push (car (cadr i)) record)))
        )
       )
        )))
record))


(defun CREATE-COMBINATIONS  (ls)
  (cond
    ((null   (cdr ls)) nil)
    (t (cons (combination-for-one (car ls) ls)
             (CREATE-COMBINATIONS (cdr ls))
       )
    )
  )
)


(defun combination-for-one (atom  ls)
  (let ((result nil))

    (loop for i in (cdr ls)
          do
        (push (list atom i) result))
result))




(defun combination-for-one (atom  ls)
  (let ((result nil))

    (loop for i in (cdr ls)
          do
        (push (list atom i) result))
result))
```

```
(defun squeeze-to-string-TOP (ls)
  (loop for comment-or-restriction   in ls
        collect
        (squeeze-to-string   comment-or-restriction)))


(defun squeeze-to-string (ls1)
  (let ((result  (car ls1)))

    (loop for word in (cdr ls1)
          do
      (setq result (string-append word result)))
    (list ls1 result)
))

;*****************************************************************************************
;&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
;&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
;*****************************************************************************************




;Command: (part-apart-the-list '("D" "N" " " " " " " ...))
;(("D" "N" " " " " " " " " " ")
; (((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "Z96B" "="
"R00E")))
;  ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U"))))
;Command: (part-apart-the-list '("D" "G" "M" " " " " ...))
;(("D" "G" "M" " " " " " " "c") (NIL NIL (" " NIL)))
;Command:



(defun part-apart-the-list (ls)     ;;;; ie.....
        ;;;;;;;;            ("H" "G" "M" " " " " " " "c" NIL NIL (" " NIL))
        ;;;;;;;;     OR
        ;;;;;;;;
;;;;;;;     ("D" "N" " " " " " " " " " " "
;;;    ((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" ";;;;;   ="
 "Z96B" "=" "R00E")))

  (let* (
        (var-1m (first ls)) (var-2m (second ls)) (var-3m (third ls)) (var-4m (fourth ls))
        (var-5m (fifth ls)) (var-6m (sixth ls))(var-7m (seventh ls))
        (base-features (list var-1m var-2m var-3m var-4m var-5m var-6m var-7m))
        (restrictions (cdr (cdr (cdr (cdr (cdr (cdr (cdr ls))))))))
       )
   ;(print var-1m) (print var-2m) (print var-3m) (print var-4m) (print var-5m) (print var-6m)
   ;(print var-7m)
   (list base-features restrictions))
)









;;;;;;;"(NO-opposite )   RETURNS   nil     OR      a   Base-Features   list.

(defun NO-opposite (ls1 ls2)
     (cond ((check-base-statements-TOP  ls1 ls2)
            (cond
             ;;;;;IN THE CASE where '(" " " " " ") '(" " " " " ")
             ((and (= (length (count-the-BASE-FEATURES  ls1)) 0)
                   (= (length (count-the-BASE-FEATURES  ls2)) 0))
```

```
                  "NO-BASE-FEATURE-RESTRICTIONS")
              (t (change-the-empty-values ls1 ls2))
             )
           )
          (t nil)
    )
)



(defun    change-the-empty-values (ls1 ls2)
  (let ((result nil))
    (loop for i in ls1 for j in ls2
          do
      ;;;;WE KNOW  THAT THEY   match.
      (cond ((string-equal i " ") (push j result))
            (t (push i result)))
    )
    result))



(defun check-base-statements-TOP (ls1 ls2)   ;;;;;;RETURNS  t  OR    nil.
  (prog (flag ls1-help ls2-help)
        (setq flag t)
        (setq ls1-help ls1)
        (setq ls2-help ls2)
        LOOP
           (cond ((or
                     (null ls1-help) (null ls2-help)) (return flag))
                  (t
                   (cond
                     ((check-base-statements (car ls1-help) (car ls2-help))
                               (setq ls1-help (cdr ls1-help))
                               (setq ls2-help (cdr ls2-help))
                               (go LOOP)
                     )
                     (t (return (setq flag nil)))
                   )
                  )
           )
    )
)



(defun check-base-statements (atom1-string  atom2-string)
  (cond
    ((string-equal atom1-string atom2-string))
    (t (or
          (string-equal atom1-string " ")
          (string-equal atom2-string " ")))))



(defun count-the-BASE-FEATURES (ls)
  (cond
    ((null ls) nil)
    ((string-equal (car ls) " ") (count-the-BASE-FEATURES (cdr ls)))
    (t (cons (car ls)
             (count-the-BASE-FEATURES (cdr ls))))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;***********************************************************************************************
;***********************************************************************************************


;;;;;;; CHECK THE          RESTRICTIONS
```

```
;Command: (part-apart-the-list '("D" "W" " " " " " " " ...))
;(("D" "W" " " " " " " " " " " " ") (NIL
;                                    ((:TERRITORY-RESTRICTIONS-1 ("M20E" "+T53R")))
;                                    ("+" ("M20E"))
;                                    )
;
;  )
;

;Command: (part-apart-the-list '("D" "N" " " " " " " " ...))
;(("D" "N" " " " " " " " " " " " ")
; (((:COMBINATIONS-RESTRICTIONS-1 ("P" "R00E")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "W57A" "=" "Z96B" "="
"R00E")))
;  ((:TERRITORY-RESTRICTIONS-1 ("M20U" "+T60A"))) ("+" ("M20U"))))
;Command:
;("H" "W" " " " " " " " " " " " " NIL NIL (" " NIL))
;Command: (part-apart-the-list '("H" "W" " " " " " " " ...))
;(("H" "W" " " " " " " " " " " " ") (NIL NIL (" " NIL)))
;Command:



;  THIS comes from  IMPLICIT  !!!!!!!!!!
;
;Command: '("H" "G" " " " " " "E" ...)
;("H" "G" " " " " " "E" "A" "A"
; ((:COMBINATIONS-RESTRICTIONS-1 ("P" "H50K" "=" "H50P")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "H50K" "=" "
H50P")))
; ((:TERRITORY-RESTRICTIONS-1 ("M20E" "+T53R"))) ("+" ("M32*")))
;Command: (part-apart-the-list '("H" "G" " " " " " "E" ...))
;(("H" "G" " " " " " "E" "E" "A")
; (((:COMBINATIONS-RESTRICTIONS-1 ("P" "H50K" "=" "H50P")) (:COMBINATIONS-RESTRICTIONS-2 ("R" "H50K" "="
"H50P")))
;  ((:TERRITORY-RESTRICTIONS-1 ("M20E" "+T53R"))) ("+" ("M32*"))))
;Command:
```

```
;;;-*-  Mode: lisp;  Package: austin-rover ; syntax: common-lisp-*-


(defun  print-the-information-known-for-the-other-models (pathname Secondary-Description-ID-entity &optio
nal (stream *standard-output*))
  (let ((maybes-others (get-MAYBES-TOP-others pathname Secondary-Description-ID-entity))
        (musts-other (get-MUSTS-TOP-others pathname Secondary-Description-ID-entity))
        (the-feature-groups-are-not-going-with-this-model nil))
    (let ((all-feature-groups nil))
      (loop for i in musts-other
       ;;;;;;;(("METRO" (NIL)) ("ROVER-800-SERIES" (NIL)))    OR
      ;;;;;;;(("New-Rover200-or-Rover400-Series" ("B51")) ("ROVER-800-series" (NIL)))
            do
            (push (second i) all-feature-groups))
      (loop for i in maybes-others
            do
            (push (second i) all-feature-groups))
      (let* ((vpq-codes-all (remove nil (remove-duplicates (only-atoms all-feature-groups) :test #'string
-equal) stream)))
        (loop for group in vpq-codes-all
              do
          (let ((group-line  (block GROUP-LINE-in-*the-GROUP-headers*
                                 (loop for i in *the-GROUP-headers*
                                       do
                                   (cond ((string-equal group (substring i 0 3))
                                          (return-from GROUP-LINE-in-*the-GROUP-headers* i))))
                                 nil)))
            (cond ((not (DOES-the-GROUPS-go-TO-this-MODEL? (find-the-model-number *model*)  group-line))
                   (push (return-the-name-of-each-FEATURE-GROUP (substring group-line 0 3))
                         the-feature-groups-are-not-going-with-this-model)))))))

    (terpri stream)
    (print (format nil "THE MUSTS FEATURE GROUPS KNOWN FOR OTHER MODELS FOR THE PART DESCRIPTION  ~s  ARE
:" (SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION  Secondary-Description-ID-entity)))

    (loop for model-with-musts in musts-other
          do
      (print (format nil "FOR THE MODEL :  ~s"  (car model-with-musts)))
      (loop for i   in  (return-the-name-of-ALL-FEATURE-GROUPS (cadr model-with-musts))
            do
        (print i stream)))

      (terpri stream) (print (format nil "THE MAYBES FEATURE GROUPS KNOWN FOR OTHER MODELS FOR THE
 PART DESCRIPTION  ~s  ARE :" (SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION  Secondary-Description-ID-en
tity)))

    (loop for model-with-maybes in maybes-others
          do
      (print (format nil "FOR THE MODEL :  ~s"  (car model-with-maybes)))
        (loop for i   in  (return-the-name-of-ALL-FEATURE-GROUPS (cadr model-with-maybes))
              do
        (print i stream)))
  (terpri stream)
  (print (format nil "Also, CHECKED FROM THE Additional Feature Group Index Report  CHART, the FEATURE GR
OUPS which are NOT GO with the model ~s  ARE:" *model*) stream)
  (loop for i in the-feature-groups-are-not-going-with-this-model
        do
    (print i stream))
  (terpri stream)
the-feature-groups-are-not-going-with-this-model)) ;;;;RETURNS the FEATURE-GROUPS in English which are no
t GOING with this MODEL.
;;;;;;;ie.. ("B69-SEAT HEATING                                "    "B53-TONNEAU COVER                                "
)



(defun get-MAYBES-TOP-others (pathname Secondary-Description-ID-entity)
  (let ((result-on-the-top nil) (OTHER-MODELS-WITH-FEATURE-GROUP-NUMBERS (get-MAYBES-others pathname Seco
ndary-Description-ID-entity)))
    (loop for list-of-model-with-feature-numbers  in  OTHER-MODELS-WITH-FEATURE-GROUP-NUMBERS
          do
      (let* ((model (car list-of-model-with-feature-numbers ))
             (numbers (cadr list-of-model-with-feature-numbers))
             (the-feature-code-expression-of-the-numbers (let ((result nil))
                                                           (loop for i in numbers
```

```
                                                                    do
                                                                 (push (given-a-number-return-the-Feature-Gro
up i) result))
                                                              result)))
        (push (list model the-feature-code-expression-of-the-numbers) result-on-the-top)))
result-on-the-top))



;Command: (setq *model* 'r8)
;R8
;;Command: (with-database (db *the-pathname-of-the-ROVERs-APN-CATALOGUE*)
;        (with-transaction () (get-MAYBES-others *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathnam
e* #<PART-DESCRIPTI
;ON:    SEAT-FRONT COMPLETE   MANUAL          >)))
;((ROVER-800-SERIES (-1)) (METRO (-1)))
;Command:



(defun  get-MAYBES-others (pathname  Secondary-Description-ID-entity)
  (let ((the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-MAYBES-of-all-the-others
         (READ-THE-MAYBE-FEATURES-from-THE-ROVER-CARS----actually-Austin-Rover-other pathname Secondary-
Description-ID-entity))
        (result  nil))
    (BLOCK takis
      (cond ((equal the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-MAYBES-of-all-the-
others nil) ;;AREN'T ANY maybes ??   ;;;;;THIS COND is rubbish ... Is only kept from the previous old  ge
t-MAYBES function.
                  (return-from takis nil)))
      (loop for list-of-MODEL-with-VECTOR  in
                 the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-MAYBES-of-all-the-ot
hers
            do   ;;;;   ((NEW-ROVER200-OR-ROVER400-SERIES #<ART-FIXNUM-5 54430165>) (ROVER-800-SERIES NIL
))

        (let ((model (car list-of-MODEL-with-VECTOR))
              (the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-MAYBES
                (cond ((equal (cadr list-of-MODEL-with-VECTOR) nil) (make-array 2 :initial-contents '(2 -
1)))
       ;;;;;;;;; A pseudo array which eventually the EVALUATION in FEature Groups of the value -1 will re
turn  NIL.
                      (t (cadr list-of-MODEL-with-VECTOR)))))

          (let* ((VECTOR-dimensions (aref the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-
the-MAYBES 0))
                 (the-number-codes-of-features (let ((temp nil))
                                                 (loop for i from 1 to (- VECTOR-dimensions 1)
                                                       do
                                                   (push
                                                     (aref the-VECTOR-of-the--Secondary-Description-ID-en
tity-which-KEEPS-the-MAYBES i) temp)) temp)))
            (push (list model the-number-codes-of-features) result)))))
        result))



(defun READ-THE-MAYBE-FEATURES-from-THE-ROVER-CARS----actually-Austin-Rover-other (pathname  Secondary-De
scription-ID-entity)
  (let ((help (get-THE-ROVER-CARS----actually-Austin-Rover---entity pathname Secondary-Description-ID-ent
ity)))
    ;;;returns-the-actual THE-ROVER-CARS----actually-Austin-Rover statice-object to reference to the
    ;;;;    actual statice object of Secondary-Description-ID-entity as argument to the function.
    (with-database (db pathname)
      (with-transaction ()
        (BLOCK  DOES-IT-HAVE-A-statice-entity-type-LINKED-TO?
          (cond ((equal help nil) (return-from DOES-IT-HAVE-A-statice-entity-type-LINKED-TO? nil)))
          (case *model*
            (R8 (list (list "METRO" (THE-ROVER-CARS----actually-Austin-Rover-METRO-maybes   help))
                      (list "ROVER-800-series" (THE-ROVER-CARS----actually-Austin-Rover-Rover-800-maybes
 help))))
            (metro (list (list "New-Rover200-or-Rover400-Series"
                               (THE-ROVER-CARS----actually-Austin-Rover-New-ROVER-200-or-ROVER-400--XW-ma
```

```
ybes   help))
                          (list "ROVER-800-series"
                                 (THE-ROVER-CARS----actually-Austin-Rover-Rover-800-maybes   help))))
            (Rover800 (list  (list "New-Rover200-or-Rover400-Series"
                                   (THE-ROVER-CARS----actually-Austin-Rover-New-ROVER-200-or-ROVER-400--X
W-maybes   help))
                              (list "METRO"  (THE-ROVER-CARS----actually-Austin-Rover-METRO-maybes   help)
))))))))


;   (METRO-maybes  (vector fixnum))
;   (R6-maybes     (vector fixnum))
;   (MAESTRO-maybes   (vector fixnum))
;   (MONTEGO-maybes   (vector fixnum))
;   (old-ROVER-200--XH-maybes    (vector fixnum))
;   (New-ROVER-200-or-ROVER-400--XW-maybes   (vector fixnum))
;   (ROVER-800-maybes   (vector  fixnum))
;   (R17-maybes       (vector fixnum))

;*****************************************************************************************************
***************

(defun get-MUSTS-TOP-others (pathname Secondary-Description-ID-entity)
  (let ((result-on-the-top nil) (OTHER-MODELS-WITH-FEATURE-GROUP-NUMBERS (get-MUSTS-others pathname Secon
dary-Description-ID-entity)))
    (loop for list-of-model-with-feature-numbers  in  OTHER-MODELS-WITH-FEATURE-GROUP-NUMBERS
          do
      (let* ((model (car list-of-model-with-feature-numbers ))
             (numbers (cadr list-of-model-with-feature-numbers))
             (the-feature-code-expression-of-the-numbers (let ((result nil))
                                                          (loop for i in numbers
                                                                do
                                                            (push (given-a-number-return-the-Feature-Gro
up i) result))
                                                          result)))
        (push (list model the-feature-code-expression-of-the-numbers) result-on-the-top)))
result-on-the-top))



;Command: (setq *model* 'r8)
;R8
;;Command: (with-database (db *the-pathname-of-the-ROVERs-APN-CATALOGUE*)
;        (with-transaction () (get-MUSTS-others *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname
* #<PART-DESCRIPTI
;ON:    SEAT-FRONT COMPLETE   MANUAL          >)))
;((ROVER-800-SERIES (-1)) (METRO (-1)))
;Command:



(defun  get-MUSTS-others (pathname  Secondary-Description-ID-entity)
  (let ((the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-MUSTS-of-all-the-others
          (READ-THE-MUST-FEATURES-from-THE-ROVER-CARS----actually-Austin-Rover-other pathname Secondary-D
escription-ID-entity))
        (result  nil))
    (BLOCK takis
      (cond ((equal the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-MUSTS-of-all-the-o
thers nil) ;;AREN'T ANY musts ??   ;;;;;THIS COND is rubbish ... Is only kept from the previous old  get-
MUSTS function.
                (return-from takis nil)))
      (loop for list-of-MODEL-with-VECTOR  in
                the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-MUSTS-of-all-the-oth
ers
          do   ;;;;   ((NEW-ROVER200-OR-ROVER400-SERIES #<ART-FIXNUM-5 54430165>)  (ROVER-800-SERIES NIL
))


        (let ((model (car list-of-MODEL-with-VECTOR))
              (the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-the-MUSTS
                (cond ((equal (cadr list-of-MODEL-with-VECTOR) nil) (make-array 2 :initial-contents '(2 -
1)))

      ;;;;;;;;; A pseudo array which eventually the EVALUATION in FEature Groups of the value -1 will re
turn  NIL.

                      (t (cadr list-of-MODEL-with-VECTOR)))))
```

```
          (let* ((VECTOR-dimensions (aref the-VECTOR-of-the--Secondary-Description-ID-entity-which-KEEPS-
the-MUSTS 0))
                 (the-number-codes-of-features (let ((temp nil))
                                                 (loop for i from 1 to (- VECTOR-dimensions 1)
                                                       do
                                                   (push
                                                     (aref the-VECTOR-of-the--Secondary-Description-ID-en
tity-which-KEEPS-the-MUSTS i) temp)) temp)))
            (push (list model the-number-codes-of-features) result)))))
        result))



(defun READ-THE-MUST-FEATURES-from-THE-ROVER-CARS----actually-Austin-Rover-other (pathname  Secondary-Des
cription-ID-entity)
  (let ((help (get-THE-ROVER-CARS----actually-Austin-Rover---entity pathname Secondary-Description-ID-ent
ity)))
    ;;;returns-the-actual THE-ROVER-CARS----actually-Austin-Rover statice-object to reference to the
    ;;;;     actual statice object of Secondary-Description-ID-entity as argument to the function.
    (with-database (db pathname)
      (with-transaction ()
        (BLOCK  DOES-IT-HAVE-A-statice-entity-type-LINKED-TO?
          (cond ((equal help nil) (return-from DOES-IT-HAVE-A-statice-entity-type-LINKED-TO? nil)))
          (case *model*
            (R8 (list (list "METRO" (THE-ROVER-CARS----actually-Austin-Rover-METRO-musts   help))
                      (list "ROVER-800-series" (THE-ROVER-CARS----actually-Austin-Rover-Rover-800-musts
 help))))
            (metro (list (list "New-Rover200-or-Rover400-Series"
                               (THE-ROVER-CARS----actually-Austin-Rover-New-ROVER-200-or-ROVER-400--XW-mu
sts   help))
                         (list "ROVER-800-series"
                               (THE-ROVER-CARS----actually-Austin-Rover-Rover-800-musts   help))))
            (Rover800 (list  (list "New-Rover200-or-Rover400-Series"
                                   (THE-ROVER-CARS----actually-Austin-Rover-New-ROVER-200-or-ROVER-400--X
W-musts   help))
                             (list "METRO"  (THE-ROVER-CARS----actually-Austin-Rover-METRO-musts   help))
))))))))
```

```
  ;;;; -*- Syntax: Common-Lisp; Package: austin-rover; Base: 10; Mode: LISP -*-
;
;;(defflavor sub-part
;;       (primary-description
;;        secondary-description
;;        super-part
;;        things-that-are-special)
;;       ()
;;  :initable-instance-variables
;;  :writable-instance-variables)
;;
;;(defmethod (sys:print-self sub-part) (stream &rest ignore)
;;  (format stream "#<SUB-PART ~A ~A>" primary-description secondary-description))
;;
;;(defflavor super-part
;;       (name
;;        vpg
;;        base-features
;;        additional-feature-groups)
;;       ()
;;  :initable-instance-variables :writable-instance-variables)
;;
;;(defmethod (sys:print-self super-part) (stream &rest ignore)
;;  (format stream "#<SUPER-PART ~A ~A>" name vpg))
;;
;;
;;;*****************************************************************************************
;
;(defflavor PART-DESCRIPTION
;       (name) ()
;  :initable-instance-variables
;  :writable-instance-variables)
;
;(defvar *PART-DESCRIPTIONS* nil)
;
;(defun make-some-PART-DESCRIPTIONS ()
;  (setq *PART-DESCRIPTIONS*
;       (loop for name in '("one" "two" "three" "four")
;             collect
;               (make-instance 'PART-DESCRIPTION :name name))))
;
;
;
;(defun match-PART-DESCRIPTIONS ()
;  (let ((help (accept 'PART-DESCRIPTION)))
;    (print '******)
;     help))
;
;
;  ;(find (accept 'string) *foos* :test #'string-equal :key #'foo-name))
;
;
;
;(define-presentation-type PART-DESCRIPTION ()
;   :no-deftype t
;   :parser ((stream)
;            (dw:complete-from-sequence *PART-DESCRIPTIONS* stream
;                                       :type 'PART-DESCRIPTION
;                                       :name-key #'PART-DESCRIPTION-name
;                                       :complete-activates t))
;   :printer ((object stream)
;             (format stream (PART-DESCRIPTION-name  object))))
;
;
;
;
;
;
;;;*****************************************************************************************
;
;(defvar *PART-DESCRIPTION* '("bar 1" "exterior mirror" "bar 3"))
;
;(define-presentation-type PART-DESCRIPTION ()
;   :parser ((stream)
;            (dw:complete-from-sequence *PART-DESCRIPTION* stream
;                                       :type 'PART-DESCRIPTION
```

```
;                                        :complete-activates t))
;   :printer ((object stream)
;             (format stream object)))
;
;
;(defun match-PRESENTATIONS ()
;  (find (accept 'string) *PART-DESCRIPTION*));; :test #'string-equal))
;
;;;;(accept 'PART-DESCRIPTION ):stream stream)
;*****************************************************************************************

(defparameter *FRAMEWORK-DISPLAY-margin-components*
              'dw:((margin-ragged-borders)
                   (margin-scroll-bar  :history-noun "display")
                   (margin-scroll-bar  :history-noun "display"
                                       :margin :bottom)
                   (margin-white-borders  :thickness 2)))


(DW:DEFINE-PROGRAM-FRAMEWORK ENGINEERING-SPEC-------AUDITING-VALIDATION
  :SELECT-KEY
  #\circle
  :COMMAND-DEFINER
  T
  :COMMAND-TABLE
  (:INHERIT-FROM '("colon full command" "standard arguments" "input editor compatibility")
   :KBD-ACCELERATOR-P 'NIL)
  :STATE-VARIABLES
  NIL
  :PANES
  ((PANE-2 :TITLE :HEIGHT-IN-LINES 1 :REDISPLAY-AFTER-COMMANDS NIL)
   (PANE-3 :DISPLAY
           :margin-components *FRAMEWORK-DISPLAY-margin-components*)

   (PANE-4 :COMMAND-MENU :MENU-LEVEL :TOP-LEVEL) (PANE-5 :INTERACTOR :HEIGHT-IN-LINES 4))
  :CONFIGURATIONS
  '((DW::MAIN (:LAYOUT (DW::MAIN :COLUMN PANE-2 PANE-3 PANE-4 PANE-5))
     (:SIZES
      (DW::MAIN (PANE-2 1 :LINES) (PANE-4 :ASK-WINDOW SELF :SIZE-FOR-PANE PANE-4)
       (PANE-5 4 :LINES) :THEN (PANE-3 :EVEN))))))


;(define-auditor-command (com-new-statement :menu-accelerator t) ()
;   (let ((stream (dw:get-program-pane 'pane-3))
;         (input nil))
;    (send stream :clear-history)
;     (setq input (multiple-value-list (menu2 (dw:get-program-pane 'pane-3))))
;     (format stream "Your choice was ~A" input)))




;(define--command (com-start  :menu-accelerator t) ()
;   (let ((stream (dw:get-program-pane 'pane-3)))
;     (send stream :clear-history)
;     (menu-framework stream)))


;(define-auditor-command (com-begin  :menu-accelerator t) ()
;   (let ((stream (dw:get-program-pane 'pane-3)))
;     (graph-framework stream)
;     (send stream :clear-history)
;     (menu-framework stream)))


;*****************************************************************************************


;(define-ENGINEERING-SPEC-------AUDITING-VALIDATION-command
```

```
;  (com-ENTER-FEATURE-GROUP--- :menu-accelerator t) ()
;   (let ((stream (dw:get-program-pane 'pane-3)))
;     (accept 'feature-group-string)))

(define-ENGINEERING-SPEC-------AUDITING-VALIDATION-command
  (com-GIVE-NEGATIVES-with-MENUS--- :menu-accelerator t) ()
   (i-am  *no-negatives*)
)

(define-ENGINEERING-SPEC-------AUDITING-VALIDATION-command
  (com-CLEAR-DISPLAY--- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-3)))
     (send stream :clear-history)))

(defun menu-QUERY-for-VPG ()  ;;;;;; YOU MUST NOT put different character format to this menu ie. BOLD ,
italic etc.
  ;;;;;;;; only  nil.nil.nil
  (values
    (dw:menu-choose '(("Yes" . T) ("No" . nil)) :prompt (zl:format nil "DO YOU WANT TO ENTER THE PART DES
CRIPTION USING THE VPG CODE IT BELONGS  ?") :near-mode '(:point 100 700))))



(defun show-all-VPGS ()
  (let ((result nil))
    (with-database  (db  *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*)
                  (with-transaction ()
                    (for-each ((SSSVPG Sub-Sub-Sub-Vpg ))
                      (push (Sub-Sub-Sub-Vpg-VPG-code  SSSVPG) result))))
result))


(defun menu-VPGs (list &optional (stream *standard-output*))
  ;;(let ((stream (dw:get-program-pane 'pane-3)))
  (let ((answers nil))
    (push
      ;;(multiple-value-list
        (dw:accepting-values (stream :own-window t
                                     :near-mode '(:point 100 700)
                                     :label
                                     "THE  ROVER'S VPG CODES  AVAILABLE IN STATICE DATA BASE. ")
          (values
            (accept '(alist-member :alist ,list) :prompt "ENTER THE VPG CODE YOUR PART DESCRIPTION BELONG
S"
                    :stream stream )

          )
       )
      answers)
    answers
  )
)

(defun only-atoms (ls)
  (cond ((null ls) nil)
        ((atom (car ls))
            (cons (car ls)
                  (only-atoms (cdr ls))))
        (t (append (only-atoms (car ls))
                   (only-atoms (cdr ls))))))



(define-ENGINEERING-SPEC-------AUDITING-VALIDATION-command
  (com-GRAPH-a-VPG---  :menu-accelerator t) ()
 (let ((stream (dw:get-program-pane 'pane-3)))
(terpri stream)
       (print (format nil "THE SYSTEM SEARCHES THE BASE and ADDITIONAL FEATURES OF THE MODEL FOR ALL THE
AVAILABLE FEATURE GROUPS") stream)
                  (terpri stream)
   (let ((temp (car (menu-VPGs (show-all-VPGS)  stream))))
     (the-top-function-to-draw-the-VPG  temp stream))))


(define-ENGINEERING-SPEC-------AUDITING-VALIDATION-command
```

```
  (com-START-PROGRAM---  :menu-accelerator t) ()
  ;;;;;;;(setq *part-description-in-concern*   nil)
  (let ((stream (dw:get-program-pane 'pane-3)))
    (send stream :clear-history)
    (let ((PART-DESCRIPTION-we-CONCERN nil))
      (cond ((menu-QUERY-for-VPG)
             (print (format nil "THE SYSTEM IS WORKING OUT ALL THE POSSIBLE  VPG  CODES WITHIN T
HE Statice Data Base ") stream) (terpri stream)
             (let ((temp-vpg (car (menu-VPGs (show-all-VPGS)  stream))))
               (setq PART-DESCRIPTION-we-CONCERN  (to-get-the-PART-DESCRIPTION-FROM-A-VPG-graph-tree tem
p-vpg stream))))
            (t

     (print (format nil "PLEASE ENTER a PART DESCRIPTION FROM THE Automatic Part Numbering CATALOGUE:" )
 stream)
     (terpri stream)
     (setq PART-DESCRIPTION-we-CONCERN  (values (match-PART-DESCRIPTIONS stream))))
       (print (format nil "You Have ALREADY Enter the DESCRIPTION of the PART YOU CONCERN,  PLEASE INDICA
TE now THE *CAR MODEL* WHICH THIS  PART HAS BEEN DESIGNED  FOR ."))
      (terpri) (terpri)
  (setq *model* (car (menu-models)))       ;;;;;;FIRST WE HAVE TO assign the *model* and later to assign t
he *BAFCs-PATHNAME*
       (work    PART-DESCRIPTION-we-CONCERN)
       (STATICE-do-the-begining stream   PART-DESCRIPTION-we-CONCERN ));;;;;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!
))



(define-ENGINEERING-SPEC-------AUDITING-VALIDATION-command
  (com-GIVE-VALID-SPECS-in-ENGLISH--- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-3)))
(cond ((equal *feature-chart-entries* nil)
             (print (format nil "It SEEMS the FEATURE GROUPS WHICH YOU GAVE IN ORDER TO SPECIFY THE PAR
T ,though  THERE EXIST  IN THE Base and Additionals Features CHARTS OF THE Model  -s  , THEY HAVE BEEN DE
LETED OUT !!!!!" *model*) stream))
           (t

   (print (format nil "THE SYSTEM NOW CONSULTS THE  DATA FILES :   "stream))
     (print (format nil "Territory Code Index Report ,  Territory Group Index Report ") stream)
     (print (format nil "Additional Feature Chart,      Base Feature Chart") stream)
     (terpri stream) (terpri stream)

      (cond
       ((equal *common-ground-record* nil)  ;;;;Try the last chance to see if there is
              ;;;;;;;;only ONE FEATURE GROUP for the SPECIFICATION of the Part
          (setq *common-ground-record* (the-Specification-of-a-part-with-only-ONE-FEATURE-GROUP))
           (finish-it-ONLY-FOR-ONE-FEATURE-GROUP-SPECIFICATION *common-ground-record* stream))
       (t
  (cond ((and    ;;; ONLY ONE RUN ?
           (equal (length (give-the-priority-for-the-SEARCH-of-ALGORITHM)) 1)
                ;;; NUMBER of RESULTS NOT only  1
           (> *result-of-RUNS*   1)
          )
                (finish-it ;;;;;-IN-PIMS
                           (remove "nothing" *common-ground-record*;;;  PUT OUT THE
                                   :test #'string-equal                 ;;;;;  "nothing"s
                                   :key #'first) stream) )
       (t
            ;;;;;;;;;;;(finish-it-IN-PIMS  *common-ground-record* stream))
        (finish-it   ;;;-IN-PIMS
          (remove "nothing" *common-ground-record*;;;  PUT OUT THE
                                   :test #'string-equal                 ;;;;;  "nothing"s
                                   :key #'first) stream))

  )
))  ;;;;;;;;    cond
     ))
))


;(cond
;       ((equal *common-ground-record* nil)  ;;;;Try the last chance to see if there is
;              ;;;;;;;;only ONE FEATURE GROUP for the SPECIFICATION of the Part
;         (setq *common-ground-record* (the-Specification-of-a-part-with-only-ONE-FEATURE-GROUP))
;           (finish-it *common-ground-record* stream))
```

```
;        (t ;;;;   NOT equal *common-ground-record* nil
;
;;        (and    ;;; ONLY ONE RUN ?
;;           (equal (length (give-the-priority-for-the-SEARCH-of-ALGORITHM)) 1)
;;                ;;; NUMBER of RESULTS NOT only  1
;;           (> *result-of-RUNS*   1)
;;          )
;        (cond ((equal (length (give-the-priority-for-the-SEARCH-of-ALGORITHM)) 1) ;;; ONLY ONE RUN ?
;               (finish-it (remove "nothing" *common-ground-record*                ;;;  PUT OUT THE
;                                  :test #'string-equal                            ;;;;;  "nothing"s
;                                  :key #'first) stream)  )
;              (t
;               (finish-it *common-ground-record* stream))
;        )     ;;;;;;; cond
;        )))))))))))))   t
;     )

(define-ENGINEERING-SPEC-------AUDITING-VALIDATION-command
  (com-GIVE-VALID-SPECS-in-PIMS--- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-3)))
     (cond ((equal *feature-chart-entries* nil)
            (print (format nil "It SEEMS the FEATURE GROUPS WHICH YOU GAVE IN ORDER TO SPECIFY THE PART ,
though  THERE EXIST  IN THE Base and Additionals Features CHARTS OF THE Model  -s  , THEY HAVE BEEN  DELE
TED OUT !!!!!" *model*) stream))
           (t
     (print (format nil "THE  PIMS  CODE FOR  -S  IS :      "  *model*) stream)
     (prinl (MODEL-IN-PIMS) stream)
     (terpri stream) (terpri  stream)

     (print (format nil "THE SYSTEM NOW CONSULTS THE  DATA FILES :   "stream))
     (print (format nil "Territory Code Index Report ,  Territory Group Index Report ") stream)
     (print (format nil "Additional Feature Chart,      Base Feature Chart") stream)
     (terpri stream) (terpri stream)
(cond
       ((equal *common-ground-record* nil)  ;;;;Try the last chance to see if there is
              ;;;;;;;;only ONE FEATURE GROUP for the SPECIFICATION of the Part
          (setq *common-ground-record* (the-Specification-of-a-part-with-only-ONE-FEATURE-GROUP))
            (finish-it-ONLY-FOR-ONE-FEATURE-GROUP-SPECIFICATION *common-ground-record* stream))
       (t
   (cond ((and    ;;; ONLY ONE RUN ?
            (equal (length (give-the-priority-for-the-SEARCH-of-ALGORITHM)) 1)
                ;;; NUMBER of RESULTS NOT only  1
            (> *result-of-RUNS*   1)
           )
                (finish-it-IN-PIMS (remove "nothing" *common-ground-record*;;;  PUT OUT THE
                                   :test #'string-equal                  ;;;;;  "nothing"s
                                   :key #'first) stream) )
        (t
         ;;;;;;(finish-it-IN-PIMS  *common-ground-record* stream))
         (finish-it-IN-PIMS (remove "nothing" *common-ground-record*;;;  PUT OUT THE
                                   :test #'string-equal                  ;;;;;  "nothing"s
                                   :key #'first) stream))
  )
))    ;;;;cond
)) ))


(defun MODEL-IN-PIMS ()
  (case *model*
    (metro  'XF)
    (r8  'XW)))


(define-ENGINEERING-SPEC-------AUDITING-VALIDATION-command
  (com-CHECK-FOR-A-specific-TRIM-LEVEL---  :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-3)))

     (print (format nil "THE SYSTEM NOW CONSULTS THE   Base  Feature  Chart   DATA FILE")
            stream)
     (terpri) (terpri)
     ;(send stream :clear-history)
      (graph stream)))
```

```
;;;;;;;;;;;
```

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: austin-rover; Base: 10 -*-


;;; MOMENTARY MENU CHOOSE WITH HIGHLIGHTED ITEMS
;;;
;;; Try doing: (menu-choose-with-highlight '(a b c) ; the items
;;;                                        '(c))    ; the ones to highlight
;
;(defflavor momentary-menu-with-highlighting ()
;          (tv:momentary-multiple-menu
;           tv:menu-highlighting-mixin))





(tv:defwindow-resource tv:momentary-multiple-menu ()
         :make-window (tv:momentary-multiple-menu)
         :reusable-when :deexposed
         :deinitializer (send object :set-geometry nil nil))

(defun menu-choose-with-highlight (item-list
                                   &optional (highlighted-items nil)
                                             (label "PLEASE ENTER THE FEATURES WHICH ARE NEGATIVE TO THE
PART DESCRIPTION:")
                                             (near-mode '(:point 100 700))
                                             ;;;(near-mode '(:mouse))     ;;;;;;!!!!!
                                             (default-item nil)

                                             (superior *graphic* )  ;;;(tv:mouse-default-superior))
                                             )
  (when (eq (car near-mode) :window)
    (setq superior (tv:sheet-superior (cadr near-mode))))
  (using-resource (m-menu tv:momentary-multiple-menu superior)
    (send m-menu :set-item-list item-list)
    (send m-menu :set-label label)
    (send m-menu :set-last-item default-item)
    (tv:expose-window-near m-menu near-mode)
    (send m-menu :set-highlighted-items highlighted-items)
    (tv:saving-mouse-position m-menu
      (and default-item (not (zl:memq (car near-mode) '(:mouse :point)))
           (multiple-value-bind (x y)
               (send m-menu :item-cursorpos default-item)
             (and x y
                  (send m-menu :set-mouse-position (+ x (tv:sheet-inside-left m-menu))
                        (+ y (tv:sheet-inside-top m-menu))))))
      (let ((values (multiple-value-list (send m-menu :choose))))
        (tv:restore-mouse-position-if-necessary (first values))
        (cl:apply #'values values)))))



;;*****************************************************************************************

(defvar  *takis-negatives* '("B51F-----AUTOLUX LEATHER                AUTLET  "))


;;;;;;;; *no-negatives*  REPRESENTS all THE Maybes and Musts in CONCERN

(defvar  *no-negatives*  '(
  "B51F-----AUTOLUX LEATHER                AUTLET  "
  "B51R-----LIGHTENING VELVET              LITVEL  "
  "B51L-----MORSE                          MORSE   "
  "B51L-----MORSE                          MORSE   "
  "B51K-----ZENITH                         ZENITH  "
  "B69D-----HEATED DRIVERS & PASS SEAT     TWOHOT  "
  "B44R-----PAIR REC FT SEATS LUMB&HT ADJ  LUMBER  "
  "B44B-----PAIR RECLINING SEATS           RCSEAT  " ))
```

```
;(define-ENGINEERING-SPEC-------AUDITING-VALIDATION-command
;  (com-GIVE-NEGATIVES-with-MENUS--- :menu-accelerator t) ()
;   (i-am  "no-negatives")
;)

;;;;;  EXAMPLE OF  "seat-front  complete  manual"


(defun i-am   (ls )
  (let ((result  nil))
    (loop for i in ls do
      (loop for j in  *takis-negatives*  ;;;;;;;; REPRESENTS ALL   negatives THAT THE SYSTEM Knows
            do

        (cond ((string-equal (substring i 0 4) (substring j 0 4) )
                (push i result)))))
    ;;;;;;;(print result)
    (values (menu-choose-with-highlight   ls result))))



::Command: (i-am *no-negatives* )      ;;;;;;; THE TOP FUNCTION.
```

```
;;; -*- MODE:LISP; PACKAGE:USER; SYNTAX:COMMON-LISP; BASE:10; -*-

;;; This file demonstrates the sort of thing you need to do to save and restore
;;; flavor oriented connected data structures.


;*****************************************************************************


;*****************************************************************************

;;; A flavor representing nodes
(DEFFLAVOR NODE (ID                  ;; unique identity
                 (NODE-LINKS NIL)) ;; nodes
            ()
  :INITABLE-INSTANCE-VARIABLES
  :READABLE-INSTANCE-VARIABLES
  :WRITABLE-INSTANCE-VARIABLES
 (:REQUIRED-INIT-KEYWORDS :ID)
 (:CONC-NAME   nil)
 )


;(defvar *nodes* nil)
;(defmethod (make-instance node) (&rest ignore)
;  (push self *nodes*))



;;; A method to add a new connection to the node
(DEFMETHOD (ADD-NODE-LINK NODE) (NODE)
  ;; can go from self to node
  (PUSH NODE NODE-LINKS))


;;; Method to obtain a connected node
(DEFMETHOD (NODE-LINK NODE) (ID)
  (FIND-NODE ID NODE-LINKS))


;;; A macro to find a node with a given id from a list of nodes
;(DEFMACRO FIND-NODE (ID LIST-OF-NODES)
;  `(FIND ,ID ,LIST-OF-NODES :KEY #'ID))
;*****************************************************************************


;;; A flavor representing networks
(DEFFLAVOR NETWORK
        ((NODES NIL)) ;; a list of node instances linked in some way
            ()
  :INITABLE-INSTANCE-VARIABLES
  :READABLE-INSTANCE-VARIABLES
  :WRITABLE-INSTANCE-VARIABLES)

                        ;;; A macro to find a node with a given id from a list of nodes
                                    (DEFMACRO FIND-NODE (ID LIST-OF-NODES)
                                    `(FIND ,ID ,LIST-OF-NODES :KEY #'ID))

;;; A Method to find a node
(DEFMETHOD (NODE NETWORK) (ID)
  (FIND-NODE ID NODES))


;;; A method to add a new node, with the supplied keyword initialisations
(DEFMETHOD (ADD-NODE NETWORK) (&REST KEYWORD-ARGS)
 (PUSH (APPLY #'MAKE-INSTANCE
               'NODE
               KEYWORD-ARGS)
       NODES))

;;; A method to connect two node that are part of a network
(DEFMETHOD (CONNECT-NODES NETWORK) (ID-1 ID-2)
  (LET ((FROM (NODE SELF ID-1))
```

```
      (TO   (NODE SELF ID-2)))
    (WHEN (AND FROM TO)
      (ADD-NODE-LINK ;;;;;;;;;;;;;;????????  SELF
             FROM TO))))
                                    ;;; A method to add a new connection to the node
                                    ;(DEFMETHOD (ADD-NODE-LINK NODE) (NODE)
                                    ;      ;; can go from self to node
                                    ;  (PUSH NODE NODE-LINKS))


;***********************************************************************************


;;; Top level function to make a network from a textual 'description'
(DEFUN MAKE-NETWORK (DESCRIPTION)
  (LET ((NETWORK (MAKE-INSTANCE 'NETWORK)))
    ;; first create all the nodes with
    ;; the links unfilled
    (LOOP FOR (ID . LINK-IDS) IN DESCRIPTION    ;; first pass
        DO
      (ADD-NODE NETWORK :ID ID))

    ;; then fill in all the links
    (LOOP FOR (ID . LINK-IDS) IN DESCRIPTION  ;; second pass
        DO
                             ;;;;;;;;(print link-ids)



      (LET ((NODE (NODE NETWORK ID)))
                             ;;;;;;;;;(print 'HELP_NODE_IS ) (prinl help-node)
        (LOOP FOR L-ID IN LINK-IDS       ;********************
            DO
                             ;;;;;;;;;(print 'L-ID_IS) (prinl L-ID)

          (ADD-NODE-LINK NODE ;;;; (make-instance 'node :id L-ID) ))
                             (NODE NETWORK L-ID))))


    ;;;(loop for node in (network-nodes network) do
    ;;;;   ;;;;(node-node-links node)))                 ;;;;?????????????????

;    (loop for (nodel . rest-of-nodes) in (network-nodes network) do
;      (connect-nodes network
;                   nodel
;                   (node-links (cadr (network-nodes network)))))

    ;; finally return the network object
    NETWORK)))


                                    ;;; A method to add a new connection to the node
                                    ;(DEFMETHOD (ADD-NODE-LINK NODE) (NODE)
                                    ;      ;; can go from self to node
                                    ;  (PUSH NODE NODE-LINKS))


;;; Top level function to obtain a network description from a network
(DEFUN NETWORK-DESCRIPTION (NETWORK)
  (LOOP FOR NODE IN (NETWORK-NODES NETWORK)
        COLLECT
        (CONS (ID NODE)
              (LOOP FOR LINK-NODE IN (NODE-LINKS NODE)
                    COLLECT (ID LINK-NODE)))))


;;; A network description
(DEFCONSTANT EXAMPLE-NETWORK-DESCRIPTION
             '((A B C)  ;; node id followed by
               (B C D)  ;;  the ids of the connected nodes
               (C D)
```

```
                (D A)))

;;; A network instance
(DEFPARAMETER
 EXAMPLE-NETWORK (MAKE-NETWORK EXAMPLE-NETWORK-DESCRIPTION))

;*****************************************************************************************
(defmethod  (printing-start network) ()
  (loop for i in (network-nodes self)
        do
    (printing i)))


(defun printing (a-node)
 (let ((record (node-links a-node)))
     (graphics:with-room-for-graphics ()

     (format-graph-from-root a-node
                           #'(lambda (o str) (princ (id o)  str))

                            #'(lambda (o)
                                (append (if
                                            (and (node-links o)
                                                 (not (member o  record)))

                                                (node-links o))

                                ))


))))
```

APPENDIX 5

LISTING OF THE STATISTICAL ANALYSIS PROGRAM

```
  ;;;; -*- Syntax: Common-Lisp; Package: austin-rover; Base: 10; Mode: LISP -*-

;;;;;THE  function REASON-UP-ON-THE-PRIMARIES-WITHIN-THE-VPG-GROUP-TOP  is the   TOP  function.
;;;;;;  ;;;;;;;       L O O K      REFERENCE   ######


(defun REASON-UP-ON-THE-PRIMARIES-WITHIN-THE-VPG-GROUP-TOP (a-string)    ;;;;ie "1109AA"
  (let ((temp (REASON-UP-ON-THE-PRIMARIES-WITHIN-THE-VPG-GROUP a-string)) (result nil))
    (let ((ONEs-car-VPG  (car temp)) (OTHERs-car-VPG (cadr temp)))
      (loop for i in ONEs-car-VPG for j in OTHERs-car-VPG
          do
        (push (list
                (car i)
                (get-the-intersection-of-a-VARIABLE-length-list (list (cadr i) (cadr j)))
                (get-the-intersection-of-a-VARIABLE-length-list (list (third i) (third j))))
              result)))
    result))


(defun  REASON-UP-ON-THE-PRIMARIES-WITHIN-THE-VPG-GROUP  (a-string)
  (let ((temp  (reasoning-for-a-single-VPG a-string))  (result nil))
    (loop for vpg in temp
        do
      (push (the-common-features-of-primary vpg) result))
result))




(defun reasoning-for-a-single-VPG (a-string)
  (let ((family-entities nil) (result nil) (result-final nil) (metro nil) (r8 nil) )
    (with-database  (db  *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*)
      (with-transaction ()
        (let ((THE-actual-VPG-entity (car (the-entity-type-of-the-VPG-CODE a-string))))
          (setq family-entities (Sub-Sub-Sub-Vpg-HAS-Families-within THE-actual-VPG-entity))

                      (loop for entity  in family-entities
                            do
                        (let* ((primaries-entities (FAMILY-within-a-SSS-Vpg-Has-Primary-Descriptions  ent
ity)))


                          (push primaries-entities  result))))
    ;;;result  ====
;(((#<Entity-Handle of PRIMARY-DESCRIPTION-ID 35/4 374236363> #<Entity-Handle of PRIMARY-DESCRIPTION-ID 35
/5 374236423>)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 35/3 374236323>) (#<Entity-Handle of PRIMARY-DESCRIPTION-ID
34/6 374236373>)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 34/5 374236333>) (#<Entity-Handle of PRIMARY-DESCRIPTION-ID
34/4 374236273>)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 34/3 374236243>) (#<Entity-Handle of PRIMARY-DESCRIPTION-ID
33/6 374236303>)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 33/4 374236533> #<Entity-Handle of PRIMARY-DESCRIPTION-ID 33
/5 374236253>)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 32/7 374236263> #<Entity-Handle of PRIMARY-DESCRIPTION-ID 32
/8 374236313>
;  #<Entity-Handle of PRIMARY-DESCRIPTION-ID 33/3 374236473>)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 32/5 374236503> #<Entity-Handle of PRIMARY-DESCRIPTION-ID 32
/6 374236543>))
    (loop for  family in result
        do
      (loop for primary in family
          do
        (let ((secondaries  (Primary-Description-ID-Has-Secondary-Descriptions  primary)))
          (push (list primary secondaries) result-final))))
    ;;; result-final
;(((#<Entity-Handle of PRIMARY-DESCRIPTION-ID 32/6 374236543> (#<PART-DESCRIPTION:    INSTR DRG-FIXING
 SEAT FRONT      >))
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 32/5 374236503>
;  (#<PART-DESCRIPTION:    INSTR DRG              ELEC-NO MEMORY  > #<PART-DESCRIPTION:    INSTR DRG
      ELEC-MEMORY
;   >
;   #<PART-DESCRIPTION:    INSTR DRG              MAN-NO L/SUPP   >
;   #<PART-DESCRIPTION:    INSTR DRG              MAN L/SUPP      >))
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 33/3 374236473> (#<PART-DESCRIPTION:    ARRGT DRG-SEAT FRONT
```

```
  TRIM WIRE/CLIPS >))
; ( ..........................................................   ...............................
    (setq *model* 'r8)
    (loop for primary-with-its-secondaries in result-final
          do
      (push (list (car primary-with-its-secondaries)
                  (delete nil      ;;;;ie.   DELETE   the UNKNOWN  information  .
                  (loop for secondary  in  (cadr primary-with-its-secondaries)
                        collect
                        (get-MUSTS-top *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname* seconda
ry)))
                  (delete nil
                  (loop for secondary  in  (cadr primary-with-its-secondaries)
                        collect
                          (get-MAYBES-top *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*  sec
ondary))))
            r8))
    (setq  *model* 'metro)
    (loop for primary-with-its-secondaries in result-final
          do
      (push (list (car primary-with-its-secondaries)
                  (delete nil
                  (loop for secondary  in  (cadr primary-with-its-secondaries)
                        collect
                        (get-MUSTS-top *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname* seconda
ry)))
                  (delete nil
                  (loop for secondary  in  (cadr primary-with-its-secondaries)
                        collect
                        (get-MAYBES-top *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname* second
ary))))
            metro))

)) ;;;;with-database
(list metro r8)))


;;;(sort '(("B69" "B51") ("B51") ("B51" "B44") ("B51" "B44")) #'(lambda (ls) (length ls)))


(defun  new-intersection (ls1 ls2)
  (cond
    ((equal ls1  nil) ls2)
    ((equal ls2  nil) ls1)
    (t (intersection ls1 ls2 :test #'string-equal))))


(defun  get-the-intersection-of-a-VARIABLE-length-list  (ls) ;; ie '(("b51" "b45") ("b51"))  OR  '(("b51"
"b45"))  OR
  (PROG (result temp)
        LOOP
           (cond ((null ls) (return result))
                 (t
                    (setq temp (new-intersection (car ls) (cadr ls)))
                    (setq result (new-intersection result temp))))
           (setq ls (cddr ls))
           (go LOOP)))


(defun  the-common-features-of-primary (ls-of-primarys)
  (let ((result nil))
    (loop for i in ls-of-primarys
          do
      (push (list (car i)
                  (get-the-intersection-of-a-VARIABLE-length-list (cadr i))
                  (get-the-intersection-of-a-VARIABLE-length-list (third i))) result))
result))
```

```
;Command: (reasoning-for-a-single-VPG "1109aa")
;(((#<Entity-Handle of PRIMARY-DESCRIPTION-ID 35/4 374236363> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 35/5 374236423> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 35/3 374236323> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 34/6 374236373> (("B25")) (("NO-SPECIFICATION")))
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 34/5 374236333> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 34/4 374236273> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 34/3 374236243> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 33/6 374236303> (("B51")) (("B45")))
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 33/4 374236533> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 33/5 374236253> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 32/7 374236263> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 32/8 374236313> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 33/3 374236473> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 32/5 374236503> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 32/6 374236543> NIL NIL))
; ((#<Entity-Handle of PRIMARY-DESCRIPTION-ID 35/4 374236363> (("B45")) (("NO-SPECIFICATION")))
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 35/5 374236423> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 35/3 374236323> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 34/6 374236373> (("B51" "B25")) (("B53" "B45")))
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 34/5 374236333> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 34/4 374236273> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 34/3 374236243> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 33/6 374236303> (("B69" "B51") ("B51") ("B51" "B44") ("B51"
 "B44"))
;   (("B44") ("B69" "B44" "B53" "B45") ("B53" "B45" "B69") ("B53" "B45" "B69")))
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 33/4 374236533> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 33/5 374236253> (("NO-SPECIFICATION")) (("NO-SPECIFICATION"
)))
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 32/7 374236263> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 32/8 374236313> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 33/3 374236473> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 32/5 374236503> NIL NIL)
;  (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 32/6 374236543> NIL NIL)))
;Command: Logout


;;;;;;;   REFERENCE   ######

;Command: (REASON-UP-ON-THE-PRIMARIES-WITHIN-THE-VPG-GROUP-top "1109aa")
;(((#<Entity-Handle of PRIMARY-DESCRIPTION-ID 35/4 374236363> ("B45") ("NO-SPECIFICATION"))
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 35/5 374236423> NIL NIL)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 35/3 374236323> NIL NIL)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 34/6 374236373> ("B25") NIL)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 34/5 374236333> NIL NIL)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 34/4 374236273> NIL NIL)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 34/3 374236243> NIL NIL)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 33/6 374236303> ("B51") ("B45"))  ;;;SEAT FRONT CO
MPLETE
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 33/4 374236533> NIL NIL)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 33/5 374236253> ("NO-SPECIFICATION") ("NO-SPECIFICATION"))
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 32/7 374236263> NIL NIL)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 32/8 374236313> NIL NIL)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 33/3 374236473> NIL NIL)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 32/5 374236503> NIL NIL)
; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 32/6 374236543> NIL NIL))
;Command:


:#<Entity-Handle of PRIMARY-DESCRIPTION-ID 33/6 374236303>
;Command: (DESCRIBE #<Table 108/111 :LOCKING NIL 367342401>)
:#<Table 108/111 :LOCKING NIL 367342401> is a table with 108 entries out of a possible 111 (97%).
;Growth threshold = 1.00, Growth factor = 1.3
;Test function for comparing keys = EQL, hash function = CLI::XEQLHASH
;Do you want to see the contents of the hash table? (Y or N) Yes.
;Do you want it sorted? (Y or N) Yes.
; FAMILY-WITHIN-A-SSS-VPG-BELONGS-TO-SSS-VPG-IS → #<Attribute BELONGS-TO-SSS-VPG of FAMILY-WITHIN-A-SSS-V
PG 367332527>
; FAMILY-WITHIN-A-SSS-VPG-D → #<Domain FAMILY-WITHIN-A-SSS-VPG 367330537>
; FAMILY-WITHIN-A-SSS-VPG-FAMILY-CODE-IS → #<Attribute FAMILY-CODE of FAMILY-WITHIN-A-SSS-VPG 367332505>
; FAMILY-WITHIN-A-SSS-VPG-HAS-PRIMARY-DESCRIPTIONS-IS → #<Attribute IS of %$FAMILY-WITHIN-A-SSS-VPG-HAS-P
RIMARY-DESCRIPTIONS 36734
;4100>
; FAMILY-WITHIN-A-SSS-VPG-HAS-PRIMARY-DESCRIPTIONS-OF → #<Attribute OF of %$FAMILY-WITHIN-A-SSS-VPG-HAS-P
RIMARY-DESCRIPTIONS 36734
;4056>
```

```
; FAMILY-WITHIN-A-SSS-VPG-HAS-PRIMARY-DESCRIPTIONS-RF → #<Relation %$FAMILY-WITHIN-A-SSS-VPG-HAS-PRIMARY-
DESCRIPTIONS 367342770>
; FAMILY-WITHIN-A-SSS-VPG-MAYBE-FEATURES-IS → #<Attribute MAYBE-FEATURES of FAMILY-WITHIN-A-SSS-VPG 36733
2573>
; FAMILY-WITHIN-A-SSS-VPG-MUST-FEATURES-IS → #<Attribute MUST-FEATURES of FAMILY-WITHIN-A-SSS-VPG 3673325
51>
; FAMILY-WITHIN-A-SSS-VPG-R → #<Relation FAMILY-WITHIN-A-SSS-VPG 367332463>
; FAMILY-WITHIN-A-SSS-VPG-SA → #<Attribute %$OF of FAMILY-WITHIN-A-SSS-VPG 367332615>
; PRIMARY-DESCRIPTION-ID-BELONGS-TO-FAMILY-IS → #<Attribute BELONGS-TO-FAMILY of PRIMARY-DESCRIPTION-ID 3
67332717>
; PRIMARY-DESCRIPTION-ID-D → #<Domain PRIMARY-DESCRIPTION-ID 367330560>
; PRIMARY-DESCRIPTION-ID-HAS-SECONDARY-DESCRIPTIONS-IS → #<Attribute IS of %$PRIMARY-DESCRIPTION-ID-HAS-S
ECONDARY-DESCRIPTIONS 367
;345645>
; PRIMARY-DESCRIPTION-ID-HAS-SECONDARY-DESCRIPTIONS-OF → #<Attribute OF of %$PRIMARY-DESCRIPTION-ID-HAS-S
ECONDARY-DESCRIPTIONS 367
;345731>
; PRIMARY-DESCRIPTION-ID-HAS-SECONDARY-DESCRIPTIONS-RF → #<Relation %$PRIMARY-DESCRIPTION-ID-HAS-SECONDAR
Y-DESCRIPTIONS 367345667>
; PRIMARY-DESCRIPTION-ID-MAYBE-FEATURES-FOR-ALL-ROVER-VEHICLES-IS → #<Attribute MAYBE-FEATURES-FOR-ALL-RO
VER-VEHICLES of PRIMARY-D
;ESCRIPTION-ID 367332763>
; PRIMARY-DESCRIPTION-ID-MUST-FEATURES-FOR-ALL-ROVER-VEHICLES-IS → #<Attribute MUST-FEATURES-FOR-ALL-ROVE
R-VEHICLES of PRIMARY-DES
;CRIPTION-ID 367332741>
; PRIMARY-DESCRIPTION-ID-PRIMARY-DESCRIPTION-OF-THE-PART-IS → #<Attribute PRIMARY-DESCRIPTION-OF-THE-PART
 of PRIMARY-DESCRIPTION-I
;D 367332675>
; PRIMARY-DESCRIPTION-ID-R → #<Relation PRIMARY-DESCRIPTION-ID 367332653>
; PRIMARY-DESCRIPTION-ID-SA → #<Attribute %$OF of PRIMARY-DESCRIPTION-ID 367333005>
; SECONDARY-DESCRIPTION-ID-APPROVED-FOR-INDICATOR-IS → #<Attribute APPROVED-FOR-INDICATOR of SECONDARY-DE
SCRIPTION-ID 367337502>
; SECONDARY-DESCRIPTION-ID-BELONGS-TO-PRIMARY-DESCRIPTION-IS → #<Attribute BELONGS-TO-PRIMARY-DESCRIPTION
 of SECONDARY-DESCRIPTION
;-ID 367337130>
; SECONDARY-DESCRIPTION-ID-COLOUR-CHART-KEY-IS → #<Attribute COLOUR-CHART-KEY of SECONDARY-DESCRIPTION-ID
 367337460>
; SECONDARY-DESCRIPTION-ID-COLOURED-INDICATOR-IS → #<Attribute COLOURED-INDICATOR of SECONDARY-DESCRIPTIO
N-ID 367337612>
; SECONDARY-DESCRIPTION-ID-D → #<Domain SECONDARY-DESCRIPTION-ID 367330601>
; SECONDARY-DESCRIPTION-ID-DATE-CREATED-IS → #<Attribute DATE-CREATED of SECONDARY-DESCRIPTION-ID 3673375
24>
; SECONDARY-DESCRIPTION-ID-DE-SEGMENT-KEY-IS → #<Attribute DE-SEGMENT-KEY of SECONDARY-DESCRIPTION-ID 367
333327>
; SECONDARY-DESCRIPTION-ID-DESIGN-RESPONSIBILITY-IS → #<Attribute DESIGN-RESPONSIBILITY of SECONDARY-DESC
RIPTION-ID 367337240>
; SECONDARY-DESCRIPTION-ID-EMISSIONS-INDICATOR-IS → #<Attribute EMISSIONS-INDICATOR of SECONDARY-DESCRIPT
ION-ID 367333415>
; SECONDARY-DESCRIPTION-ID-FAMILY-IS → #<Attribute FAMILY of SECONDARY-DESCRIPTION-ID 367333131>
; SECONDARY-DESCRIPTION-ID-FLAG-FOR-INFORMATION-OF-FEATURES-GATHERED-FROM-THE-TAPE-IS → #<Attribute FLAG-
FOR-INFORMATION-OF-FEATUR
;ES-GATHERED-FROM-THE-TAPE of SECONDARY-DESCRIPTION-ID 367333107>
; SECONDARY-DESCRIPTION-ID-HAS-MAYBES-FEATURES-IS → #<Attribute HAS-MAYBES-FEATURES of SECONDARY-DESCRIPT
ION-ID 367337174>
; SECONDARY-DESCRIPTION-ID-HAS-MUST-FEATURES-IS → #<Attribute HAS-MUST-FEATURES of SECONDARY-DESCRIPTION-
ID 367337216>
; SECONDARY-DESCRIPTION-ID-HONDA-FUNC-AREA-CODE-IS → #<Attribute HONDA-FUNC-AREA-CODE of SECONDARY-DESCRI
PTION-ID 367337326>
; SECONDARY-DESCRIPTION-ID-IN-ENGLISH-THE-DETAILED-INFORMATION-OF-HOW-THE-MUST-AND-MAYBES-FEATURES-DERIVE
D-IS → #<Attribute IN-ENG
;LISH-THE-DETAILED-INFORMATION-OF-HOW-THE-MUST-AND-MAYBES-FEATURES-DERIVED of SECONDARY-DESCRIPTION-ID 36
7333065>
; SECONDARY-DESCRIPTION-ID-INITIAL-STOCK-ORDER-INDICATOR-IS → #<Attribute INITIAL-STOCK-ORDER-INDICATOR o
f SECONDARY-DESCRIPTION-I
;D 367337700>
; SECONDARY-DESCRIPTION-ID-MAIN-GROUP-DESCRIPTION-IS → #<Attribute MAIN-GROUP-DESCRIPTION of SECONDARY-DE
SCRIPTION-ID 367333217>
; SECONDARY-DESCRIPTION-ID-MAIN-GROUP-IS → #<Attribute MAIN-GROUP of SECONDARY-DESCRIPTION-ID 367333175>
; SECONDARY-DESCRIPTION-ID-MP17-INDICATOR-IS → #<Attribute MP17-INDICATOR of SECONDARY-DESCRIPTION-ID 367
337372>
; SECONDARY-DESCRIPTION-ID-NEGATIVE-ADDITIONAL-FEATURES-IS → #<Attribute NEGATIVE-ADDITIONAL-FEATURES of
SECONDARY-DESCRIPTION-ID
;367337152>
; SECONDARY-DESCRIPTION-ID-OBSOLETE-INDICATOR-IS → #<Attribute OBSOLETE-INDICATOR of SECONDARY-DESCRIPTIO
N-ID 367337546>
; SECONDARY-DESCRIPTION-ID-PANEL-PART-INDICATOR-IS → #<Attribute PANEL-PART-INDICATOR of SECONDARY-DESCRI
```

```
PTION-ID 367337304>
; SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION-IS → #<Attribute PART-DESCRIPTION of SECONDARY-DESCRIPTION-ID
 367337064>
; SECONDARY-DESCRIPTION-ID-PRIMARY-DESCRIPTION-IS → #<Attribute PRIMARY-DESCRIPTION of SECONDARY-DESCRIPT
ION-ID 367333373>
; SECONDARY-DESCRIPTION-ID-PRIMARY-LETTER-IS → #<Attribute PRIMARY-LETTER of SECONDARY-DESCRIPTION-ID 367
333263>
; SECONDARY-DESCRIPTION-ID-R → #<Relation SECONDARY-DESCRIPTION-ID 367333043>
; SECONDARY-DESCRIPTION-ID-RECNO-IS → #<Attribute RECNO of SECONDARY-DESCRIPTION-ID 367333153>
; SECONDARY-DESCRIPTION-ID-REFERENCE-TO-THE-ROVER-VEHICLES-IS → #<Attribute REFERENCE-TO-THE-ROVER-VEHICL
ES of SECONDARY-DESCRIPTI
;ON-ID 367337106>
; SECONDARY-DESCRIPTION-ID-REGULATIONS-INDICATOR-IS → #<Attribute REGULATIONS-INDICATOR of SECONDARY-DESC
RIPTION-ID 367337656>
; SECONDARY-DESCRIPTION-ID-SA → #<Attribute %$OF of SECONDARY-DESCRIPTION-ID 367337042>
; SECONDARY-DESCRIPTION-ID-SAFETY-INDICATOR-IS → #<Attribute SAFETY-INDICATOR of SECONDARY-DESCRIPTION-ID
 367337634>
; SECONDARY-DESCRIPTION-ID-SECONDARY-DESCRIPTION-IS → #<Attribute SECONDARY-DESCRIPTION of SECONDARY-DESC
RIPTION-ID 367333351>
; SECONDARY-DESCRIPTION-ID-SECONDARY-LETTER-IS → #<Attribute SECONDARY-LETTER of SECONDARY-DESCRIPTION-ID
 367333241>
; SECONDARY-DESCRIPTION-ID-SEGMENT-SEQUENCE-NUMBER-IS → #<Attribute SEGMENT-SEQUENCE-NUMBER of SECONDARY-
DESCRIPTION-ID 367337436>
; SECONDARY-DESCRIPTION-ID-SERVICE-INDICATOR-IS → #<Attribute SERVICE-INDICATOR of SECONDARY-DESCRIPTION-
ID 367337262>
; SECONDARY-DESCRIPTION-ID-SHORT-DESCRIPTION-IS → #<Attribute SHORT-DESCRIPTION of SECONDARY-DESCRIPTION-
ID 367337414>
; SECONDARY-DESCRIPTION-ID-UNIT-OF-MEASURE-IS → #<Attribute UNIT-OF-MEASURE of SECONDARY-DESCRIPTION-ID 3
67337570>
; SECONDARY-DESCRIPTION-ID-VPG-CODE-IS → #<Attribute VPG-CODE of SECONDARY-DESCRIPTION-ID 367333305>
; SECONDARY-DESCRIPTION-ID-ZONE-CODE-IS → #<Attribute ZONE-CODE of SECONDARY-DESCRIPTION-ID 367337350>
; SUB-SUB-SUB-VPG-D → #<Domain SUB-SUB-SUB-VPG 367330516>
; SUB-SUB-SUB-VPG-FEATURE-GROUPS-LINKED-TO-THE-SUB-SUB-SUB-VPG-IS → #<Attribute FEATURE-GROUPS-LINKED-TO-
THE-SUB-SUB-SUB-VPG of SU
;B-SUB-SUB-VPG 367332403>
; SUB-SUB-SUB-VPG-HAS-FAMILIES-WITHIN-IS → #<Attribute IS of %$SUB-SUB-SUB-VPG-HAS-FAMILIES-WITHIN 367342
313>
; SUB-SUB-SUB-VPG-HAS-FAMILIES-WITHIN-OF → #<Attribute OF of %$SUB-SUB-SUB-VPG-HAS-FAMILIES-WITHIN 367342
335>
; SUB-SUB-SUB-VPG-HAS-FAMILIES-WITHIN-RF → #<Relation %$SUB-SUB-SUB-VPG-HAS-FAMILIES-WITHIN 367342357>
; SUB-SUB-SUB-VPG-R → #<Relation SUB-SUB-SUB-VPG 367332337>
; SUB-SUB-SUB-VPG-SA → #<Attribute %$OF of SUB-SUB-SUB-VPG 367332425>
; SUB-SUB-SUB-VPG-VPG-CODE-IS → #<Attribute VPG-CODE of SUB-SUB-SUB-VPG 367332361>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-D → #<Domain THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER 367330622>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-MAESTRO-MAYBES-IS → #<Attribute MAESTRO-MAYBES of THE-ROVER-CAR
S----ACTUALLY-AUSTIN-ROVE
;R 367335044>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-MAESTRO-MUSTS-IS → #<Attribute MAESTRO-MUSTS of THE-ROVER-CARS-
---ACTUALLY-AUSTIN-ROVER
;367335572>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-METRO-MAYBES-IS → #<Attribute METRO-MAYBES of THE-ROVER-CARS---
-ACTUALLY-AUSTIN-ROVER 36
;7335110>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-METRO-MUSTS-IS → #<Attribute METRO-MUSTS of THE-ROVER-CARS----A
CTUALLY-AUSTIN-ROVER 3673
;35636>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-MINI-MAYBES-IS → #<Attribute MINI-MAYBES of THE-ROVER-CARS----A
CTUALLY-AUSTIN-ROVER 3673
;35132>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-MINI-MUSTS-IS → #<Attribute MINI-MUSTS of THE-ROVER-CARS----ACT
UALLY-AUSTIN-ROVER 367335
;660>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-MONTEGO-MAYBES-IS → #<Attribute MONTEGO-MAYBES of THE-ROVER-CAR
S----ACTUALLY-AUSTIN-ROVE
;R 367335022>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-MONTEGO-MUSTS-IS → #<Attribute MONTEGO-MUSTS of THE-ROVER-CARS-
---ACTUALLY-AUSTIN-ROVER
;367335550>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-1-MAYBES-IS → #<Attribute NEW-ROVER-1-MAYBES of THE-R
OVER-CARS----ACTUALLY-AUS
;TIN-ROVER 367334670>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-1-MUSTS-IS → #<Attribute NEW-ROVER-1-MUSTS of THE-ROV
ER-CARS----ACTUALLY-AUSTI
;N-ROVER 367335416>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-10-MAYBES-IS → #<Attribute NEW-ROVER-10-MAYBES of THE
-ROVER-CARS----ACTUALLY-A
```

```
;USTIN-ROVER 367334426>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-10-MUSTS-IS → #<Attribute NEW-ROVER-10-MUSTS of THE-R
OVER-CARS----ACTUALLY-AUS
;TIN-ROVER 367335154>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-2-MAYBES-IS → #<Attribute NEW-ROVER-2-MAYBES of THE-R
OVER-CARS----ACTUALLY-AUS
;TIN-ROVER 367334646>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-2-MUSTS-IS → #<Attribute NEW-ROVER-2-MUSTS of THE-ROV
ER-CARS----ACTUALLY-AUSTI
;N-ROVER 367335374>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-200-OR-ROVER-400--XW-MAYBES-IS → #<Attribute NEW-ROVE
R-200-OR-ROVER-400--XW-MA
;YBES of THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER 367334756>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-200-OR-ROVER-400--XW-MUSTS-IS → #<Attribute NEW-ROVER
-200-OR-ROVER-400--XW-MUS
;TS of THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER 367335504>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-3-MAYBES-IS → #<Attribute NEW-ROVER-3-MAYBES of THE-R
OVER-CARS----ACTUALLY-AUS
;TIN-ROVER 367334624>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-3-MUSTS-IS → #<Attribute NEW-ROVER-3-MUSTS of THE-ROV
ER-CARS----ACTUALLY-AUSTI
;N-ROVER 367335352>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-4-MAYBES-IS → #<Attribute NEW-ROVER-4-MAYBES of THE-R
OVER-CARS----ACTUALLY-AUS
;TIN-ROVER 367334602>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-4-MUSTS-IS → #<Attribute NEW-ROVER-4-MUSTS of THE-ROV
ER-CARS----ACTUALLY-AUSTI
;N-ROVER 367335330>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-5-MAYBES-IS → #<Attribute NEW-ROVER-5-MAYBES of THE-R
OVER-CARS----ACTUALLY-AUS
;TIN-ROVER 367334560>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-5-MUSTS-IS → #<Attribute NEW-ROVER-5-MUSTS of THE-ROV
ER-CARS----ACTUALLY-AUSTI
;N-ROVER 367335306>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-6-MAYBES-IS → #<Attribute NEW-ROVER-6-MAYBES of THE-R
OVER-CARS----ACTUALLY-AUS
;TIN-ROVER 367334536>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-6-MUSTS-IS → #<Attribute NEW-ROVER-6-MUSTS of THE-ROV
ER-CARS----ACTUALLY-AUSTI
;N-ROVER 367335264>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-7-MAYBES-IS → #<Attribute NEW-ROVER-7-MAYBES of THE-R
OVER-CARS----ACTUALLY-AUS
;TIN-ROVER 367334514>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-7-MUSTS-IS → #<Attribute NEW-ROVER-7-MUSTS of THE-ROV
ER-CARS----ACTUALLY-AUSTI
;N-ROVER 367335242>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-8-MAYBES-IS → #<Attribute NEW-ROVER-8-MAYBES of THE-R
OVER-CARS----ACTUALLY-AUS
;TIN-ROVER 367334472>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-8-MUSTS-IS → #<Attribute NEW-ROVER-8-MUSTS of THE-ROV
ER-CARS----ACTUALLY-AUSTI
;N-ROVER 367335220>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-9-MAYBES-IS → #<Attribute NEW-ROVER-9-MAYBES of THE-R
OVER-CARS----ACTUALLY-AUS
;TIN-ROVER 367334450>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-NEW-ROVER-9-MUSTS-IS → #<Attribute NEW-ROVER-9-MUSTS of THE-ROV
ER-CARS----ACTUALLY-AUSTI
;N-ROVER 367335176>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-OLD-ROVER-200--XH-MAYBES-IS → #<Attribute OLD-ROVER-200--XH-MAY
BES of THE-ROVER-CARS----
;ACTUALLY-AUSTIN-ROVER 367335000>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-OLD-ROVER-200--XH-MUSTS-IS → #<Attribute OLD-ROVER-200--XH-MUST
S of THE-ROVER-CARS----AC
;TUALLY-AUSTIN-ROVER 367335526>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-R → #<Relation THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER 36733424
2>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-R17-MAYBES-IS → #<Attribute R17-MAYBES of THE-ROVER-CARS----ACT
UALLY-AUSTIN-ROVER 367334
;712>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-R17-MUSTS-IS → #<Attribute R17-MUSTS of THE-ROVER-CARS----ACTUA
LLY-AUSTIN-ROVER 36733544
;0>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-R6-MAYBES-IS → #<Attribute R6-MAYBES of THE-ROVER-CARS----ACTUA
LLY-AUSTIN-ROVER 36733506
;6>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-R6-MUSTS-IS → #<Attribute R6-MUSTS of THE-ROVER-CARS----ACTUALL
```

```
Y-AUSTIN-ROVER 367335614>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-ROVER-800-MAYBES-IS → #<Attribute ROVER-800-MAYBES of THE-ROVER
-CARS----ACTUALLY-AUSTIN-
;ROVER 367334734>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-ROVER-800-MUSTS-IS → #<Attribute ROVER-800-MUSTS of THE-ROVER-C
ARS----ACTUALLY-AUSTIN-RO
;VER 367335462>
; THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER-SA → #<Attribute $$OF of THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVE
R 367334277>
;#<Table 108/111 :LOCKING NIL 367342401>
;Command:
```

```
;;;-*- Mode: LISP; Syntax: Common-lisp; Package: austin-rover; Base: 10



;;*****************************************************************************************

;;;;; The function  which  RUN  the rest of the Program is  Judge-from-the-tape-the-MUSTS-and-MAYBES-of-t
he-MODEL-in-concern

(defun Judge-from-the-tape-the-MUSTS-and-MAYBES-of-the-MODEL-in-concern (mepg10t.zbe-file    xwuc.dat-fil
e model model)
  ;;;; model is coming from the User Interface when you insert the model of the tape
  ;;;;  is      is the result of the read-the-EJA18V-tape-of-the-specific-model function.
  (read-the-EJA18V-tape-of-the-specific-model  mepg10t.zbe-file xwuc.dat-file model) ;;;;ie. ASSIGN the v
ariable  katerina
  (translate-make-the-link-TOP katerina))

;;*****************************************************************************************


;;;(defvar *model* nil)   This it does exist at the other part of the program ie "atc-0:>takis>statice>th
e-program-the-menus.lisp"
;;          We need it for     LATER  for the  files ".... statice-R8.dat" and " ...statice-R8-continued
.dat" later , as well.



(defvar *var1* nil)
(defvar *var2* nil)

(defvar paul1 nil)
(defvar paul nil)
(defvar katerina nil)
(defvar ann nil)

;;;It is the  TOP function which reads the dape of the model  (about 2 - 3 hours) and keeps the result in
 the
;;;    global variable  katerina
;;;;


(defun reset-the-EJA18V-tape ()
  (setq *model* nil)
  (setq *var1* nil)
  (setq *var2* nil)
  (setq paul1 nil)
  (setq paul nil)
  (setq katerina nil)
  (setq ann nil))


(defun read-the-EJA18V-tape-of-the-specific-model (mepg10t.zbe-file    xwuc.dat-file model model)

  (reset-the-EJA18V-tape)

  (setq *model* model)
  (read-file-to-another-file-AUTOMATICALLY-1 mepg10t.zbe-file 240)
  (read-file-to-another-file-AUTOMATICALLY-2  xwuc.dat-file 129)   ;;;; ASSIGN  *var1* *var2*
  (read-the-EJA18-data-AUTOMATICALLY-1  *var1*)
  (read-the-EJA18-data-AUTOMATICALLY-2  *var2*)            ;;;; ASSIGN  *paul1* *ann*
  (setq paul (put-together-the-two-physical-records-to-create-the-LOGICAL-EEJA18V-record  paul1))
  (setq katerina (NOW-LINK-the-data-from-mepg01tDOTzbe-with-XWUCdotDAT-in-one-LOGICAL-and-PHYSICAL-file
paul ann)))
!!!!!!!!!!!!
!!!!!!!!!!!!
!!!!!!!!!!!!!      (translate-make-the-link-TOP katerina)    ;;;the file is "....>statice-R8.dattt"
!!!!!!!!!!!!

       !!!!!!!!!!!!!!
```

```
 INSTRUCTION DRAWING LABELS              ARTHED ETBAA10028      B                    PIECE      NTLR9278
 500 061189                    A


(defun read-file-to-another-file-AUTOMATICALLY-1 (input-filename record-number global-VAR) ;;;; output-fi
lename) ;;240
  ;;;;(with-open-file (file-stream output-filename :direction :output)
    (read-the-file-AUTOMATICALLY-1 input-filename record-number global-VAR)) ;;;;file-stream))



(defun read-the-file-AUTOMATICALLY-1 (input-filename record-number global-VAR);;;; to-stream)
  (with-open-file (tape-stream input-filename
                                :direction :input)

    (loop as record = (read-records-from-file-AUTOMATICALLY tape-stream record-number)
          while record
          do
      (push record  *var1*))))  ;;;;;FOR SOME STRANGE REASON IT DOES NOT push record TO global-VAR but on
ly to the SPECIFIC
;;;;;either *var1*  OR   *var2*

;;  ************************************************************

(defun read-file-to-another-file-AUTOMATICALLY-2 (input-filename record-number global-VAR) ;;;; output-fi
lename)  ;;129
  ;;;;(with-open-file (file-stream output-filename :direction :output)
    (read-the-file-AUTOMATICALLY-2 input-filename record-number global-VAR)) ;;;;file-stream))



(defun read-the-file-AUTOMATICALLY-2 (input-filename record-number global-VAR);;;; to-stream)
  (with-open-file (tape-stream input-filename
                                :direction :input)

    (loop as record = (read-records-from-file-AUTOMATICALLY tape-stream record-number)
          while record
          do
      (push record  *var2*))))  ;;;;;FOR SOME STRANGE REASON IT DOES NOT push record TO global-VAR but on
ly to the SPECIFIC
;;;;;either *var1*  OR   *var2*



(defun read-records-from-file-AUTOMATICALLY (stream record-number)
  ;; Skip past any non digit characters.

  (loop
        as char = (send stream :tyi)
        when (null char) do (return-from read-records-from-file-AUTOMATICALLY nil)       ;
        until (and (not (digit-char-p  char)) (not (char-equal char #\page)) (not (char-equal char #\retu
rn)))
        finally
          (send stream :untyi char))


  (let* ((record-length  record-number)
         (record (make-array  record-length)))
    (send  stream :string-in t record)
   (from-array-to-string-AUTOMATICALLY record record-number )
  )
)


(defun from-array-to-string-AUTOMATICALLY (array number)
  (let ((result ""))
    (loop for i from 0 to (- number 1)
          do
      (setq result (string-append (aref array i) result)))
    (reverse result)))
```

```
;*****************************************************************************************************
****************
;*****************************************************************************************************
****************


(defun read-the-EJA18-data-AUTOMATICALLY-1 (ls-of-strings)   ;;ie  *var1*    ;;;;;;;(input-filename   out
put-filename )
       ;;;"atc-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish.dat"
       ;;;;  "atc-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish-LISTS-NOW.dat"

;;;;;; We now take as input file the file
;;;;;; "atc-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish.dat" AS A RESULT OF THE FUNCT
ION:
;Command: (read-file-to-another-file "atc-0:>takis>statice>mepg01tDOTzbe-only-a-very-small-part-IE-from-M
IRRORS1109AA-to-1110DA.d
;at" "atc-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish.dat")
;NIL
;Command:
; ;The function :  "read-file-to-another-file"     READS THE ORIGINAL FILE FROM THE TAPE FROM THE vax  AN
D creates STRINGS
;;;;;;   PUTING OUT the   page  AND return.

 (let ((record-number 0) (result nil))
;  (with-open-file (output-stream output-filename :direction :output)
;    (with-open-file (input-stream input-filename :direction :input)
      (loop for i in     ls-of-strings ;;;;;;;;repeat 5
                                       ;;;;;;;;as rec = (read  input-stream   nil :eof)
                                       ;;;;;;;;until (eq rec :eof)
        do
        (setq record-number (+ record-number 1))
        (let ((parsed-rec
                (cond
                   ((oddp record-number) (parse-record  i   *EJA18-1st-Physical-record-FORMAT*))
                   (t (parse-record  i   *EJA18-2nd-Physical-record-FORMAT*)))))

          (push  parsed-rec result))) (setq paul1 result)))


(defun read-the-EJA18-data-AUTOMATICALLY-2 (ls-of-strings)   ;;ie  *var2*    ;;;;;;;(input-filename   out
put-filename )
       ;;;"atc-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish.dat"
       ;;;;  "atc-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish-LISTS-NOW.dat"


 (let ((record-number 0) (result nil))
   (loop for i in     ls-of-strings ;;;;;;;;repeat 5
        do
        (setq record-number (+ record-number 1))
        (let ((parsed-rec (parse-record i   *MODEL-RANGE-USAGE-CONDITION-DATA* )))
          (push  parsed-rec result))) (setq  ann result)))




(defun parse-record (string format)
  (loop as string-left = string then (substring string-left length)
        for (field-name length) in format
        collect (list field-name (substring string-left 0 length))))


(defvar *EJA18-1st-Physical-record-FORMAT*
        '((:Continuation-Marker 1)
          (:User-ID 6) (:Transaction-ID 8) (:Request-ID 4) (:Processing-Date 6) (:Model-range/change 4)
          (:VPG-code 6) (:Major-Unit-Identifier 4) (:VPS 4) (:Part-Number 12) (:Fitment-Number 4) (:relea
se-responsibility 6)
          (:Part-Description 40) (:Fitment-Description 40) (:Design-Responsibility 6) (:Part-Indicator 1)
 (:status 1)
          (:Drawing-Indicator 1) (:Drawing-Number 12)  (:Drawing-Issue-Level 3) (:Material-Specification
20) (:unit-of-measure 5)
          (:rest 46)))  ;;;;  (:vax-characters   2)))
```

```
;          (:Safety-Indicators 3) (:Colour-Key-Indicator 1) (:Assembly-Complete-Indicator 1) (:Other-Fitme
nts-Indicators 1)
;          (:Next-Assembly-Category 2) (:Last-Part-Release-Number 12) (:Last-Part-Release-Date 6) (:Spare
34)
;          (:Termination-character 1)  (:vax-characters   2)))


(defvar *EJA18-2nd-Physical-record-FORMAT*
        '((:Continuation-Marker 1)
          (:Collaborator-Part-Number 25) (:UC-number 6) (:US-number 2) (:Action-code/Release-stage-in 2)
(:Dep-in 8)
          (:Release-number-in 12) (:Release-Date-In 6) (:PIDATE-IN 6) (:Action-code/Release-stagout 2) (:
DEP-out 8)
          (:Release-number-out  12) (:Release-Date-out 6) (:PIDATE-OUT 6) (:Replaces-Part-Number  12) (:R
eplaced-by-Part-Number 12)
          (:MP17 12) (:Source-Code 45) (:usage-Quantity 8) (:Non-Specific-Quantity 3) (:other-Sources-Ind
 1) (:spare 44)
          (:Termination-Character 1)))  ;;;;;;; (:vax-characters   2)))


(defvar   *MODEL-RANGE-USAGE-CONDITION-DATA*
          '((:MODEL-RANGE 2)
            (:model-change 2) (:model-range-description 20) (:UC-table-identifier 7) (:Usage-Condition-Nu
mber 6)
            (:UC-approved-indicator 1)  (:feature-codes 90) ;;;;;;(:sign 1) (:feature-code 4)
            (:termination-character 1))) ;;; (:vax-characters   2))) ;;;;;;;THEY ARE  returns.

;;*****************************************************************************************************
*****************
;;*****************************************************************************************************
*****************
;;*****************************************************************************************************
*****************



;*****************************************************************************************************
***************
;;;;; BECAUSE of the problem to print the result into the
;;;;;;;;  "atc-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish-LISTS-NOW.dat"    FILE.  I
 do
;;;;;;(setq takis (read-the-EJA18-data "atc-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbi
sh.dat"))


;Command: (setq paul (put-together-the-two-physical-records-to-create-the-LOGICAL-EEJA18V-record takis)
;((((:CONTINUATION-MARKER "B") (:COLLABORATOR-PART-NUMBER "0042Z-SK4 -E010          ") (:UC-NUMBER "XW0001
") (:US-NUMBER "99")
;  (:ACTION-CODE/RELEASE-STAGE-IN "IP") (:DEP-IN "XW9B1   ") (:RELEASE-NUMBER-IN "R4389   500 ") (:RELEAS
E-DATE-IN "050589")
;  (:PIDATE-IN "      ") (:ACTION-CODE/RELEASE-STAGOUT "  ") (:DEP-OUT "        ") (:RELEASE-NUMBER-OUT "
            ")
;  (:RELEASE-DATE-OUT "      ") (:PIDATE-OUT "      ") (:REPLACES-PART-NUMBER "             ")
;  (:REPLACED-BY-PART-NUMBER "             ") (:MP17 "            ")
;  (:SOURCE-CODE "LXL                                          ") (:USAGE-QUANTITY "0000.000") (:NON-SPEC
IFIC-QUANTITY "R/O")
;  (:OTHER-SOURCES-IND "N") (:SPARE "                                                ") (:TERMINATION-CHARACT
ER "B")
;  (:CONTINUATION-MARKER "A") (:USER-ID "LMLS01") (:TRANSACTION-ID "EJA18V  ") (:REQUEST-ID "ST01") (:PRO
CESSING-DATE "210290")
;Back to Lisp Top Level in Dynamic Lisp Listener 1.
;
;Command: (sys:dump-forms-to-file "atc-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish-LI
STS-NOW.dat" '((setq pau
;l ',paul)))
;#P"ATC-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish-LISTS-NOW.dat.1"
;(#P"ATC-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish-LISTS-NOW.dat.1" . 2853145445)
;Command: (setq paul nil)
;NIL



(defun put-together-the-two-physical-records-to-create-the-LOGICAL-EEJA18V-record (ls) ;;;;;;;;; ie. taki
```

```
=
  (prog (result  TEMP)
        (setq temp ls)
        LOOP
        (cond
          ((null temp) (return result))
          (t (setq result (cons (append (car temp) (cadr temp))
                                result))))
        (setq temp (cddr temp))
        (go LOOP)))

;*****************************************************************************************************
****************
;;Command:
;         (setq katerina (NOW-LINK-the-data-from-mepg01tDOTzbe-with-XWUCdotDAT-in-one-LOGICAL-and-PHYSICA
L-file  paul ann
;)
;
;( ((:FEATURE-CODES "+A10X+A20W     +A30W+A50B+D92C
     ")
;  (:CONTINUATION-MARKER "B") (:COLLABORATOR-PART-NUMBER "91567-SE0 -0032          ") (:UC-NUMBER "XW2975
") (:US-NUMBER "99")
;  (:ACTION-CODE/RELEASE-STAGE-IN "IP") (:DEP-IN "XW9A5    ") (:RELEASE-NUMBER-IN "W2473   300 ") (:RELEAS
E-DATE-IN "241088")
;  (:PIDATE-IN "       ") (:ACTION-CODE/RELEASE-STAGOUT "  ") (:DEP-OUT "         ") (:RELEASE-NUMBER-OUT "
            ")
;  (:RELEASE-DATE-OUT "       ") (:PIDATE-OUT "       ") (:REPLACES-PART-NUMBER "             ")
;  (:REPLACED-BY-PART-NUMBER "             ") (:MP17 "             ")
;  (:SOURCE-CODE "LDL                                          ") (:USAGE-QUANTITY "0001.000") (:NON-SPEC
IFIC-QUANTITY "   ")
;  (:OTHER-SOURCES-IND "N") (:SPARE "                                                  ") (:TERMINATION-CHARACT
ER "B")
;  (:CONTINUATION-MARKER "A") (:USER-ID "LMLS01") (:TRANSACTION-ID "EJA18V  ") (:REQUEST-ID "ST01") (:PRO
CESSING-DATE "210290")
;  (:MODEL-RANGE/CHANGE "XW  ") (:VPG-CODE "1110EA") (:MAJOR-UNIT-IDENTIFIER "    ") (:VPS "0605") (:PART
-NUMBER "DJP8758     ")
;  (:FITMENT-NUMBER "0005") (:RELEASE-RESPONSIBILITY "**BDAA") (:PART-DESCRIPTION "BAND HARN (UND-13)
                ")
;  (:FITMENT-DESCRIPTION "MOTOR ASSY WINDOW REGULATOR - L FRONT DR") (:DESIGN-RESPONSIBILITY "HMSP**") (:
PART-INDICATOR " ")
;  (:STATUS "T") (:DRAWING-INDICATOR " ") (:DRAWING-NUMBER "DJP8758     ") (:DRAWING-ISSUE-LEVEL "  A")
;  (:MATERIAL-SPECIFICATION "                    ") (:UNIT-OF-MEASURE "PIECE")
;  (:REST "     YWAR9613   501 040190                    A"))
; ((:FEATURE-CODES "+A10X+A20W     +A50B+D92B
     ")
;  (:CONTINUATION-MARKER "B") (:COLLABORATOR-PART-NUMBER "91567-SE0 -0032          ") (:UC-NUMBER "XW3004
") (:US-NUMBER "99")
;  (:ACTION-CODE/RELEASE-STAGE-IN "IP") (:DEP-IN "XW9A1    ") (:RELEASE-NUMBER-IN "VAA05   313 ") (:RELEAS
E-DATE-IN "131187")
;  (:PIDATE-IN "       ") (:ACTION-CODE/RELEASE-STAGOUT "  ") (:DEP-OUT "         ") (:RELEASE-NUMBER-OUT "
            ")
;  (:RELEASE-DATE-OUT "       ") (:PIDATE-OUT "       ") (:REPLACES-PART-NUMBER "             ")
;  (:REPLACED-BY-PART-NUMBER "             ") (:MP17 "             ")
;  (:SOURCE-CODE "LDL                                          ") (:USAGE-QUANTITY "0001.000") (:NON-SPEC
IFIC-QUANTITY "   ")
;  (:OTHER-SOURCES-IND "N") (:SPARE "                                                  ") (:TERMINATION-CHARACT
ER "B")
;  (:CONTINUATION-MARKER "A") (:USER-ID "LMLS01") (:TRANSACTION-ID "EJA18V  ") (:REQUEST-ID "ST01") (:PRO
CESSING-DATE "210290")
;  (:MODEL-RANGE/CHANGE "XW  ") (:VPG-CODE "1110EA") (:MAJOR-UNIT-IDENTIFIER "    ") (:VPS "0605") (:PART
-NUMBER "DJP8758     ")
;  (:FITMENT-NUMBER "0005") (:RELEASE-RESPONSIBILITY "**BDAA") (:PART-DESCRIPTION "BAND HARN (UND-13)
                ")
;  (:FITMENT-DESCRIPTION "MOTOR ASSY WINDOW REGULATOR - L FRONT DR") (:DESIGN-RESPONSIBILITY "HMSP**") (:
PART-INDICATOR " ")
;  (:STATUS "T") (:DRAWING-INDICATOR " ") (:DRAWING-NUMBER "DJP8758     ") (:DRAWING-ISSUE-LEVEL "  A")
;  (:MATERIAL-SPECIFICATION "                    ") (:UNIT-OF-MEASURE "PIECE")
;  (:REST "     YWAR9613   501 040190                    A"))      ....................................
.........................

;Command: (sys:dump-forms-to-file "atc-0:>takis>statice>data-EJA18V-ALL-ie-THE-mepg01tDOTzbe-AND-xwucDOTd
at-TOGETHER-LINKED.dat"
;`((setq  katerina ',katerina)))
;#P"ATC-0:>takis>statice>data-EJA18V-ALL-ie-THE-mepg01tDOTzbe-AND-xwucDOTdat-TOGETHER-LINKED.dat.1"
;(#P"ATC-0:>takis>statice>data-EJA18V-ALL-ie-THE-mepg01tDOTzbe-AND-xwucDOTdat-TOGETHER-LINKED.dat.1" . 28
53149921)
```

```
;Command:

(defun NOW-LINK-the-data-from-mepg01tDOTzbe-with-XWUCdotDAT-in-one-LOGICAL-and-PHYSICAL-file (mepg01t.zbe
 XWUC.dat)
  (let ((result nil))
    (loop for list in mepg01t.zbe
          do
      (let* ((UC-code (cadr (assoc ':UC-NUMBER list)))
             (feature-codes (given-the-Usage-condition-code-return-the-data-list-in-ANN-ie-XWUC.dat UC-co
de XWUC.dat)))
        (push (append (list (list ':FEATURE-CODES feature-codes))
                      list) result)))
result))



;Command: (given-the-Usage-condition-code-return-the-data-list-in-ANN-ie-XWUC.dat "xw0001" Ann)
;"+A10X+A20W                                                                                    "
;Command:

(defun given-the-Usage-condition-code-return-the-data-list-in-ANN-ie-XWUC.dat (a-string XWUC.dat)
  (block TAKIS
    (loop for list in XWUC.dat
          do
      (cond
        ((string-equal a-string (cadr (assoc ':USAGE-CONDITION-NUMBER list)))
                    (return-from TAKIS (cadr (assoc  ':FEATURE-CODES list))))))
    nil))>
```

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER; Base: 10 -*-

       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;ALMLS01EJA18V  ST01210290XW  1001AA    0000BAA10028     0001ARTHEDINSTR DRG-LABELS          BON LOCK PLTFM
 INSTRUCTION DRAWING LABELS                ARTHED ETBAA10028      B                      PIECE      NTLR9278
 500 061189                   A
;ALMLS01EJA18V  ST01210290XW  1001AA    0000BAA10028     0001ARTHEDINSTR DRG-LABELS          BON LOCK PLTFM
 INSTRUCTION DRAWING LABELS                ARTHED ETBAA10028      B                      PIECE      NTLR9278
 500 061189                   A


(defun read-file-to-another-file (input-filename output-filename)
  (with-open-file (file-stream output-filename :direction :output)
    (read-the-file input-filename file-stream)))




(defun read-the-file (input-filename to-stream)
  (with-open-file (tape-stream input-filename
                               :direction :input)

    (loop as record = (read-records-from-file tape-stream)
          while record do
      (when to-stream
        (print record to-stream)))))


(defun read-records-from-file (stream)
  ;; Skip past any non digit characters.

  (loop
        as char = (send stream :tyi)
        when (null char) do (return-from read-records-from-file nil)     ;
        until (and (not (digit-char-p  char)) (not (char-equal char #\page)) (not (char-equal char #\retu
rn)))
        finally
          (send stream :untyi char))


  (let* ((record-length  240)
         (record (make-array  record-length)))
    (send  stream :string-in t record)
   (from-array-to-string record 240 )
  )
)


(defun from-array-to-string (array number)
  (let ((result ""))
    (loop for i from 0 to (- number 1)
          do
      (setq result (string-append (aref array i) result)))
    (reverse result)))



;*****************************************************************************************************
****************




(defun read-the-EJA18-data (input-filename   output-filename )
       ;;;"atc-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish.dat"
       ;;;;  "atc-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish-LISTS-NOW.dat"

;;;;;; We now take as input file the file
;;;;;; "atc-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish.dat" AS A RESULT OF THE FUNCT
```

```
ION:
;Command: (read-file-to-another-file "atc-0:>takis>statice>mepg01tDOTzbe-only-a-very-small-part-IE-from-M
IRRORS1109AA-to-1110DA.d
;at" "atc-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish.dat")
;NIL
;Command:
; ;The function :  "read-file-to-another-file"     READS THE ORIGINAL FILE FROM THE TAPE FROM THE vax  AN
D creates STRINGS
;;;;;;   PUTING OUT the   page  AND return.

 (let ((record-number 0));; (result nil))
  (with-open-file (output-stream output-filename :direction :output)
    (with-open-file (input-stream input-filename :direction :input)
      (loop ;;;;;;;;repeat 5
        as rec = (read  input-stream   nil :eof)
        until (eq rec :eof)
        do
        (setq record-number (+ record-number 1))
        (let ((parsed-rec
                (cond
                   ((oddp record-number) (parse-record  rec   *EJA18-1st-Physical-record-FORMAT*))
                   (t (parse-record  rec   *EJA18-2nd-Physical-record-FORMAT*)))))

;          (push  parsed-rec result)))) result))

            (print parsed-rec output-filename)))))))


(defun parse-record (string format)
  (loop as string-left = string then (substring string-left length)
        for (field-name length) in format
        collect (list field-name (substring string-left 0 length))))


(defvar *EJA18-1st-Physical-record-FORMAT*
        '((:Continuation-Marker 1)
          (:User-ID 6) (:Transaction-ID 8) (:Request-ID 4) (:Processing-Date 6) (:Model-range/change 4)
          (:VPG-code 6) (:Major-Unit-Identifier 4) (:VPS 4) (:Part-Number 12) (:Fitment-Number 4) (:relea
se-responsibility 6)
          (:Part-Description 40) (:Fitment-Description 40) (:Design-Responsibility 6) (:Part-Indicator 1)
 (:status 1)
          (:Drawing-Indicator 1) (:Drawing-Number 12)  (:Drawing-Issue-Level 3) (:Material-Specification
20) (:unit-of-measure 5)
          (:rest 46)))  ;;;; (:vax-characters   2)))

;         (:Safety-Indicators 3) (:Colour-Key-Indicator 1) (:Assembly-Complete-Indicator 1) (:Other-Fitme
nts-Indicators 1)
;         (:Next-Assembly-Category 2) (:Last-Part-Release-Number 12) (:Last-Part-Release-Date 6) (:Spare
34)
;         (:Termination-character 1)  (:vax-characters   2)))


(defvar *EJA18-2nd-Physical-record-FORMAT*
        '((:Continuation-Marker 1)
          (:Collaborator-Part-Number 25) (:UC-number 6) (:US-number 2) (:Action-code/Release-stage-in 2)
(:Dep-in 8)
          (:Release-number-in 12) (:Release-Date-In 6) (:PIDATE-IN 6) (:Action-code/Release-stagout 2) (:
DEP-out 8)
          (:Release-number-out  12) (:Release-Date-out 6) (:PIDATE-OUT 6) (:Replaces-Part-Number  12) (:R
eplaced-by-Part-Number 12)
          (:MP17 12) (:Source-Code 45) (:usage-Quantity 8) (:Non-Specific-Quantity 3) (:other-Sources-Ind
 1) (:spare 44)
          (:Termination-Character 1)))  ;;;;;;; (:vax-characters   2)))

;;*****************************************************************************************************
*****************
;;*****************************************************************************************************
*****************
;;*****************************************************************************************************
*****************


;(defun read-file-to-another-file (input-filename output-filename)
;;;;;;;  "tom:atc:[katerina]xwuc.dat"  "atc-0:>takis>statice>xwucDOTdat-after-out-the-rubbish-IE-page-and
-return.dat"
```

```
;  (with-open-file (file-stream output-filename :direction :output)
;    (read-the-file input-filename file-stream)))
;
;
;
;(defun read-the-file (input-filename to-stream)
;  (with-open-file (tape-stream input-filename
;                                :direction :input)
;
;    (loop as record = (read-records-from-file tape-stream)
;         while record do
;      (when to-stream
;       (print record to-stream)))))
;

(defun read-records-from-file (stream)
  ;; Skip past any non digit characters.

  (loop
        as char = (send stream :tyi)
        when (null char) do (return-from read-records-from-file nil)    ;
        until (and (not (digit-char-p  char)) (not (char-equal char #\page)) (not (char-equal char #\retu
rn)))
        finally
          (send stream :untyi char))


  (let* ((record-length  129)
         (record (make-array  record-length)))
    (send  stream :string-in t record)
   (from-array-to-string record 129 )
  )
)


;(defun from-array-to-string (array number)
;  (let ((result ""))
;    (loop for i from 0 to (- number 1)
;         do
;      (setq result (string-append (aref array i) result)))
;    (reverse result)))
;*****************************************************************************************************
****************


(defun read-the-EJA18-data-other (input-filename  output-filename)

;;;;;;"atc-0:>takis>statice>xwucDOTdat-after-out-the-rubbish-IE-page-and-return.dat"
;;;;;;"atc-0:>takis>statice>xwucDOTdat-after-out-the-rubbish-IE-page-and-return-LISTS-NOW.dat"

  (with-open-file (output-stream output-filename :direction :output)
    (with-open-file (input-stream input-filename :direction :input)
        (loop
              as rec = (read input-stream  nil :eof)
              until (eq rec :eof)
              as parsed-rec = (parse-record  rec   *MODEL-RANGE-USAGE-CONDITION-DATA* )
           do   ;;collect
             (print parsed-rec output-stream)))))




(defvar    *MODEL-RANGE-USAGE-CONDITION-DATA*
           '((:MODEL-RANGE 2)
             (:model-change 2) (:model-range-description 20) (:UC-table-identifier 7) (:Usage-Condition-Nu
mber 6)

             (:UC-approved-indicator 1)  (:feature-codes 90) ;;;;;;(:sign 1) (:feature-code 4)
             (:termination-character 1))) ;;; (:vax-characters   2))) ;;;;;;;THEY ARE  returns.


;*****************************************************************************************************
****************#
```

```
;*****************************************************************************************************
***************
;;;;; BECAUSE of the problem to print the result into the
;;;;;;;;  "atc-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish-LISTS-NOW.dat"    FILE.  I
 do
;;;;;;(setq takis (read-the-EJA18-data "atc-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbi
sh.dat"))


;Command: (setq paul (put-together-the-two-physical-records-to-create-the-LOGICAL-EEJA18V-record takis)
;((((:CONTINUATION-MARKER "B") (:COLLABORATOR-PART-NUMBER "0042Z-SK4 -E010          ") (:UC-NUMBER "XW0001
") (:US-NUMBER "99")
;  (:ACTION-CODE/RELEASE-STAGE-IN "IP") (:DEP-IN "XW9B1   ") (:RELEASE-NUMBER-IN "R4389   500 ") (:RELEAS
E-DATE-IN "050589")
;  (:PIDATE-IN "       ") (:ACTION-CODE/RELEASE-STAGOUT "  ") (:DEP-OUT "         ") (:RELEASE-NUMBER-OUT "
            ")
;  (:RELEASE-DATE-OUT "       ") (:PIDATE-OUT "       ") (:REPLACES-PART-NUMBER "             ")
;  (:REPLACED-BY-PART-NUMBER "             ") (:MP17 "             ")
;  (:SOURCE-CODE "LXL                                                ") (:USAGE-QUANTITY "0000.000") (:NON-SPEC
IFIC-QUANTITY "R/O")
;  (:OTHER-SOURCES-IND "N") (:SPARE "                                                          ") (:TERMINATION-CHARACT
ER "B")
;  (:CONTINUATION-MARKER "A") (:USER-ID "LMLS01") (:TRANSACTION-ID "EJA18V  ") (:REQUEST-ID "ST01") (:PRO
CESSING-DATE "210290")
;Back to Lisp Top Level in Dynamic Lisp Listener 1.
;
;Command: (sys:dump-forms-to-file "atc-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish-LI
STS-NOW.dat" '((setq pau
;l ',paul)))
;#P"ATC-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish-LISTS-NOW.dat.1"
;(#P"ATC-0:>takis>statice>data-mepg01tDOTzbe-EJA18-after-out-the-rubbish-LISTS-NOW.dat.1" . 2853145445)
;Command: (setq paul nil)
;NIL



(defun put-together-the-two-physical-records-to-create-the-LOGICAL-EEJA18V-record (ls) ;;;;;;;;; ie. taki
s
  (prog (result  TEMP)
        (setq temp ls)
        LOOP
        (cond
          ((null temp) (return result))
          (t (setq result (cons (append (car temp) (cadr temp))
                                result))))
        (setq temp (cddr temp))
        (go LOOP)))

;*****************************************************************************************************
***************
;;Command:
;          (setq katerina (NOW-LINK-the-data-from-mepg01tDOTzbe-with-XWUCdotDAT-in-one-LOGICAL-and-PHYSICA
L-file  paul ann
;)
;
;( ((:FEATURE-CODES "+A10X+A20W      +A30W+A50B+D92C
      ")
;  (:CONTINUATION-MARKER "B") (:COLLABORATOR-PART-NUMBER "91567-SE0 -0032          ") (:UC-NUMBER "XW2975
") (:US-NUMBER "99")
;  (:ACTION-CODE/RELEASE-STAGE-IN "IP") (:DEP-IN "XW9A5   ") (:RELEASE-NUMBER-IN "W2473   300 ") (:RELEAS
E-DATE-IN "241088")
;  (:PIDATE-IN "       ") (:ACTION-CODE/RELEASE-STAGOUT "  ") (:DEP-OUT "         ") (:RELEASE-NUMBER-OUT "
            ")
;  (:RELEASE-DATE-OUT "       ") (:PIDATE-OUT "       ") (:REPLACES-PART-NUMBER "             ")
;  (:REPLACED-BY-PART-NUMBER "             ") (:MP17 "             ")
;  (:SOURCE-CODE "LDL                                                ") (:USAGE-QUANTITY "0001.000") (:NON-SPEC
IFIC-QUANTITY "   ")
;  (:OTHER-SOURCES-IND "N") (:SPARE "                                                          ") (:TERMINATION-CHARACT
ER "B")
;  (:CONTINUATION-MARKER "A") (:USER-ID "LMLS01") (:TRANSACTION-ID "EJA18V  ") (:REQUEST-ID "ST01") (:PRO
CESSING-DATE "210290")
;  (:MODEL-RANGE/CHANGE "XW  ") (:VPG-CODE "1110EA") (:MAJOR-UNIT-IDENTIFIER "    ") (:VPS "0605") (:PART
-NUMBER "DJP8758     ")
;  (:FITMENT-NUMBER "0005") (:RELEASE-RESPONSIBILITY "**BDAA") (:PART-DESCRIPTION "BAND HARN (UND-13)
```

```
                 ")
;  (:FITMENT-DESCRIPTION "MOTOR ASSY WINDOW REGULATOR - L FRONT DR") (:DESIGN-RESPONSIBILITY "HMSP**") (:
PART-INDICATOR " ")
;  (:STATUS "T") (:DRAWING-INDICATOR " ") (:DRAWING-NUMBER "DJP8758      ") (:DRAWING-ISSUE-LEVEL "  A")
;  (:MATERIAL-SPECIFICATION "                    ") (:UNIT-OF-MEASURE "PIECE")
;  (:REST "     YWAR9613   501 040190                    A"))
; ((:FEATURE-CODES "+A10X+A20W     +A50B+D92B
     ")
;  (:CONTINUATION-MARKER "B") (:COLLABORATOR-PART-NUMBER "91567-SE0 -0032          ") (:UC-NUMBER "XW3004
") (:US-NUMBER "99")
;  (:ACTION-CODE/RELEASE-STAGE-IN "IP") (:DEP-IN "XW9A1    ") (:RELEASE-NUMBER-IN "VAA05   313 ") (:RELEAS
E-DATE-IN "131187")
;  (:PIDATE-IN "       ") (:ACTION-CODE/RELEASE-STAGOUT "  ") (:DEP-OUT "         ") (:RELEASE-NUMBER-OUT "
            ")
;  (:RELEASE-DATE-OUT "       ") (:PIDATE-OUT "       ") (:REPLACES-PART-NUMBER "             ")
;  (:REPLACED-BY-PART-NUMBER "             ") (:MP17 "             ")
;  (:SOURCE-CODE "LDL                                          ") (:USAGE-QUANTITY "0001.000") (:NON-SPEC
IFIC-QUANTITY "    ")
;  (:OTHER-SOURCES-IND "N") (:SPARE "                                                  ") (:TERMINATION-CHARACT
ER "B")
;  (:CONTINUATION-MARKER "A") (:USER-ID "LMLS01") (:TRANSACTION-ID "EJA18V  ") (:REQUEST-ID "ST01") (:PRO
CESSING-DATE "210290")
;  (:MODEL-RANGE/CHANGE "XW  ") (:VPG-CODE "1110EA") (:MAJOR-UNIT-IDENTIFIER "    ") (:VPS "0605") (:PART
-NUMBER "DJP8758      ")
;  (:FITMENT-NUMBER "0005") (:RELEASE-RESPONSIBILITY "**BDAA") (:PART-DESCRIPTION "BAND HARN (UND-13)
                 ")
;  (:FITMENT-DESCRIPTION "MOTOR ASSY WINDOW REGULATOR - L FRONT DR") (:DESIGN-RESPONSIBILITY "HMSP**") (:
PART-INDICATOR " ")
;  (:STATUS "T") (:DRAWING-INDICATOR " ") (:DRAWING-NUMBER "DJP8758      ") (:DRAWING-ISSUE-LEVEL "  A")
;  (:MATERIAL-SPECIFICATION "                    ") (:UNIT-OF-MEASURE "PIECE")
;  (:REST "     YWAR9613   501 040190                    A"))      ....................................
.........................

;Command: (sys:dump-forms-to-file "atc-0:>takis>statice>data-EJA18V-ALL-ie-THE-mepg01tDOTzbe-AND-xwucDOTd
at-TOGETHER-LINKED.dat"
;`((setq  katerina ',katerina)))
;#P"ATC-0:>takis>statice>data-EJA18V-ALL-ie-THE-mepg01tDOTzbe-AND-xwucDOTdat-TOGETHER-LINKED.dat.1"
;(#P"ATC-0:>takis>statice>data-EJA18V-ALL-ie-THE-mepg01tDOTzbe-AND-xwucDOTdat-TOGETHER-LINKED.dat.1" . 28
53149921)
;Command:

(defun NOW-LINK-the-data-from-mepg01tDOTzbe-with-XWUCdotDAT-in-one-LOGICAL-and-PHYSICAL-file (mepg01t.zbe
 XWUC.dat)
  (let ((result nil))
    (loop for list in mepg01t.zbe
          do
      (let* ((UC-code (cadr (assoc ':UC-NUMBER list)))
             (feature-codes (given-the-Usage-condition-code-return-the-data-list-in-ANN-ie-XWUC.dat UC-co
de XWUC.dat)))
        (push (append (list (list ':FEATURE-CODES feature-codes))
                      list) result)))
result))



;Command: (given-the-Usage-condition-code-return-the-data-list-in-ANN-ie-XWUC.dat "xw0001" Ann)
;"+A10X+A20W                                                                                  "
;Command:

(defun given-the-Usage-condition-code-return-the-data-list-in-ANN-ie-XWUC.dat (a-string XWUC.dat)
  (block TAKIS
    (loop for list in XWUC.dat
          do
      (cond
        ((string-equal a-string (cadr (assoc ':USAGE-CONDITION-NUMBER list)))
                    (return-from TAKIS (cadr (assoc  ':FEATURE-CODES list))))))
    nil))
```

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER; Base: 10 -*-

       ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;ALMLS01EJA18V  ST01210290XW  1001AA    0000BAA10028    0001ARTHEDINSTR DRG-LABELS          BON LOCK PLTFM
 INSTRUCTION DRAWING LABELS                ARTHED ETBAA10028      B                    PIECE      NTLR9278
 500 061189                  A
;ALMLS01EJA18V  ST01210290XW  1001AA    0000BAA10028    0001ARTHEDINSTR DRG-LABELS          BON LOCK PLTFM
 INSTRUCTION DRAWING LABELS                ARTHED ETBAA10028      B                    PIECE      NTLR9278
 500 061189                  A


(defun read-file-to-another-file (input-filename output-filename)
  (with-open-file (file-stream output-filename :direction :output)
    (read-the-file input-filename file-stream)))




(defun read-the-file (input-filename to-stream)
  (with-open-file (tape-stream input-filename
                               :direction :input)

    (loop as record = (read-records-from-file tape-stream)
          while record do
      (when to-stream
        (print record to-stream)))))


(defun read-records-from-file (stream)
  ;; Skip past any non digit characters.

  (loop
        as char = (send stream :tyi)
        when (null char) do (return-from read-records-from-file nil)    ;
        until (and (not (digit-char-p  char)) (not (char-equal char #\page)) (not (char-equal char #\retu
rn)))
        finally
          (send stream :untyi char))


  (let* ((record-length  240)
         (record (make-array  record-length)))
    (send  stream :string-in t record)
   (from-array-to-string record 240 )
  )
)


(defun from-array-to-string (array number)
  (let ((result ""))
    (loop for i from 0 to (- number 1)
          do
      (setq result (string-append (aref array i) result)))
    (reverse result)))



;*****************************************************************************************************
****************
```

```
;;;-*- mode: lisp ; syntax: common-lisp ; package:  austin-rover-*-

;*********************************************************************************************
;;;;;In the following ,until the 4 rows of STARS, the progam which is written  CONNECTS  (translates) the
 global variable
;;;;;; Katerina comming from the file : "........read-the-EJA18V-AUTOMATICALLY.lisp", to the format the R
8 global variable
;;;;;; in this file has. .... ...in order  to continue the elaboration of the EJA18V tape and decides the
 MUSTS and MAUBES
;;;;;; features through  the  ALGORITHM later.
;;;;;


;&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
;; The function  which tranlates the global Katerina to the Glogal R8 is  translate-make-the-link-TOP
;;;;
;;;
;;;;;;;;Because I  did "dump-forms-to-file" and the variable Katerina loaded to the package user I do :
;;;;;;Command: katerina
;Use the value of USER::KATERINA? (Y, N, V, P, or G) [Abort] ,katerina
;Back to Lisp Top Level in Dynamic Lisp Listener 1.
;Command: (setq katerina user::katerina)

;;;;;;;;;;;;(setq R8
;(take-out-L-and-R (translate-make-the-link (cadr (assoc "1109aa" (separate-the-different-SSSVPGs-in-k
aterina  katerina) :test #'string-equal))))
;;;;^^^^^^^^^^^^^^^^^^^^^^  OR ^^^^^^^^^^^^^^^^^^^^^^
;Command: (setq R8 (PUT-OUT-ALL-RECORDS-OF-DATA-WITH-LIMITED-Eddditions
;(take-out-L-and-R (translate-make-the-link (cadr (assoc "1110da" (separate-the-different-SSSVPGs-in-k
aterina  katerina) :test
;#'string-equal)))))
;;;;;DO we need afterwards to  EXCLUDE the LIMITED EDDITIONS from the R8 ?? ?????
;;;;;;; YES . BEcause the way the MUSTs work is that they must be 100% . Also, with the limited edditions
 the MUSTs may become
;;;; OPTIONAL (MAYBES). ie .  LOOk at the file " data-manually-METRO-EJA18V-seats-and-mirrors.dat >takis>
statice ATC-0: "
;;;;;;;;;
;;;;; ("SEAT-FRONT COMPLETE   " "MANUAL          " "HAD10159" ("R03") "+R03I")  ;;; NO "B51"



(defun translate-make-the-link-TOP (ls)       ;;;  katerina
  (let (;;(result nil))
        (temporary  (separate-the-different-SSSVPGs-in-katerina ls)))
       (loop for  VPG-code-and-SSSVPG-entries in  temporary
             do
             (let* ((VPG-CODE (car temporary))
                    (SSSVPG (second temporary))
                    (R8-before-put-out--L-and-R (translate-make-the-link  SSSVPG))
                    (R8 (take-out-L-and-R R8-before-put-out--L-and-R)))
               (the-runner R8)
               (put-the-EXPERIENCE-learned-into-the-Statice-Knowledge-Base-TOP  VPG-CODE (function-on-the
-top-of-the-runner))

))))


(defun   translate-make-the-link  (ls)       ;;;;  SSSVPG
  ;;;;;; as i have written the program in the file:   ".... >statice-R8.dat"  for a specific VPG.
  (let* ((takis (collect-only-FIRST-and-SECOND-description---PART-NUMBER---FEATURE-CODE  ls))
         (temp (push-the-FEATURE-GROUPS takis)))
temp)) ;;;;;;;; EACH  temp  IS an R8 as in the file:    ".... >statice-R8.dat"


(defun PUT-OUT-ALL-RECORDS-OF-DATA-WITH-LIMITED-Eddditions    (R8) ;;;;;;

;(("SEAT-FRONT COMPLETE   " "MANUAL          " "HAD10110" ("B51" "R03") "+B51A+R03R")
; ("SEAT-FRONT COMPLETE   " "MANUAL          " "HAD10111" ("B51" "R03") "+B51A+R03R")
; ("SEAT-FRONT COMPLETE   " "MANUAL          " "HAD10438" ("B51" "B45") "+B51N+B45B")
; ("COVER ASSY-SEAT-FRONT " "CUSHION         " "HCA10010" ("B51" "R03") "+B51A+R03R")
; ("SEAT-FR ...........................................................
```

```
  (let ((result nil))
    (loop for physical-record in R8
          do
      (cond ((notany #'(lambda (string) (string-search "R0" string)) (fourth physical-record))
             (push physical-record result))))
result))



(defun take-out-L-and-R (ls) ;(......("SEAT-FRONT COMPLETE   -R " "MANUAL          " "HAD10194XXX " ("B51"
 "B44") "+B51K+B44B")...)
  (let ((result nil))
    (loop for i in ls
          do
      (push
        (list (subseq (first i) 0 22) (second i) (subseq (third i) 0 8) (fourth i) (fifth i))
        result)) result))


(defun  separate-the-different-SSSVPGs-in-katerina  (ls)          ;;;;  katerina
  (let ((result nil) (record nil))
       (loop for i in ls
             do
             (let ((SSSVPG (second (assoc ':VPG-CODE i))))
                  (cond ((not (member SSSVPG record :test #'string-equal))
                              (push SSSVPG record)
                              (push
                                 (collect-data-of-the-same-VPG-coming-from-the-EJA18V-tape SSSVPG ls) resul
t)))))
result))


(defun  collect-data-of-the-same-VPG-coming-from-the-EJA18V-tape (SSSVPG-string ls)  ;;;;; "1110DA" kater
ina
  (let ((result nil))
       (loop for i in ls
             do
             (cond ((string-equal (second (assoc ':VPG-CODE i)) SSSVPG-string)
                         (push i result))))
(list SSSVPG-string result)))


;&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
(defvar *FEATURE-GROUPS-WHICH-REPRESENT-THE-SAME-logical-GROUP*
        ' (  (("B40" "B51")  "B51");;;; others if EXIST
))



;;; Firstly, there is not DISCIPLINE. ie there are "B40" codes...... instead  for only   "B51" s. for thi
s reason we
;;;; use the function   "evaluate-a-feature-group"
;;;
;Command: (let ((result nil))  (loop for i in katerina do (let ((feature-codes (cadr (assoc ':feature-cod
es i)))) (cond ((string-
;search "b40" feature-codes) (push feature-codes result))))) result)
;("+A10X+A20W     +A30N+B40C+B44B                                                      "
; "+A10X+A20W     +A30N+B40C+B44B                                                      "
; "+A10X+A20W     +A30N+A50A+B40B+B44R-B69D                                            "
; "+A10X+A20W     +A30N+A50B+B40B+B44R-B69D                                            "
; "+A10X+A20W     +A30N+A50B+B40B+B44R-B69D                                            "
; "+A10X+A20W     +A30N+A50B+B40B+B44R+B69D                                            "
; "+A10X+A20W     +A30N+A50B+B40B+B44R+B69D                                            "
; "+A10X+A20W     +A50A+A63Z+B40Y-B69D                                                 "
; "+A10X+A20W     +A50A+A63Z+B40Y-B69D                                                 "
; "+A10X+A20W     +A50B+A63Z+B40Y-B69D                                                 "
; "+A10X+A20W     +A50B+A63Z+B40Y-B69D                                                 "
; "+A10X+A20W     +A30N+A50A+B40B+B44R+B69D                                            "
; "+A10X+A20W     +A30N+A50A+B40B+B44R+B69D                                            "
; "+A10X+A20W     +A50A+B40Y+B44R+B69D                                                 "
; "+A10X+A20W     +A50A+B40Y+B44R+B69D                                                 "
; "+A10X+A20W     +A50B+B40Y+B44R+B69D                                                 "
```

```
; "+A10X+A20W      +A50B+B40Y+B44R+B69D                                    "
; "+A10X+A20W      +A50A+A63Z+B40F+B69D                                    "
; "+A10X+A20W      +A50A+A63Z+B40F+B69D                                    "
; "+A10X+A20W      +A50B+A63Z+B40F+B69D                                    "
; "+A10X+A20W      +A50B+A63Z+B40F+B69D                                    "
; "+A10X+A20W      +B40Y                                                   "
; "+A10X+A20W      +B25G+B40Y                                              "
; "+A10X+A20W      +B40Y+B49B-F48C                                         "
; "+A10X+A20W      +B40Y+B49B+F48C                                         "
; "+A10X+A20W      +B40Y+B49B-F48C                                         "
; "+A10X+A20W      +B40Y+B49B+F48C                                         "
; "+A10X+A20W      +B40Y                                                   "
; "+A10X+A20W      +B40Y+B49B                                              "
; "+A10X+A20W      +B40Y+B49B                                              "
; "+A10X+A20W      +B40Y+F48C                                              ")
;Command:

(defun  collect-only-FIRST-and-SECOND-description---PART-NUMBER---FEATURE-CODE  (ls)   ;;;;; katerina
  (let ((result nil))
       (loop for i in ls
             do
             (let ((part-description (second (assoc ':PART-DESCRIPTION i)))
                   (feature-codes (second (assoc ':FEATURE-CODES i))))
                  (push (list (subseq  part-description 0 25)
                              (subseq  part-description 25 40)
                              (second (assoc ':PART-NUMBER i))
                              feature-codes)
                        result)))
     result))
```

```
;;;;;;;;;;;; We can see that the data (ie. different strings) to represent the same Physical PART .. Diff
erent strings have the same part
;;;;;;;; number. Exactly, the same as  I have assumed in the original (old) file :  "...... >statice-R8.d
at"  !!!!!!!!!!

;Command: (setq takis (collect-only-FIRST-and-SECOND-description---PART-NUMBER---FEATURE-CODE katerina))
;Command: takis
; ( ..........................................................
; ("SEAT-FRONT COMPLETE   -L " "ELECTRIC        " "HAD10037XXX "
;  "+A10X+A20W      +A50B+A63Z+B40F+B69D                                    ")
; ("SEAT-FRONT COMPLETE       " "MANUAL-DRIVER  " "HAD10190XXX "
;  "+A10X+A20W      +A30L+A50A+B44R+B51F-B69D                               ")
; ("SEAT-FRONT COMPLETE       " "MANUAL-DRIVER  " "HAD10190XXX "
;  "+A10X+A20W      +A30W+A50A+B44R+B51F-B69D                               ")
; ("EAT-FRONT COMPLETE       M" "ANUAL-PASSR   F" "AD10191XXX 0"
;  "+A10X+A20W      +A30L+A50A+B44R+B51F-B69D                               ")
; ("SEAT-FRONT COMPLETE       " "MANUAL-PASSR   " "HAD10191XXX "
;  "+A10X+A20W      +A30W+A50A+B44R+B51F-B69D                               ")
; ("SEAT-FRONT COMPLETE       " "MANUAL-PASSR   " "HAD10192XXX "
;  "+A10X+A20W      +A30L+A50B+B44R+B51F-B69D                               ")
; ("SEAT-FRONT COMPLETE       " "MANUAL-PASSR   " "HAD10192XXX "
;  "+A10X+A20W      +A30W+A50B+B44R+B51F-B69D                               ")
; ("SEAT-FRONT COMPLETE       " "MANUAL-DRIVER  " "HAD10193XXX "
;  "+A10X+A20W      +A30L+A50B+B44R+B51F-B69D                               ")
; ("SEAT-FRONT COMPLETE       " "MANUAL-DRIVER  " "HAD10193XXX "
;  "+A10X+A20W      +A30W+A50B+B44R+B51F-B69D                               ")
; ("HEADRSTNT-COMP-FRONT      " "BOLSTER-NODDING" "HAH10022XXX " ......................................

;;;;SECONDLY, data are coming wrong FROM THE TAPE :
;;;;;;  ie. look at the list :
;;;;("EAT-FRONT COMPLETE       M" "ANUAL-PASSR   F" "AD10191XXX 0"
;  "+A10X+A20W      +A30L+A50A+B44R+B51F-B69D                               ")
;;;;; within the variable TAKIS. Though , the rest are coming allright this has been parsed wrong !
;;
;;; FOr this reason  I manually correct it ......
;;;; ie   ........................
;Command: (delete '("EAT-FRONT COMPLETE       M" "ANUAL-PASSR   F" "AD10191XXX 0"
; #<ARRAY STRING-CHAR 90 simple 1341021520>) takis)
;;;
;Command: (push '("SEAT-FRONT COMPLETE       " "MANUAL-PASSR   " "HAD10191XXX "
```

```
;           "+A10X+A20W     +A30L+A50A+B44R+B51F-B69D                                    ")
  takis)
;;;;;



(defun evaluate-a-feature-group  (feature-group-string)
  (block takis
    (loop for similar-feature-groups in *FEATURE-GROUPS-WHICH-REPRESENT-THE-SAME-logical-GROUP*
          do
      (cond ((member feature-group-string (car similar-feature-groups)      :test #'string-equal)
                 (return-from takis (cadr similar-feature-groups)))))
    feature-group-string))


;(defun get-words-from-string-other (string)
;  (let ((words nil))
;   (flet ((whitespacep (char) (member char '(#\space #\+ #\-))))
;
;      (do ((last-pos 0 (+ 1 pos))
;           (pos (position-if #'whitespacep string)
;                (position-if #'whitespacep string
;                             :start (+ 1 pos))))
;          ((null pos) (push (subseq string last-pos) words))
;        (push (subseq string last-pos pos) words))
;      (delete "" (nreverse words) :test #'string-equal))))



(defun push-the-FEATURE-GROUPS (ls)            ;;;;; takis
  (let ((result  nil))
    (loop for physical-part-list-of-strings  in  ls
          do
      (let* ((feature-groups-related-to-this-Physical-part
                      (get-words-from-string-other (fourth  physical-part-list-of-strings)))
             (the-actual-additional-feature-groups
                      (out-the-base-features-groups-evaluate-the-rest feature-groups-related-to-this-Ph
ysical-part)))
        (push (list (car physical-part-list-of-strings)
                    (cadr physical-part-list-of-strings)
                    (third physical-part-list-of-strings)
                    the-actual-additional-feature-groups
                    (create-a-pseudo-but-representing-the-original-USAGE-STATEMENT feature-groups-related
-to-this-Physical-part))
              result)))
result))

(defun out-the-base-features-groups-evaluate-the-rest (list-of-feature-strings)    ;;;get-words-from-stri
ng-other
  (let ((result nil))
    (loop for i in list-of-feature-strings
          do
      (cond ((not (string-equal (subseq i 0 1) "A"))
             (push   (evaluate-a-feature-group  (subseq i  0 3))  result))))
result))



(defun create-a-pseudo-but-representing-the-original-USAGE-STATEMENT (list-of-feature-strings)  ;;;get-wo
rds-from-string-other
  (let ((final ""))
    (loop for i in list-of-feature-strings
          do
      (cond ((not (string-equal (subseq i 0 1) "A"))
             (setq final (string-append   i final))
             (setq final (string-append "+"  final)))))   ;;;;; PSEUDO .  We only put  "+"  !!!!!!!!!!
final))

;;;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;;;;  FROM this point on till the  4 rows of stars I put the EXPERIENCE gained from the tape
;;;;;;;;    Some thoughts before I write the Function : "put-the-EXPERIENCE-learned-into-the-St
atice-Knowledge-Base"
;;;;;;;

;;;;;;;;;;;         1 )      The function-on-the-top-of-THE-RUNNER  gives wrong result because the tape is
CORRUPTED.
```

```
;Command: (function-on-the-top-of-THE-RUNNER)  IS WRONG. FOR THIS REASON I correct it  MANUALLY.
;(("HGE" (("INSTR DRG-HEATER-HARN   " (("SEAT-FRONT     " (("OPTIONALS" NIL) ("MUSTS" ("B69"))) "INDICATI
VE")))))
; ("HAM" (("COVER-EXPENDABLE-ST/F   " (("NON ELESTICATED" (("OPTIONALS" NIL) ("MUSTS" ("B45"))) "INDICATI
VE")))))
; ("HAH"
;  (("HEADRSTNT-COMP-FRONT      "
;    (("BOLSTER-NODDING" (("OPTIONALS" ("B53" "B45")) ("MUSTS" ("B25" "B51"))) "QUITE-POSSIBLE")
;     ("BOLSTER-ADJSL  " (("OPTIONALS" NIL) ("MUSTS" ("B25" "B51"))) "INDICATIVE")))))
; ("HCA" (("COVER ASSY-SEAT-FRONT   " (("CUSHION        " (("OPTIONALS" NIL) ("MUSTS" ("B51"))) "INDICATI
VE")))))
; ("HAD"
;  (;;;;("SEAT-FRONT-COMPLETE      " (("ELECTRIC       " (("OPTIONALS" NIL) ("MUSTS" ("B51" "B69"))) "INDI
CATIVE")))
;   ("SEAT-FRONT COMPLETE      "
;    (("ELECTRIC        " (("OPTIONALS" ("B44")) ("MUSTS" ("B51" "B69"))) "QUITE-POSSIBLE")
;     ("MANUAL-DRIVER   " (("OPTIONALS" ("B45" "B53" "B69")) ("MUSTS" ("B51" "B44"))) "QUITE-POSSIBLE")
;     ("MANUAL          " (("OPTIONALS" ("B69" "B45" "B53")) ("MUSTS" ("B51" "B44"))) "QUITE-POSSIBLE")
;     ("MANUAL-PASSR    " (("OPTIONALS" ("B53" "B45" "B69")) ("MUSTS" ("B51" "B44"))) "QUITE-POSSIBLE"))))
)
; ("HBA" (("COVER ASSY-SEAT-FRONT   " (("SQUAB          " (("OPTIONALS" NIL) ("MUSTS" ("B51"))) "INDICATI
VE"))))))
;;Command:


;;;;;;;;;;;        2 )
;;;;;;The way the features are entered into the Statice DATA-BASE is: "b51" "b44"   etc. ....FROM FILE:
;;;;;;  "the-program-the-FLAVORS.lisp" >takis>statice ATC-0:
;Command: #<PART-DESCRIPTION:     SEAT-FRONT COMPLETE   MANUAL-PASSR    >
;#<PART-DESCRIPTION:     SEAT-FRONT COMPLETE   MANUAL-PASSR    >
;Command: (get-MUSTS-TOP *the-pathname-of-the-ROVERs-APN-CATALOGUE* #<PART-DESCRIPTION:     SEAT-FRONT COM
PLETE   MANUAL-PA
;SSR    >)
;("B51" "B44")
;Command:


;;;;;;;;;;;        3 )
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;Command: (get-THE-ROVER-CARS----actually-Austin-Rover---entity *the-pathname-of-the-ROVERs-APN-CATALOGUE
* #<PART-DESCRIPTI
;ON:    INSTR DRG              MAN L/SUPP       >)
;NIL
;NIL
;Command:
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


;;;;;;;;;;        4 )
;#<PART-DESCRIPTION:     SEAT-FRONT COMPLETE   MANUAL-PASSR    >
;Command: (get-THE-ROVER-CARS----actually-Austin-Rover---entity *the-pathname-of-the-ROVERs-APN-CATALOGUE
* #<PART-DESCRIPTI
;ON:    SEAT-FRONT COMPLETE   MANUAL-PASSR    >)
;#<Entity-Handle of THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER 62/3 400252533>
;T
;Command: (instancep #<Entity-Handle of THE-ROVER-CARS----ACTUALLY-AUSTIN-ROVER 62/3 400252533>)
;T
;Command:
;


;;;;;;;;;;       5)   The data will be updated all the time .... ie. It may exist already Musts and/or Ma
ybes
;;;;;;;;;;;       6)   We can use either 4 OR 5 to check if a  Statice-entity-type
;;;;;;;;;                        THE-ROVER-CARS----actually-Austin-Rover is LINKED to  the Secondary-De
scription-ID entity
;;;;;;;;;             with the SLOT : REFERENCE-TO-THE-ROVER-VEHICLES.

;;;;;;;;;;;       7)   We also, have the functions   :
;;;;;;;;;;;              "create-the-entity-type-THE-ROVER-CARS----actually-Austin-Rover"
;;;;;;;;;;;              "assign-CAR-to-SECONDARY"
;;;;;;;;;;;            and (I will change the functions)
;;;;;;;;;;;              "ASSIGN-the-VALUE-VECTOR-of-R8-maybes-to-THE-ROVER-CARS----actually-Austin-Rover-
entity"
;;;;;;;;;;;              "ASSIGN-the-VALUE-VECTOR-of-R8-musts-to-THE-ROVER-CARS----actually-Austin-Rover-e
ntity"
;;;;;;;;;;;            already made from the file :   "the-program-the-FLAVORS.lisp >takis>statice ATC-0:
```

```
;;;;;;;;;;        8)   In addition,  I have to  put INDICATIVE or QUITE-POSSIBLE  information into the dat
a Base

;;;;;;;;;;        9)   The  NOT KNOWN    flag  , as well, for the specific  Model.


;;;;;;;;;;       10)   The *APN-CATALOGUE-format* has got some mistakes.
;;;;;;;;;;       11)   Take out  -L  -R ,    xxx FROM VPG

;(defvar *APN-CATALOGUE-format*
;       '((:recno 4)
;         (:MAIN-GROUP 1)
;
;         (:MAIN-GROUP-DESCRIPTION 35)
;         (:SECONDARY-LETTER 1)
;         (:PRIMARY-LETTER 1)
;         (:VPG-CODE 6)
;         (:DE-SEGMENT-KEY 3)
;         (:SECONDARY-DESCRIPTION 15)
;         (:PRIMARY-DESCRIPTION 22)
;
;         (:EMISSIONS-INDICATOR 1)
;         (:REGULATIONS-INDICATOR 1)
;
;         (:SAFETY-INDICATOR 1)
;         (:COLOURED-INDICATOR 1)
;         (:UNIT-OF-MEASURE 5)
;         (:OBSOLETE-INDICATOR 1)
;         (:DATE-CREATED 6)
;         (:APPROVED-FOR-INDICATOR 1)
;         (:COLOUR-CHART-KEY 3)
;
;         (:SEGMENT-SEQUENCE-NUMBER   4)
;         (:SHORT-DESCRIPTION   22)
;         (:MP17-INDICATOR   1)
;         (:INITIAL-STOCK-ORDER-INDICATOR   1)
;         (:ZONE-CODE   2)
;         (:HONDA-FUNC-AREA-CODE   5)
;         (:PANEL-PART-INDICATOR   1)
;         (:SERVICE-INDICATOR   1)
;         (:DESIGN-RESPONSIBILITY   6)
;))


(defun  ASSIGN-the-VALUE-VECTOR-of-EVERYTHING-maybes-to-THE-ROVER-CARS----actually-Austin-Rover-entity
                (pathname THE-ROVER-CARS----actually-Austin-Rover    VECTOR-OF-STRINGS);;;("b69" "b53" "
b51") OR nil
  (let ((help nil))
    (loop for FEATURE-GROUP-STRING   in VECTOR-OF-STRINGS
          do
      (push (given-a-FEATURE-FROUP-STRING-return-the-NUMBER FEATURE-GROUP-STRING) help))
    (cond ((or (equal help nil) (equal help '(nil)) (equal VECTOR-OF-STRINGS nil) (equal VECTOR-OF-STRING
S '(nil)))
          ;;;;;; Is this PART-DESCRIPTION specified without features as the data coming from the tape sh
ow ?
              (let ((temporary (make-array 2 :element-type 'fixnum :initial-contents '(2 0))))
                (with-database (db pathname)
                  (with-transaction ()
                    (case *model*
                      (R8
                        (setf (THE-ROVER-CARS----actually-Austin-Rover-New-ROVER-200-or-ROVER-400--XW-may
bes

                                THE-ROVER-CARS----actually-Austin-Rover)
                              temporary))
                      (metro
                        (setf (THE-ROVER-CARS----actually-Austin-Rover-METRO-maybes
                                THE-ROVER-CARS----actually-Austin-Rover)
                              temporary))
                      (Rover800
                        (setf (THE-ROVER-CARS----actually-Austin-Rover-ROVER-800-maybes
                                THE-ROVER-CARS----actually-Austin-Rover)
                              temporary)))
                    ))))
          (t
```

```
            (let* ((length-of-array (+ 1 (length help)))
                   (array-elements (push length-of-array help))
                   (VECTOR-value-gone-in-statice
                     (make-array length-of-array :element-type 'fixnum :initial-contents array-elements)))
              (with-database (db pathname)
                (with-transaction ()
                  (case *model*
                    (R8
                      (setf (THE-ROVER-CARS----actually-Austin-Rover-New-ROVER-200-or-ROVER-400--XW-maybes

                              THE-ROVER-CARS----actually-Austin-Rover)
                            VECTOR-value-gone-in-statice))
                    (metro
                      (setf (THE-ROVER-CARS----actually-Austin-Rover-METRO-maybes
                              THE-ROVER-CARS----actually-Austin-Rover)
                            VECTOR-value-gone-in-statice))
                    (Rover800
                      (setf (THE-ROVER-CARS----actually-Austin-Rover-ROVER-800-maybes
                              THE-ROVER-CARS----actually-Austin-Rover)
                            VECTOR-value-gone-in-statice)))
              ))))
          )  ;;;;; cond
))



(defun  ASSIGN-the-VALUE-VECTOR-of-EVERYTHING-musts-to-THE-ROVER-CARS----actually-Austin-Rover-entity
                (pathname THE-ROVER-CARS----actually-Austin-Rover    VECTOR-OF-STRINGS);;;("b69" "b53" "
b51") OR nil OR '(nil)
  (let ((help nil))
    (loop for FEATURE-GROUP-STRING   in VECTOR-OF-STRINGS
          do
      (push (given-a-FEATURE-FROUP-STRING-return-the-NUMBER FEATURE-GROUP-STRING) help))
    (cond ((or (equal help nil) (equal help '(nil)) (equal VECTOR-OF-STRINGS nil) (equal VECTOR-OF-STRING
S '(nil)))
            ;;;;;; Is this PART-DESCRIPTION specified without features as the data coming from the tape s
how ?
              (let ((temporary (make-array 2 :element-type 'fixnum :initial-contents '(2 0))))
                (with-database (db pathname)
                  (with-transaction ()
                    (case *model*
                      (R8
                        (setf (THE-ROVER-CARS----actually-Austin-Rover-New-ROVER-200-or-ROVER-400--XW-mus
ts

                                THE-ROVER-CARS----actually-Austin-Rover)
                              temporary))
                      (metro
                        (setf (THE-ROVER-CARS----actually-Austin-Rover-METRO-musts
                                THE-ROVER-CARS----actually-Austin-Rover)
                              temporary))
                      (Rover800
                        (setf (THE-ROVER-CARS----actually-Austin-Rover-ROVER-800-musts
                                THE-ROVER-CARS----actually-Austin-Rover)
                              temporary)))
                  ))))
          (t
           (let* ((length-of-array (+ 1 (length help)))
                  (array-elements (push length-of-array help))
                  (VECTOR-value-gone-in-statice
                    (make-array length-of-array :element-type 'fixnum :initial-contents array-elements)))
              (with-database (db pathname)
                (with-transaction ()
                  (case *model*
                    (R8
                      (setf (THE-ROVER-CARS----actually-Austin-Rover-New-ROVER-200-or-ROVER-400--XW-musts
                              THE-ROVER-CARS----actually-Austin-Rover)
                            VECTOR-value-gone-in-statice))
                    (metro
                      (setf (THE-ROVER-CARS----actually-Austin-Rover-METRO-musts
                              THE-ROVER-CARS----actually-Austin-Rover)
                            VECTOR-value-gone-in-statice))
                    (Rover800
                      (setf (THE-ROVER-CARS----actually-Austin-Rover-ROVER-800-musts
                              THE-ROVER-CARS----actually-Austin-Rover)
                            VECTOR-value-gone-in-statice)))
```

```
            ))))
        )  ;;;;; cond
))


(defun create-the-entity-type-THE-ROVER-CARS----actually-Austin-Rover (pathname)
  (let ((result nil))
    (with-database (db pathname)
      (with-transaction ()
        (setq result (make-THE-ROVER-CARS----actually-Austin-Rover))))
result))

(defun assign-CAR-to-SECONDARY (pathname  Secondary-Description-ID-entity     THE-ROVER-CARS----actually-
Austin-Rover)
  (with-database (db pathname)
    (with-transaction ()
      (setf (Secondary-Description-ID-REFERENCE-TO-THE-ROVER-VEHICLES Secondary-Description-ID-entity)
            THE-ROVER-CARS----actually-Austin-Rover))))



;(("HGE" (("INSTR DRG-HEATER-HARN    " (("SEAT-FRONT     " (("OPTIONALS" NIL) ("MUSTS" ("B69"))) "INDICAT
IVE")))))
; ("HAM" (("COVER-EXPENDABLE-ST/F    " (("NON ELESTICATED" (("OPTIONALS" NIL) ("MUSTS" ("B45"))) "INDICAT
IVE")))))
; ("HAH"
;  (("HEADRSTNT-COMP-FRONT      "
;    (("BOLSTER-NODDING" (("OPTIONALS" ("B53" "B45")) ("MUSTS" ("B25" "B51"))) "QUITE-POSSIBLE")
;     ("BOLSTER-ADJSL  " (("OPTIONALS" NIL) ("MUSTS" ("B25" "B51"))) "INDICATIVE")))))
; ("HCA" (("COVER ASSY-SEAT-FRONT    " (("CUSHION        " (("OPTIONALS" NIL) ("MUSTS" ("B51"))) "INDICAT
IVE")))))
; ("HAD"
;  (("SEAT-FRONT-COMPLETE       " (("ELECTRIC       " (("OPTIONALS" NIL) ("MUSTS" ("B51" "B69"))) "INDICAT
IVE")))
;   ("SEAT-FRONT COMPLETE       "
;    (("ELECTRIC        " (("OPTIONALS" ("B44")) ("MUSTS" ("B51" "B69"))) "QUITE-POSSIBLE")
;     ("MANUAL-DRIVER  " (("OPTIONALS" ("B45" "B53" "B69")) ("MUSTS" ("B51" "B44"))) "QUITE-POSSIBLE")
;     ("MANUAL         " (("OPTIONALS" ("B69" "B45" "B53")) ("MUSTS" ("B51" "B44"))) "QUITE-POSSIBLE")
;     ("MANUAL-PASSR   " (("OPTIONALS" ("B53" "B45" "B69")) ("MUSTS" ("B51" "B44"))) "QUITE-POSSIBLE"))))
)
; ("HBA" (("COVER ASSY-SEAT-FRONT    " (("SQUAB          " (("OPTIONALS" NIL) ("MUSTS" ("B51"))) "INDICAT
IVE"))))))
;Command:



(defun put-the-EXPERIENCE-learned-into-the-Statice-Knowledge-Base-TOP (VPG-CODE ls)
      ;;; ls = (function-on-the-top-of-THE-RUNNER)
  (loop for each-family in ls
        do
    (put-the-EXPERIENCE-learned-into-the-Statice-Knowledge-Base-for-each-PRIMARY-DESCR   VPG-CODE each-fa
mily)))



(defun put-the-EXPERIENCE-learned-into-the-Statice-Knowledge-Base-for-each-PRIMARY-DESCR (VPG-CODE ls)
;;;;;;;;;ls = One list of one of the lists of  the    (function-on-the-top-of-THE-RUNNER)  list.
;;;;;;;;  ie.   '("HGE" (("INSTR DRG-HEATER-HARN   " (("SEAT-FRONT     " (("OPTIONALS" NIL) ("MUSTS" ("B6
9"))) "INDICATIVE")))))
 (let ((family (car ls)))  ;;;;  (record nil)(record-of-Statice-secondary-entity-types-with-no-information
-from-tape nil))
   (with-database (db *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*)
    (with-transaction ()
      (let ((the-STATICE-Secondary-Description-IDs-BELONG-to-the-same-VPG
              (entity-of-the-Secondary-Description-ID-with-this-VPG VPG-CODE)))
        (loop for PRIMARY-with-secondaries in (cadr ls)
              do
          (let  ((primary-description-string (car PRIMARY-with-secondaries))
                 (secondaries-with-features (cadr PRIMARY-with-secondaries)))
            (loop for secondary-with-feature-and-flag in secondaries-with-features
                  do
```

```
            (let* ((secondary-description-string (car secondary-with-feature-and-flag))
                   (features (cadr secondary-with-feature-and-flag))
                   (maybe-features (car features))
                   (must-features (second features ))
                   (flag-of-data-volume (car (reverse secondary-with-feature-and-flag))))

              (let ((the-actual-STATICE-Secondary-Description-ID-meet-this-REQUIREMENTS
                     (BLOCK statice-SECONDARY
                       (loop for statice-SECONDARY-entity-type  in
                                        the-STATICE-Secondary-Description-IDs-BELONG-to-the-same-VPG
                            do
                         (cond ((string-equal (Secondary-Description-ID-PART-DESCRIPTION  statice-SECO
NDARY-entity-type)
                                              (string-append (subseq primary-description-string  0 22)
                                                             secondary-description-string))
                               ;;; BECAUSE :PRIMARY-DESCRIPTION is 22 in STATICE and NOT 25.
                               (return-from statice-SECONDARY statice-SECONDARY-entity-type))))
                       nil)))

                 (cond ((not (equal the-actual-STATICE-Secondary-Description-ID-meet-this-REQUIREMENTS n
il)) ;YOU found one.
                                  ;;;;; AS THE data may be corrupted...ie NO time match between APN ca
taloque and EJA18V
;; Keep a record of the STATICE-Secondary-Description-IDs (Part Descriptions) in VPG which the tape h
as INFORMATION for them
                 ;;;;(push the-actual-STATICE-Secondary-Description-ID-meet-this-REQUIREMENTS
record)

                        (let ((THE-ROVER-CARS----actually-Austin-Rover
                                (get-THE-ROVER-CARS----actually-Austin-Rover---entity *the-pathname-of
-the-ROVERs-APN-CATALOGUE* the-actual-STATICE-Secondary-Description-ID-meet-this-REQUIREMENTS)))
                          (cond ((equal THE-ROVER-CARS----actually-Austin-Rover nil)
                                   (let ((temp (create-the-entity-type-THE-ROVER-CARS----actually-Aust
in-Rover
                                                *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-path
name*)))
                                     (assign-CAR-to-SECONDARY *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and
-MIRRORS-pathname*
                                                              the-actual-STATICE-Secondary-Description-I
D-meet-this-REQUIREMENTS
                                                              temp))))
                          (let ((again-THE-ROVER-CARS----actually-Austin-Rover
                                  (get-THE-ROVER-CARS----actually-Austin-Rover---entity *the-pathname-
of-the-ROVERs-APN-CATALOGUE* the-actual-STATICE-Secondary-Description-ID-meet-this-REQUIREMENTS)))
                            (ASSIGN-the-VALUE-VECTOR-of-EVERYTHING-musts-to-THE-ROVER-CARS----actuall
y-Austin-Rover-entity
                              *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*
                              again-THE-ROVER-CARS----actually-Austin-Rover
                              (second must-features))
                            (ASSIGN-the-VALUE-VECTOR-of-EVERYTHING-maybes-to-THE-ROVER-CARS----actual
ly-Austin-Rover-entity
                              *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*
                              again-THE-ROVER-CARS----actually-Austin-Rover
                              (second maybe-features))
                            (let (
                                  (In-English-the-detailed-information-of-HOW-the-MUST-and-MAYBES-FEAT
URES-derived
              (now-assign-the-SLOT-------In-English-the-detailed-information-of-HOW-the-MUST-and-MAYBES-
FEATURES-derived
     family primary-description-string secondary-description-string )))

              (setf (Secondary-Description-ID-In-English-the-detailed-information-of-HOW-the-MUST-and-MAYB
ES-FEATURES-derived
                     the-actual-STATICE-Secondary-Description-ID-meet-this-REQUIREMENTS)
                    In-English-the-detailed-information-of-HOW-the-MUST-and-MAYBES-FEATURES-derived)
              (setf (Secondary-Description-ID-Flag-for-information-of-FEATURES-gathered-from-the-tape
                     the-actual-STATICE-Secondary-Description-ID-meet-this-REQUIREMENTS)
                    flag-of-data-volume)

))))) ;;;; cond
)))))    (loop for Statice-secondary-entity-types-in-VPG   in the-STATICE-Secondary-Description-IDs-BELON
G-to-the-same-VPG
                do
             (setf (Secondary-Description-ID-NEGATIVE-additional-features   ;;;;; Initialize  NEGATIVES
                     Statice-secondary-entity-types-in-VPG) nil))
;(setq record-of-Statice-secondary-entity-types-with-no-information-from-tape
```

```
;         (set-difference the-STATICE-Secondary-Description-IDs-BELONG-to-the-same-VPG
;                         record))
;(loop for   STATICE-Secondary-Description-ID-with-no-information in
;                          record-of-Statice-secondary-entity-types-with-no-information-from-tape
)))))


(defun  get-SSSVPG-code-STATICE-entity (VPG-CODE)  ;;;; a-string ie "1110DA"
  (BLOCK  get-the-actual-STATICE-entity
    (with-database  (db  *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*)
      (with-transaction ()
        (for-each ((SSSVPG  Sub-Sub-Sub-Vpg )
                   (:where (string-equal  VPG-CODE  (Sub-Sub-Sub-Vpg-VPG-code  SSSVPG)   )))
          (return-from get-the-actual-STATICE-entity SSSVPG))))
    nil))



(defun now-assign-the-SLOT-------In-English-the-detailed-information-of-HOW-the-MUST-and-MAYBES-FEATURES-
derived
       (family primary-description-string secondary-description-key)  ;;;; list-of-feature-strings)
  (let ((result nil) (result-string ""))
    (loop for i in *MUST-FEATURES*
          do
      (cond ((and
               (string-equal family (car i))
               (string-equal primary-description-string (second i))
               (string-equal secondary-description-key (third i)))
                   (let ((temporary ""))
                     (loop for feature-group-string   in (sixth i)
                           do
                       (setq temporary (string-append temporary "-" feature-group-string)))
                     (push (string-append "It-is-" (fourth i) "(" (fifth i)")" "the-FEATURE-GROUPS" ".."t
emporary)
                           result)))))
    (loop for i in *THE-OPTIONAL-FEATURE-GROUPS-FROM-STATISTICS*
          do
      (cond ((and
               (string-equal family (car i))
               (string-equal primary-description-string (second i))
               (string-equal secondary-description-key (third i)))
                  (let ((temporary ""))
                     (loop for feature-group-string   in (sixth i)
                           do
                       (setq temporary (string-append temporary "-" feature-group-string)))
                     (push (string-append "It-is-" (fourth i) "(" (fifth i)")" "the-FEATURE-GROUPS" ".."t
emporary)
                           result)))))
    (loop for i in *DEFINATELY-OPTIONAL-FEATURE-GROUPS-FOUND*
          do
      (cond ((and
               (string-equal family (car i))
               (string-equal primary-description-string (second i))
               (string-equal secondary-description-key (third i)))
                     (push (string-append "It-is-" (fourth i) "(" (fifth i)")" "the-FEATURE-GROUP" ".." (
sixth i)) result))))
    (loop for i in *THE-RESULT-OF-THE-SEARCH-WITHIN-THE-SYSTEM*
          do
      (cond ((and
               (string-equal family (car i))
               (string-equal primary-description-string (second i))
               (string-equal secondary-description-key (third i)))
                     (push (string-append "It-is-" (fourth i) "(" (fifth i)")" "the-FEATURE-GROUP" ".." (
sixth i)) result))))
    (loop for string-of-comments in result
          do
      (setq result-string (string-append result-string "*" string-of-comments)))
result-string))


;;;; If the length of (get-words-from-string-.. "It-is-IT-IS-OPTIONAL(CHECKED-FROM-STATISTICS)the-FEATURE
-GROUPS..")
;;;;;  IS  1 it means that there are not feature CODES..   Look below.
```

```
;Command: #<ART-STRING-143 374014201>
;"*It-is-IT-IS-OPTIONAL(CHECKED-FROM-STATISTICS)the-FEATURE-GROUPS..*It-is-DEFINATELY-A-MUST(CHECKED-FROM
-STATISTICS-HISTORY)the-F
;EATURE-GROUPS.."
;Command: (get-words-from-string-* #<ART-STRING-143 374014201>)
;("It-is-IT-IS-OPTIONAL(CHECKED-FROM-STATISTICS)the-FEATURE-GROUPS.."
; "It-is-DEFINATELY-A-MUST(CHECKED-FROM-STATISTICS-HISTORY)the-FEATURE-GROUPS..")
;
;Command:
;("It-is-IT-IS-OPTIONAL(CHECKED-FROM-STATISTICS)the-FEATURE-GROUPS")
;Command:


;*****************************************************************************************************
;*****************************************************************************************************
;*****************************************************************************************************
;*****************************************************************************************************

;;;    FROM HERE STARTS THE  old file: "....... statice-R8.dat.43"  which is not going to be deleted ie.
is @ in the directory STATICE

(defvar *parsed-input-from-VPGs*  nil)


;Command:          (the-runner)      ;MUST HAVE    2   ARGUMENTS  : 1)  THE *model*       2) THE  Sub-Sub-S
ub-VPG
;NIL
;Command: *DEFINATELY-OPTIONAL-FEATURE-GROUPS-FOUND*
;(("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-DRIVER  " "IS-DEFINATELY-OPTIONAL-THE-FEATURE-GROUP" "CHECKED
-FROM-HISTORY" "B69")
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-PASSR   " "IS-DEFINATELY-OPTIONAL-THE-FEATURE-GROUP" "CHECKED
-FROM-HISTORY" "B69"))
;Command: *MUST-FEATURES*
;(("HGE" "INSTR DRG-HEATER-HARN    " "SEAT-FRONT     " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B69"))
; ("HAM" "COVER-EXPENDABLE-ST/F    " "NON ELESTICATED" "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B45"))
; ("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-ADJSL  " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B25" "B51"))
; ("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B25" "B51"))
; ("HCA" "COVER ASSY-SEAT-FRONT    " "CUSHION        " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B51"))
; ("HAD" "SEAT-FRONT-COMPLETE      " "ELECTRIC       " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B51" "B69"))
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-PASSR   " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B51" "B44"))
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL         " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B51" "B44"))
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-DRIVER  " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B51" "B44"))
; ("HBA" "COVER ASSY-SEAT-FRONT    " "SQUAB          " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B51")))
;Command: *THE-OPTIONAL-FEATURE-GROUPS-FROM-STATISTICS*
;(("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-DRIVER  " "IT-IS-OPTIONAL" "CHECKED-FROM-STATISTICS" ("B45" "
B53"))
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL         " "IT-IS-OPTIONAL" "CHECKED-FROM-STATISTICS" ("B69" "
B45" "B53"))
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-PASSR   " "IT-IS-OPTIONAL" "CHECKED-FROM-STATISTICS" ("B53" "
B45"))
; ("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" "IT-IS-OPTIONAL" "CHECKED-FROM-STATISTICS" ("B45"))
)

;;Command: *FEATURE-GROUPS-NEED-TO-BE-CHECKED-IN-THE-SYSTEM*
;;(("HAD" "SEAT-FRONT-COMPLETE      " "ELECTRIC       " "NEEDS-MORE-EXAMINATION-THE-FEATURE-GROUP" "B44")
;; ("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" "NEEDS-MORE-EXAMINATION-THE-FEATURE-GROUP" "B53"))
;;Command:

;Command: *THE-RESULT-OF-THE-SEARCH-WITHIN-THE-SYSTEM*
;(("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" "MAY-A-Must" "CHECKED-IN-THE-SYSTEM" "B53")
; ("HAD" "SEAT-FRONT-COMPLETE      " "ELECTRIC       " "DEFINATELY-OPTIONAL" "CHECKED-IN-THE-SYSTEM" "B44"
))
;Command:
```

```
;Command: *variable-freezed*
;(("HBA" "COVER ASSY-SEAT-FRONT    " "SQUAB           " (("B51" 100)) 1 "INDICATIVE")
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-DRIVER   " (("B53" 33) ("B45" 33) ("B69" 89) ("B44" 100) ("B51
" 100)) 3
;  "QUITE-POSSIBLE")
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL          " (("B53" 31) ("B45" 31) ("B69" 69) ("B44" 100) ("B51
" 100)) 4
;  "QUITE-POSSIBLE")
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-PASSR    " (("B45" 38) ("B53" 38) ("B69" 88) ("B44" 100) ("B51
" 100)) 4
;  "QUITE-POSSIBLE")
; ("HAD" "SEAT-FRONT-COMPLETE      " "ELECTRIC        " (("B69" 100) ("B44" 50) ("B51" 100)) 2 "QUITE-POSSI
BLE")
; ("HCA" "COVER ASSY-SEAT-FRONT    " "CUSHION         " (("B51" 100)) 1 "INDICATIVE")
; ("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" (("B45" 40) ("B53" 80) ("B51" 100) ("B25" 100)) 3 "
QUITE-POSSIBLE")
; ("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-ADJSL   " (("B51" 100) ("B25" 100)) 1 "INDICATIVE")
; ("HAM" "COVER-EXPENDABLE-ST/F    " "NON ELESTICATED" (("B45" 100)) 1 "INDICATIVE")
; ("HGE" "INSTR DRG-HEATER-HARN    " "SEAT-FRONT      " (("B69" 100)) 1 "INDICATIVE"))
;Command:



;*****************************************************************************************y
(defun the-runner (R8)
  (initialisation)
  (reset-base-features-nested-TABLE)
  (INITIALIZE-MODEL-AND-DATA R8)
  (INITIALIZE-FROM-SCRATCH)
  (create-the-hash-table-for-VPGs)
  (setq *VARIABLE-FREEZED*
        (append-indicative-FLAG
        (collect-all-combinations-for-statistics *SECONDARY-DESCRIPTION-tables*)))
  (RUN-all
   (EXPLODE-the-feature-groups-need-reexamination
    (which-of-the-FEATURE-GROUPS-need-reexamination      *VARIABLE-FREEZED*)))
;     (append-indicative-FLAG
;      (COLLECT-ALL-COMBINATIONS-FOR-STATISTICS *SECONDARY-DESCRIPTION-TABLES*)))))
  (another-start-for-EXAMINATION-of-FEATURES)
  (the-MUSTS-from-statistics)
  (which-of-the-FEATURE-GROUPS-are-OPTIONAL *VARIABLE-FREEZED*)
)

(defun initialisation ()
  (setq *DEFINATELY-OPTIONAL-FEATURE-GROUPS-FOUND* nil)
  (setq *FEATURE-GROUPS-NEED-TO-BE-CHECKED-IN-THE-SYSTEM*  nil)
  (setq *MUST-FEATURES* nil)
  (setq *THE-OPTIONAL-FEATURE-GROUPS-FROM-STATISTICS* nil)
  (setq *THE-RESULT-OF-THE-SEARCH-WITHIN-THE-SYSTEM* nil))



(defun INITIALIZE-MODEL-AND-DATA (R8)
  ;;;;;;(setq *model* 'r8)          ;;;;;;;;;;;;;;;;;;;;;;;;;;;    DYNAMICALLY with a window OR more FEASIBL
E   Sequentially.
  (setq   *parsed-input-from-VPGs*   R8)
              ;;;;;THIS MUST BE DONE  dynamically ie.
              ;                        (case   model
              ;                           (R8 R8)
              ;                           (metro metro)
              ;                           (Rover800  Rover800))
)

(defun INITIALIZE-FROM-SCRATCH ()
  (setq *array* nil)
  (setq *grouped* nil)
  (setq *data-we-concern* nil)
  (setq *THE-COMMON-GROUND-FOR-STATISTICS-EXAMINATION* nil))

;*****************************************************************************************

(SETQ   R8 '(
  /("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10002"  ("B51" "B44") "B40C+B44B" )
```

```
;("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10003"  ("B51" "B44") "B40C+B44B" )
;("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10318"  ("B51" "B44")  "B44S+B51Y" "OLD")
;("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10318"  ("B51" "B44" "B69") "B40B+B44R-B69D" )
;("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10009"  ("B51" "B44" "B69") "B40B+B44R-B69D" )
;("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10006"  ("B51" "B44" "B69") "B40B+B44R-B69D" )
;("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10319"  ("B51" "B44" "B69") "B44S+B51Y" "OLD")

("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10194"  ("B51" "B44") "B44B+B51K")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10195"  ("B51" "B44") "B44B+B51K")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10314"  ("B51" "B44") "B44B+B51X")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10315"  ("B51" "B44") "B44B+B51X")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10318"  ("B51" "B44" "B69") "B44T+B51Y-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10319"  ("B51" "B44" "B69") "B44T+B51Y-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10321"  ("B51" "B44" "B69") "B44T+B51Y+B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10322"  ("B51" "B44" "B69") "B44T+B51V-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10323"  ("B51" "B44" "B69") "B44T+B51V-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10325"  ("B51" "B44" "B69") "B44T+B51V+B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10474"  ("B51" "B44" "B69") "B44B+B51K+B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10475"  ("B51" "B44" "B69") "B44B+B51K+B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10480"  ("B51" "B44" "B69") "B44R+B51L+B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10481"  ("B51" "B44" "B69") "B44R+B51L+B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10482"  ("B51" "B44" "B69") "B44R+B51F+B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10556"  ("B51" "B44" "B69") "B44T+B51Y-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10557"  ("B51" "B44" "B69") "B44T+B51Y-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10558"  ("B51" "B44" "B69") "B44T+B51Y-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10559"  ("B51" "B44" "B69") "B44T+B51Y-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10560"  ("B51" "B44" "B69") "B44T+B51Y-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10546"  ("B51" "B44" "B45" "B53") "B44R+B45B+B51R+B
53B")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10548"  ("B51" "B44" "B45" "B53") "B44R+B45B+B51R+B
53B")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10550"  ("B51" "B44" "B45" "B53") "B44R+B45B+B51R+B
53B")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10551"  ("B51" "B44" "B45" "B53") "B44R+B45B+B51R+B
53B")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10553"  ("B51" "B44" "B45" "B53") "B44R+B45B+B51R+B
53A")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10543"  ("B51" "B44" "B45" "B53" "B69") "B44R+B45B+
B51R+B53A+B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10549"  ("B51" "B44" "B45" "B53" "B69") "B44R+B45B+
B51R+B53A-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10552"  ("B51" "B44" "B45" "B53" "B69") "B44R+B45B+
B51R+B53A-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL          "   "HAD10554"  ("B51" "B44" "B45" "B53" "B69") "B44R+B45B+
B51R+B53A-B69D")


("SEAT-FRONT COMPLETE     "   "MANUAL-PASSR    "   "HAD10187"  ("B51" "B44" "B69") "B44R+B51L-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL-PASSR    "   "HAD10187"  ("B51" "B44" "B69") "B44R+B51L-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL-PASSR    "   "HAD10188"  ("B51" "B44" "B69") "B44R+B51L-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL-PASSR    "   "HAD10191"  ("B51" "B44" "B69") "B44R+B51F-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL-PASSR    "   "HAD10192"  ("B51" "B44" "B69") "B44R+B51F-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL-PASSR    "   "HAD10568"  ("B51" "B44" "B69" "B53" "B45") "B44R+B45B+
B51R+B53A-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL-PASSR    "   "HAD10564"  ("B51" "B44" "B45" "B53" "B69") "B44R+B45B+
B51R+B53A-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL-PASSR    "   "HAD10564"  ("B51" "B44" "B45" "B53") "B44R+B45B+B51R+B
53A")

("SEAT-FRONT COMPLETE     "   "MANUAL-DRIVER   "   "HAD10178"  ("B51" "B44" "B69") "B44B+B51K-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL-DRIVER   "   "HAD10179"  ("B51" "B44" "B69") "B44B+B51K-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL-DRIVER   "   "HAD10186"  ("B51" "B44" "B69") "B44R+B51L-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL-DRIVER   "   "HAD10189"  ("B51" "B44" "B69") "B44R+B51L-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL-DRIVER   "   "HAD10190"  ("B51" "B44" "B69") "B44R+B51F-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL-DRIVER   "   "HAD10193"  ("B51" "B44" "B69") "B44R+B51F-B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL-DRIVER   "   "HAD10562"  ("B51" "B44" "B45" "B53") "B44R+B45B+B51R+B
53A")
("SEAT-FRONT COMPLETE     "   "MANUAL-DRIVER   "   "HAD10561"  ("B51" "B44" "B45" "B53" "B69") "B44R+B45B+
B51R+B53A+B69D")
("SEAT-FRONT COMPLETE     "   "MANUAL-DRIVER   "   "HAD10563"  ("B51" "B44" "B45" "B53" "B69") "B44R+B45B+
B51R+B53A-B69D")
;*****************************************

("SEAT-FRONT COMPLETE     "   "ELECTRIC        "   "HAD10010"  ("B51" "B44" "B69") "B40B+B44R+B69D" "OLD")
("SEAT-FRONT COMPLETE     "   "ELECTRIC        "   "HAD10011"  ("B51" "B44" "B69") "B40B+B44R+B69D" "OLD")
```

```
("SEAT-FRONT COMPLETE      "   "ELECTRIC        "   "HAD10012"  ("B51" "B44" "B69") "B40B+B44R+B69D" "OLD")
("SEAT-FRONT COMPLETE      "   "ELECTRIC        "   "HAD10013"  ("B51" "B44" "B69") "B40B+B44R+B69D" "OLD")
("SEAT-FRONT COMPLETE      "   "ELECTRIC        "   "HAD10034"  ("B51" "B69") "B40F+B69D" "OLD")
("SEAT-FRONT-COMPLETE      "   "ELECTRIC        "   "HAD10035"  ("B51" "B69") "B40F+B69D" "OLD")
("SEAT-FRONT-COMPLETE      "   "ELECTRIC        "   "HAD10036"  ("B51" "B69") "B40F+B69D" "OLD")
("SEAT-FRONT-COMPLETE      "   "ELECTRIC        "   "HAD10037"  ("B51" "B69") "B40F+B69D" "OLD")


("HEADRSTNT-COMP-FRONT     "   "BOLSTER-NODDING"   "HAH10022"  ("B25" "B51")     "B25G+B51F")
("HEADRSTNT-COMP-FRONT     "   "BOLSTER-NODDING"   "HAH10023"  ("B25" "B51" "B53")    "B25G+B51K")
("HEADRSTNT-COMP-FRONT     "   "BOLSTER-NODDING"   "HAH10023"  ("B25" "B51" "B53")    "B25G+B51L")
("HEADRSTNT-COMP-FRONT     "   "BOLSTER-ADJSL  "   "HAH10038"  ("B25" "B51")    "B25G+B51X")
("HEADRSTNT-COMP-FRONT     "   "BOLSTER-ADJSL  "   "HAH10040"  ("B25" "B51")    "B25G+B51Y")
("HEADRSTNT-COMP-FRONT     "   "BOLSTER-ADJSL  "   "HAH10041"  ("B25" "B51")    "B25G+B51V")
("HEADRSTNT-COMP-FRONT     "   "BOLSTER-NODDING"   "HAH10067"  ("B25" "B51" "B53" "B45")    "B25G+B51R+B45B
+B53A")
("HEADRSTNT-COMP-FRONT     "   "BOLSTER-NODDING"   "HAH10068"  ("B25" "B51" "B53" "B45")    "B25G+B51R+B45B
+B53B")

("COVER-EXPENDABLE-ST/F    "   "NON ELESTICATED"   "HAM10002"  ("B45")    "B45B")

("COVER ASSY-SEAT-FRONT    "   "SQUAB           "   "HBA10058"  ("B51") "B51V")
("COVER ASSY-SEAT-FRONT    "   "SQUAB           "   "HBA10050"  ("B51") "B51X")

("COVER ASSY-SEAT-FRONT    "   "CUSHION         "   "HCA10044"  ("B51") "B51X")
("COVER ASSY-SEAT-FRONT    "   "CUSHION         "   "HCA10054"  ("B51") "B51V")

("INSTR DRG-HEATER-HARN    "   "SEAT-FRONT      "   "HGE10001"  ("B69") "B69D")

))



;*****************************************************************************************

(defflavor    Physical-part
        (part-number
         primary-description
         secondary-description
;        drive-flag
;        fitment-number
;        departure-in
;        release-number-in
;        release-stage-in
;        departure-out
;        release-number-out
;        release-stage-out
;        replaces
;        replaced-by
         feature-groups-link-to
         closed
         feature-combination)

        ()
  :writable-instance-variables
  :initable-instance-variables)


;                ****************************************


(defmethod (sys:print-self  Physical-part)  (stream  ignore readably)
  (if readably
      (sys:printing-random-object (self stream
                                   :typep
                                   :no-pointer)
        (write-string part-number stream))
      (write-string part-number stream)))



(defvar *VPG-physical-parts* nil)
```

```
;                  ****************************************


(defmethod (make-instance   Physical-part) (&rest ignore)
  (push self *VPG-physical-parts*))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;*******************************************************************************************
;*******************************************************************************************
;*******************************************************************************************

(defun create-the-hash-table-for-VPGs ()
  (make-the-VPG-entries)
  (create-hash-table-by-part-number)
  (create-the-PRIMARY-DESCRIPTION-tables)
  (create-the-SECONDARY-DESCRIPTION-tables)
)


(defun reset-base-features-nested-TABLE ()
  (setq *parsed-input-from-VPGs* nil)
  (clrhash *VPG-hash-table*)
  (setq *VPG-physical-parts* nil)
  (setq *PRIMARY-DESCRIPTION-TABLES* nil)
  (setq *SECONDARY-DESCRIPTION-tables* nil)
)


(defvar *VPG-hash-table*        (make-hash-table    :test #'string-equal))

(defun make-the-VPG-entries ()
  (loop for list-of-values in *parsed-input-from-VPGs*
        as new-instance = (make-instance 'Physical-part)
        do
    (setf (Physical-part-part-number   new-instance) (third list-of-values))
    (setf (Physical-part-primary-description  new-instance) (first list-of-values))
    (setf (Physical-part-secondary-description  new-instance) (second list-of-values))
    (setf (Physical-part-feature-groups-link-to  new-instance) (fourth list-of-values))
    (setf (Physical-part-feature-combination  new-instance) (fifth list-of-values))
;    (cond
;      ((equal (length list-of-values) 5) (setf (Physical-part-closed  new-instance) nil))
;      (t (setf (Physical-part-CLOSED new-instance) (car (reverse list-of-values)))))
))



;; Now hash on featcode for all entries

(defun create-hash-table-by-part-number   ()
  (loop for fc in *VPG-physical-parts*
        do
    (add-to-VPG-hash-table fc)))



(defmethod (add-to-VPG-hash-table   Physical-part) ()
  (let ((entries-with-this-featcode
          (gethash  (subseq part-number 0 3)
                    *VPG-hash-table* )))
    (setf (gethash (subseq part-number 0 3)
                   *VPG-hash-table*)
          (cons self entries-with-this-featcode))))



;;*****************************************************************************************

(defun keys (h)
```

```
  (let ((l  nil))
    (maphash #'(lambda (key  value)
                 (push  key l))   h)
    l))
;;*****************************************************************************************


;; Now hash on PRIMARY-DESCRIPTION for all entries


(defvar  *PRIMARY-DESCRIPTION-TABLES* nil)


(defun create-the-PRIMARY-DESCRIPTION-tables ()
  (loop for  VPG-sub-sub-group-key   in (keys *VPG-hash-table*)
        do
    (let  ((table  (make-hash-table    :test #'string-equal)))    ;;;;;  :size 5)))
      (create-hash-table-PRIMARY-DESCRIPTION (gethash
                                              VPG-sub-sub-group-key
                                              *VPG-hash-table*)
                                              table)
      (push (cons VPG-sub-sub-group-key table) *PRIMARY-DESCRIPTION-TABLES*))
   )
)




(defun create-hash-table-PRIMARY-DESCRIPTION   (VPG-Sub-Sub-GROUP  table)
  (loop for fc in    VPG-Sub-Sub-GROUP
        do
    (add-to-hash-table-PRIMARY   fc   table)))



(defmethod (add-to-hash-table-PRIMARY
                              Physical-part) (table)
  (let ((entries-with-this-featcode
          (gethash primary-description   table)))

    (setf (gethash  primary-description  table)
                        (cons self entries-with-this-featcode))))



;*****************************************************************************************

(defvar *SECONDARY-DESCRIPTION-tables* nil)


(defun create-the-SECONDARY-DESCRIPTION-tables ()
  (loop for (Vpg-Sub-Sub-key  . Primary-Description-table) in *PRIMARY-DESCRIPTION-TABLES*
      do

    (loop for  Prim-DESCR-key   in (keys Primary-Description-table)
          do
      (let  ((table  (make-hash-table    :test #'string-equal ))) ;;; :size 5)))
        (create-hash-table-SECONDARY-DESCRIPTION (gethash
                                                  Prim-DESCR-key
                                                  Primary-Description-table)
                                                 table)
        (push (cons (cons Vpg-Sub-Sub-key  Prim-DESCR-key) table) *SECONDARY-DESCRIPTION-tables*))
    )
  )
)



;; Now hash on SECONDARY DESCRIPTION for all entries
```

```
(defun create-hash-table-SECONDARY-DESCRIPTION   (a-PRIM-region  table)
  (loop for fc in    a-PRIM-region
        do
    (add-to-hash-table-SECONDARY   fc   table)))



(defmethod (add-to-hash-table-SECONDARY
                                Physical-part) (table)
  (let ((entries-with-this-featcode
          (gethash  secondary-description  table)))

    (setf (gethash   secondary-description  table)
                        (cons self entries-with-this-featcode))))


;*****************************************************************************************
;*****************************************************************************************
;*****************************************************************************************
;*****************************************************************************************


;;;;;;;;;;   THE STATISTICAL   ANALYSIS

(defun collect-all-combinations-for-statistics  (ls)  ;;;;;;;*SECONDARY-DESCRIPTION-tables*
  (let ((result nil)  (result-final nil))
    (loop for nested-hash-tables in ls
          do
          (let* ((family-and-primary-keys (car nested-hash-tables))
                 (family-key (car family-and-primary-keys))
                 (primary-key (cdr family-and-primary-keys))
                 (table (cdr nested-hash-tables))
                 (secondary-description-keys  (keys table)))
            (loop for table-key   in  secondary-description-keys
                  do
              (push (list family-key primary-key table-key
                          (loop for entry in    (gethash table-key table)
                                collect
                                (Physical-part-feature-groups-link-to  entry)))
                          result))))
;;result))

;Command: (collect-all-combinations-for-statistics *SECONDARY-DESCRIPTION-tables*)
;(("HBA" "COVER ASSY-SEAT-FRONT   " "SQUAB          " (("B51") ("B51")))
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL-DRIVER  "
;  (("B51" "B44" "B69") ("B51" "B44" "B69") ("B51" "B44" "B69") ("B51" "B44" "B69") ("B51" "B44" "B69") (
"B51" "B44" "B69")
;   ("B51" "B44" "B45" "B53") ("B51" "B44" "B45" "B53" "B69") ("B51" "B44" "B45" "B53" "B69")))
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL         "
;  (("B51" "B44") ("B51" "B44") ("B51" "B44") ("B51" "B44") ("B51" "B44" "B69") ("B51" "B44" "B69") ("B51
" "B44" "B69")
;   ("B51" "B44" "B69") ("B51" "B44" "B69") ("B51" "B44" "B69") ("B51" "B44" "B69") ("B51" "B44" "B69") (
"B51" "B44" "B69")
;   ("B51" "B44" "B69") ("B51" "B44" "B69") ("B51" "B44" "B69") ("B51" "B44" "B69") ("B51" "B44" "B69") (
"B51" "B44" "B69")
;   ("B51" "B44" "B69") ("B51" "B44" "B45" "B53") ("B51" "B44" "B45" "B53") ("B51" "B44" "B45" "B53") ("B
51" "B44" "B45" "B53")
;   ("B51" "B44" "B45" "B53") ("B51" "B44" "B45" "B53" "B69") ("B51" "B44" "B45" "B53" "B69") ("B51" "B44
" "B45" "B53" "B69")
;   ("B51" "B44" "B45" "B53" "B69")))
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL-PASSR   "
;  (("B51" "B44" "B69") ("B51" "B44" "B69") ("B51" "B44" "B69") ("B51" "B44" "B69") ("B51" "B44" "B69")
;   ("B51" "B44" "B69" "B53" "B45") ("B51" "B44" "B45" "B53" "B69") ("B51" "B44" "B45" "B53")))
; ("HAD" "SEAT-FRONT-COMPLETE     " "ELECTRIC       "
;  (("B51" "B44" "B69") ("B51" "B44" "B69") ("B51" "B44" "B69") ("B51" "B44" "B69") ("B51" "B69") ("B51"
"B69") ("B51" "B69")
;   ("B51" "B69")))
; ("HCA" "COVER ASSY-SEAT-FRONT   " "CUSHION        " (("B51") ("B51")))
; ("HAH" "HEADRSTNT-COMP-FRONT    " "BOLSTER-NODDING"
;  (("B25" "B51") ("B25" "B51" "B53") ("B25" "B51" "B53") ("B25" "B51" "B53" "B45") ("B25" "B51" "B53" "B
45")))
; ("HAH" "HEADRSTNT-COMP-FRONT    " "BOLSTER-ADJSL  " (("B25" "B51") ("B25" "B51") ("B25" "B51")))
; ("HAM" "COVER-EXPENDABLE-ST/F   " "NON ELESTICATED" (("B45"))) ("HGE" "INSTR DRG-HEATER-HARN   " "SEAT-
FRONT     " (("B69"))))
```

```
 (loop for i in result
       do
   (let* ((group-entries  (fourth i))
          (number-of-patterns (remove-duplicates group-entries :test #'equal)))
     (push
       (list (first i) (second i) (third i)
             (some-function group-entries)
             (length number-of-patterns)
             (length group-entries)) result-final)))
result-final))



(defun some-function (ls)
  (let ((feature-groups-appeared nil)
        (number-of-records (length ls))
        (result nil) (result-statistics nil))
    (loop for list in ls
          do
      (loop for group in list
            do
        (cond
          ((not (member group feature-groups-appeared :test #'string-equal))
           (push group feature-groups-appeared)))))
    (loop for  GROUP  in   feature-groups-appeared
            do
      (push (list GROUP
      (let ((counter 0))
        (loop for list in ls
              do
          (cond
            ((member GROUP list :test #'string-equal)
             (setq counter (+ 1 counter))))) counter) ) result))
                                ;;;;;;;result))    ;;;(("B51" 29) ("B44" 29) ("B69" 20) ("B45" 9) ("B53" 9))
    (loop for i in  result
          do
      (let ((flag (/ 100 number-of-records)))
        (push
          (list (car i)
                (round (* flag (second i)))) result-statistics)))
result-statistics))



;*****************************************************************************
;Command: (collect-all-combinations-for-statistics *SECONDARY-DESCRIPTION-tables*)
;(("HGE" "INSTR DRG-HEATER-HARN   " "SEAT-FRONT      " (("B69" 100)) 1 1)
; ("HAM" "COVER-EXPENDABLE-ST/F   " "NON ELESTICATED" (("B45" 100)) 1 1)
; ("HAH" "HEADRSTNT-COMP-FRONT    " "BOLSTER-ADJSL  " (("B51" 100) ("B25" 100)) 1 3)
; ("HAH" "HEADRSTNT-COMP-FRONT    " "BOLSTER-NODDING" (("B45" 40) ("B53" 80) ("B51" 100) ("B25" 100)) 3 5
)
; ("HCA" "COVER ASSY-SEAT-FRONT   " "CUSHION         " (("B51" 100)) 1 2)
; ("HAD" "SEAT-FRONT-COMPLETE     " "ELECTRIC        " (("B69" 100) ("B44" 50) ("B51" 100)) 2 8)
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL-PASSR    " (("B45" 38) ("B53" 38) ("B69" 88) ("B44" 100) ("B51
" 100)) 4 8)
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL          " (("B53" 31) ("B45" 31) ("B69" 69) ("B44" 100) ("B51
" 100)) 4 29)
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL-DRIVER   " (("B53" 33) ("B45" 33) ("B69" 89) ("B44" 100) ("B51
" 100)) 3 9)
; ("HBA" "COVER ASSY-SEAT-FRONT   " "SQUAB           " (("B51" 100)) 1 2))


;Command: (append-indicative-FLAG (COLLECT-ALL-COMBINATIONS-FOR-STATISTICS *SECONDARY-DESCRIPTION-TABLES*
))
;(("HBA" "COVER ASSY-SEAT-FRONT   " "SQUAB           " (("B51" 100)) 1 "INDICATIVE")
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL-DRIVER   " (("B53" 33) ("B45" 33) ("B69" 89) ("B44" 100) ("B51
" 100)) 3
;  "QUITE-POSSIBLE")
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL          " (("B53" 31) ("B45" 31) ("B69" 69) ("B44" 100) ("B51
" 100)) 4
;  "QUITE-POSSIBLE")
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL-PASSR    " (("B45" 38) ("B53" 38) ("B69" 88) ("B44" 100) ("B51
```

```
" 100)) 4
;  "QUITE-POSSIBLE")
; ("HAD" "SEAT-FRONT-COMPLETE      " "ELECTRIC        " (("B69" 100) ("B44" 50) ("B51" 100)) 2 "QUITE-POSSI
BLE")
; ("HCA" "COVER ASSY-SEAT-FRONT    " "CUSHION         " (("B51" 100)) 1 "INDICATIVE")
; ("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" (("B45" 40) ("B53" 80) ("B51" 100) ("B25" 100)) 3 "
QUITE-POSSIBLE")
; ("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-ADJSL  " (("B51" 100) ("B25" 100)) 1 "INDICATIVE")
; ("HAM" "COVER-EXPENDABLE-ST/F    " "NON ELESTICATED" (("B45" 100)) 1 "INDICATIVE")
; ("HGE" "INSTR DRG-HEATER-HARN    " "SEAT-FRONT      " (("B69" 100)) 1 "INDICATIVE"))

(defvar *VARIABLE-FREEZED* nil)



(defun append-indicative-FLAG (ls)            ;;;;;(collect-all-combinations-for-statistics *SECONDARY-DES
CRIPTION-tables*)
  (let ((result nil))
    (loop for i in ls
          do
      (cond ((< (car (reverse i)) 5)
                 (push (append (list "INDICATIVE") (cdr (reverse i))) result))
            (t (push (append (list "QUITE-POSSIBLE") (cdr (reverse i))) result))
     ))
    (loop for i in result
          collect
          (reverse i))))




(defun  statistical-markers  (ls number-of-patterns)          ;;;;;;;;; ("B53" 33)  4
  (cond
    ((and
           (≤ number-of-patterns 4) (≥ (second ls) 70)) (first ls))    ;;;;75
    (t
     (cond
       ((and
          (≤ number-of-patterns 3) (≥ (second ls) 60)) (first ls))      ;;;;; 65
       (t
        (cond
          ((and
             (≤ number-of-patterns 2) (≥ (second ls) 50)) (first ls))
          (t
           (cond
             ((and
                (≤ number-of-patterns 5) (≥ (second ls) 80)) (first ls))  ;;;;90
             (t nil))))))))))

;*****************************************************************************************

;Command: (which-of-the-FEATURE-GROUPS-need-reexamination
;                 (append-indicative-FLAG (COLLECT-ALL-COMBINATIONS-FOR-STATISTICS *SECONDARY-DESCRIPTION
-TABLES*)))
;(("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-DRIVER  " ("B69")) ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-
PASSR   " ("B69"))
; ("HAD" "SEAT-FRONT-COMPLETE      " "ELECTRIC        " ("B44")) ("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER
-NODDING" ("B53")))



(defun  which-of-the-FEATURE-GROUPS-need-reexamination (ls)
                 ;;;(append-indicative-FLAG (COLLECT-ALL-COMBINATIONS-FOR-STATISTICS *SECONDARY-DESCRIPTI
ON-TABLES*))
  (let ((result nil) (result-final nil))
    (loop for i in ls
          do
      (let ((family (first i)) (secondary-description (third i))
            (primary-description (second i)) (GROUPS (fourth i))
            (number-of-patterns (second (reverse i))))
        (push
          (list family primary-description secondary-description
```

```
                (groups-needed-reexamination GROUPS number-of-patterns))
          result)))
    (loop for i in result
            do
      (cond ((not (equal (car (reverse i)) nil))
                (push i result-final))))
result-final))


(defun groups-needed-reexamination (list-of-lists  number-of-patterns)
  (let ((GROUPS nil))
    (loop for group in list-of-lists
          do
      (let ((group-needed-reexamination
                (statistical-markers group number-of-patterns)))
        (cond
          ((not (equal group-needed-reexamination nil))
             (cond
               ((not (equal (second group) 100))
                (push group-needed-reexamination
                          GROUPS)))))))
GROUPS))

;*****************************************************************************************


(defun EXPLODE-the-feature-groups-need-reexamination (ls)
;;;(which-of-the-FEATURE-GROUPS-need-reexamination
;;;          (append-indicative-FLAG (COLLECT-ALL-COMBINATIONS-FOR-STATISTICS *SECONDARY-DESCRIPTION-TABL
ES*)))
(let ((result nil))
  (loop for i in ls
        do
    (loop for feature-group-need-reexamination in (first (reverse i))
          do
      (push (list
               (first i) (second i) (third i) feature-group-need-reexamination)
            result)))
result))

;*****************************************************************************************************
*****


;;;;; NOW  START the checking for OPTIONAL indication.

(defvar  *DEFINATELY-OPTIONAL-FEATURE-GROUPS-FOUND* nil)
(defvar *FEATURE-GROUPS-NEED-TO-BE-CHECKED-IN-THE-SYSTEM*  nil)
(defvar *MUST-FEATURES* nil)
(defvar *THE-OPTIONAL-FEATURE-GROUPS-FROM-STATISTICS* nil)

(defun    RUN-all (ls)

;(EXPLODE-the-feature-groups-need-reexamination
;        (which-of-the-FEATURE-GROUPS-need-reexamination
;         (append-indicative-FLAG (COLLECT-ALL-COMBINATIONS-FOR-STATISTICS *SECONDARY-DESCRIPTION-TABLES*)
)))

(loop for I in ls        ;;;;;;    I  = ("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" "B53")
      do
  (let ((variable    (find-the-FEAT-COMBINATIONS-OF-the-relative-instances  I)))
    (cond
      ((check-from-the-data-YOU-only-have (first variable) (second variable) (third variable))
                (push
                  (list (first I) (second I) (third I) "IS-DEFINATELY-OPTIONAL-THE-FEATURE-GROUP" "CHECK
ED-FROM-HISTORY" (car (reverse I)))
                  *DEFINATELY-OPTIONAL-FEATURE-GROUPS-FOUND*))
      (t  (push
            (list (first I) (second I) (third I) "NEEDS-MORE-EXAMINATION-THE-FEATURE-GROUP" (car (reverse
I)))
            *FEATURE-GROUPS-NEED-TO-BE-CHECKED-IN-THE-SYSTEM*))))))
```

```
;Command: (RUN-all
;
;(EXPLODE-the-feature-groups-need-reexamination
;        (which-of-the-FEATURE-GROUPS-need-reexamination
;         (append-indicative-FLAG (COLLECT-ALL-COMBINATIONS-FOR-STATISTICS *SECONDARY-DESCRIPTION-TABLES*)
))))
;NIL
;Command: *DEFINATELY-OPTIONAL-FEATURE-GROUPS-FOUND*
;(("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-DRIVER   " "IS-DEFINATELY-OPTIONAL-THE-FEATURE-GROUP" "CHECKED
-FROM-HISTORY" "B69")
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-PASSR    " "IS-DEFINATELY-OPTIONAL-THE-FEATURE-GROUP" "CHECKED
-FROM-HISTORY" "B69"))
;Command: *FEATURE-GROUPS-NEED-TO-BE-CHECKED-IN-THE-SYSTEM*
;(("HAD" "SEAT-FRONT-COMPLETE      " "ELECTRIC        " "NEEDS-MORE-EXAMINATION-THE-FEATURE-GROUP" "B44")
; ("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" "NEEDS-MORE-EXAMINATION-THE-FEATURE-GROUP" "B53"))
;*****************************************************************************************************
****************


(defun the-MUSTS-from-statistics ()
    (loop for i in *VARIABLE-FREEZED*
          do
      (push (list  (first i) (second i) (third i) "DEFINATELY-A-MUST"
                           "CHECKED-FROM-STATISTICS-HISTORY"
                           (get-th-MUSTS (fourth i))
                           )
            *MUST-FEATURES*)))



(defun get-th-MUSTS (list-of-lists)
  (let ((result nil))
    (loop for j in list-of-lists
          do
      (cond ((equal (second j) 100) ;;; 100%
             (push (first j) result))))
result))



(defun  which-of-the-FEATURE-GROUPS-are-OPTIONAL (ls)  ;;*VARIABLE-FREEZED*
                     ;;;(append-indicative-FLAG (COLLECT-ALL-COMBINATIONS-FOR-STATISTICS *SECONDARY-DESCR
IPTION-TABLES*))
  (let ((result nil) (result-final nil))
    (loop for i in ls
          do
      (let ((family (first i)) (secondary-description (third i))
            (primary-description (second i)) (GROUPS (fourth i))
            (number-of-patterns (second (reverse i))))
        (push
          (list family primary-description secondary-description
                "IT-IS-OPTIONAL" "CHECKED-FROM-STATISTICS"
                (groups-HAVE-BEEN-proved-OPTIONAL-from-statistics GROUPS number-of-patterns))
          result)))
    (loop for i in result
              do
      (cond ((not (equal (car (reverse i)) nil))
               (push i  *THE-OPTIONAL-FEATURE-GROUPS-FROM-STATISTICS* ))))
))



(defun groups-HAVE-BEEN-proved-OPTIONAL-from-statistics (list-of-lists  number-of-patterns)
  (let ((GROUPS nil))
    (loop for group in list-of-lists
          do
      (let ((OPTIONAL-group
                (statistical-markers group number-of-patterns)))
        (cond
          ((equal OPTIONAL-group nil)
             (cond
               ((not (equal (second group) 100))
                   (push (car group)
                         GROUPS)))))
```

```
      ))
GROUPS))


;Command: (which-of-the-FEATURE-GROUPS-are-OPTIONAL *VARIABLE-FREEZED*)
;(("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-DRIVER  " (("B45" 33) ("B53" 33)))
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL         " (("B69" 69) ("B45" 31) ("B53" 31)))
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-PASSR   " (("B53" 38) ("B45" 38)))
; ("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" (("B45" 40))))
;Command:



;*****************************************************************************************************
****************

;;;;;;;;;;;;;;;;;;     BELOW  IS DONE One By One ;;;;;;;;;;;;;;;;;

;;;;;;  NOW find THE  FLAVOR - instances  linked to  the feature-groups NEEDED re-examination,
;;;;;;       through the nested hash tables.


(defun  find-the-FEAT-COMBINATIONS-OF-the-relative-instances (ls)
;;;("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-DRIVER  " "B69").   One LIST-element of the ...
;;;;;;;;;;;(EXPLODE-the-feature-groups-need-reexamination ls)

(let ((result nil) (result-final nil))
  (loop for i in  *SECONDARY-DESCRIPTION-tables*
      do
    (cond
      ((and
         (string-equal (car ls) (car (car i)))
         (string-equal (second ls) (cdr (car i))))
       (setq result
             (gethash (third ls) (cdr i))))))
(loop for instance in result
      do
  (cond
    ((not (member (first (reverse ls)) (Physical-part-FEATURE-GROUPS-LINK-TO instance)
                  :test #'string-equal))
     (push (Physical-part-FEATURE-COMBINATION instance)
           result-final))))

(list result-final result (first (reverse ls)))))



;Command: (find-the-FEAT-COMBINATIONS-OF-the-relative-instances '("HAD" "SEAT-FRONT COMPLETE      " "MANUA
L-DRIVER  " "B69") )
;(("B44R+B45B+B51R+B53A")              =        FEAT-COMB-ls
; (#<PHYSICAL-PART HAD10178> #<PHYSICAL-PART HAD10179> #<PHYSICAL-PART HAD10186> #<PHYSICAL-PART HAD10189
>
;  #<PHYSICAL-PART HAD10190> #<PHYSICAL-PART HAD10193> #<PHYSICAL-PART HAD10562> #<PHYSICAL-PART HAD10561
>
;  #<PHYSICAL-PART HAD10563>)  "B69)        =        INSTANCES-ls  ,  FEATURE-GROUP-IN-EXAM


;*************************************************


(defun  check-from-the-data-YOU-only-have  (FEAT-COMB-ls   INSTANCES-ls  FEATURE-GROUP-IN-EXAM)
          ;;;;;; FEAT-COMB-ls  =  ( "B44R+B45B+B53A"  "B44R+B45B+B51R+B53A" "B45B+B51R+B53A")
  (prog (FLAG)
        NEXT
           (cond
             ((null FEAT-COMB-ls) (return FLAG))
             ((check-one-FEATURE-COMBINATION-AT-A-TIME (first FEAT-COMB-ls) INSTANCES-ls FEATURE-GROUP-IN
-EXAM)
                     (RETURN T))  ;;;ie THERE IS INDICATION THAT THE FEATURE GROUP FEATURE-GROUP-IN-EXAM I
S OPTIONAL
                                  ;;; for this family, primary, secondary KEY.
            )
           (setq FEAT-COMB-ls (cdr FEAT-COMB-ls))
           (go NEXT)))
```

```
(defun  check-one-FEATURE-COMBINATION-AT-A-TIME  (feature-combination  INSTANCES-ls  FEATURE-GROUP-IN-EXA
M)
          ;;;;;feature-combination    =    "B44R+B45B+B51R+B53A"
  (prog (flag  HELP)

        (setq HELP
        (let ((instances-related-to-FEATURE-GROUP-in-examination nil))
          (loop for instance in INSTANCES-ls
                do
            (cond
              ((and
                 (member FEATURE-GROUP-IN-EXAM  (Physical-part-FEATURE-GROUPS-LINK-TO instance)
                         :test #'string-equal)
                 (equal 5
                        (-  (string-length  (Physical-part-FEATURE-COMBINATION instance))
                            (string-length  feature-combination)))  ;;;; only reasoning at one level HIGH
ER
               )
               (push (Physical-part-FEATURE-COMBINATION instance) instances-related-to-FEATURE-GROUP-in-e
xamination))))
          instances-related-to-FEATURE-GROUP-in-examination))
        NEXT
         (cond
           ((null HELP)  (return flag))
           ((search feature-combination (first HELP)) (return t)))                  ;;; THERE is reference
         (setq HELP (cdr HELP))
         (go NEXT)))


;Command: (check-one-FEATURE-COMBINATION-AT-A-TIME "B44R+B45B+B51R+B53A" '(#<PHYSICAL-PART HAD10178> #<PH
YSICAL-PART HAD10179> #<P
:HYSICAL-PART HAD10186>
; #<PHYSICAL-PART HAD10189> #<PHYSICAL-PART HAD10190> ...) "B69")
;T
;Command: (check-one-FEATURE-COMBINATION-AT-A-TIME "B44R+B45B+B53A" '(#<PHYSICAL-PART HAD10178> #<PHYSICA
L-PART HAD10179> #<PHYSIC
:AL-PART HAD10186>
; #<PHYSICAL-PART HAD10189> #<PHYSICAL-PART HAD10190> ...) "B69")
;NIL
;Command: (check-from-the-data-YOU-only-have  '( "B44R+B45B+B53A"  "B44R+B45B+B51R+B53A" "B45B+B51R+B53A"
)
;' (#<PHYSICAL-PART HAD10178> #<PHYSICAL-PART HAD10179> #<PHYSICAL-PART HAD10186>
; #<PHYSICAL-PART HAD10189> #<PHYSICAL-PART HAD10190> ...) "B69")
;T
;Command:
;;;;;;;  CHECK THE DATA IN   R 8 , UNDER "HAD" "seat-front-complete" "manual-driver   "  TO SEE THAT THE
PHYSICAL PART
;;;;;;;  "HAD10562" SEARCHING FOR optional INDICATOR FOUNDS AT LEAST ONE AT THE PHYSICAL-PARTS "HAD10561"
"HAD10563".  ie
;;;("SEAT-FRONT COMPLETE     "   "MANUAL-DRIVER  "   "HAD10562"  ("B51" "B44" "B45" "B53") "B44R+B45B+B51
R+B53A")
;;;("SEAT-FRONT COMPLETE     "   "MANUAL-DRIVER  "   "HAD10561"  ("B51" "B44" "B45" "B53" "B69") "B44R+B4
5B+B51R+B53A+B69D")
;;;("SEAT-FRONT COMPLETE     "   "MANUAL-DRIVER  "   "HAD10563"  ("B51" "B44" "B45" "B53" "B69") "B44R+B4
5B+B51R+B53A-B69D")
;*****************************************************************************


(defun TOP-find-the-FEAT-COMBINATIONS-OF-the-relative-instances-SYSTEM (ls) ;;;;;*FEATURE-GROUPS-NEED-TO-
BE-CHECKED-IN-THE-SYSTEM*
  (let ((result nil))
    (loop for i in ls
          do
      (push (list
              (first i) (second i) (third i)
              (find-the-FEAT-COMBINATIONS-OF-the-relative-instances-SYSTEM i)
              (car (reverse i))) result))
result))




(defun  find-the-FEAT-COMBINATIONS-OF-the-relative-instances-SYSTEM (ls)

;;;;;;; ("HAD" "SEAT-FRONT-COMPLETE     " "ELECTRIC        " "NEEDS-MORE-EXAMINATION-THE-FEATURE-GROUP" "B
```

```
44")
;;;;;;;;;;;;   One LIST-element of the ...      *FEATURE-GROUPS-NEED-TO-BE-CHECKED-IN-THE-SYSTEM*

(let ((result nil) (result-final nil))
  (loop for i in  *SECONDARY-DESCRIPTION-tables*
      do
    (cond
      ((and
         (string-equal (car ls) (car (car i)))
         (string-equal (second ls) (cdr (car i))))
       (setq result
             (gethash (third ls) (cdr i))))))
(loop for instance in result
      do
  (cond
    ((not (member (first (reverse ls)) (Physical-part-FEATURE-GROUPS-LINK-TO instance)
                  :test #'string-equal))
     (push (Physical-part-FEATURE-COMBINATION instance)
           result-final))))
(remove-duplicates result-final :test #'string-equal)))


;&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
;&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
;&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&


(defun find-groups-of-all-VPG-sub-sub-sub-group ()
  (let ((result nil) (takis nil)
                     (help
                       (TOP-find-the-FEAT-COMBINATIONS-OF-the-relative-instances-SYSTEM
                         *FEATURE-GROUPS-NEED-TO-BE-CHECKED-IN-THE-SYSTEM*)))
    ;;;;HELP (("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" ("B25G+B51F") "B53")
    ;;;;      ("HAD" "SEAT-FRONT-COMPLETE      " "ELECTRIC        " ("B40F+B69D") "B44"))
    (let ((variable-before nil) (variable-after nil))
      (loop for i in help
            do
        (loop for j  in  (fourth i)
              do
          (push  (get-words-from-string-other j) variable-before)
          (push  (first (reverse i)) result)))
      (setq variable-after
            (remove-duplicates
              (loop for i in  (only-atoms variable-before)
                    collect
                (substring i 0 3))
              :test #'string-equal))
      (setq takis (remove-duplicates result :test #'string-equal))
      (loop for i in variable-after
            do
        (push i takis)))
(list (remove-duplicates takis :test #'string-equal) help)
        ;;;;; WE make a LIST because we need to transfer the
        ;;;;; RECORD of KEY, PRIMARY, SECONDARY  and
        ;;;;; the Usage Statements we need to CHECK for VALIDITY ie.
        ;;;;; ("B25G+B51F") "B53"   ,    ("B40F+B69D") "B44"))
))


(defun get-words-from-string-* (string)  ;;;;;;; SEPARATE the different feature Groups with their comment
s  in the string.
  (let ((words nil))
   (flet ((whitespacep (char) (member char '(#\*)  :test #'string-equal)))

      (do ((last-pos 0 (+ 1 pos))
           (pos (position-if #'whitespacep string)
```

```
                (position-if #'whitespacep string
                             :start (+ 1 pos))))
          ((null pos) (push (subseq string last-pos) words))
        (push (subseq string last-pos pos) words))
      (delete "" (nreverse words) :test #'string-equal))))


(defun get-words-from-string-.. (string)    ;;;;separate comment from FEATURE-GROUP code
  (let ((words nil))
   (flet ((whitespacep (char) (member char '( #\.)  :test #'string-equal)))

      (do ((last-pos 0 (+ 1 pos))
           (pos (position-if #'whitespacep string)
                (position-if #'whitespacep string
                             :start (+ 1 pos))))
          ((null pos) (push (subseq string last-pos) words))
        (push (subseq string last-pos pos) words))
      (delete "" (nreverse words) :test #'string-equal))))


(defun get-words-from-string-- (string)     ;;;;;;; separate the feature groups  CODES
  (let ((words nil))
   (flet ((whitespacep (char) (member char '( #\~)  :test #'string-equal)))

      (do ((last-pos 0 (+ 1 pos))
           (pos (position-if #'whitespacep string)
                (position-if #'whitespacep string
                             :start (+ 1 pos))))
          ((null pos) (push (subseq string last-pos) words))
        (push (subseq string last-pos pos) words))
      (delete "" (nreverse words) :test #'string-equal))))



(defun get-words-from-string-other (string)
  (let ((words nil))
   (flet ((whitespacep (char) (member char '(#\space #\+ #\-) :test #'string-equal)))

      (do ((last-pos 0 (+ 1 pos))
           (pos (position-if #'whitespacep string)
                (position-if #'whitespacep string
                             :start (+ 1 pos))))
          ((null pos) (push (subseq string last-pos) words))
        (push (subseq string last-pos pos) words))
      (delete "" (nreverse words) :test #'string-equal))))


;(defun only-atoms (ls)
;  (cond ((null ls) nil)
;        ((atom (car ls))
;            (cons (car ls)
;                  (only-atoms (cdr ls))))
;        (t (append (only-atoms (car ls))
;                   (only-atoms (cdr ls))))))


;(defun  collect-the-groups-we-concern   (list-of-groups)
;  ;;;;;; i. e...   ("b15" "b12" "b11" "b02")
;  (let ((result nil))
;    (loop for group in list-of-groups
;          do
;      (push (collect-subgroups-related   group) result)
;    )
;result))
;
;
;
;(defun  collect-subgroups-related (group)
;  (let ((result   nil))
;    (loop  for line-of-DATA  in   (find-the-ADDITIONALS-of-model)          ;;;;;;;;;;"R8"
;          do
;      (cond
```

```
;        ((string-equal   group   (substring (cadr (assoc ':FEATURE-DESCRIPTION line-of-DATA))
;                                          0 3))
;              (push line-of-DATA
;                    result))))
;result))


;*****************************************************************************************************
****************

(defun STATICE-get-only-the-data-we-need-for-the-EXAMINATION-of-the-VPG-sub-group ()
  (let (
        ;;;THIS FUNCTION initialises the *feature-chart-entries*  instead the old code which initialized
the
        ;;;;;;   *data-we-concern*  variable
        (GROUPS-we-exam-in-VPG-sub-group  (first
                                           (find-groups-of-all-VPG-sub-sub-sub-group)))
        (RECORD (second (find-groups-of-all-VPG-sub-sub-sub-group))))
    (setq *feature-chart-entries*
          (create-the-STATICE-*feature-chart-entries* GROUPS-we-exam-in-VPG-sub-group
                                                      (The-DATA-BASE-pathname-of-the-BASE-and-ADDITIONAL-
features-RESTRICTIONS-of-the-model-we-concern *model*)))
    ;;;;;;;;(print GROUPS-we-exam-in-VPG-sub-group)
record))



;(defun get-only-the-data-we-need-for-the-EXAMINATION-of-the-VPG-sub-group ()
;  (let (            ;;;THIS FUNCTION initialises the *data-we-concern*  variable
;        (GROUPS-we-exam-in-VPG-sub-group  (first
;                                           (find-groups-of-all-VPG-sub-sub-sub-group)))
;        (RECORD (second (find-groups-of-all-VPG-sub-sub-sub-group))))
;    (loop for i in (collect-the-groups-we-concern GROUPS-we-exam-in-VPG-sub-group)
;          do
;      (loop for j in i
;            do
;        (push j *data-we-concern*)))
;RECORD))

(defun STATICE-after-reset-for-the-examination-of-musts-and-maybes ()
   (STATICE-make-feature-chart-entries-for-the-examination-of-musts-and-maybes )  ;;exactly like  the fun
ction
;;;;;;   STATICE-after-reset  but it uses the function
;;;;STATICE-make-feature-chart-entries-for-the-examination-of-musts-and-maybes    instead of   STATICE-ma
ke-feature-chart-entries
   (create-the-subDescription-tables)
   (create-the-class-tables)
)


(defun STATICE-make-feature-chart-entries-for-the-examination-of-musts-and-maybes ()
;;;;; exactly the same as STATICE-make-feature-chart-entries but we don't bother with the  DELETED out fe
atures.
;  ;;;;;; check-if-the-entry-is-not-deleted
;  (let ((help nil))
;    (loop for fc in *feature-chart-entries*
;          do
;      (let ((DEPARTURE-OUT-OF-THE-FEATURE (Feature-Chart-ENTRY-dep-out fc)))  ;;a  string  of length 8
;       (cond ((IS-IT-EMPTY-the-STRING?  DEPARTURE-OUT-OF-THE-FEATURE)
;                (push fc help)))))
;    (setq *feature-chart-entries* help))

  (loop for fc in *feature-chart-entries*
        do
            (add-to-hash-table fc)))


;*****************************************************************************************************
****************

(defun CREATE-THE-Data-Base-FOR-THE-EXAMINATION-of-THE-FEATURES-IN-Sub-Sub-Sub-VPG  ()

  (STATICE-reset-feature-chart-db)   ;;;; WE NEED this function because the variable *CLASS-tables* is ge
tting very HUGE.

  (let ((RECORD     (STATICE-get-only-the-data-we-need-for-the-EXAMINATION-of-the-VPG-sub-group)))
```

```
    (STATICE-after-reset-for-the-examination-of-musts-and-maybes   ))))) It is not the same exactly as th
e STATICE-after-reset
;;;;;;;function because we dont want to EXCLUDE the possibility of searching through OLD information if t
his EXISTS .  ie the
;;;; B40 group is deleted out for both METRO and R8 but there is reference for it into their EJA18V files
 and by chance into their
;;;;;   BAFCs. Hence, we can check for musts and maybes , so we don't exclude the deleted out for the sea
rch.

    (setq *implicit-trim-levels* (give-the-implicit-trim-levels-TOP))

    (setq *array* (create-the-array))

    (setq  *grouped* (group-together-GROUPS  (group-together-SUBGROUPS *array*)))

    (setq *territories* (do-it-new))

    (setq *final-territories* (supergroups-in))

    (setq *countries* (remove-duplicates (all-the-countries) :test #'string-equal))
 RECORD)   ;;;;;;;;;; Returns also, the Record of Features to be COMPARED.
)

;*****************************************************************************************************
;*****************************************************************************************************
;*****************************************************************************************************


(defvar  *THE-COMMON-GROUND-FOR-STATISTICS-EXAMINATION*   nil)
(defvar *SCORE*  nil)


(defvar *THE-RESULT-OF-THE-SEARCH-WITHIN-THE-SYSTEM* nil)


(defun  another-start-for-EXAMINATION-of-FEATURES ()
  (let ((HELP nil)  (result nil))
    (let  ((record-of-features-to-be-compared
             (CREATE-THE-Data-Base-FOR-THE-EXAMINATION-of-THE-FEATURES-IN-Sub-Sub-Sub-VPG)))
     ;;;;;;  record-of-features-to-be-compared   ==  (("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING"
("B25G+B51F") "B53")
    ;;;;;;                                            ("HAD" "SEAT-FRONT-COMPLETE      " "ELECTRIC        "
("B40F+B69D") "B44"))

      (MUST-be-entered)  ;;;;;; THE  Pseudo  FEATURE-GROUP  "000" is entered in *GROUPED* in order to kee
p the format
                         ;;;;;; of the OLD ALGORITHM and keeps all TRIM levels for  R8.

          (loop for i in record-of-features-to-be-compared
                do
            (push
              (list (first i) (second i) (third i)
                    (FEATURE-COMBINATION-FOUND (fourth i)) (fifth i)) HELP)))

   ;;;;;;  HELP &&&&     ==     (("HAD" "SEAT-FRONT-COMPLETE      " "ELECTRIC        " (("B40F" "B69D")) "B4
4")
   ;;;;;;                        ("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" (("B25G" "B51F")) "B5
3"))
   ;;;;;;;    11111111111

   (loop for i in   HELP
         do
     (cond ((equal (CHECK-A-NUMBER-OF-feature-combinations-FOR-THE-SAME-PURPOSE i) t)  ;;;;; A  Maybe Fe
ature is FOUND
              (push (list (first i) (second i) (third i) "DEFINATELY-OPTIONAL" "CHECKED-IN-THE-SYSTEM" (
car (reverse i))) result))
           (t (push (list (first i) (second i) (third i) "MAY-A-Must" "CHECKED-IN-THE-SYSTEM" (car (reve
rse i))) result))))
;;;result))

(setq *THE-RESULT-OF-THE-SEARCH-WITHIN-THE-SYSTEM* result)))
```

```
(defun FEATURE-COMBINATION-FOUND (ls)
  (let ((result nil))
    (loop for i in ls
          do
      (push (get-words-from-string-other i) result))
result))


(defun score ()
  (let ((result nil))
    (loop for i in *grouped* do
      (push (list (first i) (length (second i)))
            result))
    (sort result #'> :key #'second)))



(defun PUT-IN-THE-SEQUENCE (ls)
  (let ((result nil) (score (score)))
    (loop for i in score
          do
      (push (car (member (car i) ls :test #'string-equal
                         :key #'(lambda (x) (subseq x 0 3))))
            result)) (remove 'nil (reverse result))))


;;;;(defvar *THE-PSEUDO-FEATURE* '("0000" NIL NIL (" " NIL) NIL ("M" "S" "W" "X" "Y" "Z")))  ;;;;;;THIS M
UST BECOME dynamically.

(defun MUST-be-entered ()
  (let* ((result nil) (TRIMS-OF-MODEL (get-all-trims-for-that-model))
         (variable '(NIL NIL (" " NIL) NIL FLAG))
         (NEW-VARIABLE (replace variable (list (get-all-trims-for-that-model)) :start1 4 :end1 5)))
    (setq result (list "000"
                       (list
                         (list
                           "0000"
                           (list
                             (list
                               "0000"  " " NIL (list NEW-VARIABLE)))))))
(push result *GROUPED*)))



;Command: (MUST-be-entered)
;            '("000" (("0000" (("0000" " " NIL ((NIL NIL (" " NIL) NIL FLAG))))))))   ; TO HAVE THE SAME
 format with
;                                                                                     ; THE rest of entri
es in *GROUPED*





;Command: (get-all-trims-for-that-model)
;("Z" "Y" "X" "W" "S" "M")


(defun get-all-trims-for-that-model ()
  (let ((result nil)  (MODEL-VARIANT-Base-Features  (find-the-base-of-model)))
    (loop for i in MODEL-VARIANT-Base-Features
          do
      (let ((TRIM (second (assoc ':CLASS i))))
        (cond
          ((not (string-equal " " TRIM))
                (push TRIM result)))))
    (remove-duplicates result :test #'string-equal)))

;;;;*****************************************************************
```

```
(defun  CHECK-A-NUMBER-OF-feature-combinations-FOR-THE-SAME-PURPOSE (ls)
                                                ;;;; A    list - element  of  HELP &&&&  above.
                                                ;;;; ie. ("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" (
**("B25G" "B51F")) "B53")
  (setq *THE-COMMON-GROUND-FOR-STATISTICS-EXAMINATION* nil)
 (BLOCK  FLAG

  (let* ((the-data-base-reference-to-the-FEATURE-GROUP
           (let ((var  nil))
                        (loop for i in  *grouped*
                              do
                          (cond
                            ((string-equal (first i) (first (reverse ls)))
                               (setq var i)))) var))

       ;;;;;;;;) the-data-base-reference-to-the-FEATURE-GROUP   =====   IT IS  NESTED the second item
;Command: (CHECK-A-NUMBER-OF-feature-combinations-FOR-THE-SAME-PURPOSE '("HAH" "HEADRSTNT-COMP-FRONT     "
 "BOLSTER-NODDING" (**("B
;25G" "B51F")) "B53"))
;(B53
;(("B53B" (("B53B" "W" (#<FEATURE-CHART-ENTRY 304002720>) NIL)))
; ("B53A" (("B53A" "W" (#<FEATURE-CHART-ENTRY 304000772>) NIL) ("B53A" "X" (#<FEATURE-CHART-ENTRY 3040007
13>) NIL)))))

          (BASE-FEATURES-and-RESTRICTIONS-list-OF-THE-Feature-group
                      (ALL-the-DATA-known-FOR-the-FEATURE-GROUP-in-EXAM the-data-base-reference-to-the-FEAT
URE-GROUP))
        )
    ;;;;; THE NEXT FUNCTION only GENERATES the  *THE-COMMON-GROUND-FOR-STATISTICS-EXAMINATION* variable.


    (CHECK-ALL-DIFFERENT-COMBINATIONS-with-the-same-reference-for-the-SAME-FEATURE-GROUP (fourth ls))

    (loop for i in *THE-COMMON-GROUND-FOR-STATISTICS-EXAMINATION*
          do
      (loop for j in   BASE-FEATURES-and-RESTRICTIONS-list-OF-THE-Feature-group
            do
        (cond ((and
                 (no-opposite (car i) (car j))
                 (intersection (cadr i) (cadr j) :test #'string-equal))
               (RETURN-FROM FLAG t))))))
nil))



;;Command: (ALL-the-DATA-known-FOR-the-FEATURE-GROUP-in-EXAM
;  '("B53"
;     (("B53B" (("B53B" "W" (#<FEATURE-CHART-ENTRY 374007300>) NIL)))
;      ("B53A" (("B53A" "W" (#<FEATURE-CHART-ENTRY 374005464>) NIL) ("B53A" "X" (#<FEATURE-CHART-ENTRY 374
005405>) NIL))))))
;((("X" " " " " " " " " " " " " " " " ")
;  ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T5
3G" "T54C" "T56J" "T56V"
;   "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E" "T59G" "T5
1R" "T51V" "T60E" "T50A"
;   "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R" "T56L" "T58C" "T6
0R" "T56C" "T54V" "T50Z"
;   "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T5
8J")
;  NIL)
; (("W" " " " " " " " " " " " " " " " ")
;  ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T5
3G" "T54C" "T56J" "T56V"
;   "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E" "T59G" "T5
1R" "T51V" "T60E" "T50A"
;   "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R" "T56L" "T58C" "T6
0R" "T56C" "T54V" "T50Z"
;   "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T5
8J")
;  NIL))


(defun   ALL-the-DATA-known-FOR-the-FEATURE-GROUP-in-EXAM (ls) ;;;;;;   reference to %%%%%%%%%%
  (let ((result nil) (final-result nil))
    (loop for i in (second ls)
```

```
          do
        (push (trial-before  i) result))
     (loop for i in result
          do
       (loop for j in i
            do
         (push  j final-result)))
     (remove-duplicates final-result :test #'equal)))


(defun trial-before  (feature-possibility-COMMON)  ;;;;; ("B53B" (("B53B" "W" (#<FEATURE-CHART-ENTRY 3740
07300>) NIL)))
;;;;;;;  Coming  from one of the  *GROUPED*   FEATURE-GROUPS   ie.  B53:
      ;;;;;;; ("B53"        ;;;;;;; %%%%%%%%%%
      ;;;;;(("B53B" (("B53B" "W" (#<FEATURE-CHART-ENTRY 374007300>) NIL)))
      ;;;;;("B53A" (("B53A" "W" (#<FEATURE-CHART-ENTRY 374005464>) NIL) ("B53A" "X" (#<FEATURE-CHART-ENTR
Y 374005405>) NIL))))))
      ;;;;;   For the same reason we  FORMAT in this way the "0000"   Pseudo-Feature , from "000" pseudo
Group, from *GROUPED*
  (let ((help nil) (result nil))
    (loop for i in      *grouped*
         do
      (cond ((string-equal (car i) "000")
               (setq help (car (cadr i))))))

      ;;;;;;  help  ----  ("0000" (("0000" " " NIL ((NIL NIL (" " NIL) NIL ("M" "S" "W" "X" "Y" "Z"))))))


    (let* ((PSEUDO-COMMON (test (all-trims-common-with-RESTRICTIONS
                                  feature-possibility-COMMON help))))
PSEUDO-COMMON)))



;;;;   NOW  WE WORK  One BY One the  Functions


;;;;;;;;;;    2222222222222

(defun   CHECK-ALL-DIFFERENT-COMBINATIONS-with-the-same-reference-for-the-SAME-FEATURE-GROUP (ls)
         ;;;;;;;;;;  ie. (**("B25G" "B51F") ("B40G" "B16y")),  with reference to the same FEATURE GROUP
ie. "B53"
        (loop for i in ls
             do
          (CHECK-THE-SPECIFIC-in-sequence-COMBINATION-first i)))


;;;;;;;;;;   33333333333333

(defun   CHECK-THE-SPECIFIC-in-sequence-COMBINATION-first   (ls)
        ;;;;  The FOURTH of each element of the "CHECK-A-NUMBER-OF-feature-combinations-FOR-THE-SAME-PURP
OSE" ' (ls)
        ;;;;  ie.      (("B25G" "B51F"))
        ;;;;  IT COULD be :        (("B25G" "B51F"    "G37G" "W67W"))  ,   FOR THIS REASON THE function:
        ;;;;   "CHECK-THE-SEQUENCE-OF-THE-COMBINATION"     does exist.   WE HAVE TO   score  THEM, FIRST.
  (let ((help (FIRST (create-combinations  (PUT-IN-THE-SEQUENCE ls)))))  ;;; WE need only the combination
 of the first
                                                                        ;;;  Master feature with the res
t of the others ie.
                        ;;;;;Command: (CREATE-COMBINATIONS '(a b c d))
                        ;;;;;(((A D) (A C) (A B)) ((B D) (B C)) ((C D)))
    (loop for i in help
         do
      (CHECK-a-SPECIFIC-feature-combination (first i) (second i)))))



;;;;;;;;     444444444444

(defun    CHECK-a-SPECIFIC-feature-combination   (master-feature  other-feature)
   (let* ((result nil)
          (help1 (GET-FROM-*GROUPED*-global-the-Feature-we-concern  master-feature))
```

```
          (help2 (GET-FROM-*GROUPED*-global-the-Feature-we-concern  other-feature))
          (common (all-trims-common-with-restrictions help1 help2))
          (common-TRIMS-common-RESTRICTIONS (let ((temporary nil)) ;BECAUSE  WE DON'T WANT TO CHECK the C
OMBINATION-restrictions
                                              (loop for i in (TEST common)
                                                    do
                                                (push (list (first i) (second i)) temporary))
                                              temporary)))
;     (print 'TEMPORARY) (prin1 common-TRIMS-common-RESTRICTIONS)
;     (print '*THE-COMMON-GROUND-FOR-STATISTICS-EXAMINATION*) (prin1  *THE-COMMON-GROUND-FOR-STATISTICS-E
XAMINATION*)
;     (print '******)

     (cond ((equal  *THE-COMMON-GROUND-FOR-STATISTICS-EXAMINATION* nil)  ;;;;;; FIRST   run?
              (setq    *THE-COMMON-GROUND-FOR-STATISTICS-EXAMINATION*  common-TRIMS-common-RESTRICTIONS))
           (t
            (loop for   list-of-BASE-and-TERRITORIES   in  common-TRIMS-common-RESTRICTIONS
                  do
              (loop for   list-of-BASE-and-TERRITORIES-ALREADY-in   in    *THE-COMMON-GROUND-FOR-STATISTI
CS-EXAMINATION*
                  do

                (let ((COMMON-BASE-FEATURES (reverse   ;;;; IT is coming in the other way around
                                              (no-opposite (first list-of-BASE-and-TERRITORIES)
                                                           (first  list-of-BASE-and-TERRITORIES-ALREADY-in)
)))
                      (COMMON-TERRITORIES  (intersection (second  list-of-BASE-and-TERRITORIES)
                                                         (second  list-of-BASE-and-TERRITORIES-ALREADY-in
)
                                                         :test  #'string-equal)))
;                 (print COMMON-BASE-FEATURES) (print COMMON-TERRITORIES)
;                 (print '****************************************)
                  (cond
                    ((and COMMON-BASE-FEATURES COMMON-TERRITORIES)
                        (push (list COMMON-BASE-FEATURES COMMON-TERRITORIES)
                              result))))))
           (setq *THE-COMMON-GROUND-FOR-STATISTICS-EXAMINATION* (remove-duplicates result :test #'equal)
)))
))



(defun  GET-FROM-*GROUPED*-global-the-Feature-we-concern (a-feature)
  (let ((result nil))
    (loop for  FEATURE-GROUP in  *grouped*
          do
      (loop for FEATURE in  (second FEATURE-GROUP)
            do
        (cond ((string-equal (first FEATURE) a-feature)
                    (setq result FEATURE)))))
result))



;*****************************************************************************************


;Command: (TOP-find-the-FEAT-COMBINATIONS-OF-the-relative-instances-SYSTEM *FEATURE-GROUPS-NEED-TO-BE-CHE
CKED-IN-THE-SYSTEM*)
;(("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" ("B25G+B51F") "B53")
; ("HAD" "SEAT-FRONT-COMPLETE      " "ELECTRIC        " ("B40F+B69D") "B44"))
;Command: (another-start-for-EXAMINATION-of-FEATURES)
;(("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" MAY-A-MUST "B53")
; ("HAD" "SEAT-FRONT-COMPLETE      " "ELECTRIC        " OPTIONAL "B44"))
;*************************************************************************
;Command: (CHECK-A-NUMBER-OF-feature-combinations-FOR-THE-SAME-PURPOSE '("HAH" "HEADRSTNT-COMP-FRONT     "
 "BOLSTER-NODDING" (("B25
;G" "B51F")) "B53"))
;NIL
;Command: *grouped*
;(("B40"
;  (("B40F" (("B40F" "Y" (#<FEATURE-CHART-ENTRY 54302375>) NIL) ("B40F" "Z" (#<FEATURE-CHART-ENTRY 543024
36>) NIL)))
```

```
;   ("B40B" (("B40B" "W" (#<FEATURE-CHART-ENTRY 54302273>) NIL)))
;   ("B40Y"
;    (("B40Y" "Z" (#<FEATURE-CHART-ENTRY 54302477> #<FEATURE-CHART-ENTRY 54302540>) NIL)
;     ("B40Y" "Y" (#<FEATURE-CHART-ENTRY 54302601>) NIL)))
;   ("B40C" (("B40C" "M" (#<FEATURE-CHART-ENTRY 54302334>) NIL)))))
; ("B51"
;  (("B51F" (("B51F" "Z" (#<FEATURE-CHART-ENTRY 54301416>) NIL) ("B51F" "Y" (#<FEATURE-CHART-ENTRY 543014
57>) NIL)))
;   ("B51L"
;    (("B51L" "Z" (#<FEATURE-CHART-ENTRY 54301622> #<FEATURE-CHART-ENTRY 54301663>) NIL)
;     ("B51L" "Y" (#<FEATURE-CHART-ENTRY 54301724>) NIL)))
;   ("B51R"
;    (("B51R" "X" (#<FEATURE-CHART-ENTRY 54301765> #<FEATURE-CHART-ENTRY 54302026>) NIL)
;     ("B51R" "W" (#<FEATURE-CHART-ENTRY 54302067>) NIL)))
;   ("B51K" (("B51K" "S" (#<FEATURE-CHART-ENTRY 54301520>) NIL) ("B51K" "M" (#<FEATURE-CHART-ENTRY 543015
61>) NIL)))))
; ("B69" (("B69D" ("B69D" "Y" (#<FEATURE-CHART-ENTRY 54302171>) NIL) ("B69D" "X" (#<FEATURE-CHART-ENTRY
54302232>) NIL)))))
; ("B44"
;  (("B44B" (("B44B" "S" (#<FEATURE-CHART-ENTRY 54302642>) NIL) ("B44B" "M" (#<FEATURE-CHART-ENTRY 543027
03>) NIL)))
;   ("B44R"
;    (("B44R" "Z" (#<FEATURE-CHART-ENTRY 54302744>) NIL) ("B44R" "Y" (#<FEATURE-CHART-ENTRY 54303005>) NI
L)
;     ("B44R" "X" (#<FEATURE-CHART-ENTRY 54303046>) NIL) ("B44R" "W" (#<FEATURE-CHART-ENTRY 54303107>) NI
L)))))
; ("B25" (("B25G" (("B25G" " " NIL ((NIL NIL (" " NIL) NIL ("M" "S" "W" "X" "Y" "Z")))))))
; ("B53"
;  (("B53B" (("B53B" "W" (#<FEATURE-CHART-ENTRY 54303252>) NIL)))
;   ("B53A" (("B53A" "W" (#<FEATURE-CHART-ENTRY 54303150>) NIL) ("B53A" "X" (#<FEATURE-CHART-ENTRY 543032
11>) NIL))))))
;Command: (test (all-trims-common-with-RESTRICTIONS '("B51F" (("B51F" "Z" # NIL) ("B51F" "Y" # NIL))) '("
B25G" (("B25G" " " NIL #)
;))))
;((("Y" " " " " " " " " " " " " " ")
;  ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T5
3G" "T54C" "T56J" "T56V"
;   "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E" "T59G" "T5
1R" "T51V" "T60E" "T50A"
;   "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R" "T56L" "T58C" "T6
0R" "T56C" "T54V" "T50Z"
;   "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T5
8J")
;  (("+" "B53B")))
; (("Z" " " " " " " " " " " " " " ")
;  ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "T5
3G" "T54C" "T56J" "T56V"
;   "T57X" "T59L" "T59V" "T53N" "T50R" "T53C" "T57E" "T59E" "T59G" "T51R" "T51V" "T60E" "T50A" "T50V" "T5
2R" "T55T" "T57G" "T57V"
;   "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R" "T56L" "T58C" "T60R" "T56C" "T54V" "T50Z" "T51G" "T5
2G" "T54N" "T55J" "T56E"
;   "T58G" "T58R" "T50X" "T53A" "T54H" "T54T" "T56U" "T58N" "T57J" "T58J")
;  (("+" "B53B"))))
;Command: (test (all-trims-common-with-RESTRICTIONS '("B40F" (("B40F" "Y" # NIL) ("B40F" "Z" # NIL))) '("
B69D" (("B69D" "Y" # NIL)
; ("B69D" "X" # NIL)))))
;((("Y" " " " " " " " "D" "B" " ") ("T50R" "T57E" "T59E" "T59G" "T53N" "T52J") NIL))
;Command:
```

```
;*****************************************************************************************************
****************
;*****************************************************************************************************
****************
;*****************************************************************************************************
****************

;Command: (setq R8 (translate-make-the-link (cadr  (ASSOC "1109aa" *SEPARATE-THE-DIFFERENT-SSSVPGS-IN-
KATERINA* :TEST #'STRING-
;EQUAL))))
;(("CHART-ARRANGEMENTS        " "ST/F COMP       " "HAC10004     " NIL "")
; ("SEAT-FRONT COMPLETE   -R " "MANUAL          " "HAD10194XXX " ("B51" "B44") "+B51K+B44B")
; ("SEAT-FRONT COMPLETE   -L " "MANUAL          " "HAD10195XXX " ("B51" "B44") "+B51K+B44B")
```

```
; ("SEAT-FRONT COMPLETE    -R " "MANUAL          " "HAD10474XXX " ("B69" "B51" "B44") "+B69D+B51K+B44B")
; ("SEAT-FRONT COMPLETE    -L " "MANUAL          " "HAD10475XXX " ("B69" "B51" "B44") "+B69D+B51K+B44B")
; ("SEAT-FRONT COMPLETE       " "MANUAL-PASSR    " "HAD10480XXX " ("B69" "B51" "B44") "+B69D+B51L+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL-PASSR    " "HAD10480XXX " ("B69" "B51" "B44") "+B69D+B51L+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL-DRIVER   " "HAD10481XXX " ("B69" "B51" "B44") "+B69D+B51L+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL-DRIVER   " "HAD10481XXX " ("B69" "B51" "B44") "+B69D+B51L+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL-PASSR    " "HAD10482XXX " ("B69" "B51" "B44") "+B69D+B51F+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL-PASSR    " "HAD10482XXX " ("B69" "B51" "B44") "+B69D+B51F+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL-DRIVER   " "HAD10483XXX " ("B69" "B51" "B44") "+B69D+B51F+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL-DRIVER   " "HAD10483XXX " ("B69" "B51" "B44") "+B69D+B51F+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL          " "HAD10543XXX " ("B69" "B53" "B51" "B45" "B44") "+B69D+B5
3A+B51R+B45B+B44R")
; ("SEAT-FRONT COMPLETE    -R " "MANUAL          " "HAD10546XXX " ("B53" "B51" "B45" "B44") "+B53B+B51R+B45
B+B44R")
; ("SEAT-FRONT COMPLETE    -L " "MANUAL          " "HAD10547XXX " ("B53" "B51" "B45" "B44") "+B53B+B51R+B45
B+B44R")
; ("SEAT-FRONT COMPLETE    -R " "MANUAL          " "HAD10548XXX " ("B53" "B51" "B45" "B44") "+B53A+B51R+B45
B+B44R")
; ("SEAT-FRONT COMPLETE    -L " "MANUAL          " "HAD10549XXX " ("B69" "B53" "B51" "B45" "B44") "+B69D+B5
3A+B51R+B45B+B44R")
; ("SEAT-FRONT COMPLETE    -R " "MANUAL          " "HAD10550XXX " ("B53" "B51" "B45" "B44") "+B53B+B51R+B45
B+B44R")
; ("SEAT-FRONT COMPLETE    -L " "MANUAL          " "HAD10551XXX " ("B53" "B51" "B45" "B44") "+B53B+B51R+B45
B+B44R")
; ("SEAT-FRONT COMPLETE    -R " "MANUAL          " "HAD10552XXX " ("B69" "B53" "B51" "B45" "B44") "+B69D+B5
3A+B51R+B45B+B44R")
; ("SEAT-FRONT COMPLETE    -L " "MANUAL          " "HAD10553XXX " ("B53" "B51" "B45" "B44") "+B53A+B51R+B45
B+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL          " "HAD10554XXX " ("B69" "B53" "B51" "B45" "B44") "+B69D+B5
3A+B51R+B45B+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL-DRIVER   " "HAD10561XXX " ("B69" "B53" "B51" "B45" "B44") "+B69D+B5
3A+B51R+B45B+B44R")
; ("SEAT-FRONT COMPLETE    -R " "MANUAL-DRIVER   " "HAD10562XXX " ("B53" "B51" "B45" "B44") "+B53A+B51R+B45
B+B44R")
; ("SEAT-FRONT COMPLETE    -L " "MANUAL-DRIVER   " "HAD10563XXX " ("B69" "B53" "B51" "B45" "B44") "+B69D+B5
3A+B51R+B45B+B44R")
; ("SEAT-FRONT COMPLETE    -R " "MANUAL-PASSR    " "HAD10564XXX " ("B69" "B53" "B51" "B45" "B44") "+B69D+B5
3A+B51R+B45B+B44R")
; ("SEAT-FRONT COMPLETE    -L " "MANUAL-PASSR    " "HAD10565XXX " ("B53" "B51" "B45" "B44") "+B53A+B51R+B45
B+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL-PASSR    " "HAD10568XXX " ("B69" "B53" "B51" "B45" "B44") "+B69D+B5
3A+B51R+B45B+B44R")
; ("HEADRSTNT-COMP-FRONT      " "BOLSTER-NODDING" "HAH10067XXX " ("B53" "B51" "B45" "B25") "+B53A+B51R+B45
B+B25G")
; ("HEADRSTNT-COMP-FRONT      " "BOLSTER-NODDING" "HAH10068XXX " ("B53" "B51" "B45" "B25") "+B53B+B51R+B45
B+B25G")
; ("COVER-EXPENDABLE-ST/F     " "NON ELESTICATED" "HAM10002    " ("B45") "+B45B")
; ("INSTR DRG-HEATER HARN     " "SEAT-FRONT      " "HGE10001    " ("B69") "+B69D")
; ("SEAT-FRONT COMPLETE    -R " "MANUAL          " "HAD10002XXX " ("B44" "B51") "+B44B+B40C")
; ("SEAT-FRONT COMPLETE    -L " "MANUAL          " "HAD10003XXX " ("B44" "B51") "+B44B+B40C")
; ("SEAT-FRONT COMPLETE    -R " "MANUAL-DRIVER   " "HAD10178XXX " ("B69" "B51" "B44") "+B69D+B51K+B44B")
; ("SEAT-FRONT COMPLETE    -R " "MANUAL-DRIVER   " "HAD10178XXX " ("B69" "B51" "B44") "+B69D+B51K+B44B")
; ("SEAT-FRONT COMPLETE    -L " "MANUAL-DRIVER   " "HAD10179XXX " ("B69" "B51" "B44") "+B69D+B51K+B44B")
; ("SEAT-FRONT COMPLETE    -L " "MANUAL-DRIVER   " "HAD10179XXX " ("B69" "B51" "B44") "+B69D+B51K+B44B")
; ("SEAT-FRONT COMPLETE    -R " "MANUAL          " "HAD10006XXX " ("B69" "B44" "B51") "+B69D+B44R+B40B")
; ("SEAT-FRONT COMPLETE    -R " "MANUAL          " "HAD10008XXX " ("B69" "B44" "B51") "+B69D+B44R+B40B")
; ("SEAT-FRONT COMPLETE    -L " "MANUAL          " "HAD10009XXX " ("B69" "B44" "B51") "+B69D+B44R+B40B")
; ("SEAT-FRONT COMPLETE    -R " "ELECTRIC        " "HAD10012XXX " ("B69" "B44" "B51") "+B69D+B44R+B40B")
; ("SEAT-FRONT COMPLETE    -L " "ELECTRIC        " "HAD10013XXX " ("B69" "B44" "B51") "+B69D+B44R+B40B")
; ("SEAT-FRONT COMPLETE       " "MANUAL-DRIVER   " "HAD10186XXX " ("B69" "B51" "B44") "+B69D+B51L+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL-DRIVER   " "HAD10186XXX " ("B69" "B51" "B44") "+B69D+B51L+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL-PASSR    " "HAD10187XXX " ("B69" "B51" "B44") "+B69D+B51L+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL-PASSR    " "HAD10187XXX " ("B69" "B51" "B44") "+B69D+B51L+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL-PASSR    " "HAD10188XXX " ("B69" "B51" "B44") "+B69D+B51L+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL-PASSR    " "HAD10188XXX " ("B69" "B51" "B44") "+B69D+B51L+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL-DRIVER   " "HAD10189XXX " ("B69" "B51" "B44") "+B69D+B51L+B44R")
; ("SEAT-FRONT COMPLETE       " "MANUAL-DRIVER   " "HAD10189XXX " ("B69" "B51" "B44") "+B69D+B51L+B44R")
; ("SEAT-FRONT COMPLETE    -R " "MANUAL          " "HAD10030XXX " ("B69" "B51") "+B69D+B40Y")
; ("SEAT-FRONT COMPLETE    -L " "MANUAL          " "HAD10031XXX " ("B69" "B51") "+B69D+B40Y")
; ("SEAT-FRONT COMPLETE    -R " "MANUAL          " "HAD10032XXX " ("B69" "B51") "+B69D+B40Y")
; ("SEAT-FRONT COMPLETE    -L " "MANUAL          " "HAD10033XXX " ("B69" "B51") "+B69D+B40Y")
; ("SEAT-FRONT COMPLETE    -R " "ELECTRIC        " "HAD10010XXX " ("B69" "B44" "B51") "+B69D+B44R+B40B")
; ("SEAT-FRONT COMPLETE    -L " "ELECTRIC        " "HAD10011XXX " ("B69" "B44" "B51") "+B69D+B44R+B40B")
; ("SEAT-FRONT COMPLETE    -R " "ELECTRIC        " "HAD10026XXX " ("B69" "B44" "B51") "+B69D+B44R+B40Y")
; ("SEAT-FRONT COMPLETE    -L " "ELECTRIC        " "HAD10027XXX " ("B69" "B44" "B51") "+B69D+B44R+B40Y")
; ("SEAT-FRONT COMPLETE    -L " "ELECTRIC        " "HAD10029XXX " ("B69" "B44" "B51") "+B69D+B44R+B40Y")
```

```
; ("SEAT-FRONT COMPLETE   -R " "ELECTRIC        " "HAD10028XXX " ("B69" "B44" "B51") "+B69D+B44R+B40Y")
; ("SEAT-FRONT COMPLETE   -R " "ELECTRIC        " "HAD10034XXX " ("B69" "B51") "+B69D+B40F")
; ("SEAT-FRONT COMPLETE   -L " "ELECTRIC        " "HAD10035XXX " ("B69" "B51") "+B69D+B40F")
; ("SEAT-FRONT COMPLETE   -R " "ELECTRIC        " "HAD10036XXX " ("B69" "B51") "+B69D+B40F")
; ("SEAT-FRONT COMPLETE   -L " "ELECTRIC        " "HAD10037XXX " ("B69" "B51") "+B69D+B40F")
; ("SEAT-FRONT COMPLETE      " "MANUAL-DRIVER   " "HAD10190XXX " ("B69" "B51" "B44") "+B69D+B51F+B44R")
; ("SEAT-FRONT COMPLETE      " "MANUAL-DRIVER   " "HAD10190XXX " ("B69" "B51" "B44") "+B69D+B51F+B44R")
; ("SEAT-FRONT COMPLETE      " "MANUAL-PASSR    " "HAD10191XXX " ("B69" "B51" "B44") "+B69D+B51F+B44R")
; ("SEAT-FRONT COMPLETE      " "MANUAL-PASSR    " "HAD10192XXX " ("B69" "B51" "B44") "+B69D+B51F+B44R")
; ("SEAT-FRONT COMPLETE      " "MANUAL-PASSR    " "HAD10192XXX " ("B69" "B51" "B44") "+B69D+B51F+B44R")
; ("SEAT-FRONT COMPLETE      " "MANUAL-DRIVER   " "HAD10193XXX " ("B69" "B51" "B44") "+B69D+B51F+B44R")
; ("SEAT-FRONT COMPLETE      " "MANUAL-DRIVER   " "HAD10193XXX " ("B69" "B51" "B44") "+B69D+B51F+B44R")
; ("HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" "HAH10022XXX " ("B51" "B25") "+B51F+B25G")
; ("HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" "HAH10023XXX " ("B51" "B25") "+B51K+B25G")
; ("HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" "HAH10023XXX " ("B51" "B25") "+B51L+B25G"))
;Command:
;Command: (the-runner      R8)
;NIL
;Command: *DEFINATELY-OPTIONAL-FEATURE-GROUPS-FOUND*
;NIL
;Command: *MUST-FEATURES*
;(("HGE" "INSTR DRG-HEATER HARN    " "SEAT-FRONT     " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTO
RY" ("B69"))
; ("HAM" "COVER-EXPENDABLE-ST/F    " "NON ELESTICATED" "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTO
RY" ("B45"))
; ("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTO
RY" ("B51" "B25"))
; ("HAC" "CHART-ARRANGEMENTS       " "ST/F COMP      " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTO
RY" NIL)
; ("HAD" "SEAT-FRONT COMPLETE   -L " "MANUAL-PASSR   " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTO
RY"
;  ("B53" "B51" "B45" "B44"))
; ("HAD" "SEAT-FRONT COMPLETE   -L " "MANUAL-DRIVER  " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTO
RY" ("B69" "B51" "B44"))
; ("HAD" "SEAT-FRONT COMPLETE   -L " "MANUAL         " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTO
RY" ("B51"))
; ("HAD" "SEAT-FRONT COMPLETE   -L " "ELECTRIC       " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTO
RY" ("B69" "B51"))
; ("HAD" "SEAT-FRONT COMPLETE   -R " "MANUAL-PASSR   " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTO
RY"
;  ("B69" "B53" "B51" "B45" "B44"))
; ("HAD" "SEAT-FRONT COMPLETE   -R " "MANUAL-DRIVER  " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTO
RY" ("B51" "B44"))
; ("HAD" "SEAT-FRONT COMPLETE   -R " "MANUAL         " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTO
RY" ("B51"))
; ("HAD" "SEAT-FRONT COMPLETE   -R " "ELECTRIC       " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTO
RY" ("B69" "B51"))
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-PASSR   " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTO
RY" ("B69" "B51" "B44"))
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL         " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTO
RY"
;  ("B69" "B53" "B51" "B45" "B44"))
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-DRIVER  " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTO
RY" ("B69" "B51" "B44")))
;Command: *THE-OPTIONAL-FEATURE-GROUPS-FROM-STATISTICS*
;(("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-DRIVER  " "IT-IS-OPTIONAL" "CHECKED-FROM-STATISTICS" ("B53"
"B45"))
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-PASSR   " "IT-IS-OPTIONAL" "CHECKED-FROM-STATISTICS" ("B53"
"B45"))
; ("HAD" "SEAT-FRONT COMPLETE   -R " "MANUAL         " "IT-IS-OPTIONAL" "CHECKED-FROM-STATISTICS" ("B44"
"B69" "B53" "B45"))
; ("HAD" "SEAT-FRONT COMPLETE   -R " "MANUAL-DRIVER  " "IT-IS-OPTIONAL" "CHECKED-FROM-STATISTICS" ("B53"
"B45"))
; ("HAD" "SEAT-FRONT COMPLETE   -L " "MANUAL         " "IT-IS-OPTIONAL" "CHECKED-FROM-STATISTICS" ("B44"
"B69" "B53" "B45"))
; ("HAD" "SEAT-FRONT COMPLETE   -L " "MANUAL-DRIVER  " "IT-IS-OPTIONAL" "CHECKED-FROM-STATISTICS" ("B53"
"B45"))
; ("HAH" "HEADRSTNT-COMP-FRONT     " "BOLSTER-NODDING" "IT-IS-OPTIONAL" "CHECKED-FROM-STATISTICS" ("B53"
"B45")))
;Command: *FEATURE-GROUPS-NEED-TO-BE-CHECKED-IN-THE-SYSTEM*
;(("HAD" "SEAT-FRONT COMPLETE   -R " "ELECTRIC       " "NEEDS-MORE-EXAMINATION-THE-FEATURE-GROUP" "B44")
; ("HAD" "SEAT-FRONT COMPLETE   -R " "MANUAL-DRIVER  " "NEEDS-MORE-EXAMINATION-THE-FEATURE-GROUP" "B69")
; ("HAD" "SEAT-FRONT COMPLETE   -L " "ELECTRIC       " "NEEDS-MORE-EXAMINATION-THE-FEATURE-GROUP" "B44"))
;Command: *THE-RESULT-OF-THE-SEARCH-WITHIN-THE-SYSTEM*
;(("HAD" "SEAT-FRONT COMPLETE   -L " "ELECTRIC       " "DEFINATELY-OPTIONAL" "CHECKED-IN-THE-SYSTEM" "B44
")
```

```
; ("HAD" "SEAT-FRONT COMPLETE   -R " "MANUAL-DRIVER  " "DEFINATELY-OPTIONAL" "CHECKED-IN-THE-SYSTEM" "B69
")
; ("HAD" "SEAT-FRONT COMPLETE   -R " "ELECTRIC       " "DEFINATELY-OPTIONAL" "CHECKED-IN-THE-SYSTEM" "B44
"))
;Command:
;Command: (function-on-the-top-of-the-runner)
;(("HGE" (("INSTR DRG-HEATER HARN " (("SEAT-FRONT     " (("OPTIONALS" NIL) ("MUSTS" ("B69"))) "INDICATIVE
")))))
; ("HAM" (("COVER-EXPENDABLE-ST/F " (("NON ELESTICATED" (("OPTIONALS" NIL) ("MUSTS" ("B45"))) "INDICATIVE
")))))
; ("HAH" (("HEADRSTNT-COMP-FRONT  " (("BOLSTER-NODDING" (("OPTIONALS" ("B53" "B45")) ("MUSTS" ("B51" "B25
"))) "QUITE-POSSIBLE"))))
;  )
; ("HAC" (("CHART-ARRANGEMENTS    " (("ST/F COMP      " (("OPTIONALS" NIL) ("MUSTS" (NIL))) "INDICATIVE")
))))
; ("HAD"
;  (("SEAT-FRONT COMPLETE   "
;    (("ELECTRIC       " (("OPTIONALS" ("B44")) ("MUSTS" ("B69" "B51"))) "QUITE-POSSIBLE")
;     ("MANUAL         " (("OPTIONALS" ("B69" "B44" "B53" "B45")) ("MUSTS" ("B51"))) "QUITE-POSSIBLE")
;     ("MANUAL-DRIVER  " (("OPTIONALS" ("B53" "B45" "B69")) ("MUSTS" ("B51" "B44"))) "QUITE-POSSIBLE")
;     ("MANUAL-PASSR   " (("OPTIONALS" ("B53" "B45" "B69")) ("MUSTS" ("B51" "B44"))) "QUITE-POSSIBLE"))))
))
;Command:
```

```
 ;;;-*- mode: lisp ; syntax: common-lisp ; package:  austin-rover-*-

;(defun the-Master-function  () ;;;;; PROBABLY  2 arguments as the  "function-on-the-top-of-THE-RUNNER" f
unction should have.
;  (the-runner)        ;;;;;;;IT IS in the file : "statice-R8.dat.43"
;  (function-on-the-top-of-THE-RUNNER)





;*****************************************************************************************************
;Command: (function-on-the-top-of-THE-RUNNER)
;(("HGE" (("INSTR DRG-HEATER-HARN   " (("SEAT-FRONT      " (("OPTIONALS" NIL) ("MUSTS" ("B69"))) "INDICATI
VE")))))
; ("HAM" (("COVER-EXPENDABLE-ST/F   " (("NON ELESTICATED" (("OPTIONALS" NIL) ("MUSTS" ("B45"))) "INDICATI
VE")))))
; ("HAH"
;  (("HEADRSTNT-COMP-FRONT     "
;    (("BOLSTER-NODDING" (("OPTIONALS" ("B53" "B45")) ("MUSTS" ("B25" "B51"))) "QUITE-POSSIBLE")
;     ("BOLSTER-ADJSL  " (("OPTIONALS" NIL) ("MUSTS" ("B25" "B51"))) "INDICATIVE")))))
; ("HCA" (("COVER ASSY-SEAT-FRONT   " (("CUSHION         " (("OPTIONALS" NIL) ("MUSTS" ("B51"))) "INDICATI
VE")))))
; ("HAD"
;  (("SEAT-FRONT-COMPLETE      " (("ELECTRIC       " (("OPTIONALS" NIL) ("MUSTS" ("B51" "B69"))) "INDICATI
VE")))
;   ("SEAT-FRONT COMPLETE      "
;    (("ELECTRIC        " (("OPTIONALS" ("B44")) ("MUSTS" ("B51" "B69"))) "QUITE-POSSIBLE")
;     ("MANUAL-DRIVER  " (("OPTIONALS" ("B45" "B53" "B69")) ("MUSTS" ("B51" "B44"))) "QUITE-POSSIBLE")
;     ("MANUAL         " (("OPTIONALS" ("B69" "B45" "B53")) ("MUSTS" ("B51" "B44"))) "QUITE-POSSIBLE")
;     ("MANUAL-PASSR   " (("OPTIONALS" ("B53" "B45" "B69")) ("MUSTS" ("B51" "B44"))) "QUITE-POSSIBLE"))))
)
; ("HBA" (("COVER ASSY-SEAT-FRONT   " (("SQUAB           " (("OPTIONALS" NIL) ("MUSTS" ("B51"))) "INDICATI
VE"))))))
;Command:


;Command: (group-second-time *VARIABLE-FREEZED*)
;(("HGE" (("INSTR DRG-HEATER-HARN   " (("SEAT-FRONT      " "INDICATIVE")))))
; ("HAM" (("COVER-EXPENDABLE-ST/F   " (("NON ELESTICATED" "INDICATIVE")))))
; ("HAH" (("HEADRSTNT-COMP-FRONT    " (("BOLSTER-NODDING" "QUITE-POSSIBLE") ("BOLSTER-ADJSL  " "INDICATIV
E")))))
; ("HCA" (("COVER ASSY-SEAT-FRONT   " (("CUSHION         " "INDICATIVE")))))
; ("HAD"
;  (("SEAT-FRONT-COMPLETE      " (("ELECTRIC       " "INDICATIVE")))
;   ("SEAT-FRONT COMPLETE      "
;    (("ELECTRIC        " "QUITE-POSSIBLE") ("MANUAL-DRIVER  " "QUITE-POSSIBLE") ("MANUAL         " "QUITE
-POSSIBLE")
;     ("MANUAL-PASSR   " "QUITE-POSSIBLE")))))
; ("HBA" (("COVER ASSY-SEAT-FRONT   " (("SQUAB           " "INDICATIVE"))))))
;Command:


(defun   function-on-the-top-of-THE-RUNNER  ()      ;;;;; PROBABLY   2 arguments  as  THE-RUNNER function
 should have.
  (let* ((HELP   (group-second-time *VARIABLE-FREEZED*))
         (VARIABLE-HELP  HELP))

    (loop for i in HELP
          do
      (let* ((family  (car i)) (other (second i)))
        (loop for PRIMARY-reference in other
              do
          (let (
                (PRIMARY-description (car PRIMARY-reference))
                (SECONDARYs-description-references (second PRIMARY-reference)))
;       (print 'family) (prinl family)(print other)(print 'PRIMARY-description) (prinl PRIMARY-descriptio
n) (print 'SECONDARYs-description-references)
;       (prinl SECONDARYs-description-references)
;       (print '*****)
        (loop for  SECONDARY-Description-ref   in    SECONDARYs-description-references
              do
          (let* ((secondary-description  (car SECONDARY-Description-ref))
```

```
                (MUSTS-AND-MAYBES
                  (COLLECT-BOTH-Musts-AND-Maybes  family PRIMARY-description secondary-description)))
                  ;;;; Command: (COLLECT-BOTH-Musts-AND-Maybes "HAD" "SEAT-FRONT COMPLETE      " "MANUAL-
PASSR    ")
                  ;;;; (("OPTIONALS" ("B53" "B45" "B69")) ("MUSTS" ("B51" "B44")))
            (push MUSTS-AND-MAYBES

(cdr (assoc secondary-description
       (second (assoc PRIMARY-description
                      (second (assoc family
                                     VARIABLE-HELP
                                     :test #'string-equal))
                      :test #'string-equal))
       :test #'string-equal))
           ) ;;;;;;   push
))))))
VARIABLE-HELP))


(defun    COLLECT-BOTH-Musts-AND-Maybes  (family PRIMARY-description secondary-description)
  (let ((OPTIONAL-FEATURES-of-PART-DESCRIPTION  nil)
        (MUST-FEATURES-of-PART-DESCRIPTION  nil))
    (loop for i in *DEFINATELY-OPTIONAL-FEATURE-GROUPS-FOUND*
          do
      (cond ((and
                (string-equal family (car i))
                (string-equal PRIMARY-description (second i))
                (string-equal secondary-description (third i)))
             ;;;;;;;;;;;;;; I MAKE simple HERE without  introducing
             ;;;;;;;;;;;;;; the comments ie. "DEFINATELY-A-MUST", "IS-DEFINATELY-OPTIONAL-THE-FEATURE-GROU
P" etc.

             (push (list (car  (reverse i))) OPTIONAL-FEATURES-of-PART-DESCRIPTION))))
    (loop for i in *THE-OPTIONAL-FEATURE-GROUPS-FROM-STATISTICS*
          do
      (cond ((and
                (string-equal family (car i))
                (string-equal PRIMARY-description (second i))
                (string-equal secondary-description (third i)))
             ;;;;;;;;;;;;;; I MAKE simple HERE without  introducing
             ;;;;;;;;;;;;;; the comments ie. "DEFINATELY-A-MUST", "IS-DEFINATELY-OPTIONAL-THE-FEATURE-GROU
P" etc.

             (push (car  (reverse i)) OPTIONAL-FEATURES-of-PART-DESCRIPTION))))
    (loop for i in *MUST-FEATURES*
          do
      (cond ((and
                (string-equal family (car i))
                (string-equal PRIMARY-description (second i))
                (string-equal secondary-description (third i)))
             ;;;;;;;;;;;;;; I MAKE simple HERE without  introducing
             ;;;;;;;;;;;;;; the comments ie. "DEFINATELY-A-MUST", "IS-DEFINATELY-OPTIONAL-THE-FEATURE-GROU
P" etc.

             (push (car  (reverse i)) MUST-FEATURES-of-PART-DESCRIPTION))))
    (loop for i in *THE-RESULT-OF-THE-SEARCH-WITHIN-THE-SYSTEM*
          do
      (cond ((and
                (string-equal family (car i))
                (string-equal PRIMARY-description (second i))
                (string-equal secondary-description (third i)))
                     (cond
                       ((search  "MUST"  (fourth i))
                        (push (list (car  (reverse i)))  MUST-FEATURES-of-PART-DESCRIPTION))
                       (t (push (list (car  (reverse i)))  OPTIONAL-FEATURES-of-PART-DESCRIPTION))))))
(list (list "OPTIONALS" (only-atoms OPTIONAL-FEATURES-of-PART-DESCRIPTION))
      (list "MUSTS" (only-atoms  MUST-FEATURES-of-PART-DESCRIPTION)))
))


;Command:          (the-runner)      ;MUST HAVE    2   ARGUMENTS  : 1)  THE *model*        2) THE  Sub-Sub-S
ub-VPG
;NIL
;Command: *DEFINATELY-OPTIONAL-FEATURE-GROUPS-FOUND*
```

```
;(("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-DRIVER  " "IS-DEFINATELY-OPTIONAL-THE-FEATURE-GROUP" "CHECKED
-FROM-HISTORY" "B69")
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-PASSR   " "IS-DEFINATELY-OPTIONAL-THE-FEATURE-GROUP" "CHECKED
-FROM-HISTORY" "B69"))
;Command: *MUST-FEATURES*
;(("HGE" "INSTR DRG-HEATER-HARN   " "SEAT-FRONT      " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B69"))
; ("HAM" "COVER-EXPENDABLE-ST/F   " "NON ELESTICATED" "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B45"))
; ("HAH" "HEADRSTNT-COMP-FRONT    " "BOLSTER-ADJSL  " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B25" "B51"))
; ("HAH" "HEADRSTNT-COMP-FRONT    " "BOLSTER-NODDING" "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B25" "B51"))
; ("HCA" "COVER ASSY-SEAT-FRONT   " "CUSHION         " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B51"))
; ("HAD" "SEAT-FRONT-COMPLETE     " "ELECTRIC        " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B51" "B69"))
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL-PASSR    " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B51" "B44"))
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL          " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B51" "B44"))
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL-DRIVER   " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B51" "B44"))
; ("HBA" "COVER ASSY-SEAT-FRONT   " "SQUAB           " "DEFINATELY-A-MUST" "CHECKED-FROM-STATISTICS-HISTOR
Y" ("B51")))
;Command: *THE-OPTIONAL-FEATURE-GROUPS-FROM-STATISTICS*
;(("HAD" "SEAT-FRONT COMPLETE     " "MANUAL-DRIVER  " "IT-IS-OPTIONAL" "CHECKED-FROM-STATISTICS" ("B45" "
B53"))
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL          " "IT-IS-OPTIONAL" "CHECKED-FROM-STATISTICS" ("B69" "
B45" "B53"))
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL-PASSR    " "IT-IS-OPTIONAL" "CHECKED-FROM-STATISTICS" ("B53" "
B45"))
; ("HAH" "HEADRSTNT-COMP-FRONT    " "BOLSTER-NODDING" "IT-IS-OPTIONAL" "CHECKED-FROM-STATISTICS" ("B45"))
)

;;Command: *FEATURE-GROUPS-NEED-TO-BE-CHECKED-IN-THE-SYSTEM*
;;(("HAD" "SEAT-FRONT-COMPLETE     " "ELECTRIC        " "NEEDS-MORE-EXAMINATION-THE-FEATURE-GROUP" "B44")
;; ("HAH" "HEADRSTNT-COMP-FRONT    " "BOLSTER-NODDING" "NEEDS-MORE-EXAMINATION-THE-FEATURE-GROUP" "B53"))
;;Command:

;Command: *THE-RESULT-OF-THE-SEARCH-WITHIN-THE-SYSTEM*
;(("HAH" "HEADRSTNT-COMP-FRONT    " "BOLSTER-NODDING" "MAY-A-Must" "CHECKED-IN-THE-SYSTEM" "B53")
; ("HAD" "SEAT-FRONT-COMPLETE     " "ELECTRIC        " "DEFINATELY-OPTIONAL" "CHECKED-IN-THE-SYSTEM" "B44"
))


;;Command: *variable-freezed*
;;(("HBA" "COVER ASSY-SEAT-FRONT   " "SQUAB           " (("B51" 100)) 1 "INDICATIVE")
; ("HAD" "SEAT-FRONT COMPLETE     " "ELECTRIC        " (("B69" 100) ("B44" 80) ("B51" 100)) 2 "QUITE-POSSI
BLE")
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL-DRIVER   " (("B53" 33) ("B45" 33) ("B69" 89) ("B44" 100) ("B51
" 100)) 3
;  "QUITE-POSSIBLE")
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL          " (("B53" 31) ("B45" 31) ("B69" 69) ("B44" 100) ("B51
" 100)) 4
;  "QUITE-POSSIBLE")
; ("HAD" "SEAT-FRONT COMPLETE     " "MANUAL-PASSR    " (("B45" 38) ("B53" 38) ("B69" 88) ("B44" 100) ("B51
" 100)) 4
;  "QUITE-POSSIBLE")
; ("HAD" "SEAT-FRONT-COMPLETE     " "ELECTRIC        " (("B69" 100) ("B51" 100)) 1 "INDICATIVE")
; ("HCA" "COVER ASSY-SEAT-FRONT   " "CUSHION         " (("B51" 100)) 1 "INDICATIVE")
; ("HAH" "HEADRSTNT-COMP-FRONT    " "BOLSTER-NODDING" (("B45" 40) ("B53" 80) ("B51" 100) ("B25" 100)) 3 "
QUITE-POSSIBLE")
; ("HAH" "HEADRSTNT-COMP-FRONT    " "BOLSTER-ADJSL  " (("B51" 100) ("B25" 100)) 1 "INDICATIVE")
; ("HAM" "COVER-EXPENDABLE-ST/F   " "NON ELESTICATED" (("B45" 100)) 1 "INDICATIVE")
; ("HGE" "INSTR DRG-HEATER-HARN   " "SEAT-FRONT      " (("B69" 100)) 1 "INDICATIVE"))
;Command:
;
;;*****************************************************************************************


!!!!!!!!!!        WE  HAVE  ALREADY   run  THE FUNCTION  :   (the-runner)  for the  MODEL  and SUB....VPG
 we want.

;Command: (group-second-time *VARIABLE-FREEZED*)
```

```
;(("HGE" (("INSTR DRG-HEATER-HARN    " (("SEAT-FRONT      " "INDICATIVE")))))
; ("HAM" (("COVER-EXPENDABLE-ST/F    " (("NON ELESTICATED" "INDICATIVE")))))
; ("HAH" (("HEADRSTNT-COMP-FRONT     " (("BOLSTER-NODDING" "QUITE-POSSIBLE") ("BOLSTER-ADJSL  " "INDICATIV
E")))))
; ("HCA" (("COVER ASSY-SEAT-FRONT    " (("CUSHION         " "INDICATIVE")))))
; ("HAD"
;  (("SEAT-FRONT-COMPLETE      " (("ELECTRIC        " "INDICATIVE")))
;   ("SEAT-FRONT COMPLETE      "
;    (("ELECTRIC        " "QUITE-POSSIBLE") ("MANUAL-DRIVER  " "QUITE-POSSIBLE") ("MANUAL         " "QUITE
-POSSIBLE")
;     ("MANUAL-PASSR    " "QUITE-POSSIBLE")))))
; ("HBA" (("COVER ASSY-SEAT-FRONT    " (("SQUAB           " "INDICATIVE")))))))
;Command:


(defun group-second-time (ls)                ;;;;;*variable-freezed*
  (let ((result nil) (help (group-first-time ls)))
    (loop for i in help
          do
      (let ((family (car i)) (others (second i)))
        (push
          (list family
                (group-of-secondary-description others))
        result)))
result))


;;Command: *variable-freezed*
;;(("HBA" "COVER ASSY-SEAT-FRONT    " "SQUAB           " (("B51" 100)) 1 "INDICATIVE")
; ("HAD" "SEAT-FRONT COMPLETE      " "ELECTRIC        " (("B69" 100) ("B44" 80) ("B51" 100)) 2 "QUITE-POSSI
BLE")
; ("HAD" "SEAT-FRONT COMPLETE      " "MANUAL-DRIVER   " (("B53" 33) ("B45" 33) ("B69" 89) ("B44" 100) ("B51
" 100)) 3
;  "QUITE-POSSIBLE")
;
(defun  group-first-time (ls)   ;;;;;;      *variable-freezed*
   (let ((result nil) (record nil))
     (loop for i in ls
           do
       (push (car i) record))

     (loop for j in (remove-duplicates record :test #'string-equal)
           do
       (push (list j (the-list-of-inferiors-FIRST-TIME j ls)) result))
result))


(defun the-list-of-inferiors-FIRST-TIME (indicator ls)
  (let ((result nil))
    (loop for k in ls
          do
      (cond ((string-equal (car k) indicator)
                (push (list (second k) (third k) (sixth k)) result))))
result))




(defun  group-of-secondary-description (ls)   ;;;;;;
;  (("SEAT-FRONT-COMPLETE      " "MANUAL-DRIVER   " "QUITE-POSSIBLE")
;   ("SEAT-FRONT-COMPLETE      " "MANUAL-DRIVER   " "QUITE-POSSIBLE")
;   ("SEAT-FRONT-COMPLETE      " "MANUAL-DRIVER   " "QUITE-POSSIBLE")
;   ("SEAT-FRONT-COMPLETE      " "MANUAL-DRIVER   " "QUITE-POSSIBLE"))
;    ("SEAT-FRONTnot COMPLETE      " "MANUAL          " "QUITE-POSSIBLE")
;    ("SEAT-FRONTnot COMPLETE      " "MANUAL-takis    " "QUITE-POSSIBLE")  )  ;;;;IT COULD BE   nested.


   (let ((result nil) (record nil))
     (loop for i in ls
           do
       (push (car i) record))
```

```
    (loop for j in (remove-duplicates record :test #'string-equal)
          do
      (push (list j (the-list-of-inferiors j ls)) result))
result))


(defun the-list-of-inferiors (indicator ls)
  (let ((result nil))
    (loop for k in ls
          do
      (cond ((string-equal (car k) indicator)
             (push (list (second k) (third k)) result))))
result))


;(("HGE" (("INSTR DRG-HEATER-HARN   " (("SEAT-FRONT      " "INDICATIVE")))))
; ("HAM" (("COVER-EXPENDABLE-ST/F   " (("NON ELESTICATED" "INDICATIVE")))))
; ("HAH" (("HEADRSTNT-COMP-FRONT    " (("BOLSTER-NODDING" "QUITE-POSSIBLE") ("BOLSTER-ADJSL  " "INDICATIV
E")))))
; ("HCA" (("COVER ASSY-SEAT-FRONT   " (("CUSHION         " "INDICATIVE")))))
; ("HAD"
;  (("SEAT-FRONT-COMPLETE     " (("ELECTRIC        " "QUITE-POSSIBLE")))
;   ("SEAT-FRONT-COMPLETE     "
;    (("MANUAL-DRIVER  " "QUITE-POSSIBLE") ("MANUAL          " "QUITE-POSSIBLE") ("MANUAL-PASSR   " "QUITE
-POSSIBLE")))))
; ("HBA" (("COVER ASSY-SEAT-FRONT   " (("SQUAB           " "INDICATIVE")))))
;Command:
```

```
;;;; -*- Syntax: Common-Lisp; Package: USER; Base: 10; Mode: LISP -*-

(make-package 'austin-rover :use '(symbolics-common-lisp common-lisp common-lisp-user statice statice-use
r))
```

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: statice-user; Base: 10; Lowercase: T -*-

;;; This example demonstrates the use of the image type.  setup-image-database
;;; makes a database containing two images, and show-image-database displays
;;; the contents of the database.

;;; Make a funny-sized subset of a grid.  Show it.  Store it, get it
;;; back, and show that.

(defvar *image-test-pathname* "stat-fs:>takis>image-test")

(define-schema image-test (snapshot))

(define-entity-type snapshot ()
  ((name string :unique t :inverse-index t :cached t)
   (picture image     :area images :cached t)))

(defun setup-image-database ()
  (make-database *image-test-pathname* 'image-test)
  ;; Make a raster array holding a grid.  Note that the width is a
  ;; multiple of 32.
  (let ((base (make-raster-array 128 37 :initial-element 0 :element-type 'bit)))
    ;; Draw vertical grid lines
    (dotimes (i 128)
      (loop for j from 3 by 10 below 40 do
        (setf (raster-aref base i j) 1)))
    ;; Draw horizontal grid lines
    (dotimes (j 37)
      (loop for i from 10 by 20 below 128 do
        (setf (raster-aref base i j) 1)))
    (with-database (db *image-test-pathname*)
      (with-transaction ()
        ;; Make a snapshot whose picture is the base raster array.
        (make-snapshot :name "base" :picture base)
        ;; Make another snapshot, whose picture is a portion of the base raster array.
        (make-snapshot :name "portion"
                       :picture (make-raster-array 59 30
                                                   :displaced-to base
                                                   :displaced-conformally t
                                                   :element-type 'bit))))))
;*****************************************************************************************

(defmethod  (sys:print-self   snapshot)   (stream  &rest  ignore)
  (format stream "~A" (snapshot-name self)))


;;;;WITH THE BELOW FAUNCTION CONVERT ALL images from Ian to images which then i can put in statice.

(defun convert-Ians-arrays-to-rasters (list &aux arrays)
 (dolist (array list arrays)
   (push (make-raster-array (array-dimension array 1) ;;;width
                            (array-dimension array 0) ;;; height
                            :element-type 'bit
                            :initial-contents (top-of-all array (array-dimension array 1) (array-dimensio
n array 0)))
         arrays
)))


;(defun store-raster-in-data-base (list)
;  (with-database (db *image-test-pathname*)
;      (with-transaction ()
;       (dolist (array list)
;         (make-snapshot
;           :name (princ-to-string (gensym "piccy"))
;           :picture array)))))


(defun top-of-all (array rows columns)
  (let ((result nil) (vector (si:array-indirect-to array)))
    (let ((help (translate-vector-to-lists vector rows columns)))
      (loop for row-array in help
            do
        (push (translate-array-to-list row-array) result)))
    result))
```

```
(defun translate-vector-to-lists (vector  rows columns)
  (let ((result nil) (help (- 0 rows)))
    (loop for j from 1 to columns
          do
      (setq help (+ help rows))
        (push

            (subseq vector help (+ help rows )) result))

    result))


(defun translate-array-to-list (array)
  (let ((result nil))
    (loop for i from 0 to (- (length array) 1)
          do
      (push (aref array i) result))
    (reverse result)))


(defun find-image ()
  (with-database (db *image-test-pathname*)
      (with-transaction ()
  (let ((snapshots nil))
    (for-each ((s snapshot))
      (push s snapshots)) snapshots))))
;         (block IMAGE
;           (cond ((string-equal (snapshot-name s) name) (return-from IMAGE s)))
;           nil)))))
;




;(defun show-image-database ()
;  (let ((snapshots nil))
;    ;; Read all the snapshots into Lisp objects.
;    (with-database (db *image-test-pathname*)
;      (with-transaction ()
;       (setq snapshots nil)
;       (for-each ((s snapshot))
;         (push s snapshots)
;         (snapshot-name s)
;         (snapshot-picture s))))
;    ;; Display them on the window.
;    (dolist (s snapshots)
;      (format t "~%~A~%" (snapshot-name s))
;      (graphics:with-room-for-graphics ()
;       (graphics:draw-image (snapshot-picture s) 0 0)))))


;*****************************************************************************************
;;;; ;TRY  to graph image in Statice Browser




(defun  make-*graphic-window-for-image* () ;;;;(part-description-STRING)


  ;;;;;(let ((temp (string-append "PART DECRIPTION:" "  " part-description-STRING)))
(setq *graphic-window-for-image* (tv:make-window 'dw:dynamic-window
                                  :edges '(100 100 800 700) ;;;;;The actual superior '(64 43 1131 783)  ;
;;;;;;'(170 90 700 700)
                                                                ;;;;(multiple-value-list (send  *clear-dis
play* :inside-edges)
                                  :more-p t
                                  :margin-components
                                  '((dw:margin-borders :thickness 5)
                                    (dw:margin-white-borders :thickness 2)
                                    (dw:margin-ragged-borders :thickness 2
                                                              :horizontal-too t)
                                    (dw:margin-scroll-bar :elevator-thickness 10
                                                          :margin :left)
```

```
                                        (dw:margin-scroll-bar :elevator-thickness 10
                                                              :margin :bottom)
                                        (dw:margin-whitespace :thickness 6
                                                              :margin :left)
                                        (dw:margin-label :string
                                                   "IMAGE OF THE PART IN STATICE DATA BASE";;;; ,temp
                                                  :margin :top))
)))


(defvar *graphic-window-for-image*  (make-*graphic-window-for-image*))



(defflavor triangle
        ((name nil))
        ()
  :readable-instance-variables
  :writable-instance-variables
  :initable-instance-variables)


(defmethod  (sys:print-self   triangle)   (stream  &rest  ignore)
  (print-node  self  stream))

(defmethod  (print-node   triangle) (stream)
  (format  stream     name))

(defmethod (display triangle) ( stream &rest arguments)
  (apply #'graphics:draw-circle 600 15 10
         :stream stream
         :filled t ;;nil
         :alu :draw arguments))


(defun present-triangle (triangle stream &rest arguments)
  (graphics:with-output-as-graphics-presentation
    (stream :object triangle
            :type 'triangle)
    (apply #'display triangle stream arguments)))


(define-presentation-type pop-up-image ())
   :printer ((object stream)
;             (with-database (db *image-test-pathname*)
;       (with-transaction ()
;             (let ((*graphic-window-for-image* (make-*graphic-window-for-image* ;(snapshot-name
;                                                                                ;  (the-actual-entity-sna
pshot object)))))
;                                                   "IMAGE OF THE PART IN STATICE DATA BASE")))
             (send *graphic-window-for-image* :clear-history)
             (send *graphic-window-for-image* :expose)
             (graphics:with-room-for-graphics (*graphic-window-for-image*)
               (graphics:draw-image object  0 0 :stream *graphic-window-for-image*))
             (let ((help (make-instance 'triangle :name "Return")))
               (display help *graphic-window-for-image*)
               (present-triangle help *graphic-window-for-image*)
               (cond ((accept 'triangle :stream *graphic-window-for-image* :prompt "Click on the circle t
o EXIT (or Return if default)")
                      (send *graphic-window-for-image* :deactivate)))
             )))


(setf (get 'snapshot-picture 'statice::browser-presentation-type)  'pop-up-image)
```

APPENDIX 6

THE STATICE DATABASES DEFINITION

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER; Base: 10 -*-


;;; This program is designed to read a simple fixed format record tape
;;; containing data about additional features on Austin Rover Car
;;; Models. The tape seems to consist of several files. Each file is
;;; prefixed by a 'header file' and is followed by an 'end of file' file
;;; so: |header-file||actual data file||end of file file||header
;;; file||actual data file||end of file file||......
;;;



;;; Read the records off the tape as strings and outputs them to tape.

(defun read-tape-to-file (filename file-number)
  (with-open-file (file-stream filename :direction :output)

    (read-tape file-number file-stream)))




;;; This function reads the file that Takis was interested in from the
;;; sample tape. It prints the records in that file as string objects to
;;; the to-stream supplied as the optional argument.

(defun read-tape (&optional (file-number 2) to-stream)
  (with-open-stream (tape-stream (tape:make-stream
                                  :direction :input ))

    (send tape-stream :rewind)
    ;; Initially Takis was interested in the second data file on the
    ;; tape so that meant skipping four physical files seperated by EOF
    ;; markers.
    (send tape-stream :skip-file  file-number);; 1))

    (loop
       as record = (read-record tape-stream)
          while record do
      (when to-stream
        (print record to-stream)))))




;;; In the file that Takis was interested in, it appeared at first sight
;;; to be one long stream of ascii characters with no easy record
;;; boundary identifiers. After a while we spotted that most records
;;; begin with a four character record length number which is the number
;;; of characters in the record inclusive of the four. This was fine but
;;; there seemed to occasionally be a load of garbage in the middle of
;;; the tape like ^^^^^^^^^^^^^^^^^^^^^^^^. The simple solution to that
;;; was to read past any non digit characters before trying to read a
;;; record length field.

(defun read-record (stream)
  (loop
        as char = (send stream :tyi)
        when (null char) do (return-from read-record nil)
        until (digit-char-p (ascii-to-char char))
        finally
          (send stream :untyi char))
  (let* ((record-length  152)
         (record (make-array  record-length)))
    (send  stream :string-in t record)
      (ascii-to-string record )
  )
)


;(loop as char = (send stream :tyi)
```

```
;       when (null char) do (return-from read-record nil)
;       until (digit-char-p (ascii-to-char char))
;       finally
;         (send stream :untyi char))
;(defun read-record (stream)
;  ;; Skip past any non digit characters.
;  (loop as char = (send stream :tyi)
;       when (null char) do (return-from read-record nil)
;       until (digit-char-p (ascii-to-char char))
;       finally
;         (send stream :untyi char))
;  (let ((record-length-string (make-array 4 :element-type 'integer)))
;    (send stream :string-in t record-length-string)
;    (let* ((record-length (read-from-string (ascii-to-string record-length-string)))
;           (record (make-array (- record-length 4) :element-type 'integer)))
;      (send stream :string-in t record)
;      (ascii-to-string record))))

;;; Trivial format specification for structure of a record. Indicates
;;; sequential field lengths in terms of number of characters.

(defun takis-old (number)
  (with-open-stream (tape-stream (tape:make-stream
                                  :direction :input))
    (send tape-stream :rewind)
    (send tape-stream :skip-file  number )
;   (let ((array-var (make-array 154 )))
;    (loop for i from 1 to 150
;         as char = (send tape-stream :tyi)
;         do
;      (cond ((null char) (return nil))
;            (t
;         (send tape-stream :string-in t array-var))))
;         (ascii-to-string array-var)
;   )
          ;finally
            ;(send stream :untyi char)

    (loop
        as ascii-code = (send tape-stream :tyi)
          do
      (cond  ((null ascii-code)
                (send tape-stream  :clear-eof))
             (t (let ((char (ascii-to-char ascii-code)))
                  (princ char)))))
    )
)



;XW  A10 M              N     M    E      A      ABA R8/3P/87                P XW0A1

;

;                                MND  XW  A20 M              N     M    E        H    ABA R8/3P/87

; P XW9H1

;                                                                    MNE  XW  A25 M              N     M
    E
;      J   ABA R8/5D/88                A XW1A1

;
  MNH  X
;W  A30 M              N      E    Y      D     A ABR8/3P/87    R8/1P/88    PHXW9H1   XW9H1

;

;                              MNT  XW  A35 M              N        K     L      M   BA R8/8P/88    R8/2
P/89
;PHXW9H1   XW9H1

;                                                            MNX  XW  A40 M              N
 K
;M  D     A A R8/8P/88    R8/2P/89    PHXW9H1   XW9H1
```

```
;
 MNY  XW
;  A50 M                W    M    E      A      A A R8/3P/87                P XW0A1

;

;                              MWD  XW  A60 M               W    M    E        H    ABA R8/3P/87
       P
; XW9A1

;                                                              MWE  XW  A65 M               W    M
  E
;    J   ABA R8/5D/88                A XW1A1

;
MWH  XW
; A70 M                W     E    Y      D     A ABR8/3P/87    R8/1P/88    PHXW9A1   XW9A5

;

;                            MWT  XW  A75 M               W       K     L      M   BA R8/8P/88    R8/2P/
89    PH
;XW0C1   XW0C1                                                    [Abort]
:Back to Lisp Top Level in Dynamic Lisp Listener 1.
```

```
;;;; -*- Mode: lisp;  Syntax: Common-lisp; Package: austin-rover -*-


;(make-package  'Austin-Rover
;               :use '(COMMON-LISP-USER  STATICE-USER Symbolics-common-lisp statice lisp))


(defun the-TOP-function-to-run-the-code-of-this-file (ls  pathname); ;;;*Rover-APN-Catalogue*
                                                        ;;;;;;;;ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-p
athname*
  ;;;;;; THE variable *Rover-APN-Catalogue*  is set to a list of strings ie.
  ;;;;;;;;;*Rover-APN-Catalogue*  ==  (append *APN-CATALOGUE-flat-data-SEATS*  *APN-CATALOGUE-flat-data-M
IRRORS*)
  ;;;(make-the-ROVERs-APN-database-for-seats-and-mirrors pathname)  ;;;"stat-FS:>takis>the-ROVERs-APN-CAT
ALOGUE")
  (THE-CREATION-OF-THE-ROVER-STATICE-data-base ls pathname))


(define-schema      ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS     (;;;;;;;; VPG , Sub-Vpg, Sub-Sub-Vp
g,
                                          Sub-Sub-Sub-Vpg  FAMILY-within-a-SSS-Vpg
                                          Primary-Description-ID
                                          Secondary-Description-ID
                                          THE-ROVER-CARS----actually-Austin-Rover))

(defvar   *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*  #p"stat-FS:>takis>ROVER-APN")


(defun make-ROVERs-APN-VPG-database-for-seats-and-mirrors (PATHNAME)  ; SYMBOL
                        ;;ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*
  (make-database  PATHNAME   'ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS))




(define-entity-type      Sub-Sub-Sub-Vpg ()
  ((VPG-code   string :inverse  the-entity-type-of-the-VPG-CODE :inverse-index t)
   (HAS-Families-within     (set-of FAMILY-within-a-SSS-Vpg) :inverse t)
   ;;;;(belongs ........)
   (FEATURE-GROUPS-LINKED-TO-THE-Sub-Sub-Sub-Vpg  (vector fixnum))))

(define-entity-type      FAMILY-within-a-SSS-Vpg  ()
  ((FAMILY-CODE string :inverse the-entity-type-of-the-family-code :inverse-index t)
   (Has-Primary-Descriptions  (set-of   Primary-Description-ID) :inverse t)
   (belongs-to-SSS-VPG   Sub-Sub-Sub-Vpg)
   (MUST-features (vector fixnum))
   (MAYBE-features  (vector fixnum))))


(define-entity-type   Primary-Description-ID ()
  ((PRIMARY-DESCRIPTION-OF-THE-PART   string :inverse The-entity-type-of-the-PRIMARY-DESCRIPTION :inverse
-index t)
   (Has-Secondary-Descriptions   (set-of    Secondary-Description-ID) :inverse t)
   (Belongs-to-Family    FAMILY-within-a-SSS-Vpg)
   ;;;;;;;(The-matrix-MAPPING-of-PARTS-with-FEATURE-GROUPS   matrix)
   (MUST-FEATURES-for-ALL-Rover-vehicles  (vector fixnum))
   (MAYBE-FEATURES-for-ALL-Rover-vehicles  (vector fixnum))))




(define-entity-type    Secondary-Description-ID ()
  ((FAMILY string)
   (RECNO  string)
   (MAIN-GROUP  string)
   (MAIN-GROUP-DESCRIPTION  string)
   (SECONDARY-LETTER string)
   (PRIMARY-LETTER string)
   (VPG-CODE string :inverse   entity-of-the-Secondary-Description-ID-with-this-VPG :inverse-index t)
```

```
   (DE-SEGMENT-KEY string)
   (SECONDARY-DESCRIPTION string)
   (PRIMARY-DESCRIPTION string)
   (EMISSIONS-INDICATOR string)
   (REGULATIONS-INDICATOR string)
   (SAFETY-INDICATOR string)
   (COLOURED-INDICATOR  string)
   (UNIT-OF-MEASURE string)
   (OBSOLETE-INDICATOR string)
   (DATE-CREATED string)
   (APPROVED-FOR-INDICATOR string)
   (COLOUR-CHART-KEY string)
   (SEGMENT-SEQUENCE-NUMBER string)
   (SHORT-DESCRIPTION string)
   (MP17-INDICATOR string)
   (INITIAL-STOCK-ORDER-INDICATOR string)
   (ZONE-CODE string)
   (HONDA-FUNC-AREA-CODE string)
   (PANEL-PART-INDICATOR string)
   (SERVICE-INDICATOR string)
   (DESIGN-RESPONSIBILITY  string)
   (Has-Must-Features (vector fixnum))
   (Has-Maybes-Features (vector fixnum))
   (NEXT-ASSEMBLY-RELATIONSHIPS (vector fixnum))
   (Flag-for-information-of-FEATURES-gathered-from-the-tape string)
   (In-English-the-detailed-information-of-HOW-the-MUST-and-MAYBES-FEATURES-derived  string)
   (NEGATIVE-ADDITIONAL-FEATURES  (vector fixnum))  ;;;;;; WE give the same  attribute with the OLD-PROGR
AM  manually entered
                                                        ;;;;;;   FLAVORS
   (Belongs-to-Primary-Description   Primary-Description-ID)
   (REFERENCE-TO-THE-ROVER-VEHICLES   THE-ROVER-CARS----actually-Austin-Rover :inverse t)
   (PART-DESCRIPTION string  :unique t :no-nulls t :cached t :inverse The-PART-DESCRIPTIONs-entity-type :
inverse-index t )
;;;;; :unique t
))

;;;;;;;;WE DEFINE THE   below   ENTITY-TYPE    because of the INABILITY of  STATICE to handle  matrices,
especially
;;;;;;;;     adjustable-matrices  AS   LISP  does.


(define-entity-type       THE-ROVER-CARS----actually-Austin-Rover ()
  ((MINI-musts   (vector fixnum))
   (METRO-musts  (vector fixnum))
   (R6-musts     (vector fixnum))
   (MAESTRO-musts   (vector fixnum))
   (MONTEGO-musts   (vector fixnum))
   (old-ROVER-200--XH-musts    (vector fixnum))
   (New-ROVER-200-or-ROVER-400--XW-musts   (vector fixnum))
   (ROVER-800-musts   (vector  fixnum))
   (R17-musts       (vector fixnum))
   (NEW-ROVER-1-musts (vector fixnum))
   (NEW-ROVER-2-musts (vector fixnum))
   (NEW-ROVER-3-musts (vector fixnum))
   (NEW-ROVER-4-musts (vector fixnum))
   (NEW-ROVER-5-musts (vector fixnum))
   (NEW-ROVER-6-musts (vector fixnum))
   (NEW-ROVER-7-musts (vector fixnum))
   (NEW-ROVER-8-musts (vector fixnum))
   (NEW-ROVER-9-musts (vector fixnum))
   (NEW-ROVER-10-musts (vector fixnum))
   (MINI-maybes   (vector fixnum))
   (METRO-maybes  (vector fixnum))
   (R6-maybes     (vector fixnum))
   (MAESTRO-maybes   (vector fixnum))
   (MONTEGO-maybes   (vector fixnum))
   (old-ROVER-200--XH-maybes    (vector fixnum))
   (New-ROVER-200-or-ROVER-400--XW-maybes   (vector fixnum))
   (ROVER-800-maybes   (vector fixnum))
   (R17-maybes       (vector fixnum))
    (NEW-ROVER-1-maybes (vector fixnum))
   (NEW-ROVER-2-maybes (vector fixnum))
   (NEW-ROVER-3-maybes (vector fixnum))
   (NEW-ROVER-4-maybes (vector fixnum))
   (NEW-ROVER-5-maybes (vector fixnum))
   (NEW-ROVER-6-maybes (vector fixnum))
```

```
    (NEW-ROVER-7-maybes (vector fixnum))
    (NEW-ROVER-8-maybes (vector fixnum))
    (NEW-ROVER-9-maybes (vector fixnum))
    (NEW-ROVER-10-maybes (vector fixnum))))


(defun check-if-like-flavors-to-collect-the-part-description-which-matches-something-in-STRING----provide
d-CACHED-is-NOT-in
          (a-string)  ;;;;ie.  "seat"
  (let ((result nil))
  (with-database  (db  *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*)
                  (with-transaction ()
                    (for-each ((ADD Secondary-Description-ID )
                               (:where (string-search  a-string  (Secondary-Description-ID-PART-DESCRIPTI
ON  ADD)   )))
                      (push ADD result))))
result))



(defun get-rid-of-vectors-in-MUSTS-and-MAYBES-within-Secondary  (ls)  ;;;;;;;*Secondary-Description-IDs*
  (with-database  (db  *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*)
                  (with-transaction ()
  (loop for entity-type in ls
        do
    (setf (Secondary-Description-ID-Has-Must-Features entity-type) nil)
    (setf (Secondary-Description-ID-Has-Maybes-Features entity-type) nil)))))



;;;;;;;;;list  ie.  the  argument for the  BELOW function
; ((:FAMILY "HAA") (:RECNO "0154") (:MAIN-GROUP "H") (:MAIN-GROUP-DESCRIPTION "SEATING
        ")
;  (:SECONDARY-LETTER "A") (:PRIMARY-LETTER "A") (:VPG-CODE "1109AA") (:DE-SEGMENT-KEY "001")
;  (:SECONDARY-DESCRIPTION "MAN L/SUPP      ") (:PRIMARY-DESCRIPTION "INSTR DRG              ") (:EMISSIONS
-INDICATOR " ")
;  (:REGULATIONS-INDICATOR " ") (:SAFETY-INDICATOR " ") (:COLOURED-INDICATOR "Y") (:UNIT-OF-MEASURE "PIEC
E")
;  (:OBSOLETE-INDICATOR "C") (:DATE-CREATED "870212") (:APPROVED-FOR-INDICATOR "P") (:COLOUR-CHART-KEY "-
---")
;  (:SEGMENT-SEQUENCE-NUMBER "0050") (:SHORT-DESCRIPTION "INSTR DRG MAN L/SUPP  ") (:MP17-INDICATOR "N")
;  (:INITIAL-STOCK-ORDER-INDICATOR " ") (:ZONE-CODE "  ") (:HONDA-FUNC-AREA-CODE "8101Z") (:PANEL-PART-IN
DICATOR " ")
;  (:SERVICE-INDICATOR " ") (:DESIGN-RESPONSIBILITY "ARTHSA"))


(defun  make-APN-CATALOGUE-entry (list   data-base-pathname)
  (with-database  (db   data-base-pathname)   ;;;;;;;;; *ROVER-TRY-pathname*)
                  (with-transaction ()
                                  (make-Secondary-Description-ID
                                    :FAMILY (put-the-right-INDICATOR ':FAMILY list)
                                    :RECNO (put-the-right-INDICATOR ':RECNO list)
                                    :MAIN-GROUP
                                          (put-the-right-INDICATOR ':MAIN-GROUP list)
                                    :MAIN-GROUP-DESCRIPTION
                                          (put-the-right-INDICATOR ':MAIN-GROUP-DESCRIPTION list)
                                    :SECONDARY-LETTER
                                          (put-the-right-INDICATOR ':SECONDARY-LETTER list)
                                    :PRIMARY-LETTER (put-the-right-INDICATOR ':PRIMARY-LETTER list)
                                    :VPG-CODE (put-the-right-INDICATOR ':VPG-CODE list)
                                    :DE-SEGMENT-KEY (put-the-right-INDICATOR ':DE-SEGMENT-KEY list)
                                    :SECONDARY-DESCRIPTION   (put-the-right-INDICATOR ':SECONDARY-DESCR
IPTION list)
                                    :PRIMARY-DESCRIPTION (put-the-right-INDICATOR ':PRIMARY-DESCRIPTION
 list)
                                    :EMISSIONS-INDICATOR (put-the-right-INDICATOR ':EMISSIONS-INDICATOR
 list)
                                    :REGULATIONS-INDICATOR  (put-the-right-INDICATOR ':REGULATIONS-INDI
CATOR list)
                                    :SAFETY-INDICATOR (put-the-right-INDICATOR ':SAFETY-INDICATOR list)
                                    :COLOURED-INDICATOR (put-the-right-INDICATOR ':COLOURED-INDICATOR l
ist)
                                    :UNIT-OF-MEASURE   (put-the-right-INDICATOR ':UNIT-OF-MEASURE list)
                                    :OBSOLETE-INDICATOR   (put-the-right-INDICATOR ':OBSOLETE-INDICATOR
 list)
                                    :DATE-CREATED (put-the-right-INDICATOR ':DATE-CREATED list)
```

```
                                        :APPROVED-FOR-INDICATOR  (put-the-right-INDICATOR ':APPROVED-FOR-IN
DICATOR list)
                                        :COLOUR-CHART-KEY  (put-the-right-INDICATOR ':COLOUR-CHART-KEY list
)
                                        :SEGMENT-SEQUENCE-NUMBER (put-the-right-INDICATOR ':SEGMENT-SEQUENC
E-NUMBER list)
                                        :SHORT-DESCRIPTION (put-the-right-INDICATOR ':SHORT-DESCRIPTION lis
t)
                                        :MP17-INDICATOR  (put-the-right-INDICATOR ':MP17-INDICATOR list)
                                        :INITIAL-STOCK-ORDER-INDICATOR
                                                  (put-the-right-INDICATOR ':INITIAL-STOCK-ORDER-INDICATOR
 list)
                                        :ZONE-CODE  (put-the-right-INDICATOR ':ZONE-CODE list)
                                        :HONDA-FUNC-AREA-CODE (put-the-right-INDICATOR ':HONDA-FUNC-AREA-CO
DE list)
                                        :PANEL-PART-INDICATOR (put-the-right-INDICATOR ':PANEL-PART-INDICAT
OR list)
                                        :SERVICE-INDICATOR (put-the-right-INDICATOR ':SERVICE-INDICATOR lis
t)
                                        :DESIGN-RESPONSIBILITY   (put-the-right-INDICATOR ':DESIGN-RESPONSI
BILITY list)
                                        :PART-DESCRIPTION (put-the-right-INDICATOR ':PART-DESCRIPTION list)

))))



(defun put-the-right-INDICATOR    (INDICATOR ls)
  (second (assoc INDICATOR ls)))

;*****************************************************************************************************
****************
;*****************************************************************************************************
****************
;*****************************************************************************************************
****************

;
;
;
;
;Command: (setq *help* (parse-everything *APN-CATALOGUE-flat-data-MIRRORS* *APN-CATALOGUE-flat-data-SEATS
*))
;
;;;;;;; The PARSE-EVERYTHING function GROUPS-together  data from the *ROVER-APN-CATALOGUE* variable which
 have the same
;;;;;;;  :VPG-CODE (ie. SOME (subseq APN-CATALOGUE-string 42 48) must be equal  .... see function BREAK-.
..) and  PARSES the STRINGS TO lists of keywords and values
;;;;;;; HENCE in OUR case , because we did append two variables which have the same  *help* has two eleme
nts lists  which EACH of THEM has as elements LISTS of lists of KEYWORDS and VALUES
;Command: *help*
;((((:PART-DESCRIPTION "CVR-EXPND-HEADRSTNT-FRNON ELASTICATED") (:FAMILY "HAM") (:RECNO "0154") (:MAIN-GR
OUP "H")
;    (:MAIN-GROUP-DESCRIPTION "SEATING                                ") (:SECONDARY-LETTER "A") (:PRIMARY-LET
TER "M")
;    (:VPG-CODE "1109AA") (:DE-SEGMENT-KEY "004") (:SECONDARY-DESCRIPTION "NON ELASTICATED")
;    (:PRIMARY-DESCRIPTION "CVR-EXPND-HEADRSTNT-FR") (:EMISSIONS-INDICATOR " ") (:REGULATIONS-INDICATOR "
")
;    (:SAFETY-INDICATOR " ") (:COLOURED-INDICATOR "N") (:UNIT-OF-MEASURE "PIECE") (:OBSOLETE-INDICATOR "C"
)
;    (:DATE-CREATED "890203") (:APPROVED-FOR-INDICATOR "P") (:COLOUR-CHART-KEY "    ") (:SEGMENT-SEQUENCE-N
UMBER "1206")
;    (:SHORT-DESCRIPTION "CVR-EXPND-H/R-FR-N/E  ") (:MP17-INDICATOR "N") (:INITIAL-STOCK-ORDER-INDICATOR "
 ") (:ZONE-CODE "  ")
;    (:HONDA-FUNC-AREA-CODE "00000") (:PANEL-PART-INDICATOR " ") (:SERVICE-INDICATOR " ") (:DESIGN-RESPONS
IBILITY "ARTHSA"))
;  ((:PART-DESCRIPTION "CVR-EXPND-HEADRSTNT-FRELASTICATED    ") (:FAMILY "HAM") (:RECNO "0154") (:MAIN-GR
OUP "H")
;    (:MAIN-GROUP-DESCRIPTION "SEATING                                ") (:SECONDARY-LETTER "A") (:PRIMARY-LET
TER "M")
;    (:VPG-CODE "1109AA") (:DE-SEGMENT-KEY "003") (:SECONDARY-DESCRIPTION "ELASTICATED    ")
;    (:PRIMARY-DESCRIPTION "CVR-EXPND-HEADRSTNT-FR") (:EMISSIONS-INDICATOR " ") (:REGULATIONS-INDICATOR "
")
;    (:SAFETY-INDICATOR " ") (:COLOURED-INDICATOR "N") (:UNIT-OF-MEASURE "PIECE") (:OBSOLETE-INDICATOR "C"
```

```
)
;    (:DATE-CREATED "890203") (:APPROVED-FOR-INDICATOR "P") (:COLOUR-CHART-KEY "    ") (:SEGMENT-SEQUENCE-N
UMBER "1204")
;    (:SHORT-DESCRIPTION "CVR-EXPND-H/R-FR-ELAST") (:MP17-INDICATOR "N") (:INITIAL-STOCK-ORDER-INDICATOR "
 ") (:ZONE-CODE "  ")
;    (:HONDA-FUNC-AREA-CODE "00000") (:PANEL-PART-INDICATOR " ") (:SERVICE-INDICATOR " ") (:DESIGN-RESPONS
IBILITY "ARTHSA"))
;  ((:PART-DESCRIPTION "COVER-EXPENDABLE-ST/F NON ELESTICATED") (:FAMILY "HAM") (:RECNO "0154") (:MAIN-GR
OUP "H")
;    (:MAIN-GROUP-DESCRIPTION "SEATING                                ") (:SECONDARY-LETTER "A") (:PRIMARY-LET
TER "M")
;    (:VPG-CODE "1109AA") (:DE-SEGMENT-KEY "002") (:SECONDARY-DESCRIPTION "NON ELESTICATED")
;    (:PRIMARY-DESCRIPTION "COVER-EXPENDABLE-ST/F ") (:EMISSIONS-INDICATOR " ") (:REGULATIONS-INDICATOR "
")
;    (:SAFETY-INDICATOR " ") (:COLOURED-INDICATOR "N") (:UNIT-OF-MEASURE "PIECE") (:OBSOLETE-INDICATOR "C"
)
;    (:DATE-CREATED "890203") (:APPROVED-FOR-INDICATOR "P") (:COLOUR-CHART-KEY "    ") (:SEGMENT-SEQUENCE-N
UMBER "1202")
;    (:SHORT-DESCRIPTION "CVR-EXPND-ST/F-N/ELAST") (:MP17-INDICATOR "N") (:INITIAL-STOCK-ORDER-INDICATOR "
 ") (:ZONE-CODE "  ")
;    (:HONDA-FUNC-AREA-CODE "00000") (:PANEL-PART-INDICATOR " ") (:SERVICE-INDICATOR " ") (:DESIGN-RESPONS
IBILITY "ARTHSA"))
;  ((:PART-DESCRIPTION "COVER-EXPENDABLE-ST/F ELASTICATED    ") (:FAMILY "HAM") (:RECNO "0154") (:MAIN-GR
OUP "H")
;    (:MAIN-GROUP-DESCRIPTION "SEATING                                ") (:SECONDARY-LETTER "A") (:PRIMARY-LET
TER "M")
;    (:VPG-CODE "1109AA") (:DE-SEGMENT-KEY "001") (:SECONDARY-DESCRIPTION "ELASTICATED    ")
;    (:PRIMARY-DESCRIPTION "COVER-EXPENDABLE-ST/F ") (:EMISSIONS-INDICATOR " ") (:REGULATIONS-INDICATOR "R
")
;    (:SAFETY-INDICATOR " ") (:COLOURED-INDICATOR "Y") (:UNIT-OF-MEASURE "PIECE") (:OBSOLETE-INDICATOR "C"
)
;    (:DATE-CREATED "       ") (:APPROVED-FOR-INDICATOR "P") (:COLOUR-CHART-KEY "    ") (:SEGMENT-SEQUENCE-N
UMBER "1200")
;    (:SHORT-DESCRIPTION "COVER-EXPENDABLE-ST/F ") (:MP17-INDICATOR "N") (:INITIAL-STOCK-ORDER-INDICATOR "
 ") (:ZONE-CODE "  ")
;    (:HONDA-FUNC-AREA-CODE "     ") (:PANEL-PART-INDICATOR " ") (:SERVICE-INDICATOR " ") (:DESIGN-RESPONS
IBILITY "ARTHSA"))
;  ((:PART-DESCRIPTION "SQUAB-COMP-SEAT FRONT MANUAL         ") (:FAMILY "HAJ") (:RECNO "0154") (:MAIN-GR
OUP "H")
;    (:MAIN-GROUP-DESCRIPTION "SEATING                                ") (:SECONDARY-LETTER "A") (:PRIMARY-LET
TER "J")
;    (:VPG-CODE "1109AA") (:DE-SEGMENT-KEY "004") (:SECONDARY-DESCRIPTION "MANUAL          ")
;    (:PRIMARY-DESCRIPTION "SQUAB-COMP-SEAT FRONT ") (:EMISSIONS-INDICATOR " ") (:REGULATIONS-INDICATOR "
")
;    (:SAFETY-INDICATOR " ") (:COLOURED-INDICATOR "Y") (:UNIT-OF-MEASURE "PIECE") (:OBSOLETE-INDICATOR "C"
)
;    (:DATE-CREATED "890315") (:APPROVED-FOR-INDICATOR "P") (:COLOUR-CHART-KEY "S/D") (:SEGMENT-SEQUENCE-N
UMBER "0904")
;[Abort]
;Back to Lisp Top Level in Dynamic Lisp Listener 1.
;
;Command:
;
;
;
;
;
;;;;   THE PROGRAM STARTSW ITH THE FUNCTION  :THE-CREATION-OF-THE-ROVER-STATICE-data-base
;;;;; WE TRY TO CREATE A tree-database  FOR EACH Element within  "help" . ALL THOSE trees belong to the s
ame data-base
;;;;RUN the:
;
;;(THE-CREATION-OF-THE-ROVER-STATICE-data-base  (append *APN-CATALOGUE-flat-data-SEATS*  *APN-CATALOGUE-f
lat-data-MIRRORS*)
;;                                                ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname
*)
;
```

```
;(defun  THE-CREATION-OF-THE-ROVER-STATICE-data-base (ls  pathname) ;;;*Rover-APN-Catalogue*
;                                                   ;;;;;;;;ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-
pathname*
;
;  (make-ROVER-APN-try-database   PATHNAME)
;  (let ((help (parse-everything ls)))
;    (loop for PARSED-Sub-Sub-Sub-VPG-data  in help
;          do
;       (CREATE-THE-SUB-SUB-SUB-VPG-STATICE-DATA-BASE  PARSED-Sub-Sub-Sub-VPG-data   PATHNAME)))) ;; *ROVE
R-APN-CATALOGUE-pathname*
;
;
;(loop for PARSED-Sub-Sub-Sub-VPG-data  in *help*
;          do
;       (CREATE-THE-SUB-SUB-SUB-VPG-STATICE-DATA-BASE  PARSED-Sub-Sub-Sub-VPG-data   *ROVERs-APN-VPG-CATAL
OGUE-only-SEATS-and-MIRRORS-pathname*))
;
;
;(defun parse-everything (ls)        ;;;;;;*ROVER-APN-CATALOGUE*
;  (let ((result nil) (help (BREAK-the-*ROVER-APN-CATALOGUE*-in-SUB-SUB-SUB-VPGs ls)))
;    (loop for Sub-Sub-Sub-VPG-flat-data   in  help
;          do
;       (push
;        (CREATE-THE-family-and-part-description   (the-creation-of-the-Sub-Sub-Sub-VPG-data-base  Sub-Sub
-Sub-VPG-flat-data))
;        result))
;result))
;
;
;(defun  BREAK-the-*ROVER-APN-CATALOGUE*-in-SUB-SUB-SUB-VPGs  (ls)   ;;;;;;*ROVER-APN-CATALOGUE*
;  (let ((result nil)  (final-result nil))
;    (loop for  APN-CATALOGUE-string   in  ls
;          do
;       (let ((temp (subseq APN-CATALOGUE-string 42 48)))
;       (cond ((not (member temp result :test #'string-equal))
;                (push temp result)))))
;    (loop for Sub-Sub-Sub-VPG  in result
;          do
;       (push (the-APN-DATA-OF-A-SPECIFIC-VPG  Sub-Sub-Sub-VPG ls)
;            final-result))
;final-result))
;
;
;(defun the-APN-DATA-OF-A-SPECIFIC-VPG (a-SSS-VPG-string  ls)
;  (let ((result  nil))
;    (loop for i in  ls
;          do
;       (cond
;        ((string-equal a-SSS-VPG-string (subseq i 42 48))
;             (push i result))))
;result))
;
;
;
;;(BREAK-the-*ROVER-APN-CATALOGUE*-in-SUB-SUB-SUB-VPGs     *ROVER-APN-CATALOGUE*)
;;(("0154ABODY STRUCTURES                    AZ1102AA002REAR END        SURFACE CO-ORDINATES     NPIECEC88
1021P   1020SURFACE CO-ORD
;;-RR END N   0059Z  ARBDAAE"
;;  "0154ABODY STRUCTURES                    AZ1102AA001BODY            SURFACE CO-ORDINATES     NPIECEC87
0629P   1000SURFACE CO-ORD
;;INATES  N   0051ZY ARBDAAE"
;;  "0154ABODY STRUCTURES                    AX1102AA002BODY SIDE-REAR EXTN                      NPIECEC88
0603P   0620EXTN-BODY SIDE
;;-REAR   N   00000Y ARBDAAE"
;;  "0154ABOD    ..............................................................................
..
;
;
;;;;;;;;;;;;          ******************************************************************************
;
;
;(defun CREATE-THE-family-and-part-description  (ls)    ;;;;;ie. (the-creation-of-the-Sub-Sub-Sub-VPG-dat
a-base  *APN-CATALOGUE-flat-data*)
;  (let ((result nil))              ;;;;;   FOR    *APN-CATALOGUE-flat-data*   look at the END.
;    (loop for  PARSED-DATA  in ls
;          do
```

```
;       (let* ((MAIN-GROUP  (assoc ':main-group PARSED-DATA))
;              (SECONDARY-LETTER (assoc ':secondary-letter PARSED-DATA))
;              (PRIMARY-LETTER  (assoc  ':primary-letter  PARSED-DATA))
;              (family  (list
;                          ':FAMILY
;                          (string-append
;                            (second MAIN-GROUP)
;                            (second SECONDARY-LETTER)
;                            (second PRIMARY-LETTER))))
;              (part-description (list
;                                  ':PART-DESCRIPTION
;                                  (string-append
;                                    (second (assoc ':PRIMARY-DESCRIPTION PARSED-DATA))
;                                    (second (assoc ':SECONDARY-DESCRIPTION PARSED-DATA))))))
;;       (delete MAIN-GROUP PARSED-DATA :test #'equal)
;;       (delete SECONDARY-LETTER PARSED-DATA :test #'equal)
;;       (delete PRIMARY-LETTER PARSED-DATA :test #'equal)
;        (push family PARSED-DATA)
;        (push part-description PARSED-DATA)
;        (push PARSED-DATA result)))
;result))
;
;
;
;
;
;(defun the-creation-of-the-Sub-Sub-Sub-VPG-data-base   (ls) ;; It gets a Sub..Sub.. VPG   flat list of d
ata from the
;                                                        ;;;;; APN Catalogue.  In this case 1109AA, ie. *APN-
CATALOGUE-flat-data*
;  (let ((result nil))
;    (loop for flat-data  in  ls
;          do
;      (push (parse-record  flat-data *APN-CATALOGUE-format*) result))
;result))
;
;
;
;
;(defun parse-record (string format)
;  (loop as string-left = string then (substring string-left length)
;        for (field-name length) in format
;        collect (list field-name (substring string-left 0 length))))
;
;
;;;; Trivial format specification for structure of a record. Indicates
;;;; sequential field lengths in terms of number of characters.
;
;(defvar *APN-CATALOGUE-format*
;        '((:recno 4)
;          (:MAIN-GROUP 1)
;
;          (:MAIN-GROUP-DESCRIPTION 35)
;          (:SECONDARY-LETTER 1)
;          (:PRIMARY-LETTER 1)
;          (:VPG-CODE 6)
;          (:DE-SEGMENT-KEY 3)
;          (:SECONDARY-DESCRIPTION 15)
;          (:PRIMARY-DESCRIPTION 22)
;
;          (:EMISSIONS-INDICATOR 1)
;          (:REGULATIONS-INDICATOR 1)
;
;          (:SAFETY-INDICATOR 1)
;          (:COLOURED-INDICATOR 1)
;          (:UNIT-OF-MEASURE 5)
;          (:OBSOLETE-INDICATOR 1)
;          (:DATE-CREATED 6)
;          (:APPROVED-FOR-INDICATOR 1)
;          (:COLOUR-CHART-KEY 3)
;
;          (:SEGMENT-SEQUENCE-NUMBER   4)
;          (:SHORT-DESCRIPTION   22)
;          (:MP17-INDICATOR   1)
;          (:INITIAL-STOCK-ORDER-INDICATOR   1)
;          (:ZONE-CODE   2)
```

```
;          (:HONDA-FUNC-AREA-CODE    5)
;          (:PANEL-PART-INDICATOR    1)
;          (:SERVICE-INDICATOR    1)
;          (:DESIGN-RESPONSIBILITY    6)
;))
;
;
;;;;;;;;;(defvar *data*      (CREATE-THE-family  (the-creation-of-the-Sub-Sub-Sub-VPG-data-base  *APN-CATA
LOGUE-flat-data*)))
;
;
;;*****************************************************************************************************
*****************
;;*****************************************************************************************************
*****************
;
;(defun CREATE-THE-SUB-SUB-SUB-VPG-STATICE-DATA-BASE  (list-data   PATHNAME)    ;;;*data*  ,   *ROVER-try
-pathname*
; (THE-TOP-create-the-STATICE-DATA-BASE-FOR-THE-SPECIFIC-Sub-Sub-Sub-VPG list-data PATHNAME))
;
;
;
;
;(defun THE-TOP-create-the-STATICE-DATA-BASE-FOR-THE-SPECIFIC-Sub-Sub-Sub-VPG  (list-of-lists  data-base-
pathname)       ;;;;;*DATA*
;  (let* ((help   (create-the-STATICE-DATA-BASE-FOR-THE-SPECIFIC-Sub-Sub-Sub-VPG  list-of-lists data-base
-pathname))
;         (secondaries-ENTITIES-in-STATICE  (first  help))
;         (primaries-entities-in-STATICE (second help))
;         (VPG-sub-sub-sub-group-entity-in-STATICE (first (third help)))
;         (family-entities-in-STATICE  (fourth help)))
;    (make-the-links-PRIMARIES-SECONDARIES primaries-entities-in-STATICE secondaries-ENTITIES-in-STATICE
data-base-pathname)
;    (make-the-links-PRIMARIES-FAMILIES primaries-entities-in-STATICE family-entities-in-STATICE data-bas
e-pathname)
;    (make-the-links-FAMILIES-Sub-Sub-Sub-VPG family-entities-in-STATICE VPG-sub-sub-sub-group-entity-in-
STATICE data-base-pathname)))
;
;
;;Command:*data*
;;(((:FAMILY "HAA") (:RECNO "0154") (:MAIN-GROUP "H") (:MAIN-GROUP-DESCRIPTION "SEATING
            ")
;;  (:SECONDARY-LETTER "A") (:PRIMARY-LETTER "A") (:VPG-CODE "1109AA") (:DE-SEGMENT-KEY "001")
;;  (:SECONDARY-DESCRIPTION "MAN L/SUPP      ") (:PRIMARY-DESCRIPTION "INSTR DRG              ") (:EMISSION
S-INDICATOR " ")
;;  (:REGULATIONS-INDICATOR " ") (:SAFETY-INDICATOR " ") (:COLOURED-INDICATOR "Y") (:UNIT-OF-MEASURE "PIE
CE")
;;  (:OBSOLETE-INDICATOR "C") (:DATE-CREATED "870212") (:APPROVED-FOR-INDICATOR "P") (:COLOUR-CHART-KEY "
---")
;;  (:SEGMENT-SEQUENCE-NUMBER "0050") (:SHORT-DESCRIPTION "INSTR DRG MAN L/SUPP  ") (:MP17-INDICATOR "N")
;;  (:INITIAL-STOCK-ORDER-INDICATOR " ") (:ZONE-CODE "  ") (:HONDA-FUNC-AREA-CODE "8101Z") (:PANEL-PART-I
NDICATOR " ")
;;  (:SERVICE-INDICATOR " ") (:DESIGN-RESPONSIBILITY "ARTHSA"))
;; ((:FAMILY "HAA") (:RECNO "0154") (:MAIN-GROUP "H") (:MAIN-GROUP-DESCRIPTION "SEATING
            ")
;;  (:SECONDARY-LETTER "A") (:PRIMARY-LETTER "A") (:VPG-CODE "1109AA") (:DE-SEGMENT-KEY "002")
;;  (:SECONDARY-DESCRIPTION "MAN-NO L/SUPP  ") (:PRIMARY-DESCRIPTION "INSTR DRG              ") (:EMISSION
S-INDICATOR " ")
;;  (:REGULATIONS-INDICATOR " ") (:SAFETY-INDICATOR " ") (:COLOURED-INDICATOR "Y") (:UNIT-OF-MEASURE "PIE
CE")
;;  (:OBSOLETE-INDICATOR "C") (:DATE-CREATED "870212") (:APPROVED-FOR-INDICATOR "P") (:COLOUR-CHART-KEY "
---")
;;  (:SEGMENT-SEQUENCE-NUMBER "0100") (:SHORT-DESCRIPTION "INSTR DRG MAN-NO L/SUP") (:MP17-INDICATOR "N")
;;  (:INITIAL-STOCK-ORDER-INDICATOR " ") (:ZONE-CODE "  ") (:HONDA-FUNC-AREA-CODE "8101Z") (:PANEL-PART-I
NDICATOR " ")
;;  (:SERVICE-INDICATOR " ") (:DESIGN-RESPONSIBILITY "ARTHSA"))
;; ((:FAMIL
;
;;;;;;;  *data*   REPRESENT  the   1109AA  V.P.G.
;;Command: (create-the-STATICE-DATA-BASE-FOR-THE-SPECIFIC-Sub-Sub-Sub-VPG *data*)
;;(((#<Entity-Handle of SECONDARY-DESCRIPTION-ID 67/2 54244205> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 67/1 54244171>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 66/3 54243373> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 66/2 54243357>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 66/1 54243343> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 65/3 54242545>
```

```
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 65/2 54242531> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 65/1 54242515>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 64/3 54241713> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 64/2 54241677>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 64/1 54241663> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 63/3 54240575>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 63/2 54240561> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 63/1 54240545>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 62/3 54237747> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 62/2 54237733>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 62/1 54237717> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 61/3 54237121>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 61/2 54237105> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 61/1 54237071>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 60/3 54236273> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 60/2 54236257>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 60/1 54236243> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 59/3 54235445>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 59/2 54235431> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 59/1 54235415>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 58/3 54234617> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 58/2 54234603>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 58/1 54234567> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 57/3 54233771>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 57/2 54233755> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 57/1 54233741>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 56/3 54233143> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 56/2 54233127>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 56/1 54233113> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 55/3 54232315>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 55/2 54232301> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 55/1 54232265>
;;  #<Entity-Handle of SECONDARY-DESCRIPTION-ID 54/4 54231467> #<Entity-Handle of SECONDARY-DESCRIPTION-I
D 54/3 54231453>)
;; (#<Entity-Handle of PRIMARY-DESCRIPTION-ID 65/4 54244675> #<Entity-Handle of PRIMARY-DESCRIPTION-ID 64
/4 54244661>
;;  #<Entity-Handle of PRIMARY-DESCRIPTION-ID 63/4 54244645> #<Entity-Handle of PRIMARY-DESCRIPTION-ID 62
/4 54244631>
;;  #<Entity-Handle of PRIMARY-DESCRIPTION-ID 61/4 54244615> #<Entity-Handle of PRIMARY-DESCRIPTION-ID 60
/4 54244601>
;;  #<Entity-Handle of PRIMARY-DESCRIPTION-ID 59/4 54244565> #<Entity-Handle of PRIMARY-DESCRIPTION-ID 58
/4 54244551>
;;  #<Entity-Handle of PRIMARY-DESCRIPTION-ID 57/4 54244535> #<Entity-Handle of PRIMARY-DESCRIPTION-ID 56
/4 54244521>
;;  #<Entity-Handle of PRIMARY-DESCRIPTION-ID 55/4 54244505> #<Entity-Handle of PRIMARY-DESCRIPTION-ID 54
/8 54244471>
;;  #<Entity-Handle of PRIMARY-DESCRIPTION-ID 54/7 54244455> #<Entity-Handle of PRIMARY-DESCRIPTION-ID 54
/6 54244441>
;;  #<Entity-Handle of PRIMARY-DESCRIPTION-ID 54/5 54244425>)
;; (#<Entity-Handle of SUB-SUB-SUB-VPG 41/5 54244221>)
;; (#<Entity-Handle of FAMILY-WITHIN-A-SSS-VPG 51/5 54244411> #<Entity-Handle of FAMILY-WITHIN-A-SSS-VPG
50/5 54244375>
;;  #<Entity-Handle of FAMILY-WITHIN-A-SSS-VPG 49/5 54244361> #<Entity-Handle of FAMILY-WITHIN-A-SSS-VPG
48/5 54244345>
;;  #<Entity-Handle of FAMILY-WITHIN-A-SSS-VPG 47/5 54244331> #<Entity-Handle of FAMILY-WITHIN-A-SSS-VPG
46/5 54244315>
;;  #<Entity-Handle of FAMILY-WITHIN-A-SSS-VPG 45/5 54244301> #<Entity-Handle of FAMILY-WITHIN-A-SSS-VPG
44/5 54244265>
;;  #<Entity-Handle of FAMILY-WITHIN-A-SSS-VPG 43/5 54244251> #<Entity-Handle of FAMILY-WITHIN-A-SSS-VPG
42/5 54244235>))
;;Command:
;
;
;
;
;(defun create-the-STATICE-DATA-BASE-FOR-THE-SPECIFIC-Sub-Sub-Sub-VPG  (list-of-lists  data-base-pathname
)   ;;;;;;;; *data*
;  (let ((THE-FINAL-RESULT nil)
;        (result-SECONDARY-entities nil) (result-primary-entities nil) (result-VPGs  nil) (result-families
 nil)
;        (record-of-SUB-SUB-SUB-VPGS   nil)
;        (record-of-FAMILIES-within-the-SSS-VPG  nil)
;        (record-of-PRIMARIES-DESCRIPTIONS nil))
;  (loop for APN-Catalogue-line  in list-of-lists
;       do
;    (let (
```

```
;           (VPG (second (assoc ':VPG-CODE APN-Catalogue-line)))
;           (FAMILY (second (assoc ':FAMILY APN-Catalogue-line)))
;           (PRIMARY-DESCRIPTION (second (assoc ':PRIMARY-DESCRIPTION APN-Catalogue-line))))
;       (cond
;        ((not (member VPG record-of-SUB-SUB-SUB-VPGS  :test #'string-equal))
;            (push  VPG record-of-SUB-SUB-SUB-VPGS)))
;       (cond
;        ((not (member  FAMILY record-of-FAMILIES-within-the-SSS-VPG :test #'string-equal))
;            (push  FAMILY record-of-FAMILIES-within-the-SSS-VPG)))
;       (cond
;        ((not (member PRIMARY-DESCRIPTION record-of-PRIMARIES-DESCRIPTIONS  :test #'string-equal))
;            (push PRIMARY-DESCRIPTION record-of-PRIMARIES-DESCRIPTIONS))))
;
;     (push (make-APN-CATALOGUE-entry APN-Catalogue-line    data-base-pathname) result-SECONDARY-entities)
)
;   (loop for VPG in record-of-SUB-SUB-SUB-VPGS
;        do
;     (push
;       (with-database  (db    data-base-pathname)     ;;;;;;; *ROVER-TRY-pathname*
;                   (with-transaction ()
;                     (make-Sub-Sub-Sub-Vpg  :VPG-code  VPG))) result-VPGs))
;   (loop for family in record-of-FAMILIES-within-the-SSS-VPG
;        do
;     (push
;       (with-database  (db   data-base-pathname)   ;;;;;;; *ROVER-TRY-pathname*)
;        (with-transaction ()
;          (make-FAMILY-within-a-SSS-Vpg  :FAMILY-CODE  family))) result-families))
;   (loop for primary-DESCRIPTION   in   record-of-PRIMARIES-DESCRIPTIONS
;        do
;     (push
;       (with-database  (db   data-base-pathname)    ;;;;;; *ROVER-TRY-pathname*)
;        (with-transaction ()
;          (make-Primary-Description-ID  :PRIMARY-DESCRIPTION-OF-THE-PART  primary-DESCRIPTION))) result-p
rimary-entities))
;
;(push result-families THE-FINAL-RESULT) (push result-VPGs THE-FINAL-RESULT)
;(push result-primary-entities  THE-FINAL-RESULT) (push result-SECONDARY-entities THE-FINAL-RESULT)
;THE-FINAL-RESULT
;;(print result-SECONDARY-entities) (print  result-primary-entities) (print result-VPGs) (print result-fa
milies)
;))
;
;
;
;
;
;
;
;
;
;(defun make-the-links-PRIMARIES-SECONDARIES   (primaries-entities-in-STATICE secondaries-ENTITIES-in-STA
TICE data-base-pathname)
;   (with-database  (db data-base-pathname)
;     (with-transaction ()
;       (loop for PRIMARY-ENTITY    in  primaries-entities-in-STATICE
;            do
;        (loop for SECONDARY-ENTITY    in   secondaries-ENTITIES-in-STATICE
;             do
;          (cond
;            ((string-equal  (Secondary-Description-ID-PRIMARY-DESCRIPTION SECONDARY-ENTITY)
;                            (Primary-Description-ID-PRIMARY-DESCRIPTION-OF-THE-PART  PRIMARY-ENTITY))
;                (add-to-set  (Primary-Description-ID-Has-Secondary-Descriptions  PRIMARY-ENTITY)
;                             SECONDARY-ENTITY)
;                (setf (Secondary-Description-ID-Belongs-to-Primary-Description   SECONDARY-ENTITY)
;                      PRIMARY-ENTITY))))))))
;
;
;(defun  make-the-links-PRIMARIES-FAMILIES (primaries-entities-in-STATICE family-entities-in-STATICE data
-base-pathname)
;   (with-database  (db data-base-pathname)
;     (with-transaction ()
;       (loop for FAMILY  in family-entities-in-STATICE
;            do
;        (loop for PRIMARY in primaries-entities-in-STATICE
;             do
;          (let ((help (Primary-Description-ID-Has-Secondary-Descriptions  PRIMARY)))
```

```
;           (cond ((check-if-everyone-has-the-same-family-ID help  (FAMILY-within-a-SSS-Vpg-FAMILY-CODE F
AMILY))
;
;             (add-to-set (FAMILY-within-a-SSS-Vpg-Has-Primary-Descriptions  FAMILY)
;                         PRIMARY)
;             (setf (Primary-Description-ID-Belongs-to-Family  PRIMARY)  FAMILY)))))))))
;
;
;
;;Command: (with-database  (db    *ROVER-TRY-pathname*) (with-database  (db    *ROVER-TRY-pathname*)
;;                  (with-transaction ()  (check-if-everyone-has-the-same-family-ID (Primary-Description-ID
-Has-Secondary-Descripti
;;ons #<Entity-Handle of PRIMARY-DESCRIPTION-ID 58/4 454176260>) "HAD")))
;;T
;;Command:
;
;
;
;(defun check-if-everyone-has-the-same-family-ID (list-of-secondaries   family-string)
;  (every  #'(lambda (a-secondary)
;             (string-equal (Secondary-Description-ID-FAMILY a-secondary) family-string))
;         list-of-secondaries))
;
;
;
;(defun make-the-links-FAMILIES-Sub-Sub-Sub-VPG   (family-entities-in-STATICE
;                                                 VPG-sub-sub-sub-group-entity-in-STATICE data-base-pathn
ame)
;  (with-database  (db data-base-pathname)
;    (with-transaction ()
;      (loop for family in family-entities-in-STATICE
;          do
;       (setf (FAMILY-within-a-SSS-Vpg-belongs-to-SSS-VPG   family)   VPG-sub-sub-sub-group-entity-in-STA
TICE))
;      (loop for family in family-entities-in-STATICE
;          do
;       (add-to-set (Sub-Sub-Sub-Vpg-HAS-Families-within   VPG-sub-sub-sub-group-entity-in-STATICE)
;                   family)))))
;
;
;
;;*****************************************************************************************
;
;;;;;;;;      THIS   IS   COMING  FROM THE   A.P.N. tape,   ellectronically.
;
;
;
;
;;*****************************************************************************************************
*****************
;;*****************************************************************************************************
*****************
;;;;;;;;*headings*   COMING from the file :      K87.dat
;
;
;
;;"A10-MARQUE                                  "
;;"A10AAUSTIN                          A       "
;;"A10HHONDA                           HONDA   "
;;"A10MMORRIS                          MORRIS  "
;;"A10RROVER                           ROVER   "
;;"A10XAUSTIN ROVER                    AUSROV  "
;;"A20-MODEL RANGE (AUSTIN ROVER)              "
;;"A20AADO 73                          A73     "
;;"A20BLC8                             LC8     "
;;"A20CMAESTRO                         LM10    "
;;"A20DACCLAIM                         ACL     "
;
;
;
;
;;(defun  each-Feature-Group-has-its-identical-Integer   (ls)   ;;;;*headings*
;;  (let ((pointer 0) (result nil))
;;    (loop for i in ls
;;        do
;;      (cond
```

```
;;      ((string-equal "-" (subseq i 3 4))
;;          (push (list (subseq i 0 3) (incf pointer))
;;                result))))
;;))        ;;;;;;THE result MUST be a  global  variable.
;;
;;(defvar  *FEATURE-GROUP-WITH-NUMBER*   (each-Feature-Group-has-its-identical-Integer *headings*))
;;
;;
;;(defun given-a-number-return-the-Feature-Group (a-number)
;;  (let ((result nil))
;;    (loop for pair in *FEATURE-GROUP-WITH-NUMBER*
;;        do
;;      (cond
;;      ((equal (second pair) a-number)
;;           (setq result (first pair)))))
;;result))
;;
;;
;;;*****************************************************************************************************
*****************
;;
;;;
;;;
;;;;;;;;   all the 1110DA mirrors are in.............................
;;;;;
;Command: (length        (with-database (db *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*)
;    (with-transaction ()
;(entity-of-the-Secondary-Description-ID-with-this-VPG  "1110da"))))
;111
;
;Command: (length *APN-CATALOGUE-flat-data-MIRRORS*)
;111
;Command:


;(defvar *APN-CATALOGUE-flat-data-MIRRORS*  '(
;
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RA1110DA001MIRRORS EXT    INSTR DRG                NPIECEC87040
6P   0050INSTR DRG MIRRORS EXT N   00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA001H/S-SELF COLR  MIRROR ASSY-EXT-DRIVER   NPIECEC87040
6P   0250MIRROR ASSY-EXT-DRIVERNY  76200 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA002H/S-BODY COLR  MIRROR ASSY-EXT-DRIVER   NPIECEC87040
6P   0300MIRROR ASSY-EXT-DRIVERNY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA003H/S-SELF COLR  MIRROR ASSY-EXT-PASSR    NPIECEC87040
6P   0350MIRROR ASSY-EXT-PASSR NY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA004H/S-BODY COLR  MIRROR ASSY-EXT-PASSR    NPIECEC87040
6P   0400MIRROR ASSY-EXT-PASSR NY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA005H/S-LESS COVER MIRROR ASSY-EXT-DRIVER   NPIECEC87040
6P   0450MIRROR ASSY-EXT-DRIVERNY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA006H/S-LESS COVER MIRROR ASSY-EXT-PASSR    NPIECEC87040
6P   0500MIRROR ASSY-EXT-PASSR NY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA007SELF COLOUR    MIRROR ASSY-EXT-H/S      YPIECEC87040
6P   0550MIRROR ASSY-EXT-H/S SENY  00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA008BODY COLOUR    MIRROR ASSY-EXT-H/S      YPIECEC87040
6P---0600MIRROR ASSY-EXT-H/S BONY  76200  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA009LESS COVER     MIRROR ASSY-EXT-H/S      NPIECEC87040
6P   0650MIRROR ASSY-EXT-H/S LENY  00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA010R/C-SELF COLR  MIRROR ASSY-EXT-DRIVER   NPIECEC87040
6P   0700MIRROR ASSY-EXT-DRIVERNY  76200 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA011R/C-SELF COLR  MIRROR ASSY-EXT-PASSR    NPIECEC87040
6P   0750MIRROR ASSY-EXT-PASSR NY  76250 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA012R/C-BODY COLR  MIRROR ASSY-EXT-DRIVER   NPIECEC87040
6P   0800MIRROR ASSY-EXT-DRIVERNY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA013R/C-BODY COLOURMIRROR ASSY-EXT-PASSR    NPIECEC87040
6P   0850MIRROR ASSY-EXT-PASSR NY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA014R/C-BRIGHT FIN MIRROR ASSY-EXT-DRIVER   NPIECEC87040
6P   0900MIRROR ASSY-EXT-DRIVERNY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA015R/C-BRIGHT FIN MIRROR ASSY-EXT-PASSR    NPIECEC87040
6P   0950MIRROR ASSY-EXT-PASSR NY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA016R/C-LESS COVER MIRROR ASSY-EXT-DRIVER   NPIECEC87040
6P   1000MIRROR ASSY-EXT-DRIVERNY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA017R/C-LESS COVER MIRROR ASSY-EXT-PASSR    NPIECEC87040
6P   1050MIRROR ASSY-EXT-PASSR NY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA018R/C-SELF COLOURMIRROR ASSY-EXT          NPIECEC87040
6P   1100MIRROR ASSY-EXT R/C-SENY  00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA019R/C-BODY COLOURMIRROR ASSY-EXT          YPIECEC87040
```

```
6P---1150MIRROR ASSY-EXT R/C-BONY  76200  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA020R/C-BRIGHT FIN MIRROR ASSY-EXT          NPIECEC87040
6P   1200MIRROR ASSY-EXT R/C-BRNY  00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA021R/C-LESS COVER MIRROR ASSY-EXT          NPIECEC87040
6P   1250MIRROR ASSY-EXT R/C-LENY  00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA022E/C-LESS COVER MIRROR ASSY-EXT-DRIVER   NPIECEC87040
6P   1300MIRROR ASSY-EXT-DRIVERNY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA023E/C-LESS COVER MIRROR ASSY-EXT-PASSR    NPIECEC87040
6P   1350MIRROR ASSY-EXT-PASSR NY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA024E/C-LESS COVER MIRROR ASSY-EXT          NPIECEC87040
6P   1400MIRROR ASSY-EXT E/C-LENY  00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA025MEM-LESS COVER MIRROR ASSY-EXT-DRIVER   NPIECEC87040
6P   1450MIRROR ASSY-EXT-DRIVERNY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA026MEM-LESS COVER MIRROR ASSY-EXT-PASSR    NPIECEC87040
6P   1500MIRROR ASSY-EXT-PASSR NY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA027MEM-LESS COVER MIRROR ASSY-EXT          NPIECEC87040
6P   1550MIRROR ASSY-EXT MEM-LENY  00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA028E/C SELF COLOURMIRROR ASSY-EXT-DRIVER   NPIECEC87041
4P   1570MIRROR ASSY-EXT-E/C-SENY  76200 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA029E/C SELF COLOURMIRROR ASSY-EXT-PASSR    NPIECEC87041
4P   1590MIRROR ASSY-EXT-E/C-SENY  76250 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA030H/S BODY COLR  MIRROR ASSY-EXT-KIT    R NPIECEC87111
7P   0200MIRROR ASSY-EXT-KIT   YY  00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA031R/C-BODY COLR  MIRROR ASSY-EXT-KIT      NPIECEC87112
4P   0310MIRROR ASSY-EXT-KIT   NY  00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA032E/C SELF COLR  MIRROR ASSY-EXT-KIT      NPIECEC87112
7P   0100MIRROR ASSY-EXT-KIT   NY  00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA033E/C BODY COLOURMIRROR ASSY-EXT          YPIECEC88051
2P---1420MIRROR ASSY-EXT-E/C   N   76200  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA034EXTERNAL       MIRROR ASSY-HEAD         NPIECEC88080
1P   1595MIRROR ASSY-HEAD-EXT  N   00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA035R/C-SELF COLR  MIRROR ASSY-EXT-KIT      NPIECEC89042
5P   0145MIRROR ASSY-EXT-KIT   N   00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RB1110DA036ACCESSORY      MIRROR-TOWING            NPIECEC89072
8P   5000MIRROR-TOWING-ACCESSRYN   00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RC1110DA001SELF COLR-GRAINCOVER-EXT MIRROR         NPIECEC87040
6P   1600COVER-EXT MIRROR SELF N   76221  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RC1110DA002SELF COLR-UNFINCOVER-EXT MIRROR          PIECEC87040
6    1650COVER-EXT MIRROR SELF N          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RC1110DA003BODY COLOUR    COVER-EXT MIRROR          PIECEC87040
6    1700COVER-EXT MIRROR BODY N          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RC1110DA004BRIGHT FIN     COVER-EXT MIRROR          PIECEC87040
6    1750COVER-EXT MIRROR BRIGHN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RD1110DA001MIRROR         GLASS ASSY-FLAT          NPIECEC87040
6P   1800GLASS ASSY-FLAT MIRRORNY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RD1110DA002EXT MIRROR ASSYGLASS-FLAT               NPIECEC87040
6P   1850GLASS-FLAT-FSC EXT MIRNY  00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RD1110DA003MIRROR         GLASS ASSY-CONVEX        NPIECEC87040
6P   1900GLASS ASSY-CONVEX MIRRNY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RD1110DA004EXT MIRROR ASSYGLASS-CONVEX             NPIECEC87040
6P   1950GLASS-CONVEX-FSC EXT MNY  00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RD1110DA005MIRROR         GLASS ASSY-FLAT-HEATED   NPIECEC87040
6P   2000GLASS ASSY-FLAT-HEATEDNY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RD1110DA006MIRROR         GLASS ASSY-CONVX-HEATD   NPIECEC87040
6P   2050GLASS ASSY-CONVX-HEATDNY  00000 YARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RE1110DA001EXT MIRROR     PLATE-RETAINING           PIECEC87040
6    2100PLATE-RETAINING EXT MIN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RF1110DA001EXT MIRROR     CHOCK                     PIECEC87040
6    2150FINISHER-CHEATER INT-FN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RG1110DA001EXT MIRROR     KNOB-CONTROL             NPIECEC87040
6P   2200KNOB-CONTROL EXT MIRRON   88154  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RH1110DA001EXT MIRROR ASSYCASE                       PIECEC87040
6    2250CASE EXT MIRROR ASSY  N          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RJ1110DA001EXT MIRROR ASSYBASE-DRIVERS               NPIECEC87040
6P   2300BASE-DRIVERS EXT MIRRON   7621Z  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RJ1110DA002EXT MIRROR ASSYBASE-PASSRS                PIECEC87040
6    2350BASE-PASSRS EXT MIRRORN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RJ1110DA003EXT MIRROR ASSYBASE                       NPIECEC88080
1P   3300BASE-EXT MIRROR ASSY  N   00000  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RK1110DA001EXT MIRROR ASSYCOVER-BASE                 NPIECEC87040
6P   2400COVER-BASE EXT MIRROR N   7622Z  ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RL1110DA001EXT MIRROR ASSYSPRING-TENSION             PIECEC87040
6    2450SPRING-TENSION EXT MIRN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RL1110DA002EXT MIRROR ASSYSPRING-COMPRESSION         PIECEC87040
6    2500SPRING-COMPRESSION EXTN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS    RM1110DA001EXT MIRROR ASSYSLEEVING-PLASTIC           PIECEC87040
```

```
6     2550SLEEVING-PLASTIC EXT MN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RN1110DA001EXT MIRROR ASSYCLAMP-CABLE                PIECEC87040
6     2600CLAMP-CABLE EXT MIRRORN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RP1110DA001EXT MIRROR ASSYMOTOR & CABLE ASSY         PIECEC87040
6     2650MOTOR & CABLE ASSY EXTN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RQ1110DA001EXT MIRROR ASSYMOTOR ASSY-ELECTRIC        PIECEC87040
6     2700MOTOR ASSY-ELECTRIC EXN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RR1110DA001EXT MIRROR ASSYCABLE ASSY                 PIECEC87040
6     2750CABLE ASSY EXT MIRROR N          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RS1110DA001EXT MIRROR ASSYELEMENT-HEATER             PIECEC87040
6     2800ELEMENT-HEATER EXT MIRN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RT1110DA001EXT MIRROR ASSYPAD-ADHESIVE               PIECEC87040
6     2850PAD-ADHESIVE EXT MIRRON          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RT1110DA002EXT MIRROR      PAD-PRESSURE             NPIECEC87091
7P    4550PAD-PRESSR-EXT MIRROR N    00000 ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RT1110DA003INTERIOR MIRRORPAD-SUCTION               NPIECEC89042
4P    6200PAD-SUCTION-INT MIRRORN    00000 ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RT1110DA004EXTERIOR MIRRORPAD-SUPPORT               NPIECEC89092
7P    4650PAD-SUPPORT-EXT MIRRORN    00000 ARBDAAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RU1110DA001EXT MIRROR ASSYPIVOT-SPHERICAL            PIECEC87040
6     2900PIVOT-SPHERICAL EXT MIN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RV1110DA001EXT MIRROR ASSYHANDLE & INNER SEAT        PIECEC87040
6     2950HANDLE & INNER SEAT EXN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RW1110DA001EXT MIRROR ASSYBACKPLATE-FLAT             PIECEC87040
6     3000BACKPLATE-FLAT EXT MIRN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RW1110DA002EXT MIRROR ASSYBACKPLATE-CONVEX           PIECEC87040
6     3050BACKPLATE-CONVEX EXT MN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RX1110DA001EXT MIRROR ASSYSTRIP-ADHESIVE             PIECEC87040
6     3100STRIP-ADHESIVE EXT MIRN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RY1110DA001EXT MIRROR ASSYLABEL-IDENTITY             PIECEC87040
6     3150LABEL-IDENTITY EXT MIRN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RZ1110DA001EXT MIRROR ASSYPIN-TERMINAL               NPIECEC87040
6P    3200PIN-TERMINAL EXT MIRRON    00000 ARELCDE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RZ1110DA002EXT MIRROR ASSYHSG-TERMINAL PIN           NPIECEC87040
6P    3250HOUSING-TERMINAL PIN EN    00000 ARELCDE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RZ1110DA003EXT MIRROR ASSYSOCKET-TERMINAL            NPIECEC87040
6P    3300SOCKET-TERMINAL EXT MIN    00000 ARELCDE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     RZ1110DA004EXT MIRROR ASSYHSG-SOCKET                 NPIECEC87040
6P    3350HOUSING-SOCKET EXT MIRN    00000 ARELCDE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SB1110DA001EXT MIRROR ASSYCABLE-MOVEMENT-UPPER       PIECEC87040
6     3400CABLE-MOVEMENT-UPPER EN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SB1110DA002EXT MIRROR ASSYCABLE-MOVEMENT-LOWER       PIECEC87040
6     3450CABLE-MOVEMENT-LOWER EN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SB1110DA003EXT MIRROR ASSYCABLE ASSY-CONTROL         PIECEC87040
6     3500CABLE ASSY-CONTROL EXTN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SC1110DA001EXT MIRROR ASSYHSG-CONTROL                PIECEC87040
6     3550HOUSING-CONTROL EXT MIN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SD1110DA001EXT MIRROR ASSYJOYSTICK-CONTROL           PIECEC87040
6     3600JOYSTICK-CONTROL EXT MN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SE1110DA001EXT MIRROR ASSYCOVER ASSY-KNOB            PIECEC87040
6     3650COVER ASSY-KNOB EXT MIN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SE1110DA002EXT MIRROR ASSYCOVER-KNOB                 NPIECEC87062
6P    3670COVER-KNOB-EXT MIRROR N    88159 ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SF1110DA001EXT MIRROR ASSYSPRING-CONTROL HSG         PIECEC87040
6     3700SPRING-CONTROL HOUSINGN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SG1110DA001EXT MIRROR ASSYTRIGGER-RELEASE            PIECEC87040
6     3750TRIGGER-RELEASE EXT MIN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SH1110DA001EXT MIRROR ASSYSEAT-INNER-RETAINER        PIECEC87040
6     3800SEAT-INNER-RETAINER EXN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SJ1110DA001EXT MIRROR ASSYLINK-SPRING                PIECEC87040
6     3850LINK-SPRING EXT MIRRORN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SK1110DA001EXT MIRROR ASSYPLATE-ANCHOR-SPRING        PIECEC87040
6     3900PLATE-ANCHOR-SPRING EXN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SL1110DA001EXT MIRROR ASSYPIVOT ASSY-GLASS           PIECEC87040
6     3950PIVOT ASSY-GLASS EXT MN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SL1110DA002EXT MIRROR ASSYPIVOT-GLASS                PIECEC87040
6     4000PIVOT-GLASS EXT MIRRORN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SM1110DA001EXT MIRROR ASSYLEVER-R/C                  PIECEC87040
6     4050LEVER-R/C EXT MIRROR AN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SN1110DA001EXT MIRROR ASSYBUSH-PIVOT                 PIECEC87040
6     4100BUSH-PIVOT EXT MIRROR N          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SP1110DA001EXT MIRROR ASSYFORK-GLASS ADJUSTING       PIECEC87040
6     4150FORK-GLASS ADJUSTING EN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SQ1110DA001EXT MIRROR ASSYPLATE-SPRING LOAD          PIECEC87040
6     4200PLATE-SPRING LOAD EXT N          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SR1110DA001EXT MIRROR ASSYROD-SPRING ANCHOR          PIECEC87040
```

```
6     4250ROD-SPRING ANCHOR EXT N          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SS1110DA001EXT MIRROR ASSYADAPTOR-MOTOR MTG          PIECEC87040
6     4300ADAPTOR-MOTOR MTG EXT N          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     ST1110DA001EXT MIRROR ASSYTHERMOSTAT-HTR CNTRL       PIECEC87040
6     4350THERMO-HEATER CONTROL N          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SU1110DA001EXT MIRROR ASSYGAITER-EXTERNAL            PIECEC87040
6     4400GAITER-EXTERNAL EXT MIN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SV1110DA001EXT MIRROR ASSYSLEEVING-CABLE             PIECEC87040
6     4450SLEEVING-CABLE EXT MIRN          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SW1110DA001EXT MIRROR      GASKET                    NPIECEC87091
7P    4500GASKET-EXT MIRROR      N   00000 ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SX1110DA001EXT MIRROR      SLIDE-PAD-PRESSURE        NPIECEC87091
7P    4600SLIDE-PAD-PRESSR-EXT MN   00000 ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SY1110DA001EXT MIRROR      CHEATER                   NPIECEC88012
7P    4490CHEATER-EXT MIRROR     N   76220 ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     SZ1110DA001EXT MIRROR ASSYCLIP-RETAINING             NPIECEC88051
1P    4460CLIP-RETG-EXT MIRROR   N   00000 ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     TB1110DA001DIPPING         MIRROR ASSY-INTERIOR      NPIECEC87040
6P    6000MIRROR ASSY-INT-DIPPNGN   76400 ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     TB1110DA002DIPPING         MIRROR-INTERIOR           NPIECEC88070
5P    6050MIRROR-INT-DIPPING     N   00000 ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     TB1110DA003NON DIPPING     MIRROR ASSY-INTERIOR      NPIECEC88070
5P    6100MIRROR ASSY-INT-N/DIP N   00000 ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     TB1110DA004NON DIPPING     MIRROR-INTERIOR           NPIECEC88070
5P    6150MIRROR-INTERIOR-N/DIP N   00000 ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     TC1110DA001INT MIRROR      HEAD ASSY-DIPPING         PIECEC87040
6     6050HEAD ASSY-DIPPING INT N          ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     TD1110DA001INT MIRROR      STEM                      PIECEC87040
6     6100STEM INT MIRROR        N         ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     TE1110DA001INT MIRROR      PAD-ADHESIVE              PIECEC87040
6     6150STEM INT MIRROR        N         ARTHEAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     UA1110EA001DOOR GLASS      INSTRUCTION DRG           NPIECEC87040
6P    0050INSTRUCTION DRG-DOOR GN          ARBDAAE"
;"0154CT & H EXTERIOR, GLASS & MIRRORS     UA1110EA002FRONT DOOR      INSTRUCTION DRG-SASH      NPIECEC87112
3P    0100INSTR DRG-SASH-FRONT   N   00000 ARBDAAE"
;
;
;))
;
;
;


;;;; ALL the seats  1109AA   are in ..............................
;Command: (length *APN-CATALOGUE-flat-data-SEATS*)
;40
;Command: (length (with-database (db *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*)
;     (with-transaction ()
;(entity-of-the-Secondary-Description-ID-with-this-VPG  "1109AA"))))
;40
;Command:
;
;
;(defvar *APN-CATALOGUE-flat-data-SEATS* '(
;
;"0154HSEATING                             AA1109AA001MAN L/SUPP      INSTR DRG                 YPIECEC87021
2P---0050INSTR DRG MAN L/SUPP   N   8101Z ARTHSAE"
;"0154HSEATING                             AA1109AA002MAN-NO L/SUPP   INSTR DRG                 YPIECEC87021
2P---0100INSTR DRG MAN-NO L/SUPN   8101Z ARTHSAE"
;"0154HSEATING                             AA1109AA003ELEC-MEMORY     INSTR DRG                 PIECEC87021
2     0150INSTR DRG ELEC-MEMORY N          ARTHSAE"
;"0154HSEATING                             AA1109AA004ELEC-NO MEMORY  INSTR DRG                 PIECEC87021
2     0200INSTR DRG ELEC-NO MEMON          ARTHSAE"
;"0154HSEATING                             AA1109AA005SEAT FRONT      INSTR DRG-FIXING          NPIECEC89061
6P    0040INSTR DRG-FIXING-ST/F N   00000 ARTHSAE"
;"0154HSEATING                             AB1109AA001FIXED           ARRGT DRG-ST/F-MANUAL     PIECEC87021
2     0250ARRGT DRG-ST/F-MANUAL N          ARTHSAE"
;"0154HSEATING                             AB1109AA002L/SUPP          ARRGT DRG-ST/F-MANUAL     PIECEC87021
2     0300ARRGT DRG-ST/F-MANUAL N          ARTHSAE"
;"0154HSEATING                             AB1109AA003NO L/SUPP       ARRGT DRG-ST/F-MANUAL     PIECEC87021
2     0350ARRGT DRG-ST/F-MANUAL N          ARTHSAE"
;"0154HSEATING                             AB1109AA004L/SUPP-MEMORY   ARRGT DRG-ST/F-ELEC        PIECEC87021
2     0400ARRGT DRG-ST/F-ELEC L/N          ARTHSAE"
;"0154HSEATING                             AB1109AA005L/SUPP-NO MEM   ARRGT DRG-ST/F-ELEC        PIECEC87021
2     0450ARRGT DRG-ST/F-ELEC L/N          ARTHSAE"
;"0154HSEATING                             AB1109AA006NO L/SUPP-MEM   ARRGT DRG-ST/F-ELEC        PIECEC87021
2     0500ARRGT DRG-ST/F-ELEC NON          ARTHSAE"
```

```
;"0154HSEATING                                  AB1109AA007NO L/SUPP & MEMARRGT DRG-ST/F-ELEC       PIECEC87021
2    0550ARRGT DRG-ST/F-ELEC NON                ARTHSAE"
;"0154HSEATING                                  AB1109AA008TRIM WIRE/CLIPSARRGT DRG-SEAT FRONT     NPIECEC89073
1P   0240ARRGT DRG-ST/F-T/WIRE N     00000      ARTHSAE"
;"0154HSEATING                                  AC1109AA001FIXINGS         CHART-TORQUE-ST/F        PIECEC87021
2    0600CHART-TORQUE-ST/F FIXIN                ARTHSAE"
;"0154HSEATING                                  AC1109AA002ST/F COMP       CHART-ARRANGEMENTS       NPIECEC88071
4P   0570CHART-ARRGTS-ST/F COMPN     81112      ARTHSAE"
;"0154HSEATING                                  AD1109AA001MANUAL-DRIVER   SEAT-FRONT COMPLETE    R YPIECEC
 PS/D0650SEAT-FRONT COMPLETE MAN     81100      ARTHSAE"
;"0154HSEATING                                  AD1109AA002MANUAL-PASSR    SEAT-FRONT COMPLETE    R YPIECEC
 PS/D0700SEAT-FRONT COMPLETE MAN     81500      ARTHSAE"
;"0154HSEATING                                  AD1109AA003ELECTRIC-DRIVERSEAT-FRONT COMPLETE     R YPIECEC
 PS/D0750SEAT-FRONT COMPLETE ELN     81100      ARTHSAE"
;"0154HSEATING                                  AD1109AA004ELECTRIC-PASSR SEAT-FRONT COMPLETE     R YPIECEC
 PS/D0800SEAT-FRONT COMPLETE ELN                ARTHSAE"
;"0154HSEATING                                  AD1109AA005MANUAL          SEAT-FRONT COMPLETE    R YPIECEC
 PS/D0850SEAT-FRONT COMPLETE MAN     81100      ARTHSAE"
;"0154HSEATING                                  AD1109AA006ELECTRIC        SEAT-FRONT COMPLETE    R YPIECEC
 PS/D0900SEAT-FRONT COMPLETE ELN     85500      ARTHSAE"
;"0154HSEATING                                  AE1109AA001SIDE INTRUSION CROSSBAR ASSY           R NPIECEC87033
1P   1250CROSSBAR ASSY          N               ARTHSAE"
;"0154HSEATING                                  AF1109AA001ST/F-TIP CATCH PLATE-PROTECTION          NPIECEC88112
3P   1300PLT-PROTECTN-ST/F TIP N     00000      ARTHSAE"
;"0154HSEATING                                  AG1109AA001ELECTRIC-DRIVERCUSHN-COMP-SEAT FRONT     YPIECEC89011
4PS/D0930CUSHN-COMP-ST/F-EL/DR N     81130 YARTHSAE"
;"0154HSEATING                                  AG1109AA002MANUAL-DRIVER   CUSHN-COMP-SEAT FRONT     YPIECEC89011
4PS/D0928CUSHN-COMP-ST/F-MN/DR N     77230 YARTHSAE"
;"0154HSEATING                                  AG1109AA003MANUAL-PASSR    CUSHN-COMP-SEAT FRONT     YPIECEC89011
4PS/D0929CUSHN-COMP-ST/F-MN/PSRN     77430 YARTHSAE"
;"0154HSEATING                                  AG1109AA004MANUAL          CUSHN-COMP-SEAT FRONT     YPIECEC89031
5PS/D0915CUSHN-COMP-SEAT-FR-MANN     00000      ARTHSAE"                                       ;;;;; CUSHN-COMP-S
EAT-FRONT  ;;;;; a MISTAKE
;
;"0154HSEATING                                  AH1109AA001'A'FRAME        HEADRSTNT-COMP-FRONT   R YPIECEC
 PS/D0950HEADRSTNT-COMP-FRONT 'N     77580      ARTHSAE"
;"0154HSEATING                                  AH1109AA002'A'FRM INC PADSHEADRSTNT-COMP-FRONT    R YPIECEC
 PS/D1000HEADRSTNT-COMP-FRONT 'N     81145      ARTHSAE"
;"0154HSEATING                                  AH1109AA003BOLSTER-FIXED   HEADRSTNT-COMP-FRONT    R YPIECEC
 PS/D1050HEADRSTNT-COMP-FRONT BN                ARTHSAE"
;"0154HSEATING                                  AH1109AA004BOLSTER-ADJL    HEADRSTNT-COMP-FRONT    R YPIECEC
 PS/D1100HEADRSTNT-COMP-FRONT BN     81145      ARTHSAE"
;"0154HSEATING                                  AH1109AA005BOLSTER-NODDINGHEADRSTNT-COMP-FRONT    R YPIECEC
 PS/D1150HEADRSTNT-COMP-FRONT BN     81145      ARTHSAE"
;"0154HSEATING                                  AJ1109AA001ELECTRIC-DRIVERSQUAB-COMP-SEAT FRONT     YPIECEC89011
4PS/D0910SQUAB-COMP-ST/F-EL/DR N     81120 YARTHSAE"
;"0154HSEATING                                  AJ1109AA002MANUAL-DRIVER   SQUAB-COMP-SEAT FRONT     YPIECEC89011
4PS/D0908SQUAB-COMP-SEAT FRONT N     77220 YARTHSAE"
;"0154HSEATING                                  AJ1109AA003MANUAL-PASSR    SQUAB-COMP-SEAT FRONT     YPIECEC89011
4PS/D0909SQUAB-COMP-ST/F-MN/PSRN     77420 YARTHSAE"
;"0154HSEATING                                  AJ1109AA004MANUAL          SQUAB-COMP-SEAT FRONT     YPIECEC89031
5PS/D0904SQB-COMP-SEAT-FR-MAN  N     00000      ARTHSAE"                                       ;;;;; SQUAB-COMP-S
EAT-FRONT ;;a MISTAKE
;"0154HSEATING                                  AM1109AA001ELASTICATED     COVER-EXPENDABLE-ST/F  R YPIECEC
 P   1200COVER-EXPENDABLE-ST/F N                ARTHSAE"
;"0154HSEATING                                  AM1109AA002NON ELESTICATEDCOVER-EXPENDABLE-ST/F     NPIECEC89020
3P   1202CVR-EXPND-ST/F-N/ELASTN     00000      ARTHSAE"
;"0154HSEATING                                  AM1109AA003ELASTICATED     CVR-EXPND-HEADRSTNT-FR   NPIECEC89020
3P   1204CVR-EXPND-H/R-FR-ELASTN     00000      ARTHSAE"
;"0154HSEATING                                  AM1109AA004NON ELASTICATEDCVR-EXPND-HEADRSTNT-FR    NPIECEC89020
3P   1206CVR-EXPND-H/R-FR-N/E  N     00000      ARTHSAE"
;))
;
;
;
;
;
;
;
```

```
;;;; -*- Mode: lisp;  Syntax: Common-lisp; Package: austin-rover -*-


(define-schema  The-data-base-of-the-ADDITIONAL-FEATURES-of-R8  (Feature-Chart-ENTRY))
(define-schema  The-data-base-of-the-ADDITIONAL-FEATURES-of-METRO  (Feature-Chart-ENTRY))


(defvar  *the-pathname-of-the-ADDITIONAL-FEATURES-R8*  #p"stat-FS:>takis>R8-data-only-the-ADDITIONALS")
(defvar  *the-pathname-of-the-ADDITIONAL-FEATURES-METRO*  #p"stat-FS:>takis>METRO-data-only-the-ADDITIONA
LS")


(defun make-METRO-ADDITIONALS-data-database ()  ;;;;; (PATHNAME SYMBOL)
  (make-database   *the-pathname-of-the-ADDITIONAL-FEATURES-METRO*  'The-data-base-of-the-ADDITIONAL-FEAT
URES-of-METRO))


(defun create-the-STATICE-DATA-BASE-METRO   (list-of-lists PATHNAME)   ;;;;;;; *additional-features-metro
*
                                                                  ;;;;;;;  *the-pathname-of-the-ADDITIONAL-FEATURES-M
ETRO*
  (loop for Additional-feature-line in list-of-lists
        do
    (make-new-FEATURE-CHART-ENTRY Additional-feature-line PATHNAME)))

(defun make-R8-ADDITIONALS-data-database ()  ;;;;; (PATHNAME SYMBOL)
  (make-database   *the-pathname-of-the-ADDITIONAL-FEATURES-R8*  'The-data-base-of-the-ADDITIONAL-FEATURE
S-of-R8))


(define-entity-type    Feature-Chart-ENTRY ()
  ((FEATURE-DESCRIPTION string  :cached t)   ;;;:no-nulls t not meaninigful as the DATABASE is already cr
eated )
                                                  ;;;; It is not  :Unique t
  (all-the-combinations-restrictions  list  :cached t)
  (all-the-comments-restrictions  list :cached t)
  (all-the-territory-restrictions  list :cached t)
  (recno string :cached t)
  (featcode string :cached t)  ;;:index t)  ;; for the *data-we-concern*  actually the *feature-chart-ent
ries*
  (standard string :cached t)
  (base string :cached t)
  (optional string :cached t)
  (marketing string :cached t)
  (enveng string :cached t)
  (legal string :cached t)
  (accessory string :cached t)
  (mr  string :cached t)
  (mc string :cached t)
  (class string :cached t) ;;; :index t)  ;because-of-the-hash-table
  (body string :cached t)
  (family string :cached t)
  (capacity string :cached t)
  (performance string :cached t)
  (drive string :cached t)
  (trans string :cached t)
  (auth-ref-in string :cached t)
  (auth-ref-out string :cached t)
  (rel-stage-in string :cached t)
  (rel-stage-out string :cached t)
  (dep-in string :cached t)
  (dep-out  string :cached t)
  (feat-group string :cached t :inverse   entity-of-the-FEATURE-GROUP :inverse-index t)
                     ;; for the *data-we-concern*  actually the *feature-chart-entries*
  (history  string :cached t)
  (errno  string :cached t)
  (all-the-group-restrictions  list :cached t)))


(defun create-the-STATICE-DATA-BASE   (list-of-lists PATHNAME)   ;;;;;;; *the-pathname-of-the-ADDITIONAL-
FEATURES-R8*
                                                                  ;;;;;;;  *the-pathname-of-the-ADDITIONAL-FEATURES-R
8*
  (loop for Additional-feature-line in list-of-lists
```

```
        do
    (make-new-FEATURE-CHART-ENTRY Additional-feature-line PATHNAME)))


(defun check-if-like-flavors-to-collect-the-FEAT-GROUP-provided-CACHED-is-in  (a-string)
  (let ((result nil))
  (with-database  (db  *the-pathname-of-the-ADDITIONAL-FEATURES-R8*)
                  (with-transaction ()
                    (for-each ((ADD Feature-Chart-ENTRY)
                               (:where (string-equal   (Feature-Chart-ENTRY-feat-group ADD)   a-string)))
                      (push ADD result))))
result))


(defun check-if-the-negatives-really-dint-go (ls)
  (loop for entity in ls
        collect
        (Feature-Chart-ENTRY-feature-description entity)))


(defun  make-new-FEATURE-CHART-ENTRY (list PATHNAME)

  (with-database  (db   PATHNAME)
                  (with-transaction ()
                                  (make-Feature-Chart-ENTRY
                                    :FEATURE-DESCRIPTION (put-the-right-INDICATOR ':FEATURE-DESCRIPTION
 list)
                                    :all-the-combinations-restrictions
                                          (put-the-right-INDICATOR ':all-the-combinations-restrictions
list)
                                    :all-the-comments-restrictions
                                          (put-the-right-INDICATOR ':all-the-comments-restrictions list
)
                                    :all-the-territory-restrictions
                                          (put-the-right-INDICATOR ':all-the-territory-restrictions lis
t)
                                    :recno (put-the-right-INDICATOR ':recno list)
                                    :featcode (put-the-right-INDICATOR ':featcode list)
                                    :standard (put-the-right-INDICATOR ':standard list)
                                    :base   (put-the-right-INDICATOR ':base list)
                                    :optional (put-the-right-INDICATOR ':optional list)
                                    :marketing (put-the-right-INDICATOR ':marketing list)
                                    :enveng  (put-the-right-INDICATOR ':enveng list)
                                    :legal (put-the-right-INDICATOR ':legal list)
                                    :accessory (put-the-right-INDICATOR ':accessory list)
                                    :mr   (put-the-right-INDICATOR ':mr list)
                                    :mc   (put-the-right-INDICATOR ':mc list)
                                    :class (put-the-right-INDICATOR ':class list)
                                    :body  (put-the-right-INDICATOR ':body list)
                                    :family  (put-the-right-INDICATOR ':family list)
                                    :capacity (put-the-right-INDICATOR ':capacity list)
                                    :performance (put-the-right-INDICATOR ':performance list)
                                    :drive  (put-the-right-INDICATOR ':drive list)
                                    :trans  (put-the-right-INDICATOR ':trans list)
                                    :auth-ref-in  (put-the-right-INDICATOR ':auth-ref-in list)
                                    :auth-ref-out  (put-the-right-INDICATOR ':auth-ref-out list)
                                    :rel-stage-in (put-the-right-INDICATOR ':rel-stage-in list)
                                    :rel-stage-out (put-the-right-INDICATOR ':rel-stage-out list)
                                    :dep-in   (put-the-right-INDICATOR ':dep-in list)
                                    :dep-out  (put-the-right-INDICATOR ':dep-out list)
                                    :feat-group  (put-the-right-INDICATOR ':feat-group list)
                                    :history  (put-the-right-INDICATOR ':history list)
                                    :errno   (put-the-right-INDICATOR ':errno list)
                                    :all-the-group-restrictions (put-the-right-INDICATOR ':all-the-grou
p-restrictions list)
))))


;(defun put-the-right-INDICATOR    (INDICATOR ls)
;  (second (assoc INDICATOR ls)))

;*****************************************************************************************************
```

```
****************

;;;;;  WE TRY TO   COLLECT  THE   "data-we-concern"    ,,,,,,  actually the  "feature-chart-entries"  .
;;;;;  More specifically to change  the FLAVORS in "feature-chart-entries" with ENTITIES from the equival
ent STATICE data
```

APPENDIX 7

LISTING OF THE
TRUTH
MAINTENANCE
MECHANISM

```
;;;-*-  Mode: lisp;  Package: austin-rover ; syntax: common-lisp-*-


(defflavor  Vehicle-part
        () () )


(defflavor  vpg-code
        (name
         (legal-families  nil))
        (Vehicle-part)
  :writable-instance-variables
  :initable-instance-variables)


(defvar  *vpg-codes*  nil)


(defmethod  (sys:print-self   vpg-code)   (stream  &rest  ignore)
  (print-node  self  stream))

(defmethod (make-instance  vpg-code)   (&rest  ignore)
           (push self *vpg-codes*))


(defmethod  (print-node   vpg-code) (stream)
  (format  stream  "Vpg-code ~A"  name))


(defmethod  (node-inferiors   vpg-code)  ()
  legal-families)

;*************************************************************


(defflavor  family
        (name
         (legal-Primary-Descriptions  nil))
        (Vehicle-part)
  :writable-instance-variables
  :initable-instance-variables)


(defvar  *families*  nil)

(defmethod  (sys:print-self   family)   (stream  &rest  ignore)
  (print-node  self  stream))


(defmethod (make-instance  family)  (&rest  ignore)
           (push self *families*))


(defmethod  (print-node   family) (stream)
  (format  stream  " -A"  name))


(defmethod  (node-inferiors   family)  ()
  legal-Primary-Descriptions)

;*************************************************************


(defflavor  Primary-Description
        (name
         (legal-Secondary-Descriptions  nil))
        (Vehicle-part)
  :writable-instance-variables
  :initable-instance-variables)


(defvar  *Primary-Descriptions*  nil)
```

```
(defmethod  (sys:print-self   Primary-Description)   (stream  &rest  ignore)
  (print-node  self  stream))


(defmethod (make-instance  Primary-Description) (&rest  ignore)
           (push self *Primary-Descriptions*))


(defmethod  (print-node   Primary-Description) (stream)
  (format  stream  " ~A"  name))


(defmethod  (node-inferiors   Primary-Description)  ()
  legal-Secondary-Descriptions)


;*********************************************************


(defflavor  Secondary-Description
        (name
         (legal-Pseudos  nil))
        (Vehicle-part)
  :writable-instance-variables
  :initable-instance-variables)


(defvar  *Secondary-Descriptions*  nil)

(defmethod  (sys:print-self   Secondary-Description)   (stream  &rest  ignore)
  (print-node  self  stream))


(defmethod (make-instance  Secondary-Description)   (&rest  ignore)
           (push self *Secondary-Descriptions*))


(defmethod  (print-node   Secondary-Description) (stream)
  (format  stream  "~A"  name))


(defmethod  (node-inferiors   Secondary-Description)  ()
  legal-Pseudos)


;*********************************************************


;*****************************************************************************************


(defun  graph-tree-for-VPG  (vpg-code &optional   (stream  *standard-output*))
            (fresh-line  stream)
  (format-graph-from-root   vpg-code
                            #'princ
                            #'node-inferiors
                            :dont-draw-duplicates t
                            :orientation  :horizontal
                            :stream  stream))


;(defun  restore-data  ()
;  (loop for c in  *vpg-codes*
;        do
;    (when  (not (boundp-in-instance  c 'legal-families))
;      (setf (vpg-code-legal-families  c)   nil)
;  ))
;  (loop for b in  *families*
;        do
;    (when  (not (boundp-in-instance  b 'legal-Primary-Descriptions))
;      (setf (legal-Primary-Descriptions  b)   nil)
;  ))
```

```
;  (loop for f in  *primary-Descriptions*
;       do
;    (when  (not (boundp-in-instance  f 'legal-Secondary-Descriptions))
;      (setf (legal-Secondary-Descriptions  f)   nil)
;  ))
;  (loop for cap in  *secondary-descriptions*
;       do
;    (when  (not (boundp-in-instance  cap 'legal-pseudos))
;      (setf (legal-pseudos  cap)   nil)
;  ))
;)

;*****************************************************************************************


:Command: (given-a-vpg-string-collect-ALL-families-prim-secon-AND-group-them-as-well "1109AA")
;(("HAM"
;  (("COVER-EXPENDABLE-ST/F " ("COVER-EXPENDABLE-ST/F ELASTICATED    " "COVER-EXPENDABLE-ST/F NON ELESTIC
ATED"))
;   ("CVR-EXPND-HEADRSTNT-FR" ("CVR-EXPND-HEADRSTNT-FRELASTICATED    " "CVR-EXPND-HEADRSTNT-FRNON ELASTIC
ATED"))))
; ("HAJ"
;  (("SQUAB-COMP-SEAT FRONT "
;    ("SQUAB-COMP-SEAT FRONT ELECTRIC-DRIVER" "SQUAB-COMP-SEAT FRONT MANUAL-DRIVER  " "SQUAB-COMP-SEAT FR
ONT MANUAL-PASSR   "
;     "SQUAB-COMP-SEAT FRONT MANUAL          "))))
; ("HAH"
;  (("HEADRSTNT-COMP-FRONT  "
;    ("HEADRSTNT-COMP-FRONT  'A'FRAME       " "HEADRSTNT-COMP-FRONT  'A'FRM INC PADS" "HEADRSTNT-COMP-FRO
NT  BOLSTER-FIXED  "
;     "HEADRSTNT-COMP-FRONT  BOLSTER-ADJL   " "HEADRSTNT-COMP-FRONT  BOLSTER-NODDING"))))
; ("HAG"
;  (("CUSHN-COMP-SEAT FRONT "
;    ("CUSHN-COMP-SEAT FRONT ELECTRIC-DRIVER" "CUSHN-COMP-SEAT FRONT MANUAL-DRIVER  " "CUSHN-COMP-SEAT FR
ONT MANUAL-PASSR   "
;     "CUSHN-COMP-SEAT FRONT MANUAL          "))))
; ("HAF" (("PLATE-PROTECTION      " ("PLATE-PROTECTION      ST/F-TIP CATCH "))))
; ("HAE" (("CROSSBAR ASSY         " ("CROSSBAR ASSY         SIDE INTRUSION "))))
; ("HAD"
;  (("SEAT-FRONT COMPLETE   "
;    ("SEAT-FRONT COMPLETE   MANUAL-DRIVER  " "SEAT-FRONT COMPLETE   MANUAL-PASSR   " "SEAT-FRONT COMPLET
E   ELECTRIC-DRIVER"
;     "SEAT-FRONT COMPLETE   ELECTRIC-PASSR " "SEAT-FRONT COMPLETE   MANUAL         " "SEAT-FRONT COMPLET
E   ELECTRIC        "))))
; ("HAC"
;  (("CHART-TORQUE-ST/F     " ("CHART-TORQUE-ST/F     FIXINGS         "))
;   ("CHART-ARRANGEMENTS    " ("CHART-ARRANGEMENTS    ST/F COMP       "))))
; ("HAB"
;  (("ARRGT DRG-ST/F-MANUAL "
;    ("ARRGT DRG-ST/F-MANUAL FIXED          " "ARRGT DRG-ST/F-MANUAL L/SUPP         " "ARRGT DRG-ST/F-MAN
UAL NO L/SUPP      "))
;   ("ARRGT DRG-ST/F-ELEC   "
;    ("ARRGT DRG-ST/F-ELEC   L/SUPP-MEMORY  " "ARRGT DRG-ST/F-ELEC   L/SUPP-NO MEM  " "ARRGT DRG-ST/F-ELE
C   NO L/SUPP-MEM  "
;     "ARRGT DRG-ST/F-ELEC   NO L/SUPP & MEM"))
;   ("ARRGT DRG-SEAT FRONT  " ("ARRGT DRG-SEAT FRONT  TRIM WIRE/CLIPS"))))
; ("HAA"
;  (("INSTR DRG             "
;    ("INSTR DRG             MAN L/SUPP     " "INSTR DRG             MAN-NO L/SUPP  " "INSTR DRG
   ELEC-MEMORY    "
;     "INSTR DRG             ELEC-NO MEMORY "))
;   ("INSTR DRG-FIXING      " ("INSTR DRG-FIXING      SEAT FRONT     "))))))
;Command:


(defun to-get-the-PART-DESCRIPTION-FROM-A-VPG-graph-tree (a-string &optional   (stream  *standard-output*
))
  (the-top-function-to-draw-the-VPG  a-string stream)
  (let* ((the-input-PART-DESCRIPTION (accept 'secondary-description))
         (the-PART-DESCRIPTION-name (secondary-description-name the-input-PART-DESCRIPTION)))
    (with-database  (db  *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*)
                  (with-transaction ()
                    (let ((the-actual-statice-entity-of-the-PART-DESCRIPTION
```

```
                                (The-PART-DESCRIPTIONs-entity-type the-PART-DESCRIPTION-name)))
                          (print the-actual-statice-entity-of-the-PART-DESCRIPTION))))))


(defun the-top-function-to-draw-the-VPG (a-string &optional   (stream  *standard-output*))
  (setq *vpg-codes* nil)
  (setq *families* nil)
  (setq *primary-descriptions* nil)
  (setq *secondary-descriptions* nil)
  (let ((temporary (given-a-vpg-string-collect-ALL-families-prim-secon-AND-group-them-as-well a-string)))
    (loop for i in temporary
          do
      (linkages i))
    (let ((instance-of-VPG (make-instance 'vpg-code :name a-string)))
      (setf (vpg-code-legal-families instance-of-VPG) *families*)
      (graph-tree-for-VPG  (car *vpg-codes*) stream))))



(defun given-a-vpg-string-collect-ALL-families-prim-secon-AND-group-them-as-well (a-string)
  (let ((families-all nil) (family-with-primaries nil) (family-with-primaries-with-secondaries nil)
        (family-with-primaries-with-secondaries-ONLY-STRINGS nil))
    (with-database  (db  *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*)
                  (with-transaction ()
                    (let ((THE-actual-VPG-entity (car (the-entity-type-of-the-VPG-CODE a-string))))
                      (loop for family-entity in  (Sub-Sub-Sub-Vpg-HAS-Families-within THE-actual-VPG-ent
ity)
                            do
                        (push (list family-entity (FAMILY-within-a-SSS-Vpg-FAMILY-CODE family-entity))
                              families-all))
                      (loop for entity-with-string in families-all
                            do
                        (let* ((primaries-entities (FAMILY-within-a-SSS-Vpg-Has-Primary-Descriptions (car
 entity-with-string)))
                               (primaries-strings (let ((result nil))
                                                    (loop for primary-entity  in  primaries-entities
                                                          do
                                                      (push
                                                        (Primary-Description-ID-PRIMARY-DESCRIPTION-OF-TH
E-PART primary-entity)
                                                        result)) result)))
                          (push (list
                                  (cadr entity-with-string)
                                  primaries-strings
                                  primaries-entities) family-with-primaries)))
                      (loop for Family-strings-Primary-strings-Primary-Entities in family-with-primaries
                            do
                        (let* ((primary-entities (third Family-strings-Primary-strings-Primary-Entities))
                               (secondary-entities (let ((result nil))
                                                     (loop for primary-entity in primary-entities
                                                           do
                                                       (push (Primary-Description-ID-Has-Secondary-Descri
ptions primary-entity)
                                                             result)) result)))
                          (push (list  (car Family-strings-Primary-strings-Primary-Entities)
                                       (cadr Family-strings-Primary-strings-Primary-Entities)
                                       secondary-entities)
                                family-with-primaries-with-secondaries)))
                      (loop for  i   in  family-with-primaries-with-secondaries
                            do
                        (let* ((secondaries-entities (third i))
                               (secondary-string-all (let ((result nil))
                                                       (loop for list-of-SECONDARIES in secondaries-entit
ies
                                                             do
                                                             (push (do-the-strings-from-secondaries list-
of-SECONDARIES)
                                                                   result)) result)))
                          (push (list
                                  (car i) (second i) secondary-string-all) family-with-primaries-with-sec
ondaries-ONLY-STRINGS))))))
    (afterwards-you-group-the-primaries-with-secondaries family-with-primaries-with-secondaries-ONLY-STRI
NGS)))
```

```
(defun do-the-strings-from-secondaries (list-of-secondaries)
;(#<Entity-Handle of SECONDARY-DESCRIPTION-ID 35/1 43265733> #<Entity-Handle of SECONDARY-DESCRIPTION-ID
35/2 43265152>
;    #<Entity-Handle of SECONDARY-DESCRIPTION-ID 36/1 43265137> #<Entity-Handle of SECONDARY-DESCRIPTION-
ID 36/2 43265124>
;    #<Entity-Handle of SECONDARY-DESCRIPTION-ID 37/1 43265111>)
  (let ((result nil))
    (with-database  (db  *ROVERs-APN-VPG-CATALOGUE-only-SEATS-and-MIRRORS-pathname*)
      (with-transaction ()
        (loop for  secondary  in  list-of-secondaries
              do
          (push (SECONDARY-DESCRIPTION-ID-PART-DESCRIPTION  secondary) result))))
result))



;(("HAA" ("INSTR DRG-FIXING       " "INSTR DRG              ")
;  (("INSTR DRG              MAN L/SUPP     " "INSTR DRG              MAN-NO L/SUPP  " "INSTR DRG
   ELEC-MEMORY    "
;    "INSTR DRG              ELEC-NO MEMORY ")
;   ("INSTR DRG-FIXING       SEAT FRONT     "))))
; ("HAB" ("ARRGT DRG-SEAT FRONT  " "ARRGT DRG-ST/F-ELEC   " "ARRGT DRG-ST/F-MANUAL ")
;  (("ARRGT DRG-ST/F-MANUAL FIXED          " "ARRGT DRG-ST/F-MANUAL L/SUPP          " "ARRGT DRG-ST/F-MANU
AL NO L/SUPP      ")
;   ("ARRGT DRG-ST/F-ELEC   L/SUPP-MEMORY  " "ARRGT DRG-ST/F-ELEC   L/SUPP-NO MEM  " "ARRGT DRG-ST/F-ELEC
   NO L/SUPP-MEM  "
;    "ARRGT DRG-ST/F-ELEC   NO L/SUPP & MEM")
;   ("ARRGT DRG-SEAT FRONT  TRIM WIRE/CLIPS")))
; ("HAC" ("CHART-ARRANGEMENTS     " "CHART-TORQUE-ST/F     ")
;  (("CHART-TORQUE-ST/F     FIXINGS        ") ("CHART-ARRANGEMENTS    ST/F COMP      ")))
; ("HAD" ("SEAT-FRONT COMPLETE   ")
;  (("SEAT-FRONT COMPLETE   MANUAL-DRIVER  " "SEAT-FRONT COMPLETE   MANUAL-PASSR   " "SEAT-FRONT COMPLETE
   ELECTRIC-DRIVER"
;    "SEAT-FRONT COMPLETE   ELECTRIC-PASSR " "SEAT-FRONT COMPLETE   MANUAL          " "SEAT-FRONT COMPLETE
   ELECTRIC       ")))
; ("HAE" ("CROSSBAR ASSY          ") (("CROSSBAR ASSY          SIDE INTRUSION ")))
; ("HAF" ("PLATE-PROTECTION       ") (("PLATE-PROTECTION       ST/F-TIP CATCH ")))
; ("HAG" ("CUSHN-COMP-SEAT FRONT ")
;  (("CUSHN-COMP-SEAT FRONT ELECTRIC-DRIVER" "CUSHN-COMP-SEAT FRONT MANUAL-DRIVER  " "CUSHN-COMP-SEAT FRO
NT MANUAL-PASSR   "
;    "CUSHN-COMP-SEAT FRONT MANUAL          ")))
; ("HAH" ("HEADRSTNT-COMP-FRONT  ")
;  (("HEADRSTNT-COMP-FRONT  'A'FRAME        " "HEADRSTNT-COMP-FRONT  'A'FRM INC PADS" "HEADRSTNT-COMP-FRON
T  BOLSTER-FIXED  "
;    "HEADRSTNT-COMP-FRONT  BOLSTER-ADJL   " "HEADRSTNT-COMP-FRONT  BOLSTER-NODDING")))
; ("HAJ" ("SQUAB-COMP-SEAT FRONT ")
;  (("SQUAB-COMP-SEAT FRONT ELECTRIC-DRIVER" "SQUAB-COMP-SEAT FRONT MANUAL-DRIVER  " "SQUAB-COMP-SEAT FRO
NT MANUAL-PASSR   "
;    "SQUAB-COMP-SEAT FRONT MANUAL          ")))
; ("HAM" ("CVR-EXPND-HEADRSTNT-FR" "COVER-EXPENDABLE-ST/F ")
;  (("COVER-EXPENDABLE-ST/F ELASTICATED    " "COVER-EXPENDABLE-ST/F NON ELESTICATED")
;   ("CVR-EXPND-HEADRSTNT-FRELASTICATED    " "CVR-EXPND-HEADRSTNT-FRNON ELASTICATED"))))


(defun   afterwards-you-group-the-primaries-with-secondaries  (ls) ;;ie  The list  ABOVE.
  (let ((result nil))
    (loop for i in ls
          do
      (let  ((primaries (second i))
             (secondaries (third i)))
        (push (list (car i) (connect-the-secondaries-with-the-primaries primaries secondaries))
              result)))
    result))


;Command: (connect-the-secondaries-with-the-primaries '("INSTR DRG-FIXING       " "INSTR DRG              "
; '(("INSTR DRG
;   MAN L/SUPP      " "INSTR DRG              MAN-NO L/SUPP  "
;  "INSTR DRG              ELEC-MEMORY     " "INSTR DRG              ELEC-NO MEMORY ")
; ("INSTR DRG-FIXING       SEAT FRONT      "))
;(("INSTR DRG              "
;  ("INSTR DRG              MAN L/SUPP      " "INSTR DRG              MAN-NO L/SUPP  " "INSTR DRG
```

```
  ELEC-MEMORY     "
;   "INSTR DRG                ELEC-NO MEMORY "))
; ("INSTR DRG-FIXING       " ("INSTR DRG-FIXING      SEAT FRONT      ")))
;Command:


(defun connect-the-secondaries-with-the-primaries (ls1  ls2)
;  ls1  ===  ("INSTR DRG-FIXING       " "INSTR DRG              ")
;  ls2  ===  (("INSTR DRG              MAN L/SUPP     " "INSTR DRG              MAN-NO L/SUPP  " "INSTR DRG
            ELEC-MEMORY     "
;    "INSTR DRG               ELEC-NO MEMORY ")
;   ("INSTR DRG-FIXING       SEAT FRONT      "))
  (let ((result nil))
    (loop for i in ls1
          do
      (loop for j in ls2
            do
        (let ((key (substring (car j) 0 22)))
          (cond ((string-equal i  key)
                 (push (list i j) result))))))
    result))

;;;*****************************************************************************************************
*****************
;;;;;;;;   NOW WE START TO MAKE THE   FLAVOR  LINKS
;;;;;;;;;;;       One list of the result of the function :
;          (afterwards-you-group-the-primaries-with-secondaries (GIVEN-A-VPG-STRING-COLLECT-ALL-FAMILIES-P
RIM-SECON-AND-GROUP-THEM-
;AS-WELL "1109AA"))
;ls  =======
;("HAM"
;  (("COVER-EXPENDABLE-ST/F " ("COVER-EXPENDABLE-ST/F ELASTICATED     " "COVER-EXPENDABLE-ST/F NON ELESTIC
ATED"))
;   ("CVR-EXPND-HEADRSTNT-FR" ("CVR-EXPND-HEADRSTNT-FRELASTICATED     " "CVR-EXPND-HEADRSTNT-FRNON ELASTIC
ATED"))))
;

(defun   linkages  (ls)    ;;;;;; AS  ABOVE
  (let ((primary-with-secondaries-ALL  (second ls)))
    (loop for specific-primary-with-secondaries  in primary-with-secondaries-ALL
          do
      (let ((primary  (first specific-primary-with-secondaries))
            (secondaries (second specific-primary-with-secondaries)))
        (loop for secondary in secondaries
              do
          (make-instance  'Secondary-Description :name secondary))
        (let ((instance-of-PRIMARY (make-instance 'Primary-Description :name primary)))
          (setf (Primary-Description-legal-Secondary-Descriptions  instance-of-PRIMARY)  *Secondary-Descr
iptions*)
          (setq *Secondary-Descriptions* nil))))
    (let ((instance-of-FAMILY (make-instance 'family :name (car ls))))
      (setf (family-legal-Primary-Descriptions  instance-of-FAMILY ) *Primary-Descriptions*)))
  (setq *Primary-Descriptions* nil))
```

```
;;; -*- Syntax: Common-lisp; Package: austin-rover; Base: 10; Mode: LISP -*-


;(define-ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT-command
;  (com-show-ALL-THE-TRUTH-MAINTENANCE-TREE---- :menu-accelerator t) ()
;   (let ((stream (dw:get-program-pane 'pane-3)))
;     (send stream :clear-history)
;     (create-all-tree-and-graph-it stream)))



(defflavor  takis
        (name
         (feature-combinations  nil))
        (Feature-combination)
  :writable-instance-variables
  :initable-instance-variables)



(defmethod (make-instance  takis)   (&rest  ignore)
           (push self *ROOTS*))

(defmethod  (sys:print-self  takis)   (stream  &rest  ignore)
  (print-node  self  stream))



(defmethod  (print-node   takis ) (stream)
  (format  stream  "ALL  TRUTH MAINTENACE TREE FOR MODEL : ~A" name ))

(defmethod  (node-inferiors   takis)  ()
  feature-combinations)



(defun create-all-tree-and-graph-it (&optional (stream *standard-output*))
  (setq *feature-combinations-for-DRAW* nil)
  (let ((help nil)
        (temp
          (loop for i in *choices*
                collect
                  (car i))))
          ;;;;;ie.. ((("B51R+B44R+B69D") ("B51R+B44R-B69D")) (("B51L+B44R+B69D") ("B51L+B44R-B69D")) (("B
51K+B44B")))
    (loop for i in temp
          do
      (push (one-or-two i) help))
    (setq *ADD-FEAT-COMBINATIONs* nil)
      (loop for i in help
            do
        (make-instance 'ADD-FEAT-COMBINATION :name i :feature-combinations nil))
      (setq *ROOTS* nil)
      (make-instance 'takis :name *model* :feature-combinations *ADD-FEAT-COMBINATIONs*)
        ;;;;(graph-tree-ROOT root stream)

         (loop for i in *ADD-FEAT-COMBINATIONs*
               do

        (let ((variable (ADD-FEAT-COMBINATION-name i)))
                (cond ((check-if-tree-of-a-double-combination variable)
                       (do-the-draw-tree-for-a-double-combination-draw-all-tree  variable stream
i))

                (t
              (let* ((variable-which-keeps-both-flavor-names-and-tree-structure
                  (create-the-tree-of-the-string variable))    ;;;;;;;;reference   ####
                 (flavor-names (car variable-which-keeps-both-flavor-names-and-tree-structure))
                 (tree-structure (cadr variable-which-keeps-both-flavor-names-and-tree-structure)))
            (let ((table-of-flavor-names-with-their-equivalent-lists nil) (actual-flavors+lists nil))
              (loop for i in flavor-names
                    do
                (push (list (make-names-appending-strings i) i) table-of-flavor-names-with-their-equivale
nt-lists))
; (("B51L" "B51L") ("B44R" "B44R") ("B69D" "B69D") ("B51L,B44R,B69D" ("B51L" "B44R" "B69D")) ("B51L,B44R"
 ("B51L" "B44R"))
```

```
; ("B51L,B69D" ("B51L" "B69D")))
            (loop for i in table-of-flavor-names-with-their-equivalent-lists
                  do
              (push (list (make-instance 'Usage-statement :name (car i))
                          (cond ((atom (cadr i)) (list (cadr i)))
                                (t (cadr i))))
                    actual-flavors+lists))
            ;;;;;;(print actual-flavors+lists)  ;;;;;reference &&&&&&&
            (loop for i in tree-structure  ;;  i === (("B51L" "B44R") ("B51L") ("B44R"))
                  do
              (setf (Usage-statement-feature-combinations (return-the-flavor-of-the-list (car i) actual-f
lavors+lists))
                    (list (return-the-flavor-of-the-list (second i) actual-flavors+lists)
                          (return-the-flavor-of-the-list (third i) actual-flavors+lists))))
            (let* ((final (loop for i in tree-structure
                                collect (list  (car i) (length (car i)))))
                   (root-takis (return-the-flavor-of-the-list
                                (caar (sort final #'> :key #'second)) actual-flavors+lists)))

              (setf  (ADD-FEAT-COMBINATION-feature-combinations i) (list root-takis))
)))
))
))

(draw-nothings-as-well (car *roots*) stream)
 (graph-tree-ROOT-vertical  (car *roots*) stream)
))



(defun draw-nothings-as-well (the-top-root  &optional (stream *standard-output*))
  (setq *feature-combinations-for-DRAW* nil)
  (let* ((help1 (collect-all-nothings-for-draw)) (help2 nil)
;(("NOTHING" "B51L" "B44B" "NOTHING") ("NOTHING" "B51R" "B44B" "NOTHING") ("NOTHING" "B51K" "B44R" "NOTHI
NG")
; ("NOTHING" "B51K" "B69D" "NOTHING"))
         (help nil))
    (loop for i in help1
          do
              (push (list (string-append (second i) "," (third i))) help2))

    (setq *ADD-FEAT-COMBINATIONs* nil)
    (let (
          (root (make-instance 'ADD-FEAT-COMBINATION :name "THE BELOW FEATURE COMBINATIONS DO NOT MATCH"
:feature-combinations *ADD-FEAT-COMBINATIONs*)))
      (setq *ADD-FEAT-COMBINATIONs* nil)

      (loop for i in help2
            do
        (make-instance 'ADD-FEAT-COMBINATION :name (car i) :feature-combinations nil))


        (loop for i in *ADD-FEAT-COMBINATIONs*
              do
        (let ((variable (ADD-FEAT-COMBINATION-name  i)))

                      (do-the-draw-tree-for-a-double-combination-draw-all-tree  variable stream
i)))
        (setf (ADD-FEAT-COMBINATION-feature-combinations root) *ADD-FEAT-COMBINATIONs*)

        (push root (takis-feature-combinations the-top-root)))))


;;;;;*****************************************************************************************************
************************


(defun do-the-draw-tree-for-a-double-combination-draw-all-tree (feature-string &optional (stream *standar
d-output*) i)
  (setq *feature-combinations-for-DRAW* nil)
  (let ((takis
  (cond ((string-search "-OR-" feature-string)
         (let* ((combination (car (get-words-from-string-- feature-string))) ;;;ie ("B51L+B44R" "OR" "B51
L-B44R")
```

```
                (final (get-words-from-string combination)))
            (make-names-appending-strings final)))
        (t (make-names-appending-strings (get-words-from-string feature-string))))))
    (let (
;    (tree-structure (list (get-words-from-string-comma takis)
;                          (list (substring takis 0 4))
;                          (list (substring takis 5 9))))
          (flavor-names (list  (substring takis 0 4) (substring takis 5 9)))
          )
      (loop for i in flavor-names
            do
        (make-instance 'Usage-statement :name i))
      (let ((help  *feature-combinations-for-DRAW*))
        (let ((temp (make-instance 'Usage-statement :name takis :feature-combinations help)))
          (setf  (ADD-FEAT-COMBINATION-feature-combinations i) (list temp))
          )))))


(defun  graph-tree-ROOT-vertical  (root &optional   (stream  *standard-output*))
            (fresh-line  stream)
  (format-graph-from-root   root
                            #'princ
                            #'node-inferiors
                            :dont-draw-duplicates t
                            :orientation :vertical
                            :stream  stream
;                           :row-spacing 10
;                           :within-row-spacing 20
))
```

```
;;; -*- Syntax: Common-lisp; Package: austin-rover; Base: 10; Mode: LISP -*-


(DW:DEFINE-PROGRAM-FRAMEWORK ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT
  :SELECT-KEY
  #\square
  :COMMAND-DEFINER
  T
  :COMMAND-TABLE
  (:INHERIT-FROM '("colon full command" "standard arguments" "input editor compatibility")
   :KBD-ACCELERATOR-P 'NIL)
  :STATE-VARIABLES
  NIL
  :PANES
  ((PANE-2 :TITLE :HEIGHT-IN-LINES 1 :REDISPLAY-AFTER-COMMANDS NIL)
   (PANE-3 :DISPLAY
           :margin-components *FRAMEWORK-DISPLAY-margin-components*)

   (PANE-4 :COMMAND-MENU :MENU-LEVEL :TOP-LEVEL) (PANE-5 :INTERACTOR))
                                                        ;:margin-components *FRAMEWORK-DISPLAY-margin-co
mponents*
                                                        ;:HEIGHT-IN-LINES 4))
  :CONFIGURATIONS
  '((DW::MAIN (:LAYOUT (DW::MAIN :COLUMN PANE-2 PANE-3 PANE-4 PANE-5))
     (:SIZES
      (DW::MAIN (PANE-2 1 :LINES)  ;;;(PANE-4 2 :lines)
                (PANE-4 :ASK-WINDOW SELF :SIZE-FOR-PANE PANE-4)
       (PANE-5 1 :LINES) :THEN (PANE-3 :EVEN))))))


(define-ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT-command
  (com-show-choices--- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-3))) ;;;(temp (finish-it-with-GRAPHICS *COMMON-GROUND-RECORD*
)))
     (send stream :clear-history)
     ;;;;;(setq *CHOICES* temp)
     (let* ((help (assign-the-usage-statement-with-numbers *CHOICES*))
            (other-help  (loop for i in help
                                        collect
                                          (list (car i)))))
     (print (format nil "THE SYSTEM HAS FOUND ~D CHOICES FOR THE PART SPECIFICATION ..ie ~D  different US
AGE STATEMENTS OF THE PART" (length *CHOICES*) (length *CHOICES*)) stream)

;     (loop for i in other-help
;          do
;       (print i stream)))
;     (terpri stream)))
     (nested-formatting-2-FEATURE-COMBINATIONS other-help stream))))









(defun nested-formatting-2-FEATURE-COMBINATIONS (tree &optional (stream *standard-output*))
  (fresh-line)
  (format-graph-from-root tree
                          #'(lambda (node stream)
                              (if (consp (first node))
                                  (with-character-size (:large stream :bind-line-height t)
                                    (princ "VALID-FEATURE-COMBINATIONS-FOUND-FROM-THE-SYSTEM" stream))
                                  (format-item-list node
                                                    :stream stream
                                                    :optimal-number-of-rows 2
                                                    :fresh-line nil
                                                    :additional-indentation 0)))
                          #'(lambda (node)
                              (and (consp (first node)) node))
                          :stream stream))


(defun nested-formatting-2-MUSTS (tree &optional (stream *standard-output*))
  (fresh-line)
```

```
  (format-graph-from-root tree
                          #'(lambda (node stream)
                              (if (consp (first node))
                                  (with-character-size (:large stream :bind-line-height t)
                                    (princ "MUST-FEATURES" stream))
                                  (format-item-list node
                                                    :stream stream
                                                    :optimal-number-of-rows 2
                                                    :fresh-line nil
                                                    :additional-indentation 0)))
                          #'(lambda (node)
                              (and (consp (first node)) node))
                          :stream stream))


(defun nested-formatting-2-MAYBES (tree &optional (stream *standard-output*))
  (fresh-line)
  (format-graph-from-root tree
                          #'(lambda (node stream)
                              (if (consp (first node))
                                  (with-character-size (:large stream :bind-line-height t)
                                    (princ "MAYBE(optional)-FEATURES" stream))
                                  (format-item-list node
                                                    :stream stream
                                                    :optimal-number-of-rows 2
                                                    :fresh-line nil
                                                    :additional-indentation 0)))
                          #'(lambda (node)
                              (and (consp (first node)) node))
                          :stream stream))









(defun nested-formatting-2-NEGATIVES (tree &optional (stream *standard-output*))
  (fresh-line)
  (format-graph-from-root tree
                          #'(lambda (node stream)
                              (if (consp (first node))
                                  (with-character-size (:large stream :bind-line-height t)
                                    (princ "FEATURES-NEGATIVE-TO-THE-PART" stream))
                                  (format-item-list node
                                                    :stream stream
                                                    :optimal-number-of-rows 2
                                                    :fresh-line nil
                                                    :additional-indentation 0)))
                          #'(lambda (node)
                              (and (consp (first node)) node))
                          :stream stream))









(defun nested-formatting-2 (tree &optional (stream *standard-output*))
  (fresh-line)
  (format-graph-from-root tree
                          #'(lambda (node stream)
                              (if (consp (first node))
                                  (with-character-size (:large stream :bind-line-height t)
                                    (princ "*" stream))
                                  (format-item-list node
                                                    :stream stream
                                                    :optimal-number-of-rows 2
```

```
                                                    :fresh-line nil
                                                    :additional-indentation 0)))
                                  #'(lambda (node)
                                      (and (consp (first node)) node))
                                  :stream stream))


(define-ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT-command
  (com-CLEAR-THE-DISPLAY--- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-3)))

     (send stream :clear-history)))


;(define-ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT-command
;  (com-GIVE-VALID-SPECS-in-ENGLISH-with-GRAPH--- :menu-accelerator t) ()
;   (let ((stream (dw:get-program-pane 'pane-3)) (temp (finish-it-with-GRAPHICS *COMMON-GROUND-RECORD*)))
;     (loop for i in temp
;          do
;       (send stream :clear-history)
;       (nested-formating i stream))))




(define-ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT-command
  (com-show-ALL-THE-TRUTH-MAINTENANCE-TREE---- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-3)))
     (send stream :clear-history)
     (create-all-tree-and-graph-it stream)))


(define-ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT-command
  (com-GIVE-VALID-SPECS-in-ENGLISH-and-GRAPH-FOR-A-SPECIFIC-USAGE-STATEMENT---- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-3)) (temp (assign-the-usage-statement-with-numbers *CHOICES*)
))
     ;;;temp =
        ;;  (("B51K+B44B" 4) ("B51F+B44R+B69D~OR~B51F+B44R-B69D" 3) ("B51L+B44R+B69D~OR~B51L+B44R-B69D" 2
)
        ;;   ("B51R+B44R+B69D~OR~B51R+B44R-B69D" 1))
     (send stream :clear-history)
     (let ((help (QUERY-for-the-GRAPH (loop for i in temp
                                          collect
                                            (car i))  stream)))
       (let ((number-found (BLOCK return-number
                             (loop for i in temp
                                   do
                               (cond ((string-equal (car i) help)
                                      (return-from return-number (cadr i)))))
                             nil)))
         (nested-formating  (find-the-choice  number-found  *choices*) stream)
))))


(defun QUERY-for-the-GRAPH ( list &optional (stream *standard-output*))
  ;(let ((stream (dw:get-program-pane 'pane-3)))
  (let ((answers nil))
    (push
      ;;(multiple-value-list
        (dw:accepting-values (stream :own-window t :label
                                     "THE BELOW USAGE STATEMENTS REPRESENT THE ALL POSSIBLE VALID COMBINATIO
NS THE SYSTEM FOUND !")
          (values
            (accept '(alist-member :alist ,list) :prompt "PLEASE CHOOSE THE ONE YOU WANT TO  BE DISP
LAYED."

                    :stream stream )

          )
        )
      answers)
    (car answers) ;;;;(list (car answers) (set-difference list answers :test #'string-equal))
  )
)
```

```
(defun assign-the-usage-statement-with-numbers (ls)    ;;;;*CHOICES*
  (let ((result nil) (record 0))
    (loop for i in ls
          do
      (setq record (+ 1 record))
      (push (list (one-or-two (car i)) record) result))
result))


(defun one-or-two (ls)
  (let ((result ""))
    (cond ((equal (length ls) 1) (setq result (caar ls)))
          (t (loop for j in ls
                   do
               (setq result (string-append result (car j) "~OR~")))))
  (cond ((equal (length ls) 1) result)
        (t (substring result 0 (- (string-length result) 4)) ;;;; take out the last "~OR~"
))))


(defvar *CHOICES*  nil)



;(define-ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT-command
;  (com-ASK-FOR-CHOICE-TO-BE-GRAPHED---- :menu-accelerator t) ()
;   (let ((stream (dw:get-program-pane 'pane-3))


(defun find-the-choice (number ls)
  (case number
    (1 (first ls))
    (2 (second ls))
    (3 (third ls))
    (4 (fourth ls))
    (5 (fifth ls))
    (6 (sixth ls))
    (7 (seventh ls))
    (8 (eighth ls))
    (9 (ninth ls))
    (10 (tenth ls))))


;;;;;*****************************************************************************************************
********************

;Command:          *COMMON-GROUND-RECORD*            ;;;;RUNNING B51 B44 B69 for     R8
;(("PLUS" "B51K" "B44B"
;  (((("M" " " " " " " " " " " " " " ")
;     ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "
T53G" "T54C" "T56J"
;     "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E" "
T59G" "T51R" "T51V"
;     "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R" "
T56L" "T58C" "T60R"
;     "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T" "
T56U" "T58N" "T57J"
;     "T58J")
;    (("+" "B53B")))
;   ((("S" " " " " " " " " " " " " " ")
;     ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E" "
T53G" "T54C" "T56J"
;     "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E" "
T59G" "T51R" "T51V"
;     "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R" "
T56L" "T58C" "T60R"
;     "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T" "
T56U" "T58N" "T57J"
;     "T58J")
;    (("+" "B53B")))))
; ("PLUS-AND-MINUS,PLUS" "B51F" "B69D,B44R"
;  ((("Y" " " " " " " " "D" "B" " ") ("T52J" "T53N" "T50R" "T57E" "T59E" "T59G") (("+" "B53B")))))
```

```
; ("PLUS-AND-MINUS,PLUS" "B51L" "B69D,B44R"
;  (((("Y" " " " " " " " "D" "B" " ") ("T52J" "T53N" "T50R" "T57E" "T59E" "T59G") (("+" "B53B")))))
; ("PLUS-AND-MINUS,PLUS" "B51R" "B69D,B44R"
;  (((("X" "N" " " " " " " " "B" " ") ("T52J" "T53N" "T50R" "T57E" "T59E" "T59G") (("+" "B53A")))))
; ("PLUS-AND-MINUS,PLUS" "B51R" "B69D,B44R"
;  (((("X" "W" " " " " " " " "B" " ") ("T52J" "T53N" "T50R" "T57E" "T59E" "T59G") (("+" "B53A"))))))
;Command:


(defun finish-it-with-GRAPHICS (ls  &optional (stream *standard-output*))          ;;;;;;;;*COMMON-GROUN
D-RECORD*)
  (let ((help   (PIMS-CODE ls)) (result nil) (result-final nil))
    (loop for pim-final-with-base-and-countries  in help
          do
      ;;;;;;;(print pim-final-with-base-and-countries  stream)

      (let  ((PIMS-STATEMENTS (correct (car pim-final-with-base-and-countries)))
             (TERRITORIES (cdr  pim-final-with-base-and-countries))
            )
;       (print 'PIMS-STATEMENTS) (print PIMS-STATEMENTS)
;       (print 'TERRITORIES) (print TERRITORIES)
;       (print '*****************************************************************)
;
        (push (list PIMS-STATEMENTS
                    (find-the-rest-going-with-this-PIMS-STATEMENT TERRITORIES stream))
              result)
;       (print 'takis) (print (loop for i in TERRITORIES
;                                          collect
;                                          (find-the-rest-going-with-this-PIMS-STATEMENT TERRITORIES s
tream)))


        ;;;(print 'result)(print result)   ;;;;;;
        ))
    ;;;;result))

    (loop for i in result
          do
;      (print i)
;      (let ((temp (length i)))
;       (cond
;         ((equal temp 1) (print 1);;;;; ONLY ONE LIST ie NOT  OR
;            (push (list (append-all-strings-of-list (car i))
;                        (second i)) result-final))
;         (t (push (list (append-all-strings-of-list-ALL (car i))
;                        (second i)) result-final)))))
;result-final))



      (let ((temp   (car i)))
        (cond ((atom (car temp))
                 (push (list (list (append-all-strings-of-list temp))
                             (cadr i)) result-final))
              (t
               (push (list (append-all-strings-of-list-ALL temp)
                           (cadr i)) result-final))
        )
      ))
    result-final))



(defun append-all-strings-of-list-ALL (ls)
  (let ((result nil))
    (loop for ls-of-strings in ls
          do
      (push (append-all-strings-of-list   ls-of-strings) result))
    result))


(defun append-all-strings-of-list (ls)
  (let ((result ""))
```

```
    (loop for i in ls
          do
      (setq result (string-append result i)))
    (list result)))



(defun find-the-rest-going-with-this-PIMS-STATEMENT (TERRITORIES &optional (stream *standard-output*))
  (let ((result nil))
        (loop for  COMMON-GROUND-before  in  (car TERRITORIES)
              do
;         (print COMMON-GROUND-before)
;         (print '********)
          (let* ((COMMON-GROUND (car COMMON-GROUND-before))
                 (base-statement (translate-base-to-english COMMON-GROUND)) ;; (car COMMON-GROUND)))
                 (countries (translate-to-english-COUNTRIES  (second COMMON-GROUND-before)))
                 (restrictions (TRANSLATE-TO-English-COMBINATIONS-RESTRICTIONS
                                 (caddr COMMON-GROUND-before))))
            (push (list base-statement countries restrictions) result)))
result))


;(defun nested-formating (&optional (tree '(((1 2 3 4) ((5 6 7))) ((9 10 11)))))
;  (fresh-line)
;  (format-graph-from-root tree
;
;                          #'(lambda (node stream)
;                              (if (consp (first node))
;                                  (with-character-size (:large stream :bind-line-height  t)
;                                    (princ "*" stream))
;                                  (format-item-list node
;                                                    :stream stream
;                                                    :optimal-number-of-rows  2
;                                                    :fresh-line nil
;                                                    :additional-indentation 0
;                                                    ;;;;; :orientation  :horizontal
;                                                    )))
;                          #'(lambda (node)
;                              (and (consp (first node)) node))
;                          :orientation  :horizontal))


(defun nested-formating ( tree     ;;;;;;;;&optional (tree '(((1 2 3 4) ((5 6 7))) ((9 10 11))))
                          &optional (stream *standard-output*))  ;;;;;   ls =  takis

  (fresh-line)
  (formatting-table (stream)
    (formatting-column-headings (stream)
      (loop for i in tree for j from 1 to 2
            do
        (with-character-style ('(nil :bold :large) t :bind-line-height t)
          (formatting-cell (stream  :align-x :center  :align-y :center )
            (format t "Graph #~D" j)))))
    (formatting-row (stream)
      (let ((record 0))
      (dolist (orientation '(:horizontal :horizontal))
        (setq record (+ 1 record))
        (formatting-cell (stream :align-x :center :align-y :center  )

          (cond ((equal record 1)

  (format-graph-from-root (car tree)

                          #'(lambda (node stream)
                              (if (consp (first node))
                                  (with-character-size (:large stream :bind-line-height  t)
                                    (princ "FEAT-COMBINATIONS" stream))
                                  (format-item-list node
                                                    :stream stream
                                                    :optimal-number-of-rows  2
                                                    :fresh-line nil
                                                    :additional-indentation 0
```

```
                                                    )))
                                #'(lambda (node)
                                     (and (consp (first node)) node))
                                :orientation  orientation
                                :stream stream
                                :COLUMN-SPACING  15
                                :within-column-spacing 15)
                ) ;;;;
  (t
 (format-graph-from-root (cadr tree)

                                #'(lambda (node stream)
                                     (if (consp (first node))
                                         (with-character-size (:large stream :bind-line-height  t)
                                           (princ   "RESTRICTIONS-PACK" stream))
                                         (format-item-list node
                                                    :stream stream
                                                    :optimal-number-of-rows  2
                                                    :fresh-line nil
                                                    :additional-indentation 0

                                                    )))
                                #'(lambda (node)
                                     (and (consp (first node)) node))
                                :orientation  orientation
                                :stream stream
                                :COLUMN-SPACING  15
                                :within-column-spacing 15)      )) ;;;;;cond
))))))


;;;*****************************************************************************************


;Command: (finish-it-with-GRAPHICS takis)
;((((("B51L+B44R"))
;  ((((CLASS** "TRIM LEVEL 7") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** NO-RESTRICTED) (TRANSMISION** NO-RESTRICTED))
;     ("SPAIN" "PORTUGAL" "GREECE" "GIBRALTAR" "CANARY ISLES") (("W I T H" "B53-SEAT SIDE ROLLS"))))
;   (((CLASS** "TRIM LEVEL 6") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** NO-RESTRICTED) (TRANSMISION** NO-RESTRICTED))
;     ("SOUTH AFRICA" "PAKISTAN" "SRI LANKA" "NEPAL" "INDIA" "GUYANA" "FIJI" "BANGLADESH" "ST. KITTS NEVIS
" "SINGAPORE" "MALAYSIA"
;      "JAMAICA" "HONG KONG" "CYPRUS" "BERMUDA" "BARBADOS" "INDONESIA" "MALAWI" "ZAMBIA" "SEYCHELLES" "MAU
RITIUS" "KENYA"
;      "GRENADA" "BRUNEI" "JAPAN" "ISRAEL" "AUSTRALIA" "SAUDI ARABIA" "QATAR" "OMAN" "KUWAIT" "DUBAI" "BAH
RAIN" "ABU DHABI" "USA"
;      "CANADA" "CALIFORNIA" "SWITZERLAND" "SWEDEN" "NORWAY" "FINLAND" "AUSTRIA" "GERMANY" "SPAIN" "PORTUG
AL" "GREECE" "GIBRALTAR"
;      "CANARY ISLES" "TUNISIA" "TAHITI" "REUNION" "NEW CALEDONIA" "MARTINIQUE" "GUADELOUPE" "FRENCH GUIAN
A" "FRANCE" "YUGOSLAVIA"
;      "ITALY" "TAIWAN" "NETHERLANDS" "ICELAND" "DENMARK" "BELGIUM" "NEW ZEALAND" "MALTA" "GIBRALTAR-RHD"
"GERMANY-RHD"
;      "UNITED KINGDOM" "IRISH REPUBLIC")
;     (("W I T H" "B53-SEAT SIDE ROLLS"))))))
; ((("B51R+B44R"))
;  ((((CLASS** "TRIM LEVEL 51") (BODY** "3 DOOR SALOON") (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICT
ED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** NO-RESTRICTED) (TRANSMISION** NO-RESTRICTED))
;     ("SOUTH AFRICA" "PAKISTAN" "SRI LANKA" "NEPAL" "INDIA" "GUYANA" "FIJI" "BANGLADESH" "ST. KITTS NEVIS
" "SINGAPORE" "MALAYSIA"
;      "JAMAICA" "HONG KONG" "CYPRUS" "BERMUDA" "BARBADOS" "INDONESIA" "MALAWI" "ZAMBIA" "SEYCHELLES" "MAU
RITIUS" "KENYA"
;      "GRENADA" "BRUNEI" "JAPAN" "ISRAEL" "AUSTRALIA" "SAUDI ARABIA" "QATAR" "OMAN" "KUWAIT" "DUBAI" "BAH
RAIN" "ABU DHABI" "USA"
;      "CANADA" "CALIFORNIA" "SWITZERLAND" "SWEDEN" "NORWAY" "FINLAND" "AUSTRIA" "GERMANY" "SPAIN" "PORTUG
AL" "GREECE" "GIBRALTAR"
;      "CANARY ISLES" "TUNISIA" "TAHITI" "REUNION" "NEW CALEDONIA" "MARTINIQUE" "GUADELOUPE" "FRENCH GUIAN
A" "FRANCE" "YUGOSLAVIA"
;      "ITALY" "TAIWAN" "NETHERLANDS" "ICELAND" "DENMARK" "BELGIUM" "NEW ZEALAND" "MALTA" "GIBRALTAR-RHD"
"GERMANY-RHD"
;      "UNITED KINGDOM" "IRISH REPUBLIC")
;     (("W I T H" "B53-SEAT SIDE ROLLS"))))
;    (((CLASS** "TRIM LEVEL 51") (BODY** "5 DOOR SALOON") (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICT
```

```
ED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** NO-RESTRICTED) (TRANSMISION** NO-RESTRICTED))
;     ("SOUTH AFRICA" "PAKISTAN" "SRI LANKA" "NEPAL" "INDIA" "GUYANA" "FIJI" "BANGLADESH" "ST. KITTS NEVIS
" "SINGAPORE" "MALAYSIA"
;      "JAMAICA" "HONG KONG" "CYPRUS" "BERMUDA" "BARBADOS" "INDONESIA" "MALAWI" "ZAMBIA" "SEYCHELLES" "MAU
RITIUS" "KENYA"
;      "GRENADA" "BRUNEI" "JAPAN" "ISRAEL" "AUSTRALIA" "SAUDI ARABIA" "QATAR" "OMAN" "KUWAIT" "DUBAI" "BAH
RAIN" "ABU DHABI" "USA"
;      "CANADA" "CALIFORNIA" "SWITZERLAND" "SWEDEN" "NORWAY" "FINLAND" "AUSTRIA" "GERMANY" "SPAIN" "PORTUG
AL" "GREECE" "GIBRALTAR"
;      "CANARY ISLES" "TUNISIA" "TAHITI" "REUNION" "NEW CALEDONIA" "MARTINIQUE" "GUADELOUPE" "FRENCH GUIAN
A" "FRANCE" "YUGOSLAVIA"
;      "ITALY" "TAIWAN" "NETHERLANDS" "ICELAND" "DENMARK" "BELGIUM" "NEW ZEALAND" "MALTA" "GIBRALTAR-RHD"
"GERMANY-RHD"
;      "UNITED KINGDOM" "IRISH REPUBLIC")
;     (("W I T H" "B53-SEAT SIDE ROLLS"))))
;    (((CLASS** "TRIM LEVEL 5") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** NO-RESTRICTED) (TRANSMISION** NO-RESTRICTED))
;     ("SOUTH AFRICA" "PAKISTAN" "SRI LANKA" "NEPAL" "INDIA" "GUYANA" "FIJI" "BANGLADESH" "ST. KITTS NEVIS
" "SINGAPORE" "MALAYSIA"
;      "JAMAICA" "HONG KONG" "CYPRUS" "BERMUDA" "BARBADOS" "INDONESIA" "MALAWI" "ZAMBIA" "SEYCHELLES" "MAU
RITIUS" "KENYA"
;      "GRENADA" "BRUNEI" "JAPAN" "ISRAEL" "AUSTRALIA" "SAUDI ARABIA" "QATAR" "OMAN" "KUWAIT" "DUBAI" "BAH
RAIN" "ABU DHABI" "USA"
;      "CANADA" "CALIFORNIA" "SWITZERLAND" "SWEDEN" "NORWAY" "FINLAND" "AUSTRIA" "GERMANY" "SPAIN" "PORTUG
AL" "GREECE" "GIBRALTAR"
;      "CANARY ISLES" "TUNISIA" "TAHITI" "REUNION" "NEW CALEDONIA" "MARTINIQUE" "GUADELOUPE" "FRENCH GUIAN
A" "FRANCE" "YUGOSLAVIA"
;      "ITALY" "TAIWAN" "NETHERLANDS" "ICELAND" "DENMARK" "BELGIUM" "NEW ZEALAND" "MALTA" "GIBRALTAR-RHD"
"GERMANY-RHD"
;      "UNITED KINGDOM" "IRISH REPUBLIC")
;     (("W I T H" "B53-SEAT SIDE ROLLS"))))))
; (((("B51R+B44R+B69D") ("B51R+B44R-B69D"))
;  ((((CLASS** "TRIM LEVEL 51") (BODY** "3 DOOR SALOON") (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICT
ED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** "LEFT HAND      STEERING") (TRANSMISION** NO-RESTRICTED))
;     ("SWITZERLAND" "SWEDEN" "NORWAY" "AUSTRIA" "GERMANY" "DENMARK") (("W I T H" "B53-SEAT SIDE ROLLS"))
))
;    (((CLASS** "TRIM LEVEL 51") (BODY** "5 DOOR SALOON") (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICT
ED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** "LEFT HAND      STEERING") (TRANSMISION** NO-RESTRICTED))
;     ("SWITZERLAND" "SWEDEN" "NORWAY" "AUSTRIA" "GERMANY" "DENMARK") (("W I T H" "B53-SEAT SIDE ROLLS"))
))))
; (((("B51L+B44R+B69D") ("B51L+B44R-B69D"))
;  ((((CLASS** "TRIM LEVEL 6") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;      (PERFORMANCE** "MULTI POINT INJ STD PERFORM") (DRIVE** "LEFT HAND      STEERING") (TRANSMISION** NO
-RESTRICTED))
;     ("SWITZERLAND" "SWEDEN" "NORWAY" "AUSTRIA" "GERMANY" "DENMARK") (("W I T H" "B53-SEAT SIDE ROLLS"))
))))
; ((("B51K+B44B"))
;  ((((CLASS** "TRIM LEVEL 4") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** NO-RESTRICTED) (TRANSMISION** NO-RESTRICTED))
;     ("SOUTH AFRICA" "PAKISTAN" "SRI LANKA" "NEPAL" "INDIA" "GUYANA" "FIJI" "BANGLADESH" "ST. KITTS NEVIS
" "SINGAPORE" "MALAYSIA"
;      "JAMAICA" "HONG KONG" "CYPRUS" "BERMUDA" "BARBADOS" "INDONESIA" "MALAWI" "ZAMBIA" "SEYCHELLES" "MAU
RITIUS" "KENYA"
;      "GRENADA" "BRUNEI" "JAPAN" "ISRAEL" "AUSTRALIA" "SAUDI ARABIA" "QATAR" "OMAN" "KUWAIT" "DUBAI" "BAH
RAIN" "ABU DHABI" "USA"
;      "CANADA" "CALIFORNIA" "SWITZERLAND" "SWEDEN" "NORWAY" "FINLAND" "AUSTRIA" "GERMANY" "SPAIN" "PORTUG
AL" "GREECE" "GIBRALTAR"
;      "CANARY ISLES" "TUNISIA" "TAHITI" "REUNION" "NEW CALEDONIA" "MARTINIQUE" "GUADELOUPE" "FRENCH GUIAN
A" "FRANCE" "YUGOSLAVIA"
;      "ITALY" "TAIWAN" "NETHERLANDS" "ICELAND" "DENMARK" "BELGIUM" "NEW ZEALAND" "MALTA" "GIBRALTAR-RHD"
"GERMANY-RHD"
;      "UNITED KINGDOM" "IRISH REPUBLIC")
;     (("W I T H" "B53-SEAT SIDE ROLLS"))))
;    (((CLASS** "TRIM LEVEL 3") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** NO-RESTRICTED) (TRANSMISION** NO-RESTRICTED))
;     ("SOUTH AFRICA" "PAKISTAN" "SRI LANKA" "NEPAL" "INDIA" "GUYANA" "FIJI" "BANGLADESH" "ST. KITTS NEVIS
" "SINGAPORE" "MALAYSIA"
;      "JAMAICA" "HONG KONG" "CYPRUS" "BERMUDA" "BARBADOS" "INDONESIA" "MALAWI" "ZAMBIA" "SEYCHELLES" "MAU
RITIUS" "KENYA"
;      "GRENADA" "BRUNEI" "JAPAN" "ISRAEL" "AUSTRALIA" "SAUDI ARABIA" "QATAR" "OMAN" "KUWAIT" "DUBAI" "BAH
RAIN" "ABU DHABI" "USA"
;      "CANADA" "CALIFORNIA" "SWITZERLAND" "SWEDEN" "NORWAY" "FINLAND" "AUSTRIA" "GERMANY" "SPAIN" "PORTUG
AL" "GREECE" "GIBRALTAR"
```

```
;     "CANARY ISLES" "TUNISIA" "TAHITI" "REUNION" "NEW CALEDONIA" "MARTINIQUE" "GUADELOUPE" "FRENCH GUIAN
A" "FRANCE" "YUGOSLAVIA"
;     "ITALY" "TAIWAN" "NETHERLANDS" "ICELAND" "DENMARK" "BELGIUM" "NEW ZEALAND" "MALTA" "GIBRALTAR-RHD"
"GERMANY-RHD"
;     "UNITED KINGDOM" "IRISH REPUBLIC")
;    (("W I T H" "B53-SEAT SIDE ROLLS")))))))
;Command: (append-all-strings-of-list-ALL '(("B51L+B44R+B69D") ("B51L+B44R-B69D")))
;(("B51L+B44R-B69D") ("B51L+B44R+B69D"))
;Command:
```

APPENDIX 8

LISTING OF THE INTELLIGENT NETWORKS PROTOTYPE

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: GRAPHER; Base: 10 -*-
;;; Node definitions


;*****************************************************************************************


;*****************************************************************************************


(defvar *all-the-nodes* nil
  "a list of instances of flavor node.
   only active nodes appear here")

(defvar *next-node-index* 0
  "Unique index assigned to each node.")


;;; The flavor of each node in the graph.
(defflavor node
        ((arcs nil)                          ; list of arcs attached to this node
         xpos                                ; coordinates of the center of this node
         ypos
         ;;;(TOTAL-QUANTITY 1)
         (quantity 2)  ;;;;;;;;; KEEPS the parts needed FOR ONE VEHICLE !!!!!!!!!!
         (quantity-per-usage-for-one-vehicle 2)    ;;;;;; The name is misleading !!!
                                        ;;;;;;;;;;;;IT KEEPS THE SUMMATION OF parts need in one assembly
         (part-number " ")
         (usage-condition "ALL")
         (radius :needs-calculation)         ; radius of node
         (label " ")                         ; a string
         (shape :circle)                     ; what shape border is displayed
         (unique (incf *next-node-index*)))  ; For print-self
        ()                                   ; No other flavors
  ;; You can set location, label and shape instance variables when you create
  ;; an instance of node.  After creating, you can change the label and shape.
  ;; You can always read the list of arcs, position and radius, in addition to
  ;; label and shape. The position must be specified when creating a node.
  (:initable-instance-variables xpos ypos label shape quantity quantity-per-usage-for-one-vehicle
                                usage-condition radius part-number) ;;; TOTAL-QUANTITY)
  (:writable-instance-variables label shape quantity quantity-per-usage-for-one-vehicle
                                usage-condition radius part-number) ;;;;TOTAL-QUANTITY)
  (:readable-instance-variables arcs xpos ypos radius quantity quantity-per-usage-for-one-vehicle usage-c
ondition
                                part-number);;;;;;TOTAL-QUANTITY)
  (:required-init-keywords :xpos :ypos))


;;; Called when new instances are created
(defmethod (make-instance node) (&rest ignore)
  (push self *all-the-nodes*))                ; Record each active instance





;;; Print-self routine to make nodes print prettily.
(defmethod (sys:print-self node) (stream ignore readably)
  (let ((name (or label (format nil "Unnamed node ~D" unique))))
    (if readably                              ; Print so read can "work".
        (sys:printing-random-object (self stream
                                            :typep
                                            :no-pointer)
          (write-string name stream))
        (write-string name stream))))

;;; For performing an operation to all nodes.
(defmacro map-over-nodes ((node-var) &body body)
  `(loop for ,node-var in *all-the-nodes*
         do (progn ,@body)))
```

```
;;; Need to recalculate radius
(defmethod ((setf node-label) node :after) (ignore)
  (setf radius :needs-calculation))

;;; Move a node from here to there
(defmethod (move node) (new-xpos new-ypos)
  (setf xpos new-xpos ypos new-ypos))

;;; Self-explanatory
(defmethod (add-arc node) (arc)
  (push arc arcs))

(defmethod (remove-arc node) (arc)
  (setq arcs (delete arc arcs)))
```

```
;;; Remove a node from the database
(defmethod (delete-self node) ()
  ;; (borrowed from tv:sheet's :kill method)
  ;; Do it this way to prevent CDR'ing down list structure being modified
  (loop until (null arcs)
        do (delete-self (car arcs)))
  (setq *all-the-nodes* (delete self *all-the-nodes*)))

;;; Display a node on the given window

;(defmethod (present-self node) (window)
;;;;;;;(setf (node-radius self) ':needs-calculation)
;
;  (calculate-radius self window)              ; Only required because we want to center
;                                              ; label where mouse click happened
;  ;; This is written this way because of a bug in 7.0 -- correct code below
;  (send window :set-cursorpos (- xpos radius) (- ypos radius))
;  (dw:with-output-as-presentation (:stream window :object self
;                                  :type 'node :single-box t)
;    (surrounding-output-with-border (window :shape
;                                          ;;;;;(graphics:draw-image (cadr (car statice-user::*images-in-
statice*)) 0 0) )
;    shape)
;      (send window :string-out      (string-append
;                                          (or label " ") "/"
;                                          (or  (format nil "~D" (node-quantity self)) " ") "/"
;                                          (or  (format nil "~D" (node-quantity-per-usage-for-one-vehicl
e s( f)) " ") "/"
;                                          (or usage-condition " ") ;;;
;                                          ;;;;(or (format nil "~D"  (node-TOTAL-QUANTITY self)) " "))))
;                                                  )))
;;      (send window :string-out (or  (format nil "~D" (node-quantity-per-usage-for-one-vehicle self)) "
"))
;;      (send window :string-out (or usage-condition " "))
;))

 (defmethod (present-self node) (window)
  (cond ((string-equal " " (node-label self))
         (calculate-radius self window)              ; Only required because we want to center
                                              ; label where mouse click happened

  ;; This is written this way because of a bug in 7.0 -- correct code below
  (send window :set-cursorpos (- xpos radius) (- ypos radius))
  (dw:with-output-as-presentation (:stream window :object self
                                  :type 'node :single-box t)
    (surrounding-output-with-border (window :shape shape)
      (dw::centering-lines (window)
        (format window "~A~%~D~%~D~%~A~%"
                (or label " ")
                ;;;;(or (format nil "~D"(node-quantity self) :stream window) " ")
                (or part-number " ")
                (or (format nil "~D" (node-quantity-per-usage-for-one-vehicle self) :stream window) " ")
                (or usage-condition " ")))))
        (t
  (let* ((image-in-statice (from-node-return-the-image self))

         (SMALLER-IMAGE (TV:WITH-OUTPUT-TO-BITMAP (BMSTREAM )
                        (GRAPHICS:DRAW-IMAGE image-in-statice ;;;;(cadr common-lisp-user::*images*)
                                                ;;;;;(cadr (cadr statice-user::*images-in-statice*))
                                                0 0 :STREAM BMSTREAM
                                                :SCALE 0.1))))


  (calculate-radius self window)              ; Only required because we want to center
                                              ; label where mouse click happened

  ;; This is written this way because of a bug in 7.0 -- correct code below
  (send window :set-cursorpos (- xpos radius) (- ypos radius))
  (dw:with-output-as-presentation (:stream window :object self
                                  :type 'node :single-box t)
    (surrounding-output-with-border (window :shape shape)
      (dw::centering-lines (window)
        (format window "~A~%~D~%~D~%~A~%"
                (or label " ")
                ;;;;(or (format nil "~D"(node-quantity self) :stream window) " ")
```

```
                (or part-number " ")
                (or (format nil "~D" (node-quantity-per-usage-for-one-vehicle self) :stream window) " ")
                (or usage-condition " "))))

      (graphics:draw-image ;;;*image*
                                ;;;;;(cadr (cadr statice-user::*images-in-statice*))
      SMALLER-IMAGE
                                                  ;;*IMAGE* would be a node attribute
                                                  ;;NOT a global variable
                                                  (- xpos (/ (array-dimension ;;; *image*
                                                              SMALLER-IMAGE 1) 2))
                                                  (cond ((or
                                                           (> (string-length (node-label self))
                                                            7)
                                                           (> (string-length (node-usage-condition self))
                                                            7))
                                                         (+ ypos 37)) ;;;; BECAUSE IF THE PART DESCRIPTION
IS BIG IT
                                                                          ;;;;  COVERS THE USAGES
                                                        (t
                                                         (+ ypos 20)))

                                                  ;;;;scale 0.2
                                                  :stream window)



)))))


#| Correct (i.e., Ideologically pure) version of the above:


;;;; Display a node on the given window
;(defmethod (present-self node) (window)
;  ;;;;;(setf (node-radius self) *:needs-calculation)
;  (calculate-radius self window)                  ; Only required because we want to center
;                                                  ; label where mouse click happened
;  (dw:with-output-as-presentation (:stream window :object self
;                                  :type 'node :single-box t)
;    (surrounding-output-with-border (window :shape shape)
;
;      (graphics:draw-string (or label " ") (- xpos radius) ypos :stream window)
;))))

;;;*****************************************************************************************************

;;;;;;; IN ORDER TO ECAPSULATE an IMAGE INSIDE A NODE.


;;;;;;In package  statice-user

(defvar *images-in-statice*
  (let ((snapshots nil))
    ;; Read all the snapshots into Lisp objects.
    (with-database (db *image-test-pathname*)
      (with-transaction ()
        (for-each ((s snapshot))
          (push (list (snapshot-name s) (snapshot-picture s)) snapshots)))) snapshots))
;
;(defvar *all-snapshots-in-statice*
;  (let ((snapshots nil))
;    ;; Read all the snapshots into Lisp objects.
;    (with-database (db *image-test-pathname*)
;      (with-transaction ()
;        (for-each ((s snapshot))
;          (push s snapshots))))
;    snapshots))


;Command: (export '*images-in-statice*)
;T
;Command:
;Command: (export '(*all-snapshots-in-statice*))
;T
;Command:
```

```
;Command: *images-in-statica*
;(("wheel" #<ART-1B-313-352 350130001>) ("steering-wheel" #<ART-1B-278-288 350143571>) ("spark" #<ART-1B-
271-128 350150503>)
; ("seat" #<ART-1B-285-256 350152602>))
;Command:


;(defmethod (present-self node) (window)
;          ;;THESE LABELS SHOULD BE ATTRIBUTES OF THE NODE YOU A DRAWING
;          ;;NOT LOCAL VARIABLES.
;  (let* (
;;        (label1 "LABEL 1")
;;        (label2 "LABEL 2")
;;        (label3 "LABEL 3")
;;        (label4 "LABEL 4")
;        (label1 (or label " "))
;        (label2 (or (format nil "~D" (node-quantity self) :stream window) " "))
;        (label3 (or (format nil "~D" (node-quantity-per-usage-for-one-vehicle self) :stream window) " ")
|
;        (label4 (or usage-condition " "))
;        (labels (list label4 label3 label2 label1))
;        (text-spacing-in-y 5)    ;level some space between the labels
;        (image-spacing-from-text-in-y 20)  ;leave space between text and images.
;        total-bitmap)
;    (setq total-bitmap (tv:with-output-to-bitmap (bmstream)
;                         (loop for label in labels
;                               for y-position from ypos by (- (+ text-spacing-in-y
;                                                                 (send bmstream
;                                                                       :character-height #\L)))
;                                                                 ;find out the height of the
;                                                                 ;label by asking the window
;                               do
;                           (graphics:draw-string label
;                                                 xpos
;                                                 y-position
;                                                 :stream bmstream
;                                                 :attachment-x :center
;                                                 :attachment-y :baseline))
;                           (graphics:draw-image ;;;*image*
;                             (cadr (cadr statica-user::*images-in-statica*))
;                                                 ;;*IMAGE* would be a node attribute
;                                                 ;;NOT a global variable
;                                                (- xpos (/ (array-dimension ;;; *image*
;                                                           (cadr (cadr statica-user::*images-in-statica
*)) 1) 2))
;                                                (+ ypos image-spacing-from-text-in-y)
;                                                :scale 0.2
;                                                :stream bmstream)))
;
;    (destructuring-bind (bm-height bm-width) (array-dimensions total-bitmap)
;      (setf radius  (/ (sqrt (+ (* bm-width bm-height)
;                                (* bm-height bm-height)))
;                       2))  ;;set RADIUS as half the length of the bitmap diagonal
;      (send window :set-cursorpos (- xpos radius) (- ypos radius))
;      (dw:with-output-as-presentation (:stream window :object self
;                                                 :type 'node :single-box t)
;       (surrounding-output-with-border (window :shape shape)
;         (graphics:draw-image
;           total-bitmap
;           0 0 :stream window))))))



;;;THIS BIT WAS WRONG!!!!!
;;;THIS BIT WAS WRONG!!!!!
;;; Calculate the radius of a node for a given window.
;;; We only care about this because we want to center the node at (xpos, ypos)
;;; instead of having its upper left be there.
(defmethod (calculate-radius node) (window)
  (when (and (stringp label) (zerop (string-length label)))
    (setq label nil))
  (when (eql radius :needs-calculation)
     (setq radius (ceiling
                    (multiple-value-bind (width height)
                        (dw:continuation-output-size
```

```
                          (lambda (stream)
                            (surrounding-output-with-border
                              (stream :shape shape)
                              (dw::centering-lines (stream)
                                (format stream "~A~%~D~%~D~%~A~%"
                                        (or label " ")
                                        (node-quantity self)  (node-quantity-per-usage-for-one-vehicle sel
f)

                                        (or usage-condition " ")))))
;                             (send stream :string-out       (string-append
;                                          (or label " ") "/"
;                                          (format nil "~D" (node-quantity self)) "/"
;                                          (format nil "~D" (node-quantity-per-usage-for-one-vehicle sel
f)) "/"
;                                          (or usage-condition " ") ;;;
;                                          ;;;;(or (format nil "~D"  (node-TOTAL-QUANTITY self)) " ))))
;                                                  ))))
                        window)
                      (max width height))
                  2)) )
  radius )

;;;THAT BIT WAS WRONG!!!!!
;;;THAT BIT WAS WRONG!!!!!
```

```
(defvar *all-the-arcs nil)


;;; Arc definitions
;;; The flavor of each arc in the graph
(defflavor arc
        ((mark nil)    ; arbitrary object for misc tagging purposes
         node1         ; the two "node" objects this arc connects
         node2
         (parts-per-assembly 1))
        ()
  (:writable-instance-variables mark parts-per-assembly)
  (:initable-instance-variables node1 node2 parts-per-assembly)
  (:readable-instance-variables node1 node2 parts-per-assembly))

;;; Hook this arc up to its nodes
(defmethod (make-instance arc) (&rest ignore)
  (add-arc node1 self)
  (add-arc node2 self)
  (setf (arc-parts-per-assembly  self)
        (cond ((> (node-quantity  node1) (node-quantity  node2) )
                                                    (/ (node-quantity  node1) (node-quantity  node2)))
                                              (t (/ (node-quantity  node2) (node-quantity  node1)))))
        ;;;;(+ (node-quantity  node2) (node-quantity  node1)))
)
  (push self *all-the-arcs ))


;;; Arcs always print in #<...> format.
(defmethod (sys:print-self arc) (stream &rest ignore)

  (sys:printing-random-object (self stream :typep)

    (format stream "~A  <--> ~A" node1  node2)

    ))

;;; Macro to perform an operation on all arcs.
;;; Note that this cannot be nested inside itself.  Thus, it can't be used
;;; to create relations whose domain and range are the set of arcs, for example.
(defmacro map-over-arcs ((arc-var) &body body)
  (let ((mark-var (make-symbol "MARK"))
        (node-var (make-symbol "NODE")))
    `(loop with ,mark-var = (ncons nil)          ; Create mark
           for ,node-var in *all-the-nodes*
           do (loop for ,arc-var in (node-arcs ,node-var)
                    unless (eq (arc-mark ,arc-var) ,mark-var)
                      do (progn ,@body
                                (setf (arc-mark ,arc-var)
                                      ,mark-var))))))

;;; Self-explanatory
(defmethod (delete-self arc) ()
  (remove-arc node1 self)
  (remove-arc node2 self))

;;; Currently, no good way to present arcs, just draw them instead.  We can
;;; count on the radius being OK because we always draw the nodes before the
;;; arcs; otherwise we'd have to call calculate-radius on them.
(defmethod (present-self arc) (window)
;  (let ((stream (dw:get-program-pane 'graph)))
;  (let ((radius-of-node1 (calculate-radius node1 stream))
;        (radius-of-node2 (calculate-radius node2 stream)))
  (multiple-value-bind (x1 y1 x2 y2)
      (find-edges-of-nodes   (node-radius node1)  ;;;;;radius-of-node1 ;;

                             (node-xpos node1)
                             (node-ypos node1)
                               (node-radius node2) ;;;;     radius-of-node2 ;;;;
                             (node-xpos node2)
                             (node-ypos node2))
;;;THIS BIT WAS WRONG!!!!!

    (graphics:draw-arrow  x1 y1 x2 y2 :stream window); (+ y1 (node-radius node1) ) (+ x2  (node-radius no
```

```
de2)  )  y2 :stream window)
    (graphics:draw-string (format nil "~D" (arc-parts-per-assembly self))  (abs (+ x1 (/ (- x2 x1) 2))) (
abs (+ y2 (/ (- y1 y2) 2))) :stream window)
)) ;;;))


;(defun make-the-array (number)
;  (let ((help (make-array 2)))
;    (loop for i from 0 to 1
;          do
;      (setf (aref help i) (int-char number)))
;    help))
;
;
;(defun array-to-string (array)
;(let ((string  "")) (loop for i from 0 to 1
;                          do
;                      (setq string (string-append (aref array i) string)))
;     string))
;
;
;(defun return-the-quantity-in-string (string-of-asciies)
;  (let ((string ""))
;    (loop for i from 0 to 1
;          do
;      (setq string (string-append (char-int (aref string-of-asciies i)) string)))
;    string))

;;; Subroutine for the above.
(defun find-edges-of-nodes (r1 xpos1 ypos1 r2 xpos2 ypos2)

  (let* ((dx (- xpos2 xpos1))
         (dy (- ypos2 ypos1))
         (length (isqrt (+ (* dx dx) (* dy dy)))))
    (values (+ xpos1 (ceiling (* dx r1) length))
            (+ ypos1 (ceiling (* dy r1) length))
            (- xpos2 (floor (* dx r2) length))
            (- ypos2 (floor (* dy r2) length)))))

(compile-flavor-methods node arc)



;;; Display a node on the given window
;(defmethod (present-self node) (window)
;  ;;;;;(setf (node-radius self) ':needs-calculation)
;  (calculate-radius self window)                    ; Only required because we want to center
;                                                    ; label where mouse click happened
;  (dw:with-output-as-presentation (:stream window :object self
;                                  :type 'node :single-box t)
;    (surrounding-output-with-border (window :shape shape)
;      (graphics:draw-image (cadr (cadr statice-user::*images-in-statice*)) (- xpos radius) ypos :scale 0
.3 :stream window)
;      (graphics:draw-string (or label " ") (- xpos radius) ypos :stream window)
;)))


;;;;;;;;  NOTICE, YOU COULD WRITE A PROGRAM TO define a BUTTON gesture (ie.  right button) WHERE TO CLICK
;;;;;;;;; INSIDE THE node-image AND HAVE A BIGGER MORE DETAILED  PICTURE AS IN  Statice.


;;;*****************************************************************************************************
;;;THAT BIT WAS WRONG!!!!!
;#

;;;; Calculate the radius of a node for a given window.
;;;; We only care about this because we want to center the node at (xpos, ypos)
;;;; instead of having its upper left be there.
;
;(defmethod (calculate-radius node) (window)
;  (when (and (stringp label) (zerop (string-length label)))
;    (setq label nil))
;  (when (eql radius :needs-calculation)
;    (setq radius (ceiling
;                   (multiple-value-bind (width height)
```

```
;                       (dw:continuation-output-size
;                         (lambda (stream)
;                           (surrounding-output-with-border
;                             (stream :shape shape)
;                             (send stream :string-out (or label " "))))
;                         window)
;                     (max width height))
;                 2)) )
;  radius)


;;;THIS BIT WAS WRONG!!!!!
;;;THIS BIT WAS WRONG!!!!!
;;; Calculate the radius of a node for a given window.
;;; We only care about this because we want to center the node at (xpos, ypos)
;;; instead of having its upper left be there.
(defmethod (calculate-radius node) (window)
  (when (and (stringp label) (zerop (string-length label)))
    (setq label nil))
  (when (eql radius :needs-calculation)
     (setq radius (ceiling
                 (multiple-value-bind (width height)
                     (dw:continuation-output-size
                       (lambda (stream)
                         (surrounding-output-with-border
                           (stream :shape shape)
                           (dw::centering-lines (stream)
                             (format stream "~A~%~D~%~D~%~A~%"
                                     (or label " ")
                                     (node-quantity self) (node-quantity-per-usage-for-one-vehicle sel
f)

                                     (or usage-condition " "))))))
                           (send stream :string-out       (string-append
                                         (or label " ") "/"
                                         (format nil "~D" (node-quantity self)) "/"
                                         (format nil "~D" (node-quantity-per-usage-for-one-vehicle sel
f)) "/"
                                         (or usage-condition " ") ;;;
                                         ;;;;(or (format nil "~D"  (node-TOTAL-QUANTITY self)) " "))))
                                                  ))))
                       window)
                   (max width height))
                 2)) )
  radius)


;;;THAT BIT WAS WRONG!!!!!
```

```
(defvar *all-the-arcs nil)


;;; Arc definitions
;;; The flavor of each arc in the graph
(defflavor arc
        ((mark nil)    ; arbitrary object for misc tagging purposes
         node1         ; the two "node" objects this arc connects
         node2
         (parts-per-assembly 1))
        ()
  (:writable-instance-variables mark parts-per-assembly)
  (:initable-instance-variables node1 node2 parts-per-assembly)
  (:readable-instance-variables node1 node2 parts-per-assembly))

;;; Hook this arc up to its nodes
(defmethod (make-instance arc) (&rest ignore)
  (add-arc node1 self)
  (add-arc node2 self)
  (setf (arc-parts-per-assembly  self)
        (cond ((> (node-quantity  node1) (node-quantity  node2) )
                                                    (/ (node-quantity  node1) (node-quantity  node2)))
                                              (t (/ (node-quantity  node2) (node-quantity  node1))))
        ;;;;(+ (node-quantity  node2) (node-quantity  node1)))
)
  (push self *all-the-arcs ))


;;; Arcs always print in #<...> format.
(defmethod (sys:print-self arc) (stream &rest ignore)

  (sys:printing-random-object (self stream :typep)

    (format stream "~A  → ~A" node1  node2)

    ))

;;; Macro to perform an operation on all arcs.
;;; Note that this cannot be nested inside itself.  Thus, it can't be used
;;; to create relations whose domain and range are the set of arcs, for example.
(defmacro map-over-arcs ((arc-var) &body body)
  (let ((mark-var (make-symbol "MARK"))
        (node-var (make-symbol "NODE")))
    `(loop with ,mark-var = (ncons nil)          ; Create mark
           for ,node-var in *all-the-nodes*
           do (loop for ,arc-var in (node-arcs ,node-var)
                    unless (eq (arc-mark ,arc-var) ,mark-var)
                      do (progn ,@body
                                (setf (arc-mark ,arc-var)
                                      ,mark-var))))))

;;; Self-explanatory
(defmethod (delete-self arc) ()
  (remove-arc node1 self)
  (remove-arc node2 self)
  (delete self *all-the-arcs))

;;; Currently, no good way to present arcs, just draw them instead.  We can
;;; count on the radius being OK because we always draw the nodes before the
;;; arcs; otherwise we'd have to call calculate-radius on them.
;(defmethod (present-self arc) (window)
;;  (setf (node-radius node1) ':needs-calculation)
;;  (setf (node-radius node2) ':needs-calculation)
;;  (let ((stream (dw:get-program-pane 'graph)))
;;  (let ((radius-of-node1 (calculate-radius node1 stream))
;;        (radius-of-node2 (calculate-radius node2 stream)))
;  (multiple-value-bind (x1 y1 x2 y2)
;      (find-edges-of-nodes (node-radius node1)  ;;;;;radius-of-node1 ;;
;                           (node-xpos node1)
;                           (node-ypos node1)
;                           (node-radius node2) ;;;;    radius-of-node2
;                           (node-xpos node2)
;                           (node-ypos node2))
;    (graphics:draw-arrow  x1
```

```
;                              y1 ;;;(+ y1 (node-radius node1)  (node-radius node1) ) ;;y1 (+ y1 (node-radius
node1) )
;                              x2 ;;;(+ x2  (node-radius node2) (/ (node-radius node2) 2)  ) ;;x2 (+ x2  (node
-radius node2)  )
;                              y2 :stream window)
;    (graphics:draw-string (format nil "~D" (arc-parts-per-assembly self))  (abs (+ x1 (/ (- x2 x1) 2)))
(abs (+ y2 (/ (- y1 y2) 2))) :stream window)
;)) ;;;))

;&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

;;;;;;; IN ORDER TO ECAPSULATE an IMAGE INSIDE A NODE.


;;;;;;In package  statice-user

;(defvar *images-in-statice*
;  (let ((snapshots nil))
;    ;; Read all the snapshots into Lisp objects.
;    (with-database (db *image-test-pathname*)
;      (with-transaction ()
;       (for-each ((s snapshot))
;         (push (list (snapshot-name s) (snapshot-picture s)) snapshots)))) snapshots))

;Command: (export '*images-in-statice*)
;T
;Command:

;Command: *images-in-statice*
;(("wheel" #<ART-1B-313-352 350130001>) ("steering-wheel" #<ART-1B-278-288 350143571>) ("spark" #<ART-1B-
271-128 350150503>)
; ("seat" #<ART-1B-285-256 350152602>))
;Command:





;;;THIS IS THE ALTERNATIVE VERSION THAT DISPLAYS 4 LABELS AND AN IMAGE.
;;;THE IMAGES YOU HAVE SHOULD BE SCALED AND YOU SHOULD SAVE THESE SMALLER VERSIONS.
;;;USE A FORM LIKE:
;;;   (SETQ SMALLER-IMAGE (TV:WITH-OUTPUT-TO-BITMAP (BMSTREAM)
;;;                              (GRAPHICS:DRAW-IMAGE BIG-IMAGE 0 0 :STREAM BMSTREAM
;;;                                                                 :SCALE 0.1)))
;;;YOU CAN THEN STORE THESE SMALLER IMAGES IN YOUR DATABASE.


;(defmethod (present-self node) (window)
;         ;;THESE LABELS SHOULD BE ATTRIBUTES OF THE NODE YOU A DRAWING
;         ;;NOT LOCAL VARIABLES.
;  (let* (
;;       (label1 "LABEL 1")
;;       (label2 "LABEL 2")
;;       (label3 "LABEL 3")
;;       (label4 "LABEL 4")
;        (label1 (or label " "))
;        (label2 (or (format nil "~D" (node-quantity self) :stream window) " "))
;        (label3 (or (format nil "~D" (node-quantity-per-usage-for-one-vehicle self) :stream window) " ")
)
;        (label4 (or usage-condition " "))
;        (labels (list label4 label3 label2 label1))
;        (text-spacing-in-y 5)   ;level some space between the labels
;        (image-spacing-from-text-in-y 20)  ;leave space between text and images.
;        total-bitmap)
;    (setq total-bitmap (tv:with-output-to-bitmap (bmstream)
;                         (loop for label in labels
;                               for y-position from ypos by (- (+ text-spacing-in-y
;                                                                 (send bmstream
;                                                                       :character-height #\L)))
;                                                                 ;find out the height of the
;                                                                 ;label by asking the window
;                               do
;                         (graphics:draw-string label
```

```
;                                                      xpos
;                                                      y-position
;                                                      :stream bmstream
;                                                      :attachment-x :center
;                                                      :attachment-y :baseline))
;                               (graphics:draw-image ;;;*image*
;                                 (cadr (cadr statice-user::*images-in-statice*))
;                                                      ;;*IMAGE* would be a node attribute
;                                                      ;;NOT a global variable
;                                                     (- xpos (/ (array-dimension ;;; *image*
;                                                                  (cadr (cadr statice-user::*images-in-statice
*)) 1) 2))
;                                                     (+ ypos image-spacing-from-text-in-y)
;                                                     :scale 0.2
;                                                     :stream bmstream)))
;
;     (destructuring-bind (bm-height bm-width) (array-dimensions total-bitmap)
;       (setf radius  (/ (sqrt (+ (* bm-width bm-height)
;                                  (* bm-height bm-height)))
;                        2))  ;;set RADIUS as half the length of the bitmap diagonal
;       (send window :set-cursorpos (- xpos radius) (- ypos radius))
;       (dw:with-output-as-presentation (:stream window :object self
;                                                     :type 'node :single-box t)
;        (surrounding-output-with-border (window :shape shape)
;          (graphics:draw-image
;            total-bitmap
;            0 0 :stream window))))))


;&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&


(defmethod (present-self arc) (window)
;  (let ((stream (dw:get-program-pane 'graph)))
;  (let ((radius-of-node1 (calculate-radius node1 stream))
;        (radius-of-node2 (calculate-radius node2 stream)))
  (multiple-value-bind (x1 y1 x2 y2)
      (find-edges-of-nodes   (node-radius node1)  ;;;;;radius-of-node1 ;;

                              (node-xpos node1)
                              (node-ypos node1)
                                (node-radius node2) ;;;;    radius-of-node2 ;;;;
                              (node-xpos node2)
                              (node-ypos node2))
;;;THIS BIT WAS WRONG!!!!!

    (graphics:draw-arrow  x1 y1 x2 y2 :stream window); (+ y1 (node-radius node1) ) (+ x2  (node-radius no
de2)  )  y2 :stream window)
    (graphics:draw-string (format nil "~D" (arc-parts-per-assembly self))  (abs (+ x1 (/ (- x2 x1) 2))) (
abs (+ y2 (/ (- y1 y2) 2))) :stream window)
))


;(defun make-the-array (number)
;  (let ((help (make-array 2)))
;    (loop for i from 0 to 1
;          do
;      (setf (aref help i) (int-char number)))
;    help))
;
;
;(defun array-to-string (array)
;(let ((string  "")) (loop for i from 0 to 1
;                           do
;
;                    (setq string (string-append (aref array i) string)))
;    string))
;
;
;(defun return-the-quantity-in-string (string-of-asciies)
;  (let ((string ""))
;    (loop for i from 0 to 1
;          do
```

```
;       (setq string (string-append (char-int (aref string-of-asciies i)) string)))
;     string))

;;; Subroutine for the above.
(defun find-edges-of-nodes (r1 xpos1 ypos1 r2 xpos2 ypos2)

  (let* ((dx (- xpos2 xpos1))
         (dy (- ypos2 ypos1))
         (length (isqrt (+ (* dx dx) (* dy dy)))))
    (values (+ xpos1 (ceiling (* dx r1) length))
            (+ ypos1 (ceiling (* dy r1) length))
            (- xpos2 (floor (* dx r2) length))
            (- ypos2 (floor (* dy r2) length)))))

(compile-flavor-methods node arc)
```

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10; Package: COMMON-LISP -*-

;;;;;;  I  do the next in LISP.

(defpackage grapher
  (:use scl)
  (:colon-mode :external)
  (:export *all-the-nodes*))

;(defsystem grapher
;    (:patchable t
;     :default-pathname "lisp-lore:examples;grapher;"
;     :maintaining-sites :ssf
;     :pretty-name "Graph demo program")
;  (:serial "nodes-and-arcs" "presentation-types" "display-frame"))


;;;;;;;;;;;;;;;;;;  OR ;;;;;;;;;;;;;;

;  (make-package  'grapher
;  (:use scl)
;  (:colon-mode :external)
;)
;
;
;;;;;;;;Command: Set Package (a package [default COMMON-LISP-USER]) GRAPHER
; ;;Package set to GRAPHER
;Command: (defvar *all-the-nodes*  nil)
;*ALL-THE-NODES*
;Command: (export '(*ALL-THE-NODES*))
;T
;Command:
```

```
;;; -*- Mode: LISP; Syntax: Common-Lisp; Base: 10; Package: GRAPHER -*-


;;; This is the presentation type for nodes
(define-presentation-type node
      ()
   :no-deftype t                    ; node is also a flavor
   ;; Standard parser for "you can't read one of these."
   :parser ((stream) (loop do (dw:read-char-for-accept stream))))

#|  We're not presenting any of these at the moment.

;;; This is the presentation type for arcs
(define-presentation-type arc
      ()
   :no-deftype t                    ; arc is also a flavor
   ;; Standard parser for "you can't read one of these."
   :parser ((stream) (loop do (dw:read-char-for-accept stream))))

|#
```

```
;;; -*- Mode: LISP; Syntax: Common-Lisp; Base: 10; Package: GRAPHER -*-


(make-instance 'node :label "SEAT-MANUAL"  :xpos 450 :ypos 50)

(defparameter
  *grapher-display-margin-components*
  'dw:((margin-ragged-borders )
       (margin-scroll-bar :history-noun "display")
       (margin-scroll-bar :history-noun "display"
                          :margin :bottom)
       (margin-white-borders :thickness 2)))

(dw:define-program-framework grapher
  :select-key
  #\network
  :command-definer
  t
  :command-table
  (:inherit-from '("colon full command" "standard arguments" "standard scrolling"))
  :state-variables
  nil
  :panes
  ((graph :display :redisplay-after-commands t
          :redisplay-function 'display-graph
          :incremental-redisplay nil
          :margin-components *grapher-display-margin-components*
          :typeout-window t)
   (commands :interactor)
   (menu :command-menu :menu-level :top-level :columns 1
         :margin-components *grapher-display-margin-components*))
  :configurations
  '((dw::main
     (:layout (dw::main :column graph row-1)
      (row-1 :row commands menu))
     (:sizes (dw::main (graph 0.8) :then (row-1 :even))
      (row-1 (menu  :ask-window self
                    :size-for-pane menu)
       :then (commands :even))))))



(defmethod (display-graph grapher) (display-pane)
  (map-over-nodes (node) (present-self node display-pane))
  (map-over-arcs (arc) (present-self arc display-pane)))

;;; %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

(define-grapher-command (com-create-node :menu-accelerator "Create PART")
    ((x 'integer)
     (y 'integer))
   (make-instance 'node :xpos x :ypos y))

(define-presentation-to-command-translator com-create-node-ex-nihilo
                                            (dw:no-type :blank-area t
                                                        :suppress-highlighting nil
                                                        :documentation "Create node here"
                                                        :gesture create-node-ex-nihilo)
  (ignore &key x y) (cp:build-command 'com-create-node x y))


;;; %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


(define-grapher-command (com-set-node-label :menu-accelerator "Set DESCRIPTION")
    ((node 'node :prompt "a Part")
     (label '((null-or-type string))
            :default (node-label node)
            :prompt "its Part Description" :confirm t))
   (setf (node-label node) label))

(define-grapher-command (com-set-node-label-\(prompt\) :name nil)
                        ;;;;:menu-accelerator "Set LABEL")
    ((node 'node))
   (let* ((label (accept 'string
```

```
                        :prompt (format nil "New Part Description for Physical part ~A: " node)
                        :default (node-label node))))
     (setf (node-label node) label)))

(define-presentation-to-command-translator com-set-node-label
                                            (node :gesture set-node-label)
  (node)
  (cp:build-command 'com-set-node-label-\(prompt\)
node))




;;;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

(define-grapher-command (com-set-node-USAGE-CONDITION :menu-accelerator "Set USAGE COND")
    ((node 'node :prompt "a Part")
     (UC '((null-or-type string))
            :default (node-usage-condition node)
            :prompt "its USAGE CONDITION" :confirm t))
   (setf (node-usage-condition node) UC))


(define-grapher-command (com-set-node-USAGE-CONDITION-\(prompt\) :name nil)
                         ;;;;;;:menu-accelerator "Set USAGE COND")
    ((node 'node))
   (let* ((UC (accept 'string
                        :prompt (format nil "New USAGE CONDITION for Physical part ~A: " node)
                        :default (node-usage-condition node))))

     (setf (node-usage-condition node) UC)))


(define-presentation-to-command-translator com-set-node-USAGE-CONDITION
                                            (node :gesture set-node-USAGE-CONDITION)
  (node)
  (cp:build-command 'com-set-node-USAGE-CONDITION-\(prompt\)
                    node))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

;;;;;; ***********************************************************************

(define-grapher-command (com-set-node-QUANTITY-IN-VEHICLE :menu-accelerator "Set QUANTITY")
    ((node 'node :prompt "a Part")
     (quantity '((null-or-type number))
            :default (node-quantity node)
            :prompt "its QUANTITY per UCAGE CONDITION" :confirm t))
   (setf (node-quantity node) quantity))


(define-grapher-command (com-set-node-QUANTITY-IN-VEHICLE-\(prompt\) :name nil)
                         ;;;;;;;:menu-accelerator "Set QUANTITY")
    ((node 'node))
   (let* ((quantity (accept 'string
                        :prompt (format nil "New QUANTITY per USAGE CONDITION of the Physical part ~A: "
 node)
                        :default (node-quantity node))))

     (setf (node-quantity node) quantity)))


(define-presentation-to-command-translator com-set-node-QUANTITY-IN-VEHICLE
                                            (node :gesture com-set-node-QUANTITY-IN-VEHICLE)
  (node)
  (cp:build-command 'com-set-node-USAGE-CONDITION-\(prompt\)
node))



;;;;;; ***********************************************************************
```

```
(defmacro with-mouse-tracking-node ((node pane) &body body)
  ;; Done right, we would care about the node's radius
  (let ((old-mouse-char (make-symbol "OLD-MOUSE-CHAR")))
    `(let ((,old-mouse-char (send ,pane :mouse-blinker-character)))
       (unwind-protect
           (progn (send ,pane :set-mouse-cursorpos
                        (node-xpos ,node) (node-ypos ,node))
                  (send ,pane :set-mouse-blinker-character
                        #\mouse:fat-circle)
                  (send ,pane :mouse-standard-blinker)
                  ,@body)
         (send ,pane :set-mouse-blinker-character ,old-mouse-char)
         (send ,pane :mouse-standard-blinker)))))

;;;;;; ***********************************************************************

(define-grapher-command (com-move-node :menu-accelerator "Move PART")
    ((node 'node :prompt "click on a Physical Part" :confirm t))
   (let ((pane (send dw:*program-frame* :get-pane 'graph)))
     (with-mouse-tracking-node (node pane)
       (dw:tracking-mouse (pane)
         (:who-line-documentation-string
           () "Specify new location for node.  Right aborts")
         (:mouse-click (button x y)
          (unless (char-mouse-equal button #\mouse-right)
            (move node x y))          ; Don't move if clicked right
          (return nil))))))

(define-presentation-to-command-translator com-move-node-from-here-to-there
                                            (node :gesture move-node)
  (node)
  (cp:build-command 'com-move-node node))


;;;;*****************************************************************************************************************
%%%%%%%%%%%%%%%%%%%%%%%
;;;;;;;(defvar *new-link* nil)

(define-grapher-command (com-add-arc :menu-accelerator "Add ASSY LINK")

    ((node1 'node :prompt "a Physical part")
     (node2 'node :prompt "another Physical part" :confirm t))
   (if (eq node1 node2)
       (format t "~&Sorry, you can't make an ASSEMBLY LINK from a Physical Part to itself.")
       (make-instance 'arc :node1 node1 :node2 node2)
   )
  (let ((katerina " "))
   (let ((temp-summ 0))
      (let ((arcs-of-node-which-the-link-is-put-TO (node-arcs node2)))
        (let ((result nil))
          (loop for arc in arcs-of-node-which-the-link-is-put-TO
                do
            (cond ((equal (arc-node2 arc) node2)

                (push (list (arc-node1 arc) arc) result))))

          (loop for accedant-node-and-its-belonging-arc in result
                 do
             (setq temp-summ (+ (* (node-quantity-per-usage-for-one-vehicle (car accedant-node-and-its-b
elonging-arc))
                                   ;;;;(node-quantity accedant-node))
                                   (arc-parts-per-assembly (cadr accedant-node-and-its-belonging-arc)))
                                temp-summ)))
;           (setq temp-summ (+ (* (node-quantity-per-usage-for-one-vehicle node1)
;                                 (arc-parts-per-assembly *new-link*)) ;;;WE DON'T NEED IT as the link is
 already put,
;                              temp-summ))))                          ;;;; in NODE1  , hence it will be count
ed, anyway. !!!!!!
            ))
      ;;;;(setf (node-TOTAL-QUANTITY node2) temp-summ)))
      (setf (node-quantity-per-usage-for-one-vehicle node2) temp-summ))
```

```
    (let ((usage-condition-of-node2 " "))
      (let ((arcs-of-node-which-the-link-is-put-TO (node-arcs node2)))
        (let ((result nil))
          (loop for arc in arcs-of-node-which-the-link-is-put-TO
                do
                (push (list (arc-node1 arc) arc) result))
          (loop for accendant-node in result
                do
            (setq usage-condition-of-node2 (string-append (node-usage-condition (car accendant-node))  ;;
; because it comes as a list
                                                          usage-condition-of-node2       ;;;;; of first t
he node than the arcs
;                                         Error: The generic function NODE-USAGE-CONDITION
;                                         was applied to the argument (#<NODE cushion> #<ARC #  <--> # 6001
10733>).
;                                         The argument was not of the correct type.
;                                         The function is implemented by the flavor NODE.
                                                        )
                 ))))
      (let ((usage-combination-of-node-before (get-words-from-string-other-new-old (node-usage-condition
 node2)))
            (usage-combination-of-the-links-of-the-node
              (get-words-from-string-other-new-old (combine-logically-the-feature-groups usage-condition-
of-node2))))
                    ;;;;;;;;     (combine-logically-the-feature-groups "-b34 +gty8 -kiop +b34")
       (let* ((other (remove-duplicates
                       (append usage-combination-of-node-before usage-combination-of-the-links-of-the-nod
e)
                      :test #'string-equal))


             (after-without-ALL (cond
                                  ((and (> (length other) 1)
                                        (member "ALL" other :test #'string-equal))
                                        (cond
                                          ;;;;the-rest-of-the-usage-statement-it-does-NOT-belong-to-the-
same-ALL-feature-group
;                                           (member
;                                            (car (remove "ALL" other :test #'string-equal))
;                                            (list usage-condition-of-node2) :test #'string-equal)
                                         ;;;;; We check to see if the rest of the usage condition ie. +B
69D belongs to the same
                                         ;;;;; DELETION OF each other, already happened. WE only take th
e car because we assume
                                         ;;;;; only  one feature group to take place now, ie. the B69D
                                          ((or
                                             (equal (length
                                                    (collect-all-usage-conditions-of-nodes
                                                      (function-which-collects-all-the-upward-nodes node
2)))
                                                  2)
                                             (equal (collect-all-usage-conditions-of-nodes
                                                      (function-which-collects-all-the-upward-nodes node
2))
                                                    '("ALL")))   ;;;;; ie. there is only one above with
ALL statement
                                           '("ALL"))
                                          (t
                                               (remove "ALL" other :test #'string-equal)))))))
                                         ;;;(t other))))   ;;;;;;;;;;'("ALL")

;        (print usage-combination-of-node-before)
;                 ;;;;;(print (combine-logically-the-feature-groups usage-condition-of-node2))
;        (print usage-combination-of-the-links-of-the-node)
;        (print after-without-ALL)
;      ;;;;;;;(print usage-combination-of-the-links-of-the-node)
;        (print after-without-ALL)


        ;; You are changing the label here
        ;; Therefore you need to recalculate the radius!!!
        (cond ((equal usage-combination-of-the-links-of-the-node nil)      ;;;ie. +B69D  and -B69D delete
each other ?
                 ;;;(cond ((and
               (setf (node-usage-condition  node2)
```

```
                    "ALL"))
                ((equal usage-combination-of-the-links-of-the-node (list "ALL"))
                    (setf (node-usage-condition  node2)
                    "ALL"))
                ((some #'(lambda (x) (string-search "ALLA"   x)) usage-combination-of-the-links-of-the-nod
e)
                     ;;;;; in order to avoid cases such as ALLALL
                     (setf (node-usage-condition  node2)
                    "ALL"))
                (t
                 (loop for feature-group in after-without-ALL
                       do
                   (setq katerina (string-append " " (string-append (string-append feature-group " ") kate
rina ))) )

                 (setf (node-usage-condition  node2)
                       katerina)))

        (setf (node-radius node2) :needs-calculation)
           ))))
;;;;;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
  (let ((arcs-of-the-node2 (node-arcs node2))
        (nodes-to-the-downward-direction  nil))
    (loop for arc in arcs-of-the-node2
          do
      (cond ((equal (arc-node1 arc) node2)
                 (push (arc-node2 arc)  nodes-to-the-downward-direction))))
    ;;;;;;;;(print nodes-to-the-downward-direction)
;    (loop for i in nodes-to-the-downward-direction
;         do
;      (try-one-step-downwards i)))

(recursion-on-the-network nodes-to-the-downward-direction)

))


(defun function-which-collects-all-the-upward-nodes (node)
  (let ((upward-nodes  nil))
    (loop for arc in (node-arcs node)
          do
      (cond ((equal (arc-node2 arc) node)
                 (push (arc-node1 arc) upward-nodes))))
    upward-nodes))


(defun collect-all-usage-conditions-of-nodes (list-of-nodes)
  (let ((result nil) (result-new nil))
    (loop for i in list-of-nodes
          do
           (push (node-usage-condition i) result))
    (loop for i in result
              do
      (push (string-trim '(#\space) i) result-new))
    (remove-duplicates result-new :test #'string-equal)))


(defun function-which-collects-all-the-downward-nodes (node)
  (let ((downward-nodes  nil))
    (loop for arc in (node-arcs node)
          do
      (cond ((equal (arc-node1 arc) node)
                 (push (arc-node2 arc) downward-nodes))))
    downward-nodes))



(defun  recursion-on-the-network  (list-of-nodes) ;;;;;;
  (cond
    ((null list-of-nodes) nil)
    ((equal (function-which-collects-all-the-downward-nodes (car list-of-nodes)) nil)
         (try-one-step-downwards (car list-of-nodes))
         (recursion-on-the-network   (cdr list-of-nodes)))
    (t (try-one-step-downwards (car list-of-nodes))
       (cons
       (recursion-on-the-network
```

```
        (function-which-collects-all-the-downward-nodes (car list-of-nodes)))
      (recursion-on-the-network
         (cdr list-of-nodes))))))


;;;;*******************************************************************%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%




;;;;;;;;;;;;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%


(defmethod     (try-one-step-downwards   node) ()

(let ((katerina " "))
   (let ((temp-summ 0))
      (let ((arcs-of-node-which-the-link-is-put-TO (node-arcs self)))
        (let ((result nil))
          (loop for arc in arcs-of-node-which-the-link-is-put-TO
                do
            (cond ((equal (arc-node2 arc) self)
                   (push (list (arc-nodel arc) arc) result))))

          (loop for accedant-node-and-its-belonging-arc in result
                do
              (setq temp-summ (+ (* (node-quantity-per-usage-for-one-vehicle (car accedant-node-and-its-b
elonging-arc))
                                    ;;;;(node-quantity accedant-node))
                                    (arc-parts-per-assembly (cadr accedant-node-and-its-belonging-arc)))
                                 temp-summ)))
;           (setq temp-summ (+ (* (node-quantity-per-usage-for-one-vehicle nodel)
;                                 (arc-parts-per-assembly *new-link*)) ;;;WE DON'T NEED IT as the link is
 already put,
;                              temp-summ))))                    ;;;; in NODE1  , hence it will be count
ed, anyway. !!!!!!
           ))
      ;;;;(setf (node-TOTAL-QUANTITY self) temp-summ)))
      (setf (node-quantity-per-usage-for-one-vehicle self) temp-summ))

    (let ((usage-condition-of-self " "))
      (let ((arcs-of-node-which-the-link-is-put-TO (node-arcs self)))
        (let ((result nil))
          (loop for arc in arcs-of-node-which-the-link-is-put-TO
                do
                (push (list (arc-nodel arc) arc) result))
          (loop for accendant-node in result
                do
            (setq usage-condition-of-self (string-append (node-usage-condition (car accendant-node))  ;;;
 because it comes as a list
                                                    usage-condition-of-self       ;;;;; of first th
e node then the arcs
;                                       Error: The generic function NODE-USAGE-CONDITION
;                                       was applied to the argument (#<NODE cushion> #<ARC #  ←→ # 6001
10733>).
;                                       The argument was not of the correct type.
;                                       The function is implemented by the flavor NODE.
                                                     )
                 ))))
      (let ((usage-combination-of-node-before (get-words-from-string-other-new-old (node-usage-condition
 self)))
            (usage-combination-of-the-links-of-the-node
              (get-words-from-string-other-new-old (combine-logically-the-feature-groups usage-condition-
of-self))))
        (let* ((other (remove-duplicates
                      (append usage-combination-of-node-before usage-combination-of-the-links-of-the-nod
e)
                     :test #'string-equal))
;             (after-without-ALL (cond
;                                   ((and (> (length other) 1)
;                                         (member "ALL" other :test #'string-equal)
;                                         ;;;(sure-it-represents-the-same-DELETION-with-the-rest-of-the-us
```

```
ages .....)
;                                               )
                                               (remove "ALL" other :test #'string-equal))
;                                             (t other)))
;
        ;;;;;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
              (after-without-ALL (cond
                                    ((and (> (length other) 1)
                                          (member "ALL" other :test #'string-equal))
                                          (cond
                                            ;;;;the-rest-of-the-usage-statement-it-does-NOT-belong-to-the-
same-ALL-feature-group
;                                             (member
;                                              (car (remove "ALL" other :test #'string-equal))
;                                              (list usage-condition-of-node2) :test #'string-equal)
                                           ;;;;; We check to see if the rest of the usage condition ie. +B
69D belongs to the same
                                           ;;;;; DELETION OF each other, already happened. WE only take th
e car because we assume
                                           ;;;;; only  one feature group to take place now, ie. the B69D
                                            ((or
                                               (equal (length
                                                       (collect-all-usage-conditions-of-nodes
                                                         (function-which-collects-all-the-upward-nodes self
)))
                                                     2)
                                               (equal (collect-all-usage-conditions-of-nodes
                                                         (function-which-collects-all-the-upward-nodes self
))
                                                       '("ALL")))   ;;;;; ie. there is only one above with
ALL statement
                                             '("ALL"))
                                            (t

                                                (remove "ALL" other :test #'string-equal))))))

     ;;;;;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                 )
;         (print usage-condition-of-self)
;         (print (node-usage-condition  self))
;         (print usage-combination-of-node-before)
;         (print usage-combination-of-the-links-of-the-node)
;;        (print usage-combination-of-node-before)
;         (print usage-combination-of-the-links-of-the-node)
;;;       (print (combine-logically-the-feature-groups usage-condition-of-self))
;         (print usage-combination-of-the-links-of-the-node)
;         (print after-without-ALL)
          (cond
                ;;((and (equal usage-combination-of-the-links-of-the-node nil)

                ((equal usage-combination-of-the-links-of-the-node nil)     ;;;ie. +B69D  and -B69D delete
 each other ?
                 (setf (node-usage-condition  self)
                       "ALL"))
                ((equal usage-combination-of-the-links-of-the-node (list "ALL"))
                    (setf (node-usage-condition  self)
                       "ALL")) ;;;;;;;; It must be checked properly IF the result ALL is actually covers t
he "+B69D" ie.
                               ;;;;;;;; It came from the same combination
                ((some #'(lambda (x) (string-search "ALLA"   x)) usage-combination-of-the-links-of-the-nod
e)
                        ;;;;; in order to avoid cases such as ALLALL
                        (setf (node-usage-condition  self)
                       "ALL"))

                (t
                 (loop for feature-group in after-without-ALL
                       do
                   (setq katerina (string-append " " (string-append (string-append feature-group " ") kate
rina ))) )

                 (setf (node-usage-condition  self)
                       katerina)))
          (setf (node-radius self) :needs-calculation)
```

```
          ))))

)
;;;;;;;;;;;;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%

;;;;;;;;;;;Command: (combine-logically-the-feature-groups "-b34 +gty8 -kiop +b34")

(defun combine-logically-the-feature-groups (a-long-string-of-feature-groups)
  (let ((result " "))
    (let ((temp nil))

    (let ((help (get-words-from-string-other-new-old     a-long-string-of-feature-groups)))

      (loop for feature-group in help
            do
        (cond ((member feature-group (remove feature-group help)
                       :test #'check-if-the-feature-groups-delete-each-other)
               (push feature-group temp))))    ;;;;;temp ==  ("+b34" "-b34")


      (let ((remaining-groups  (remove-duplicates (set-difference help temp :test #'string-equal)
                                               :test #'string-equal)))  ;;; IN order to avoid +B69D ,
+B69D twice or more


  (loop for feature-group in remaining-groups
        do
    (setq result (string-append " " (string-append (string-append feature-group " ") result ))) ))))
  result))


(defun check-if-the-feature-groups-delete-each-other (a b ) ;;;(strig1 string2) "+b69D" "-B69D"
   (cond
     ((and
      (or (and (string-equal (subseq a 0 1) "+")
               (string-equal (subseq b 0 1) "-"))
          (and (string-equal (subseq a 0 1) "-")
               (string-equal (subseq b 0 1) "+")))
      (string-equal (car (get-words-from-string-other-new-plus-or-minus  (subseq a 1 4)))
                    (car (get-words-from-string-other-new-plus-or-minus  (subseq b 1 4))))))
   ))


(defun get-words-from-string-other-new-plus-or-minus (string)
  (let ((words nil))
   (flet ((whitespacep (char) (member char '(#\space  #\+ #\- ))))
      (do ((last-pos 0 (+ 1 pos))
           (pos (position-if #'whitespacep string)
                (position-if #'whitespacep string
                             :start (+ 1 pos))))
          ((null pos) (push (subseq string last-pos) words))
        (push (subseq string last-pos pos) words))
      (delete "" (nreverse words) :test #'string-equal))))


(defun get-words-from-string-other-new-old (string)
  (let ((words nil))
   (flet ((whitespacep (char) (member char '(#\space ))))

      (do ((last-pos 0 (+ 1 pos))
           (pos (position-if #'whitespacep string)
                (position-if #'whitespacep string
                             :start (+ 1 pos))))
          ((null pos) (push (subseq string last-pos) words))
        (push (subseq string last-pos pos) words))
      (delete "" (nreverse words) :test #'string-equal))))


;(defflavor node
;       ((arcs nil)                          ; list of arcs attached to this node
;        xpos                                ; coordinates of the center of this node
```

```
;        ypos
;               (TOTAL-QUANTITY 1) no-needed
;        (quantity 2)
;        (quantity-per-usage-for-one-vehicle 2)
;        (usage-condition "ALL")
;
;(defflavor arc
;       ((mark nil)    ; arbitrary object for misc tagging purposes
;        node1         ; the two "node" objects this arc connects
;        node2
;        (parts-per-assembly 1))
;       ()


;;;;*****************************************************************$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$


(define-grapher-command (com-add-arc-by-mouse :name "")
    ((node1 'node))
   (let ((node2 (accept 'node :prompt "click on the other Physical part")))
     (com-add-arc node1 node2)
     (setf node1 nil)))
```

```
(define-presentation-to-command-translator com-add-arc-from-here-to-there
                                            (node :gesture add-arc
                                                  :documentation "Create ASSEMBLY LINK from this Physical
Part")
  (node)
  (cp:build-command 'com-add-arc-by-mouse node))

(define-grapher-command (com-delete-node :menu-accelerator "Delete PART")
    ((node 'node :prompt "click on the Physical Part to delete"
           :provide-default nil :confirm nil))
   (when (tv:mouse-y-or-n-p
           (format nil "Do you really want to delete the part ~'I@A~⊃"
                   node))
     (delete-self node)))

(define-presentation-to-command-translator com-delete-node
  (node :gesture delete-node)
  (node)
  (cp:build-command 'com-delete-node node))

(define-grapher-command (com-delete-arc :menu-accelerator "Delete assyLINK")
    ((node1 'node :prompt "click on one Physical part")
     (node2 'node :prompt "click on the other Physical part"
           :provide-default nil))
   (let ((n-deleted 0))
     (map-over-arcs (arc)
       (when (or (and (eq (arc-node1 arc) node1)
                      (eq (arc-node2 arc) node2))
                 (and (eq (arc-node1 arc) node2)
                      (eq (arc-node2 arc) node1)))
         (delete-self arc)
         (incf n-deleted)))
     (when (= n-deleted 0)
       (format t "~&Didn't find an ASSEMBLY LINK between-
                 ~'I@A~⊃and ~'I@A~⊃
               node1 node2)))
   (let ((katerina  " "))

   (let ((temp-summ 0))
      (let ((arcs-of-node-which-the-link-is-put-TO (node-arcs node2)))

        (let ((result nil))
          (loop for arc in arcs-of-node-which-the-link-is-put-TO
                do
            (cond ((equal (arc-node2 arc) node2)

                  (push (list (arc-node1 arc) arc) result))))
          ;;;;;;(print result)

          (loop for accedant-node-and-its-belonging-arc in result
                  do
              (setq temp-summ (+ (* (node-quantity-per-usage-for-one-vehicle (car accedant-node-and-its-b
elonging-arc))
                                    ;;;;(node-quantity accedant-node))
                                    (arc-parts-per-assembly (cadr accedant-node-and-its-belonging-arc)))
                                 temp-summ)))
;           (setq temp-summ (+ (* (node-quantity-per-usage-for-one-vehicle node1)
;                                 (arc-parts-per-assembly *new-link*)) ;;;WE DON'T NEED IT as the link is
 already put,
;                              temp-summ))))                          ;;;; in NODE1  , hence it will be count
ed, anyway. !!!!!!
          ))
      ;;;;(setf (node-TOTAL-QUANTITY node2) temp-summ)))
      (setf (node-quantity-per-usage-for-one-vehicle node2) temp-summ))

   (let ((usage-condition-of-node2 " "))
      (let ((arcs-of-node-which-the-link-is-put-TO (node-arcs node2)))
        (let ((result nil))
          (loop for arc in arcs-of-node-which-the-link-is-put-TO
                do
                (push (list (arc-node1 arc) arc) result))
          (loop for accendant-node in result
                do
            (setq usage-condition-of-node2 (string-append (node-usage-condition (car accendant-node)) ;;
; because it comes as a list
```

```
                                                   usage-condition-of-node2       ;;;;; of first t
he node then the arcs
;                                        Error: The generic function NODE-USAGE-CONDITION
;                                        was applied to the argument (#<NODE cushion> #<ARC #  ←→ # 6001
10733>).
;                                        The argument was not of the correct type.
;                                        The function is implemented by the flavor NODE.
                                                   )
          ))))
   (let ((usage-combination-of-node-before (get-words-from-string-other-new-old (node-usage-condition
 node2)))
      (usage-combination-of-the-links-of-the-node
        (get-words-from-string-other-new-old (combine-logically-the-feature-groups usage-condition-
of-node2))))
    (let* ((other (remove-duplicates
                 (append usage-combination-of-node-before usage-combination-of-the-links-of-the-nod
e)
                :test #'string-equal))

;           (after-without-ALL (cond
;                                 ((and (> (length other) 1)
;                                      (member "ALL" other :test #'string-equal))
;                                  (remove '"ALL" other :test #'string-equal))
;                                 (t other)))
   ;;;;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        (after-without-ALL (cond
                              ((and (> (length other) 1)
                                    (member "ALL" other :test #'string-equal))
                                    (cond
                                      ;;;;the-rest-of-the-usage-statement-it-does-NOT-belong-to-the-
same-ALL-feature-group
;                                        (member
;                                         (car (remove "ALL" other :test #'string-equal))
;                                         (list usage-condition-of-node2) :test #'string-equal)
                                     ;;;;; We check to see if the rest of the usage condition ie. +B
69D belongs to the same
                                     ;;;;; DELETION OF each other, already happened. WE only take th
e car because we assume
                                     ;;;;; only  one feature group to take place now, ie. the B69D
                                      ((or
                                         (equal (length
                                                (collect-all-usage-conditions-of-nodes
                                                  (function-which-collects-all-the-upward-nodes node
2)))
                                               2)
                                         (equal (collect-all-usage-conditions-of-nodes
                                                  (function-which-collects-all-the-upward-nodes node
2))
                                              '("ALL")))   ;;;;; ie. there is only one above with
ALL statement
                                       '("ALL"))
                                      (t

                                          (remove "ALL" other :test #'string-equal))))))
   ;;;;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

          )
;       (print usage-combination-of-node-before)
;       (print usage-combination-of-the-links-of-the-node)
;       (print (combine-logically-the-feature-groups usage-condition-of-node2))
;       (print usage-combination-of-the-links-of-the-node)
;       (print after-without-ALL)
        (cond ((equal usage-combination-of-the-links-of-the-node nil)     ;;;ie. +B69D  and -B69D delete
 each other ?
                (setf (node-usage-condition  node2)
                      "ALL"))
              ((equal usage-combination-of-the-links-of-the-node (list "ALL"))
                  (setf (node-usage-condition  node2)
                      "ALL"))
              ((some #'(lambda (x) (string-search "ALLA"   x)) usage-combination-of-the-links-of-the-nod
e)
                      ;;;;; in order to avoid cases such as ALLALL
                      (setf (node-usage-condition  node2)
                      "ALL"))
              (t
```

```
                (loop for feature-group in after-without-ALL
                      do
                  (setq katerina (string-append " " (string-append (string-append feature-group " ") kate
rina ))) )

                (setf (node-usage-condition  node2)
                      katerina)))
          (setf (node-radius node2) :needs-calculation)
             )))

)  ;;; katerina


(let ((arcs-of-the-node2 (node-arcs node2))
        (nodes-to-the-downward-direction  nil))
    (loop for arc in arcs-of-the-node2
          do
      (cond ((equal (arc-node1 arc) node2)
                (push (arc-node2 arc)  nodes-to-the-downward-direction))))
   ;;;;;;;;;(print nodes-to-the-downward-direction)

   (recursion-on-the-network nodes-to-the-downward-direction)

;    (loop for i in nodes-to-the-downward-direction
;         do
;      (try-one-step-downwards i)))
)
)

;;;;*****************************************************************************************************************
%%%%%%%%%%%%%%%%%%%%%%

;;;;;;;*****************************************************************************************


(define-grapher-command (com-delete-arc-by-mouse :name nil)
    ((node1 'node))
   (let ((node2 (accept 'node :prompt "click on the other Physical Part")))
     (com-delete-arc node1 node2)))

(define-presentation-to-command-translator com-delete-arc-from-here-to-there
                                            (node :gesture delete-arc)
  (node)
  (cp:build-command 'com-delete-arc-by-mouse node))

(define-grapher-command (com-reset-database :menu-accelerator "Reset ASSEMBLY" :name nil)
    ()
   (when (tv:mouse-y-or-n-p
           "Do you really want to reset the Assembly structure?")
     (setq *all-the-nodes* nil)
     (setq *all-the-arcs nil)
     (make-instance 'node :label "MANUAL-LH"  :xpos 450 :ypos 50 :usage-condition " +B69D ")
     (make-instance 'node :label "MANUAL-RH"  :xpos 100 :ypos 50 :usage-condition " +B69D ")
     (make-instance 'node :label "cushion"  :xpos 250 :ypos 200)
     (make-instance 'node :label "overlay"  :xpos 250 :ypos 400))

)



;;;;;;;;;;; the new VERSION

(define-grapher-command
  (com-BUILD-NEW-ASSY :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'graph)))
     (send stream :clear-history)
  (cond (
  (values
    (dw:menu-choose '(("Yes" . T) ("No" . nil)) :prompt (zl:format nil "DO YOU CONSIDER interchangeabilit
y IN THE STRUCTURE OF THE NEW ASSEMBLY ?") :near-mode '(:point 100 700)))
  (setq *all-the-nodes* nil)
     (setq *all-the-arcs nil)
     (let ((help1
             (make-instance 'node :label "MANUAL-LH"  :xpos 450 :ypos 50 :usage-condition " -B69D "))
           (help2
```

```
            (make-instance 'node :label "MANUAL-RH"  :xpos 100 :ypos 50 :usage-condition " -B69D "))
          (help3
            (make-instance 'node :label "cushion"  :xpos 250 :ypos 200))
          (help4
            (make-instance 'node :label "overlay"  :xpos 250 :ypos 400)))
      (add-an-arc help1 help3) (add-an-arc help2 help3) (add-an-arc help3 help4)

      )
    (let ((help11
            (make-instance 'node :label "MANUAL-LH-heated"  :xpos 600 :ypos 50 :usage-condition " +B69D
"))
          (help12
            (make-instance 'node :label "MANUAL-RH-heated"  :xpos 250 :ypos 50 :usage-condition " +B69D
"))
          (help13
            (make-instance 'node :label "cushion"  :xpos 400 :ypos 200))
          (help14
            (make-instance 'node :label "overlay"  :xpos 400 :ypos 400)))
      ;;;;;(add-an-arc help11 help13) (add-an-arc help12 help13) (add-an-arc help13 help14)

      ))
       (t
        (setq *all-the-nodes* nil)
    (setq *all-the-arcs* nil)
    (let ((help1
            (make-instance 'node :label "MANUAL-LH"  :xpos 450 :ypos 50 :usage-condition " +B69D "))
          (help2
            (make-instance 'node :label "MANUAL-RH"  :xpos 100 :ypos 50 :usage-condition " +B69D "))
          (help3
            (make-instance 'node :label "cushion"  :xpos 250 :ypos 200))
          (help4
            (make-instance 'node :label "overlay"  :xpos 250 :ypos 400)))
      (add-an-arc help1 help3) (add-an-arc help2 help3) (add-an-arc help3 help4)
      )))))


(define-grapher-command (com-reset-database :menu-accelerator "Reset ASSEMBLY" :name nil)
    ()
   (when (tv:mouse-y-or-n-p
          "Do you really want to reset the Assembly structure?")
    (setq *all-the-nodes* nil)
    (setq *all-the-arcs* nil)
    (let ((help1
            (make-instance 'node :label "MANUAL-LH"  :xpos 450 :ypos 50 :usage-condition " +B69D "))
          (help2
            (make-instance 'node :label "MANUAL-RH"  :xpos 100 :ypos 50 :usage-condition " +B69D "))
          (help3
            (make-instance 'node :label "cushion"  :xpos 250 :ypos 200))
          (help4
            (make-instance 'node :label "overlay"  :xpos 250 :ypos 400)))
      ;;;(add-an-arc help1 help3) (add-an-arc help2 help3) (add-an-arc help3 help4)

)))




(defun  add-an-arc  (node1 node2)

   (if (eq node1 node2)
       (format t "~&Sorry, you can't make an ASSEMBLY LINK from a Physical Part to itself.")
       (make-instance 'arc :node1 node1 :node2 node2)
   ))
```

```
(defmacro define-grapher-mouse-gestures (&rest gesture-list)
  `(progn ,@
     (loop for (gesture mouse-char) in gesture-list
           collect `(setf (dw:mouse-char-for-gesture ',gesture)
                          ',mouse-char))))

(define-grapher-mouse-gestures
  (create-node-ex-nihilo    #\mouse-left)
  (add-arc                  #\mouse-middle)
  (delete-node              #\control-mouse-left)
  (delete-arc               #\control-mouse-middle)
  (set-node-label           #\mouse-left)
  (move-node                #\shift-mouse-left))
```

```
(define-grapher-command (com-delete-arc-by-mouse :name nil)
    ((node1 'node))
   (let ((node2 (accept 'node :prompt "click on the other Physical Part")))
     (com-delete-arc node1 node2)))

(define-presentation-to-command-translator com-delete-arc-from-here-to-there
                                            (node :gesture delete-arc)
  (node)
  (cp:build-command 'com-delete-arc-by-mouse node))
```

```
;(define-grapher-command
;  (com-SHOW-IMAGE :menu-accelerator t) ()
;
;   (statice:with-database (db *image-test-pathname*)
;                          (statice:with-transaction ()
;   (let ((stream (dw:get-program-pane 'graph)))
;
;     (let* ((node (accept 'node :stream stream))
;            (statice-image (from-node-return-the-image node)))
;
;             (let ((*graphic-window-for-image* (make-*graphic-window-for-image* ;(snapshot-name
;                                                                                ;  (the-actual-entity-sna
pshot object)))))
;                                                  ))) ;;; "IMAGE OF THE PART IN STATICE DATA BASE")))
;             (send *graphic-window-for-image* :clear-history)
;             (send *graphic-window-for-image* :expose)
;             ;;(graphics:with-room-for-graphics (*graphic-window-for-image*)
;               (graphics:draw-image statice-image  0 2 :stream *graphic-window-for-image* :scale 0.4);;)
;             (let ((help (make-instance 'triangle :name "Return")))
;               (display help *graphic-window-for-image*)
;               (present-triangle help *graphic-window-for-image*)
;               (cond ((accept 'triangle :stream *graphic-window-for-image* :prompt "Click on the circle t
o EXIT (or Return if default)")
;                          (send *graphic-window-for-image* :deactivate)))
;             ))
;  )) ;; lets
;)))
```

```
(define-grapher-command
  (com-SHOW-IMAGE :menu-accelerator t) ()

;   (statice:with-database (db *image-test-pathname*)
;                          (statice:with-transaction ()
   (let ((stream (dw:get-program-pane 'graph)))

     (let* ((node (accept 'node :stream stream))
            (statice-image (from-node-return-the-image node)))

             (let ((*graphic-window-for-image* (make-*graphic-window-for-image* ;(snapshot-name
                                                                                ;  (the-actual-entity-sna
```

```
pshot object)))))
                                                    ))) ;;; "IMAGE OF THE PART IN STATICE DATA BASE")))
            (send *graphic-window-for-image* :clear-history)
            (send *graphic-window-for-image* :expose)
            ;;(graphics:with-room-for-graphics (*graphic-window-for-image*)
              (graphics:draw-image statice-image  0 2 :stream *graphic-window-for-image* :scale 0.4);;)
            (let ((help (make-instance 'triangle :name "Return")))
              (display help *graphic-window-for-image*)
              (present-triangle help *graphic-window-for-image*)
              (cond ((accept 'triangle :stream *graphic-window-for-image* :prompt "Click on the circle t
o EXIT (or Return if default)")
                     (send *graphic-window-for-image* :deactivate)))
            ))
  )) ;; lets
);;))



(define-grapher-command (com-SHOW-IMAGE-by-mouse :name nil)
    () ;;; (nodel 'node)
   (com-show-image))

(define-presentation-to-command-translator com-SHOW-IMAGE-by-mouse
                                            (node :gesture show-image)
  (node)
  (cp:build-command 'com-SHOW-IMAGE-by-mouse node))



(define-grapher-command (com-reset-database :menu-accelerator "Reset ASSEMBLY" :name nil)
    ()
   (when (tv:mouse-y-or-n-p
          "Do you really want to  reset the Assembly structure?")
     (setq *all-the-nodes* nil)
     (setq *all-the-arcs* nil)
     (make-instance 'node :label "SEAT-LH-NO-HT"
                    :xpos 450 :ypos 50 :usage-condition " +B69D " :part-number "HAD12345xxx")
     (make-instance 'node :label "SEAT-RH-NO-HT"
                    :xpos 100 :ypos 50 :usage-condition " +B69D " :part-number "HAD22345xxx")
     (make-instance 'node :label "no-heated-squab"  :xpos 250 :ypos 200 :part-number "HAD22445xxx")
     (make-instance 'node :label "OVERLAY-no-heat"  :xpos 250 :ypos 400 :part-number "HAD12443xxx"))

)
```

```
(defmacro define-grapher-mouse-gestures (&rest gesture-list)
  `(progn ,@
     (loop for (gesture mouse-char) in gesture-list
           collect `(setf (dw:mouse-char-for-gesture ',gesture)
                          ',mouse-char))))

(define-grapher-mouse-gestures
  (create-node-ex-nihilo   #\mouse-left)
  (add-arc                 #\mouse-middle)
  (delete-node             #\control-mouse-left)
  (delete-arc              #\control-mouse-middle)
  (set-node-label          #\mouse-left)
  (move-node               #\shift-mouse-left)
  (SHOW-IMAGE              #\shift-mouse-middle))


;;&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&


(defun  make-*graphic-window-for-image* () ;;;;(part-description-STRING)


   ;;;;;(let ((temp (string-append "PART DECRIPTION:" "   " part-description-STRING)))
(setq *graphic-window-for-image* (tv:make-window 'dw:dynamic-window
                                :edges '(100 100 800 700) ;;;;;The actual superior '(64 43 1131 783)  ;
;;;;;;'(170 90 700 700)
                                                                       ;;;;(multiple-value-list (send  *clear-di
play* :inside-edges)
                                :more-p t
                                :margin-components
                                '((dw:margin-borders :thickness 5)
                                  (dw:margin-white-borders :thickness 2)
                                  (dw:margin-ragged-borders :thickness 2
                                                            :horizontal-too t)
                                  (dw:margin-scroll-bar :elevator-thickness 10
                                                        :margin :left)
                                  (dw:margin-scroll-bar :elevator-thickness 10
                                                        :margin :bottom)
                                  (dw:margin-whitespace :thickness 6
                                                        :margin :left)
                                  (dw:margin-label :string
                                                   "IMAGE OF THE PART IN STATICE DATA BASE";;;; ,temp
                                                   :margin :top))
)))


(defvar *graphic-window-for-image*  (make-*graphic-window-for-image*))



(defflavor triangle
        ((name nil))
        ()
  :readable-instance-variables
  :writable-instance-variables
  :initable-instance-variables)


(defmethod  (sys:print-self   triangle)   (stream  &rest  ignore)
  (print-node  self  stream))

(defmethod  (print-node   triangle) (stream)
  (format  stream     name))

(defmethod (display triangle) ( stream &rest arguments)
  (apply #'graphics:draw-circle 600 25 10
         :stream stream
         :filled t ;;nil
         :alu :draw arguments))


(defun present-triangle (triangle stream &rest arguments)
```

```
  (graphics:with-output-as-graphics-presentation
    (stream :object triangle
            :type 'triangle)
    (apply #'display triangle stream arguments)))

(defvar *image-test-pathname* "stat-fs:>takis>image-test")


;(defvar *all-snapshots-in-statice*
;  (let ((snapshots nil))
;    ;; Read all the snapshots into Lisp objects.
;    (with-database (db *image-test-pathname*)
;      (with-transaction ()
;       (for-each ((s snapshot))
;         (push s snapshots))))
;    snapshots))



;(defun from-node-return-the-image (node)
;  (BLOCK image
;  (statice:with-database (db *image-test-pathname*)
;    (statice:with-transaction ()
;     (loop for s in statice-user::*images-in-statice*
;
;          do
;       (cond ((string-search (string-trim '(#\space) (node-label node)) (car s))  (return-from image (se
cond s)))))))
;  nil))


(defun from-node-return-the-image (node)
  (cond ((string-search (string-trim '(#\space) (node-label node)) "no-heated-squab")
               (second common-lisp-user::*images*))
        ((string-search (string-trim '(#\space)  (node-label node)) "squab-heat")
         (sixth common-lisp-user::*images*))
        ((string-search (string-trim '(#\space)  (node-label node)) "OVERLAY-heat")
         (third common-lisp-user::*images*))
        ((string-search (string-trim '(#\space)  (node-label node)) "OVERLAY-no-heat")
         (fifth common-lisp-user::*images*))
        ((string-search (string-trim '(#\space)  (node-label node)) "heating-element")
         (fourth common-lisp-user::*images*))
;       ((string-search "seat-L"  (string-trim '(#\space)  (node-label node)) )
;        (eighth common-lisp-user::*images*))
        ((string-search "seat"  (string-trim '(#\space)  (node-label node)) )
         (seventh  common-lisp-user::*images*))

)
)
```

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER; Base: 10 -*-


;(make-package '3d :use '(symbolics-common-lisp common-lisp common-lisp-user))
;; SET PACKAGE ..... 3D

;;;;;;; In package 3D

;(defvar *list-of-1b-images* nil)
;(export '(*list-of-1b-images*))
;; SET PACKAGE ..... C-L-U

;;;;;;; In package C-L-U

;Command: Load File (file [default ATC-0:>takis>foo]) ATC-0:>takis>images>*.bin.*
;Loading ATC-0:>takis>images>seat-1b.bin.2 into package USER (really ZETALISP-USER)
;Loading ATC-0:>takis>images>spark-plug-1b.bin.2 into package USER (really ZETALISP-USER)
;Loading ATC-0:>takis>images>steering-wheel-1b.bin.2 into package USER (really ZETALISP-USER)
;Loading ATC-0:>takis>images>wheel-1b.bin.2 into package USER (really ZETALISP-USER)
; ... Done.
;Command:

;;;; Compile Buffer of this file ie.     "let-takis-play-with-images.lisp"     >takis>images ATC-0:

;;;; Command: 3d:*list-of-1b-images*
;(#<ART-1B-313-352 330042035> #<ART-1B-278-288 330042044> #<ART-1B-271-128 330042053> #<ART-1B-285-256 33
0042062>)
;Command:

;;;; Then I can use the files :
;;;;;    image-from-statice-examples.lisp >takis>images ATC-0:       and
;;;;;    bank.lisp >takis>images ATC-0:
;;;;;  to experiment  with images.

;;;;*****************************************************************************************

(defvar 3d:*list-of-1b-images* nil)

(defun display-images (&optional (stream *standard-output*))
  (loop for image in 3d:*list-of-1b-images* do
    (graphics:with-room-for-graphics (stream)
      (graphics:draw-image image 0 0 :stream stream))))
```

```
(defvar *base-record-format*
        '((:mmr 2)
          (:mmc 2)

          (:mrecno 3)

          (:mclass1 1)
          (:mclass2 1)
          (:mclass3 1)
          (:mclass4 1)
          (:mclass5 1)
          (:mclass6 1)
          (:mclass7 1)
          (:mclass8 1)
          (:mclass9 1)
          (:mclass10 1)
          (:mclass11 1)
          (:mclass12 1)
          (:mclass13 1)
          (:mclass14 1)
          (:mclass15 1)
          (:mclass16 1)

          (:mbody1 1)
          (:mbody2 1)
          (:mbody3 1)
          (:mbody4 1)
          (:mbody5 1)
          (:mbody6 1)


          (:mfam1 1)
          (:mfam2 1)
          (:mfam3 1)
          (:mfam4 1)
          (:mfam5 1)

          (:mcap1 1)
          (:mcap2 1)
          (:mcap3 1)
          (:mcap4 1)
          (:mcap5 1)
          (:mcap6 1)
          (:mcap7 1)

          (:mperf1 1)
          (:mperf2 1)
          (:mperf3 1)
          (:mperf4 1)
          (:mperf5 1)
          (:mperf6 1)
          (:mperf7 1)

          (:mdrive1 1)
          (:mdrive2 1)

          (:mtrans1 1)
          (:mtrans2 1)

          (:mauthin 12)
          (:mauthout 12)

          (:mrelin 1)
          (:mrelout 1)

          (:mdepin 8)
          (:mdepout 8)

          (:mtergrp1 4)
          (:mtergrp2 4)
          (:mtergrp3 4)
          (:mtergrp4 4)
          (:mtergrp5 4)
          (:mtergrp6 4)
          (:mtergrp7 4)
```

```
        (:mtergrp8 4)
        (:mtergrp9 4)
        (:mtergrp10 4)
        (:mtergrp11 4)
        (:mtergrp12 4)
        (:mtergrp13 4)
        (:mtergrp14 4)
        (:mtergrp15 4)
        (:mtergrp16 4)
        (:mtergrp17 4)
        (:mtergrp18 4)

        (:mterexc1 4)
        (:mterexc2 5)
        (:mterexc3 5)
        (:mtq1 1)
        (:mterexc4 4)
        (:mterexc5 5)
        (:mterexc6 5)
        (:mtq2 1)
        (:mterexc7 4)
        (:mterexc8 5)
        (:mterexc9 5)
        (:mtq3 1)
        (:mterexc10 4)
        (:mterexc11 5)
        (:mterexc12 5)
        (:mtq4 1)
        (:mterexc13 4)
        (:mterexc14 5)
        (:mterexc15 5)
        (:mtq5 1)
        (:mterexc16 4)
        (:mterexc17 5)
        (:mterexc18 5)

        (:mtq6 1)

        (:msales 3)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defvar *new-base-record-format*
       '((:mmr 2)
        (:mmc 2)

        (:mrecno 3)

        (:mclass1 1)
        (:mclass2 1)
        (:mclass3 1)
        (:mclass4 1)
        (:mclass5 1)
        (:mclass6 1)
        (:mclass7 1)
        (:mclass8 1)
        (:mclass9 1)
        (:mclass10 1)
        (:mclass11 1)
        (:mclass12 1)
        (:mclass13 1)
        (:mclass14 1)
        (:mclass15 1)
        (:mclass16 1)

        (:mbody1 1)
        (:mbody2 1)
        (:mbody3 1)
        (:mbody4 1)
        (:mbody5 1)
        (:mbody6 1)


        (:mfam1 1)
        (:mfam2 1)
```

```
;;; -*- Syntax: Common-lisp; Package: austin-rover; Base: 10; Mode: LISP -*-

(defun remove-nothing (ls)      ;;;; *common-ground-record*
  (remove "nothing" ls :key #'car :test #'string-equal))


;Command: (do-the-pure-RUNS-for-the-facility-HOW? (give-the-priority-for-the-SEARCH-of-ALGORITHM))
;((:|RECORD-OF-PURE-RUNS  2|
;  (("NOTHING" "B51K" "B69D" "NOTHING")
;   ("PLUS-AND-MINUS" "B51R" "B69D"
;    (("X" "W" " " " " " " " " "B" " ") ("T52J" "T53N" "T50R" "T57E" "T59E" "T59G") (("+" "B53A")))
;     (("X" "N" " " " " " " " " "B" " ") ("T52J" "T53N" "T50R" "T57E" "T59E" "T59G") (("+" "B53A")))))
;   ("PLUS-AND-MINUS" "B51L" "B69D" (("Y" " " " " " " " " "D" "B" " ") ("T52J" "T53N" "T50R" "T57E" "T59E" "
T59G") (("+" "B53B"))))))
;  )
; (:|RECORD-OF-PURE-RUNS  1|
;  (("NOTHING" "B51K" "B44R" "NOTHING")
;   ("PLUS" "B51K" "B44B"
;    (("M" " " " " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G" "T54C" "T56J"
;        "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E"
 "T59G" "T51R" "T51V"
;        "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R"
 "T56L" "T58C" "T60R"
;        "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T"
 "T56U" "T58N" "T57J"
;        "T58J")
;       (("+" "B53B")))
;      (("S" " " " " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G" "T54C" "T56J"
;        "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E"
 "T59G" "T51R" "T51V"
;        "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R"
 "T56L" "T58C" "T60R"
;        "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T"
 "T56U" "T58N" "T57J"
;        "T58J")
;       (("+" "B53B")))))
;   ("PLUS" "B51R" "B44R"
;    (("W" " " " " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G" "T54C" "T56J"
;        "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E"
 "T59G" "T51R" "T51V"
;        "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R"
 "T56L" "T58C" "T60R"
;        "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T"
 "T56U" "T58N" "T57J"
;        "T58J")
;       (("+" "B53B")))
;      (("X" "W" " " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G" "T54C" "T56J"
;        "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E"
 "T59G" "T51R" "T51V"
;        "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R"
 "T56L" "T58C" "T60R"
;        "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T"
 "T56U" "T58N" "T57J"
;        "T58J")
;       (("+" "B53A")))
;      (("X" "N" " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G" "T54C" "T56J"
;        "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E"
 "T59G" "T51R" "T51V"
;        "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R"
 "T56L" "T58C" "T60R"
;        "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T"
 "T56U" "T58N" "T57J"
;        "T58J")
;       (("+" "B53A")))))
;   ("NOTHING" "B51R" "B44B" "NOTHING")
```

```
;    ("PLUS" "B51L" "B44R"
;     (("Y" " " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G" "T54C" "T56J"
;        "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E"
 "T59G" "T51R" "T51V"
;        "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R"
 "T56L" "T58C" "T60R"
;        "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T"
 "T56U" "T58N" "T57J"
;        "T58J")
;       (("+" "B53B")))
;      (("Z" " " " " " " " " " " " " ") ("T51X" "T53V" "T53Z" "T57T" "T58L") (("+" "B53B")))))
;    ("NOTHING" "B51L" "B44B" "NOTHING"))))
;Command: *how*
;((:|RECORD-OF-COMMON-GROUND  2| (("B51K" "B44B") ("B51L" "B69D,B44R") ("B51R" "B69D,B44R") ("B51R" "B69D
,B44R")) NIL
;  (("PLUS" "B51K" "B44B"
;     (("M" " " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G" "T54C" "T56J"
;        "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E"
 "T59G" "T51R" "T51V"
;        "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R"
 "T56L" "T58C" "T60R"
;        "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T"
 "T56U" "T58N" "T57J"
;        "T58J")
;       (("+" "B53B")))
;      (("S" " " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G" "T54C" "T56J"
;        "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E"
 "T59G" "T51R" "T51V"
;        "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R"
 "T56L" "T58C" "T60R"
;        "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T"
 "T56U" "T58N" "T57J"
;        "T58J")
;       (("+" "B53B")))))
;    ("PLUS-AND-MINUS,PLUS" "B51L" "B69D,B44R"
;     (("Y" " " " " " " " "D" "B" " ") ("T52J" "T53N" "T50R" "T57E" "T59E" "T59G") (("+" "B53B")))))
;    ("PLUS-AND-MINUS,PLUS" "B51R" "B69D,B44R"
;     (("X" "N" " " " " " " " "B" " ") ("T52J" "T53N" "T50R" "T57E" "T59E" "T59G") (("+" "B53A")))))
;    ("PLUS-AND-MINUS,PLUS" "B51R" "B69D,B44R"
;     (("X" "W" " " " " " " " "B" " ") ("T52J" "T53N" "T50R" "T57E" "T59E" "T59G") (("+" "B53A")))))))
; (:|RECORD-OF-COMMON-GROUND  1| (("B51K" "B44B") ("B51R" "B44R") ("B51L" "B44R")) NIL
;  (("NOTHING" "B51K" "B44R" "NOTHING")
;    ("PLUS" "B51K" "B44B"
;     (("M" " " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G" "T54C" "T56J"
;        "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E"
 "T59G" "T51R" "T51V"
;        "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R"
 "T56L" "T58C" "T60R"
;        "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T"
 "T56U" "T58N" "T57J"
;        "T58J")
;       (("+" "B53B")))
;      (("S" " " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G" "T54C" "T56J"
;        "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E"
 "T59G" "T51R" "T51V"
;        "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R"
 "T56L" "T58C" "T60R"
;        "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T"
 "T56U" "T58N" "T57J"
;        "T58J")
;       (("+" "B53B")))))
;    ("PLUS" "B51R" "B44R"
;     (("W" " " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G" "T54C" "T56J"
;        "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E"
```

```
 "T59G" "T51R" "T51V"
;        "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R"
 "T56L" "T58C" "T60R"
;        "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T"
 "T56U" "T58N" "T57J"
;        "T58J")
;       (("+" "B53B")))
;      (("X" "W" " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G" "T54C" "T56J"
;        "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E"
 "T59G" "T51R" "T51V"
;        "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R"
 "T56L" "T58C" "T60R"
;        "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T"
 "T56U" "T58N" "T57J"
;        "T58J")
;       (("+" "B53A")))
;      (("X" "N" " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G" "T54C" "T56J"
;        "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E"
 "T59G" "T51R" "T51V"
;        "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R"
 "T56L" "T58C" "T60R"
;        "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T"
 "T56U" "T58N" "T57J"
;        "T58J")
;       (("+" "B53A")))))
;   ("NOTHING" "B51R" "B44B" "NOTHING")
;   ("PLUS" "B51L" "B44R"
;    (((Y" " " " " " " " " " " " ")
;       ("T55A" "T60A" "T53R" "T53X" "T56G" "T56Z" "T51A" "T52J" "T54R" "T56T" "T59N" "T55E" "T60M" "T53E"
 "T53G" "T54C" "T56J"
;        "T56V" "T57X" "T59L" "T59V" "T51X" "T53V" "T53Z" "T57T" "T58L" "T53N" "T50R" "T53C" "T57E" "T59E"
 "T59G" "T51R" "T51V"
;        "T60E" "T50A" "T50V" "T52R" "T55T" "T57G" "T57V" "T57Z" "T50N" "T55C" "T55L" "T51N" "T54A" "T55R"
 "T56L" "T58C" "T60R"
;        "T56C" "T54V" "T50Z" "T51G" "T52G" "T54N" "T55J" "T56E" "T58G" "T58R" "T50X" "T53A" "T54H" "T54T"
 "T56U" "T58N" "T57J"
;        "T58J")
;       (("+" "B53B")))
;      (("Z" " " " " " " " " " " " " " ") ("T51X" "T53V" "T53Z" "T57T" "T58L") (("+" "B53B")))))
;   ("NOTHING" "B51L" "B44B" "NOTHING"))))
;Command:
;
;


(defvar *HOW*  nil)
(defvar *pure-RUNS* nil)

(defvar *HOW-graphs* nil)
(defvar *pure-RUNS-graphs* nil)
 (defvar *table* nil)
(defvar *records-of-pure-runs-through-ALGORITHM*    0)




(defun the-function-which-shows-HOW ()
  (setq  *CHOICES* (finish-it-with-GRAPHICS *COMMON-GROUND-RECORD*))
  (do-the-pure-RUNS-for-the-facility-HOW?   (give-the-priority-for-the-SEARCH-of-ALGORITHM))
  (setq *HOW-graphs* (get-*how*-TOP *how*))
  (setq *pure-RUNS-graphs* (get-*pure-runs*-TOP *pure-RUNS*))
  (setq *table* (remove-duplicates (append *PURE-RUNS-GRAPHS* *HOW-GRAPHS*) :test #'equal)))



(defun reset-records-of-pure-runs-through-ALGORITHM-keyword-counter ()
  (setq *records-of-pure-runs-through-ALGORITHM* 0))
```

```
(defun make-new-records-of-pure-runs-through-ALGORITHM  ()
  (intern (format nil "RECORD-OF-PURE-RUNS  ~d"
                  (incf   *records-of-pure-runs-through-ALGORITHM*))
          (find-package 'keyword)))



(defun do-the-pure-RUNS-for-the-facility-HOW? (ls &optional (stream *standard-output*)) ;; list-of-l
ists
;                ls   =====    (give-the-priority-for-the-SEARCH-of-ALGORITHM)
;               like the function begin in the file :  "statice-the-program-the-menus.lisp >takis>statice
 ATC-0: "

  (let ((result nil))
    (loop for list-of-GROUP-comparisons   in   ls
          do
      (let ((the-common-ground-as-is-is-when-you-run-only-2-feature-groups nil))  ;;;; ie (run "b51" "b44
")
        (run (car list-of-GROUP-comparisons) (cadr list-of-GROUP-comparisons))
        (setq the-common-ground-as-is-is-when-you-run-only-2-feature-groups *common-ground-record*)
        (push (list (make-new-records-of-pure-runs-through-ALGORITHM)
                    the-common-ground-as-is-is-when-you-run-only-2-feature-groups) result)
        (setq *common-ground-record* nil)))
    (setq *pure-RUNS* result)))



;Command: (get-*how*-TOP *)
;((((("B51L+B44R"))
;  ((((CLASS** "TRIM LEVEL 6") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** NO-RESTRICTED) (TRANSMISION** NO-RESTRICTED))
;     ("SOUTH AFRICA" "PAKISTAN" "SRI LANKA" "NEPAL" "INDIA" "GUYANA" "FIJI" "BANGLADESH" "ST. KITTS NEVIS
" "SINGAPORE" "MALAYSIA"
;      "JAMAICA" "HONG KONG" "CYPRUS" "BERMUDA" "BARBADOS" "INDONESIA" "MALAWI" "ZAMBIA" "SEYCHELLES" "MAU
RITIUS" "KENYA"
;      "GRENADA" "BRUNEI" "JAPAN" "ISRAEL" "AUSTRALIA" "SAUDI ARABIA" "QATAR" "OMAN" "KUWAIT" "DUBAI" "BAH
RAIN" "ABU DHABI" "USA"
;      "CANADA" "CALIFORNIA" "SWITZERLAND" "SWEDEN" "NORWAY" "FINLAND" "AUSTRIA" "GERMANY" "SPAIN" "PORTUG
AL" "GREECE" "GIBRALTAR"
;      "CANARY ISLES" "TUNISIA" "TAHITI" "REUNION" "NEW CALEDONIA" "MARTINIQUE" "GUADELOUPE" "FRENCH GUIAN
A" "FRANCE" "YUGOSLAVIA"
;      "ITALY" "TAIWAN" "NETHERLANDS" "ICELAND" "DENMARK" "BELGIUM" "NEW ZEALAND" "MALTA" "GIBRALTAR-RHD"
"GERMANY-RHD"
;      "UNITED KINGDOM" "IRISH REPUBLIC")
;     (((("W I T H" "B53-SEAT SIDE ROLLS                ")))))))
; ((("B51R+B44R"))
;  ((((CLASS** "TRIM LEVEL 5") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** NO-RESTRICTED) (TRANSMISION** NO-RESTRICTED))
;     ("SOUTH AFRICA" "PAKISTAN" "SRI LANKA" "NEPAL" "INDIA" "GUYANA" "FIJI" "BANGLADESH" "ST. KITTS NEVIS
" "SINGAPORE" "MALAYSIA"
;      "JAMAICA" "HONG KONG" "CYPRUS" "BERMUDA" "BARBADOS" "INDONESIA" "MALAWI" "ZAMBIA" "SEYCHELLES" "MAU
RITIUS" "KENYA"
;      "GRENADA" "BRUNEI" "JAPAN" "ISRAEL" "AUSTRALIA" "SAUDI ARABIA" "QATAR" "OMAN" "KUWAIT" "DUBAI" "BAH
RAIN" "ABU DHABI" "USA"
;      "CANADA" "CALIFORNIA" "SWITZERLAND" "SWEDEN" "NORWAY" "FINLAND" "AUSTRIA" "GERMANY" "SPAIN" "PORTUG
AL" "GREECE" "GIBRALTAR"
;      "CANARY ISLES" "TUNISIA" "TAHITI" "REUNION" "NEW CALEDONIA" "MARTINIQUE" "GUADELOUPE" "FRENCH GUIAN
A" "FRANCE" "YUGOSLAVIA"
;      "ITALY" "TAIWAN" "NETHERLANDS" "ICELAND" "DENMARK" "BELGIUM" "NEW ZEALAND" "MALTA" "GIBRALTAR-RHD"
"GERMANY-RHD"
;      "UNITED KINGDOM" "IRISH REPUBLIC")
;     (((("W I T H" "B53-SEAT SIDE ROLLS                ")))))))
; ((("B51R+B44R+B69D") ("B51R+B44R-B69D"))
;  ((((CLASS** "TRIM LEVEL 51") (BODY** "5 DOOR SALOON") (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICT
ED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** "LEFT HAND      STEERING") (TRANSMISION** NO-RESTRICTED))
;     ("SWITZERLAND" "SWEDEN" "NORWAY" "AUSTRIA" "GERMANY" "DENMARK") ((("W I T H" "B53-SEAT SIDE ROLLS
            ")))))))
; ((("B51L+B44R+B69D") ("B51L+B44R-B69D"))
;  ((((CLASS** "TRIM LEVEL 6") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;      (PERFORMANCE** "MULTI POINT INJ STD PERFORM") (DRIVE** "LEFT HAND      STEERING") (TRANSMISION** NO
-RESTRICTED))
;     ("SWITZERLAND" "SWEDEN" "NORWAY" "AUSTRIA" "GERMANY" "DENMARK") ((("W I T H" "B53-SEAT SIDE ROLLS
            ")))))))
```

```
; (((("B51K+B44B"))
;  ((((CLASS** "TRIM LEVEL 3") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** NO-RESTRICTED) (TRANSMISION** NO-RESTRICTED))
;     ("SOUTH AFRICA" "PAKISTAN" "SRI LANKA" "NEPAL" "INDIA" "GUYANA" "FIJI" "BANGLADESH" "ST. KITTS NEVIS
" "SINGAPORE" "MALAYSIA"
;      "JAMAICA" "HONG KONG" "CYPRUS" "BERMUDA" "BARBADOS" "INDONESIA" "MALAWI" "ZAMBIA" "SEYCHELLES" "MAU
RITIUS" "KENYA"
;      "GRENADA" "BRUNEI" "JAPAN" "ISRAEL" "AUSTRALIA" "SAUDI ARABIA" "QATAR" "OMAN" "KUWAIT" "DUBAI" "BAH
RAIN" "ABU DHABI" "USA"
;      "CANADA" "CALIFORNIA" "SWITZERLAND" "SWEDEN" "NORWAY" "FINLAND" "AUSTRIA" "GERMANY" "SPAIN" "PORTUG
AL" "GREECE" "GIBRALTAR"
;      "CANARY ISLES" "TUNISIA" "TAHITI" "REUNION" "NEW CALEDONIA" "MARTINIQUE" "GUADELOUPE" "FRENCH GUIAN
A" "FRANCE" "YUGOSLAVIA"
;      "ITALY" "TAIWAN" "NETHERLANDS" "ICELAND" "DENMARK" "BELGIUM" "NEW ZEALAND" "MALTA" "GIBRALTAR-RHD"
"GERMANY-RHD"
;      "UNITED KINGDOM" "IRISH REPUBLIC")
;     ((("W I T H" "B53-SEAT SIDE ROLLS                ")))))))
;Command:
```

```
(defun get-*how*-TOP (ls)       ;;;;  *how*
  (let ((result nil) (help (remove-duplicates (get-*how* ls) :test #'equal)) (record nil))
    (loop for combination in help
          do
      (cond ((not (assoc (list (car combination) (second combination) (third combination))
                         record
                         :test #'equal))
                 (let ((temporary nil))  ;;;;;;(block THE-SAME-FEATURE-COMBINATIONS
                        (loop for i in help
                              do
                         (cond ((and
                                  (string-equal (car i) (car combination))
                                  (string-equal (cadr i) (cadr combination))
                                  (string-equal (third i) (third combination)))
                                (push i temporary))))
                        (cond ((equal (length temporary) 1)
                           (push (car temporary) result))
                        (t (let ((combination (list (caar temporary)
                                                    (second (car temporary))
                                                    (third (car temporary))))
                                 (all-different-restrictions nil))
                             (loop for k in temporary
                                   do
                               (setq all-different-restrictions (append (fourth k) all-different-restric
tions)))
                             (push (append combination
                                           (list all-different-restrictions))  ;;;; for the same format
                                   result)))
                  )
               )
              (push (list (list (car combination) (second combination) (third combination))
                          (fourth combination)) record)
          )    ;;;;;    )
    )   ;;;;;cond
  )

(finish-it-with-GRAPHICS result)))


(defun get-*how* (ls)   ;;;;  *how*
  (let ((result nil))
    (loop for j in ls
          do
      (loop for combination in (fourth j)
                do
              (push combination result)))
    (delete "nothing" result :test #'string-equal :key #'car)))
```

```
(defun get-*pure-runs*-TOP (ls)      ;;;;  *pure-runs*
  (let ((result nil) (help (remove-duplicates (get-*pure-RUNS* ls) :test #'equal)) (record nil))
    (loop for combination in help
          do
      (cond ((not (assoc (list (car combination) (second combination) (third combination))
                         record
                         :test #'equal))
                 (let ((temporary nil))  ;;;;;;(block THE-SAME-FEATURE-COMBINATIONS
                        (loop for i in help
                              do
                          (cond ((and
                                   (string-equal (car i) (car combination))
                                   (string-equal (cadr i) (cadr combination))
                                   (string-equal (third i) (third combination)))
                                 (push i temporary))))
                        (cond ((equal (length temporary) 1)
                           (push (car temporary) result))
                        (t (let ((combination (list (caar temporary)
                                                    (second (car temporary))
                                                    (third (car temporary))))
                                 (all-different-restrictions nil))
                             (loop for k in temporary
                                   do
                               (setq all-different-restrictions (append (fourth k) all-different-restric
tions)))
                             (push (append combination
                                           (list all-different-restrictions))  ;;;; for the same format
                                   result)))
                  )
               )
              (push (list (list (car combination) (second combination) (third combination))
                          (fourth combination)) record)
          )    ;;;;;    )
    )   ;;;;;cond
  )
    (finish-it-with-GRAPHICS result)))



(defun get-*pure-RUNS* (ls)   ;;;;  *pure-RUNS*
  (let ((result nil))
    (loop for j in ls
          do
      (loop for combination in (second j)
                do
              (push combination result)))
    (delete "nothing" result :test #'string-equal :key #'car)))


;;;;;;*****************************************************************************************
********************

(defflavor circle
        ((name nil))
        ()
  :readable-instance-variables
  :writable-instance-variables
  :initable-instance-variables)


(defmethod  (sys:print-self   circle)   (stream  &rest  ignore)
  (print-node  self  stream))

(defmethod  (print-node   circle) (stream)
  (format  stream     name))

(defmethod (display circle) ( stream &rest arguments)
  (apply #'graphics:draw-circle 15 15 10
         :stream stream
         :filled t ;;nil
         :alu :flip arguments))


(defun present-circle (circle stream &rest arguments)
```

```
  (graphics:with-output-as-graphics-presentation
    (stream :object circle
            :type 'circle)
    (apply #'display circle stream arguments)))


        ;;;;; ************************************************


;(defparameter *FRAMEWORK-DISPLAY-margin-components*
;              'dw:((margin-ragged-borders)
;                   (margin-scroll-bar  :history-noun "display")
;                   (margin-scroll-bar  :history-noun "display"
;                                       :margin :bottom)
;                   (margin-white-borders  :thickness 2)))
;
;
;(DW:DEFINE-PROGRAM-FRAMEWORK ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT
;  :SELECT-KEY
;  #\square
;  :COMMAND-DEFINER
;  T
;  :COMMAND-TABLE
;  (:INHERIT-FROM '("colon full command" "standard arguments" "input editor compatibility")
;   :KBD-ACCELERATOR-P 'NIL)
;  :STATE-VARIABLES
;  NIL
;  :PANES
;  ((PANE-2 :TITLE :HEIGHT-IN-LINES 1 :REDISPLAY-AFTER-COMMANDS NIL)
;   (PANE-3 :DISPLAY
;           :margin-components *FRAMEWORK-DISPLAY-margin-components*)
;
;   (PANE-4 :COMMAND-MENU :MENU-LEVEL :TOP-LEVEL) (PANE-5 :INTERACTOR))
;                                                        ;:margin-components *FRAMEWORK-DISPLAY-margin-co
mponents*
;                                                        ;:HEIGHT-IN-LINES 4))
;  :CONFIGURATIONS
;  '((DW::MAIN (:LAYOUT (DW::MAIN :COLUMN PANE-2 PANE-3 PANE-4 PANE-5))
;     (:SIZES
;      (DW::MAIN (PANE-2 1 :LINES)  ;;;(PANE-4 2 :lines)
;                (PANE-4 :ASK-WINDOW SELF :SIZE-FOR-PANE PANE-4)
;       (PANE-5 1 :LINES) :THEN (PANE-3 :EVEN))))))
;
;


(define-ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT-command
  (com-show-how-for-choice--- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-3)))
     (send stream :clear-history)
     (create-others-and-root stream)))


;(define-ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT-command
;  (com-show-DETAILS--- :menu-accelerator t) ()
;   (let ((stream (dw:get-program-pane 'pane-3))(char (accept 'Usage-statement)))
;     ;;;(send stream :clear-history)
;     (let* ((feature-string (Usage-statement-name char))  ;;;;"B51L,B44R,B69D"
;                            (list-of-the-feature-combinations (get-words-from-string-comma feature-stri
ng)))
;                                  ;;("B51L" "B44R" "B69D")
;                        (let ((FEATURE-COMBINATIONS-and-restrictions-to-be-drawn
;                                (BLOCK RESTRICTIONS
;                                  (loop for i in *table*
;                                        do
;                                    (let ((features (only-atoms (car i))))
;                                      (let ((list-of-features-in-lists
;                                              (loop for i in features
;                                                    collect
;                                                    (get-words-from-string-other i))))
;                                        (cond ((member list-of-the-feature-combinations list-of-features
-in-lists
;                                                       :test #'equal)
;                                              (return-from RESTRICTIONS i))))))
;                                nil)))
;                        (menu-tree feature-string FEATURE-COMBINATIONS-and-restrictions-to-be-drawn st
```

```
ream)
;                                   ))))


;(define-ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT-command
;  (com-show-HOW--- :menu-accelerator t) ()
;   (let ((stream (dw:get-program-pane 'pane-3)))
;     (send stream :clear-history)
;     (menu-tree "b51K"  stream)))
;     ;(draw-the-binary-tree stream)))

;;;;(defvar *clear-display* nil)

(define-ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT-command
  (com-RESET--- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-3)))
     (send stream :clear-history)
     ;;;(defvar *clear-display* nil)
     (the-function-which-shows-HOW)))



;(defun nested-formatting-2-NEGATIVES (tree &optional (stream *standard-output*))
;  (fresh-line)
;  (format-graph-from-root tree
;                          #'(lambda (node stream)
;                              (if (consp (first node))
;                                  (with-character-size (:large stream :bind-line-height t)
;                                    (princ "FEATURES-NEGATIVE-TO-THE-PART" stream))
;                                  (format-item-list node
;                                                    :stream stream
;                                                    :optimal-number-of-rows 2
;                                                    :fresh-line nil
;                                                    :additional-indentation 0)))
;                          #'(lambda (node)
;                              (and (consp (first node)) node))
;                          :stream stream))


(defun  make-*graphic-window-for-draw*  (feature-combination)


   (let ((temp (string-append "DETAIL INFORMATION FOR THE FEATURE COMBINATION" "  " feature-combination))
)
(setq *graphic-window-for-draw* (tv:make-window 'dw:dynamic-window
                                  :edges '(100 100 800 700) ;;;;;The actual superior '(64 43 1131 783)  ;
;;;;;;'(170 90 700 700)
                                                                  ;;;;(multiple-value-list (send  *clear-dis
play* :inside-edges)
                                  :more-p t
                                  :margin-components
                                  '((dw:margin-borders :thickness 5)
                                    (dw:margin-white-borders :thickness 2)
                                    (dw:margin-ragged-borders :thickness 2
                                                              :horizontal-too t)
                                    (dw:margin-scroll-bar :elevator-thickness 10
                                                          :margin :left)
                                    (dw:margin-scroll-bar :elevator-thickness 10
                                                          :margin :bottom)
                                    (dw:margin-whitespace :thickness 6
                                                          :margin :left)
                                    (dw:margin-label :string
                                                 ,temp
                                                 :margin :top))
))))


(defun menu-tree (feature-combination tree &optional (stream *standard-output*))
  (make-*graphic-window-for-draw* feature-combination)
  (send *graphic-window-for-draw* :expose) ;;; :near-mode t)
  (nested-formatting-2-more-detailed-data   (simplify-countries tree) *graphic-window-for-draw*)
  (let ((help (make-instance 'circle :name "EXIT")))
    (present-circle  help *graphic-window-for-draw*))
  (cond ((accept 'circle :stream *graphic-window-for-draw*
                 :prompt "CLICK ON THE BLACK CIRCLE ON THE LEFT TOP CORNER TO EXIT ,OR PRESS Return (if DEFAULT)")
```

```
        (send *graphic-window-for-draw* :bury))))



(defun simplify-countries (ls)  ;;;;;paul
  (let ((result nil))
    (loop for i in (cadr ls)   ;;;restrictions
          do
      (let ((countries-which-goes (second i)))
        (cond ((equal (length countries-which-goes) 69)   ;;;; countries
               (push (list (car i)
                           (list "TO-ALL-COUNTRIES")
                           (list (third i))) result))
              (t (push i result)))
        ))
    result))



(defun nested-formatting-2-more-detailed-data (tree &optional (stream *standard-output*))
  (fresh-line)
  (format-graph-from-root  tree
                           #'(lambda (node stream)
                               (if (consp (first node))
                                   (with-character-size (:large stream :bind-line-height t)
                                     (princ "*" stream))
                                   (format-item-list node
                                                     :stream stream
                                                     :optimal-number-of-rows 2
                                                     :fresh-line nil
                                                     :additional-indentation 0

                                                     )))
                           #'(lambda (node)
                               (and (consp (first node)) node))
                           :stream stream
                           :orientation :horizontal
                           :COLUMN-SPACING  50
;                          :within-column-spacing 50
                           )
  )


;;;*****************************************************************************************************
*****************
;Command:(create-others-and-root) ......
;......... A GRAPH TREE
;Enter the Lisp representation of an add-feat-combination: FEATURE COMBINATION: B51L+B44R+B69D-OR-B51L+B4
4R-B69D
;((("B51L" "B69D") ("B51L" "B44R") ("B51L" "B44R" "B69D") "B69D" "B44R" "B51L")
; ((("B51L" "B44R") ("B51L") ("B44R")) (("B51L" "B69D") ("B51L") ("B69D"))
;  (("B51L" "B44R" "B69D") ("B51L" "B44R") ("B51L" "B69D"))))
;Command:   ;;;;;;;;reference   ####

;;;;;;;;;;;;;;;;;  reference &&&&&&&
;((FEATURE-COMBINATION B51R,B69D ("B51R" "B69D")) (FEATURE-COMBINATION B51R,B44R ("B51R" "B44R"))
; (FEATURE-COMBINATION B51R,B44R,B69D ("B51R" "B44R" "B69D")) (FEATURE-COMBINATION B69D ("B69D"))
; (FEATURE-COMBINATION B44R ("B44R")) (FEATURE-COMBINATION B51R ("B51R")))
;((FEATURE-COMBINATION B51R,B69D ("B51R" "B69D")) (FEATURE-COMBINATION B51R,B44R ("B51R" "B44R"))
; (FEATURE-COMBINATION B51R,B44R,B69D ("B51R" "B44R" "B69D")) (FEATURE-COMBINATION B69D ("B69D"))
; (FEATURE-COMBINATION B44R ("B44R")) (FEATURE-COMBINATION B51R ("B51R")))
;Command:

(defun  get-words-from-string  (string)
  (let  ((words  nil))
    (do  ((last-pos  0  (+  1  pos))
          (pos
            (or
                (position #\-  string)
                (position #\+  string)
                (position #\space string))
            (or
               (position #\- string  :start (+ 1 pos))
               (position #\space string  :start (+ 1 pos)))))
```

```
        ((null pos)  (push (subseq  string  last-pos)  words))
      (push (subseq  string  last-pos  pos)  words))
    (nreverse
    (loop for word in words
          when (not (empty word))
            collect word)
    )
))

(defun empty (s)
  (not (find-if-not #'(lambda (a) (char-equal a #\space)) s)))


(defun check-if-tree-of-a-double-combination (feature-string)  ;;;; ie "B51K+B44B" or "B51L+B44R~OR~B51L-
B44R"
  (cond ((string-search "~OR~" feature-string)
           (let ((combination (car (get-words-from-string-~ feature-string)))) ;;;ie ("B51L+B44R" "OR" "B
51L-B44R")
             (cond ((equal (string-length combination) 9)))))
        (t (cond ((equal (string-length feature-string) 9))))))




(defun do-the-draw-tree-for-a-double-combination (feature-string &optional (stream *standard-output*))
  (setq *feature-combinations-for-DRAW* nil)
  (let ((takis
  (cond ((string-search "~OR~" feature-string)
          (let* ((combination (car (get-words-from-string-~ feature-string))) ;;;ie ("B51L+B44R" "OR" "B51
L-B44R")
                 (final (get-words-from-string combination)))
           (make-names-appending-strings final)))
        (t (make-names-appending-strings (get-words-from-string feature-string))))))
    (let (
;    (tree-structure (list (get-words-from-string-comma takis)
;                          (list (substring takis 0 4))
;                          (list (substring takis 5 9))))
          (flavor-names (list  (substring takis 0 4) (substring takis 5 9)))
          )
      (loop for i in flavor-names
            do
        (make-instance 'Usage-statement :name i))
      (let ((help  *feature-combinations-for-DRAW*))
        (let ((temp (make-instance 'Usage-statement :name takis :feature-combinations help)))
          (graph-tree-for-FEATURE-COMBINATIONS temp stream)
          (loop for char = (accept '(or ;;;(dw:member-sequence (#\return))
                                        (character)
                                        (Usage-statement))
                                   :prompt "ENTER THE FEAT-COMBINATION YOU WANT TO EXAMINE IN MORE DETAIL OR Return TO E
XIT" )


                  until (dw:compare-char-for-accept char #\return)
                  do
                (cond ((instancep char)
                       (let* ((feature-string (Usage-statement-name char))  ;;;;"B51L,B44R,B69D"
                              (list-of-the-feature-combinations (get-words-from-string-comma feature-stri
ng)))
                                  ;;("B51L" "B44R" "B69D")
                        (let ((FEATURE-COMBINATIONS-and-restrictions-to-be-drawn
                               (BLOCK RESTRICTIONS
                                 (loop for i in *table*
                                       do
                                   (let ((features (only-atoms (car i))))
                                     (let ((list-of-features-in-lists
                                            (loop for i in features
                                                  collect
                                                  (get-words-from-string-other i))))
                                       (cond ((member list-of-the-feature-combinations list-of-features
-in-lists
                                                      :test #'equal)
                                              (return-from RESTRICTIONS i))))))
                                 nil)))

                        (cond ((equal (string-length feature-string) 4)
```

```
                                                (get-the-string-of-only-one-feature-combination-in-STATICE feature-s
tring stream))
                                             (t
                                              (menu-tree feature-string FEATURE-COMBINATIONS-and-restrictions-to-be-d
rawn stream)))
                              ;;;;;  before it was ONLY (menu-tree feature-string FEATURE-COMBINATIONS-and-restrictio
ns-to-be-drawn stream)
                                    ))))
                 )
)))))


(defun create-others-and-root (&optional (stream *standard-output*))
  (setq *feature-combinations-for-DRAW* nil)
  (let ((help nil)
        (temp
          (loop for i in *choices*
                collect
                  (car i))))
          ;;;;;ie.. ((("B51R+B44R+B69D") ("B51R+B44R-B69D")) (("B51L+B44R+B69D") ("B51L+B44R-B69D")) (("B
51K+B44B")))
    (loop for i in temp
          do
      (push (one-or-two i) help))
    (setq *ADD-FEAT-COMBINATIONs* nil)
      (loop for i in help
            do
        (make-instance 'ADD-FEAT-COMBINATION :name i :feature-combinations nil))
      (let ((root (make-instance 'root :feature-combinations *ADD-FEAT-COMBINATIONs*)))
        (graph-tree-ROOT root stream)
        (let ((variable (ADD-FEAT-COMBINATION-name  (accept 'ADD-FEAT-COMBINATION
                                                          :prompt "Enter the FEATURE COMBINATION for which you w
ant to see its TRUTH MAINTENANE TREE !" ))))
          ;;;;;;(print variable stream) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
          (cond ((check-if-tree-of-a-double-combination variable)
                 (do-the-draw-tree-for-a-double-combination variable stream))
                (t
          (let* ((variable-which-keeps-both-flavor-names-and-tree-structure
                  (create-the-tree-of-the-string variable))    ;;;;;;;;reference   ####
                 (flavor-names (car variable-which-keeps-both-flavor-names-and-tree-structure))
                 (tree-structure (cadr variable-which-keeps-both-flavor-names-and-tree-structure)))
            (let ((table-of-flavor-names-with-their-equivalent-lists nil) (actual-flavors+lists nil))
              (loop for i in flavor-names
                    do
                (push (list (make-names-appending-strings i) i) table-of-flavor-names-with-their-equivale
nt-lists))
; (("B51L" "B51L") ("B44R" "B44R") ("B69D" "B69D") ("B51L,B44R,B69D" ("B51L" "B44R" "B69D")) ("B51L,B44R"
 ("B51L" "B44R"))
; ("B51L,B69D" ("B51L" "B69D")))
            (loop for i in table-of-flavor-names-with-their-equivalent-lists
                  do
              (push (list (make-instance 'Usage-statement :name (car i))
                          (cond ((atom (cadr i)) (list (cadr i)))
                                (t (cadr i))))
                    actual-flavors+lists))
            ;;;;;;(print actual-flavors+lists)  ;;;;;;reference &&&&&&&
            (loop for i in tree-structure  ;;  i === (("B51L" "B44R") ("B51L") ("B44R"))
                  do
              (setf (Usage-statement-feature-combinations (return-the-flavor-of-the-list (car i) actual-f
lavors+lists))
                    (list (return-the-flavor-of-the-list (second i) actual-flavors+lists)
                          (return-the-flavor-of-the-list (third i) actual-flavors+lists))))
            (let* ((final (loop for i in tree-structure
                               collect (list  (car i) (length (car i)))))
                   (root (return-the-flavor-of-the-list
                          (caar (sort final #'> :key #'second)) actual-flavors+lists)))

              (graph-tree-for-FEATURE-COMBINATIONS root stream)
              (loop for char = (accept '(or ;;;(dw:member-sequence (#\return))
                                         (character)
                                         (Usage-statement))
```

```
                                        :prompt "ENTER THE FEATURE COMBINATION YOU WANT TO EXAMINE IN MORE DETAIL OR Ret
urn TO EXIT")


                    until (dw:compare-char-for-accept char #\return)
                    do
                (cond ((instancep char)
                       (let* ((feature-string (Usage-statement-name char))  ;;;;"B51L,B44R,B69D"
                              (list-of-the-feature-combinations (get-words-from-string-comma feature-stri
ng)))
                                  ;;("B51L" "B44R" "B69D")
                         (let ((FEATURE-COMBINATIONS-and-restrictions-to-be-drawn
                                 (BLOCK RESTRICTIONS
                                   (loop for i in *table*
                                         do
                                     (let ((features (only-atoms (car i))))
                                       (let ((list-of-features-in-lists
                                               (loop for i in features
                                                     collect
                                                     (get-words-from-string-other i))))
                                         (cond ((or
                                                  (member list-of-the-feature-combinations list-of-featur
es-in-lists
                                                          :test #'equal)
                                                  (check-if-any-of-them-is-a-member list-of-the-feature-c
ombinations
                                                                                    list-of-features-in-l
ists))
                                                 (return-from RESTRICTIONS i))))))
                                   nil)))

                          (cond ((equal (string-length feature-string) 4)
                                   (get-the-string-of-only-one-feature-combination-in-STATICE feature-s
tring stream))
                                (t

                                 (menu-tree feature-string FEATURE-COMBINATIONS-and-restrictions-to-be-d
rawn stream)))

                  ;;;;;  before it was ONLY (menu-tree feature-string FEATURE-COMBINATIONS-and-restrictio
ns-to-be-drawn stream)

                         ))))

          );;;;loop
            )

            )
)
))   ;;;t  AND  COND
))))


;;;;;FROM the file    the-algorithm
;
;(defun CREATE-COMBINATIONS  (ls)
;  (cond
;    ((null    (cdr ls)) nil)
;    (t (cons (combination-for-one (car ls) ls)
;             (CREATE-COMBINATIONS (cdr ls))
;       )
;    )
;  )
;)
;
;
;
;(defun combination-for-one (atom  ls)
;  (let ((result nil))
;
;    (loop for i in (cdr ls)
;         do
;       (push (list atom i) result))
;result))
```

```
(defun check-if-any-of-them-is-a-member (ls1 ls2)  ;;'("b51r" "b44y" "b45u") '(("b44y" "b45u" "b51r")("b4
4y" "b45u" "b51r"))
  (let ((temp (combination-of-positions-TOP ls1)))  ;;;;;;;; IT NEEDS MORE WORK FOR MORE THAN THREE atoms
 in the list
    ;;;;;(print temp)
    (BLOCK flag
      (loop for i in  temp
            do
        (cond ((member i ls2 :test #'equal) (return-from flag t))))
      nil)))


(defun combination-of-positions-TOP (ls)
  (let ((result nil))
  (cond ((or (< (length ls) 3)
             (equal (length ls) 3)) (setq result (combination-of-positions ls)))
        ((equal (length ls) 4)

           (loop for i in ls
                 do
             (let ((atom (set-difference ls (list i))))
               (let ((help (combination-of-positions atom)))
                 (loop for j in help
                       do
                   (push (only-atoms (list i j)) result)))))))

  (remove-duplicates result :test #'equal)))



(defun  combination-of-positions (ls)
  (let ((temp (CREATE-COMBINATIONS ls)) (result nil) (help nil))
    (loop for i in temp
          do
      (loop for j in i
           do
           (push j help)))
    (loop for i in help
          do
      (let ((atom (set-difference ls i)))
        (push (append atom i)
              result)))
    (loop for i in help
          do
      (let ((atom (set-difference ls i)))
        (push (append  i atom)
              result)))
    (loop for i in result
          do
      (push (reverse i) result))
    (remove-duplicates result :test #'equal)))


(defun  get-the-string-of-only-one-feature-combination-in-STATICE (a-string stream) ;;; ie "b51L"
  (let* ((help (BLOCK REFERENCE
                 (loop for i in "grouped"
                       do
                   (let ((j (cadr i)))
                     (loop for k in j
                           do
                       (cond ((string-equal a-string (car k))
                                   (return-from REFERENCE k))))))
                 nil))
         (temp (list  ;;;;; TO HAVE THE SAME FORMAT FOR THE draw
                  (specify-for-each-specific-feature-with-a-pseudo-NIL-TO_DRAW   help)))
;   help   = (list   ("B51F"
;                            (("B51F" "Z" (#<Entity-Handle of FEATURE-CHART-ENTRY 83/1 400163706>) NIL)
;                             ("B51F" "Y" (#<Entity-Handle of FEATURE-CHART-ENTRY 83/2 400163520>) NIL)))
    )

        )    ;;;;
    ;;;;;;(print temp stream)
```

```
    (let ((katerina (finish-it-with-GRAPHICS  temp)))
      ;;;;;(print katerina stream))))

      (manu-tree (car help) (car katerina) stream))))


(defun get-words-from-string-comma (string)    ;;;;separate comment from FEATURE-GROUP code
  (let ((words nil))
   (flet ((whitespacep (char) (member char '( #\,)  :test #'string-equal)))

      (do ((last-pos 0 (+ 1 pos))
           (pos (position-if #'whitespacep string)
                (position-if #'whitespacep string
                             :start (+ 1 pos))))
          ((null pos) (push (subseq string last-pos) words))
        (push (subseq string last-pos pos) words))
      (delete "" (nreverse words) :test #'string-equal))))


(defun return-the-flavor-of-the-list (ls1  ls2)   ;;;;ls2  ==== actual-flavors+lists
  (block FLAVOR
    (loop for i in ls2
          do
      (cond
        ((equal (cadr i) ls1) (return-from FLAVOR (car i)))))
     nil))

;;;;;;;;;;        ***************************************************


(defflavor  Feature-combination
        () () )


(defflavor  Usage-statement
        (name
         (feature-combinations  nil))
        (Feature-combination)
  :writable-instance-variables
  :initable-instance-variables)


(defvar  *feature-combinations-for-DRAW*  nil)


(defmethod  (sys:print-self   Usage-statement)   (stream  &rest  ignore)
  (print-node  self  stream))

(defmethod (make-instance  Usage-statement)   (&rest  ignore)
           (push self *feature-combinations-for-DRAW*))


(defmethod  (print-node   Usage-statement) (stream)
  (format  stream  "FEATURE-COMBINATION ~A"  name))


(defmethod  (node-inferiors   Usage-statement)  ()
  feature-combinations)


(defun  graph-tree-for-FEATURE-COMBINATIONS  (Usage-statement &optional   (stream  *standard-output*))
            (fresh-line  stream)
  (format-graph-from-root   Usage-statement
                            #'princ
                            #'node-inferiors
                            :dont-draw-duplicates t
                            :orientation  :vertical     ;;;;:horizontal
                            :stream  stream))
```

```
;;;;;;;;;;          ************************************************


(defun make-names-appending-strings (ls)
  (let ((result ""))
    (cond ((atom ls) (setq result ls))
          (t (loop for j in ls
                   do
                 (setq result (string-append result j ",")))))
    (cond ((not (atom ls)) (substring result  0 (- (string-length result) 1)))
          (t result))))


;;;;Command: (create-the-tree-of-the-string "B51L+B44R+B69D+b45y")
;;;;;; THE FUNCTION :   create-the-tree-of-the-string  does not work for two
;;;;;   ;;;;   variable  =  "B51R+B44R-B45B-B69D-OR-B51R+B44R-B45B+B69D-OR-B51R+B44R+B45B-B69D-OR-B51R+B4
4R+B45B+B69D"

(defun   create-the-tree-of-the-string  (a-string)   ;;;;ie the variable from the function    "create-oth
ers-and-root"...
                                  ;;;;;;;;;;;   ie   "B51L+B44R+B69D-OR-B51L+B44R-B69D"

  (let ((final-result nil))
  (let ((temporary (cond ((string-equal (car (get-words-from-string-- a-string))
                                        a-string) (get-words-from-string-other a-string)) ;;;ie THERE is
NOT  + and/OR -
                        (t (get-words-from-string-other (car (get-words-from-string-- a-string)))))))
    ;;;;(print temporary)
    (let ((help
             (get-the-string-create-the-tree (car temporary) (reverse (cdr temporary)))) ;!!!!!!!!!!!!!!!
!!!!!   ???????
          (combinations nil))
      (loop for i in help
            do
        (loop for j in i
              do
          (push j combinations)))
 ;;;;(("B51L" "b45y") ("B51L" "B69D") ("B51L" "B69D" "b45y") ("B51L" "B69D" "b45y") ("B51L" "B44R") ("B51
L" "B44R" "B69D" "b45y"))
      (let ((ready-for-flavors (remove-duplicates combinations :test #'equal))
            (doubles nil))
        (loop for i in ready-for-flavors
              do
          (cond ((equal (length i) 2)
                    (push i doubles))))  ;;;;;;(("B51L" "B44R") ("B51L" "B69D") ("B51L" "b45y"))
        (let ((more-left-right nil))
                (loop for i in doubles
                      do
                  (push (list i (list (car i)) (list (cadr i)))
                        more-left-right))
;;;;(((("B51L" "b45y") ("B51L") ("b45y")) (("B51L" "B69D") ("B51L") ("B69D")) (("B51L" "B44R") ("B51L") ("B
44R")))
       (loop for u in more-left-right
             do
         (push u help))
;;;;HELP
;;;;((("B51L" "B44R") ("B51L") ("B44R")) (("B51L" "B69D") ("B51L") ("B69D")) (("B51L" "b45y") ("B51L") ("
b45y"))
;;;;("B51L" "B44R" "B69D" "b45y") ("B51L" "B44R") ("B51L" "B69D" "b45y")) (("B51L" "B69D" "b45y") ("B51L"
"B69D") ("B51L" "b45y")))
        (let ((singles nil))
          (loop for i in doubles
                do
            (loop for j in i
                  do
              (push j singles)))
          (setq  final-result (list
                                (remove-duplicates (append ready-for-flavors singles) :test #'equal)
                                help))
        ;;;;;
;;;;;;;(("B51L" "b45y") ("B51L" "B69D") ("B51L" "B69D" "b45y") ("B51L" "B44R") ("B51L" "B44R" "B69D" "b45y
") "b45y" "B69D" "B44R"
; "B51L")
))
)))
```

```
final-result))


;Command: (get-the-string-create-the-tree 'b511 '(b44r b69d b45b))
;(((B51L B44R B69D B45B) (B51L B44R) (B51L B69D B45B)) ((B51L B69D B45B) (B51L B69D) (B51L B45B)))

(defun  get-the-string-create-the-tree (master-atom  ls)  ;;   (master-string   the-rest-of-the-string)

  (prog (result help left right)
        (setq help ls)
     LOOP
        (cond ((atom (cdr help)) (return (reverse result)))
              (t (setq left (list master-atom (car help)))
                 (setq right (only-atoms (list master-atom (cdr help))))
                 (setq result (cons  (list (only-atoms (list master-atom  help)) left right) result))))
        (setq help (cdr help))
        (go LOOP)))

            ;;;;;;;;;;;;; ************************************************



(defflavor  root
        (
         (feature-combinations  nil))
        (Feature-combination)
  :writable-instance-variables
  :initable-instance-variables)


(defmethod  (sys:print-self   root)   (stream  &rest  ignore)
  (print-node  self  stream))


(defmethod  (print-node   root) (stream)
  (format  stream  "ROOT OF FEATURE COMBINATIONS FOR MODEL -A"  *model* ))

(defmethod  (node-inferiors   root)  ()
  feature-combinations)


(defvar *ADD-FEAT-COMBINATIONs*  nil)

(defflavor  ADD-FEAT-COMBINATION
        (name
         (feature-combinations  nil))
        (Feature-combination)
  :writable-instance-variables
  :initable-instance-variables)


(defmethod  (sys:print-self  ADD-FEAT-COMBINATION)   (stream  &rest  ignore)
  (print-node  self  stream))

(defmethod (make-instance  ADD-FEAT-COMBINATION)   (&rest  ignore)
           (push self *ADD-FEAT-COMBINATIONs*))

(defmethod  (print-node   ADD-FEAT-COMBINATION ) (stream)
  (format  stream  "FEATURE COMBINATION: -A" name ))

(defmethod  (node-inferiors   ADD-FEAT-COMBINATION)  ()
  feature-combinations)


(defun  graph-tree-ROOT  (root &optional   (stream  *standard-output*))
           (fresh-line  stream)
  (format-graph-from-root   root
                            #'princ
                            #'node-inferiors
                            :dont-draw-duplicates t
                            :orientation :horizontal
```

```
                                :stream  stream
;                               :row-spacing 10
;                               :within-row-spacing 20
))

            ;;;;;;;;;;;;;; ***************************************************


;;*****************************************************************************************************
*****************
;;;;;*************************************************************


;Command: "table"
;((((("B51R+B44R+B69D") ("B51R+B44R-B69D"))
;  ((((CLASS** "TRIM LEVEL 51") (BODY** "3 DOOR SALOON") (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICT
ED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** "LEFT HAND      STEERING") (TRANSMISION** NO-RESTRICTED))
;     ("SWITZERLAND" "SWEDEN" "NORWAY" "AUSTRIA" "GERMANY" "DENMARK") ((("W I T H" "B53-SEAT SIDE ROLLS"))
))))
; ((("B51L+B69D") ("B51L-B69D"))
;  ((((CLASS** "TRIM LEVEL 6") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;      (PERFORMANCE** "MULTI POINT INJ STD PERFORM") (DRIVE** "LEFT HAND      STEERING") (TRANSMISION** NO
-RESTRICTED))
;     ("SWITZERLAND" "SWEDEN" "NORWAY" "AUSTRIA" "GERMANY" "DENMARK") ((("W I T H" "B53-SEAT SIDE ROLLS"))
))))
; ((("B51R+B69D") ("B51R-B69D"))
;  ((((CLASS** "TRIM LEVEL 51") (BODY** "5 DOOR SALOON") (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICT
ED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** "LEFT HAND      STEERING") (TRANSMISION** NO-RESTRICTED))
;     ("SWITZERLAND" "SWEDEN" "NORWAY" "AUSTRIA" "GERMANY" "DENMARK") ((("W I T H" "B53-SEAT SIDE ROLLS"))
))))
; ((("B51L+B44R"))
;  ((((CLASS** "TRIM LEVEL 6") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** NO-RESTRICTED) (TRANSMISION** NO-RESTRICTED))
;     ("SOUTH AFRICA" "PAKISTAN" "SRI LANKA" "NEPAL" "INDIA" "GUYANA" "FIJI" "BANGLADESH" "ST. KITTS NEVIS
" "SINGAPORE" "MALAYSIA"
;      "JAMAICA" "HONG KONG" "CYPRUS" "BERMUDA" "BARBADOS" "INDONESIA" "MALAWI" "ZAMBIA" "SEYCHELLES" "MAU
RITIUS" "KENYA"
;      "GRENADA" "BRUNEI" "JAPAN" "ISRAEL" "AUSTRALIA" "SAUDI ARABIA" "QATAR" "OMAN" "KUWAIT" "DUBAI" "BAH
RAIN" "ABU DHABI" "USA"
;      "CANADA" "CALIFORNIA" "SWITZERLAND" "SWEDEN" "NORWAY" "FINLAND" "AUSTRIA" "GERMANY" "SPAIN" "PORTUG
AL" "GREECE" "GIBRALTAR"
;      "CANARY ISLES" "TUNISIA" "TAHITI" "REUNION" "NEW CALEDONIA" "MARTINIQUE" "GUADELOUPE" "FRENCH GUIAN
A" "FRANCE" "YUGOSLAVIA"
;      "ITALY" "TAIWAN" "NETHERLANDS" "ICELAND" "DENMARK" "BELGIUM" "NEW ZEALAND" "MALTA" "GIBRALTAR-RHD"
"GERMANY-RHD"
;      "UNITED KINGDOM" "IRISH REPUBLIC")
;     ((("W I T H" "B53-SEAT SIDE ROLLS"))))))
; ((("B51R+B44R"))
;  ((((CLASS** "TRIM LEVEL 5") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** NO-RESTRICTED) (TRANSMISION** NO-RESTRICTED))
;     ("SOUTH AFRICA" "PAKISTAN" "SRI LANKA" "NEPAL" "INDIA" "GUYANA" "FIJI" "BANGLADESH" "ST. KITTS NEVIS
" "SINGAPORE" "MALAYSIA"
;      "JAMAICA" "HONG KONG" "CYPRUS" "BERMUDA" "BARBADOS" "INDONESIA" "MALAWI" "ZAMBIA" "SEYCHELLES" "MAU
RITIUS" "KENYA"
;      "GRENADA" "BRUNEI" "JAPAN" "ISRAEL" "AUSTRALIA" "SAUDI ARABIA" "QATAR" "OMAN" "KUWAIT" "DUBAI" "BAH
RAIN" "ABU DHABI" "USA"
;      "CANADA" "CALIFORNIA" "SWITZERLAND" "SWEDEN" "NORWAY" "FINLAND" "AUSTRIA" "GERMANY" "SPAIN" "PORTUG
AL" "GREECE" "GIBRALTAR"
;      "CANARY ISLES" "TUNISIA" "TAHITI" "REUNION" "NEW CALEDONIA" "MARTINIQUE" "GUADELOUPE" "FRENCH GUIAN
A" "FRANCE" "YUGOSLAVIA"
;      "ITALY" "TAIWAN" "NETHERLANDS" "ICELAND" "DENMARK" "BELGIUM" "NEW ZEALAND" "MALTA" "GIBRALTAR-RHD"
"GERMANY-RHD"
;      "UNITED KINGDOM" "IRISH REPUBLIC")
;     ((("W I T H" "B53-SEAT SIDE ROLLS"))))))
; ((("B51R+B44R+B69D") ("B51R+B44R-B69D"))
;  ((((CLASS** "TRIM LEVEL 51") (BODY** "5 DOOR SALOON") (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICT
ED)
;      (PERFORMANCE** NO-RESTRICTED) (DRIVE** "LEFT HAND      STEERING") (TRANSMISION** NO-RESTRICTED))
```

```
;    ("SWITZERLAND" "SWEDEN" "NORWAY" "AUSTRIA" "GERMANY" "DENMARK") ((("W I T H" "B53-SEAT SIDE ROLLS"))
))))
; ((("B51L+B44R+B69D") ("B51L+B44R-B69D"))
;  ((((CLASS** "TRIM LEVEL 6") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;     (PERFORMANCE** "MULTI POINT INJ STD PERFORM") (DRIVE** "LEFT HAND      STEERING") (TRANSMISION** NO
-RESTRICTED))
;    ("SWITZERLAND" "SWEDEN" "NORWAY" "AUSTRIA" "GERMANY" "DENMARK") ((("W I T H" "B53-SEAT SIDE ROLLS"))
))))
; ((("B51K+B44B"))
;  ((((CLASS** "TRIM LEVEL 3") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;     (PERFORMANCE** NO-RESTRICTED) (DRIVE** NO-RESTRICTED) (TRANSMISION** NO-RESTRICTED))
;    ("SOUTH AFRICA" "PAKISTAN" "SRI LANKA" "NEPAL" "INDIA" "GUYANA" "FIJI" "BANGLADESH" "ST. KITTS NEVIS
" "SINGAPORE" "MALAYSIA"
;     "JAMAICA" "HONG KONG" "CYPRUS" "BERMUDA" "BARBADOS" "INDONESIA" "MALAWI" "ZAMBIA" "SEYCHELLES" "MAU
RITIUS" "KENYA"
;     "GRENADA" "BRUNEI" "JAPAN" "ISRAEL" "AUSTRALIA" "SAUDI ARABIA" "QATAR" "OMAN" "KUWAIT" "DUBAI" "BAH
RAIN" "ABU DHABI" "USA"
;     "CANADA" "CALIFORNIA" "SWITZERLAND" "SWEDEN" "NORWAY" "FINLAND" "AUSTRIA" "GERMANY" "SPAIN" "PORTUG
AL" "GREECE" "GIBRALTAR"
;     "CANARY ISLES" "TUNISIA" "TAHITI" "REUNION" "NEW CALEDONIA" "MARTINIQUE" "GUADELOUPE" "FRENCH GUIAN
A" "FRANCE" "YUGOSLAVIA"
;     "ITALY" "TAIWAN" "NETHERLANDS" "ICELAND" "DENMARK" "BELGIUM" "NEW ZEALAND" "MALTA" "GIBRALTAR-RHD"
"GERMANY-RHD"
;     "UNITED KINGDOM" "IRISH REPUBLIC")
;    ((("W I T H" "B53-SEAT SIDE ROLLS")))))))
;Command:
;
;


;Command: *choices*
;(((("B51R+B44R+B69D") ("B51R+B44R-B69D"))
;  ((((CLASS** "TRIM LEVEL 51") (BODY** "5 DOOR SALOON") (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICT
ED)
;     (PERFORMANCE** NO-RESTRICTED) (DRIVE** "LEFT HAND      STEERING") (TRANSMISION** NO-RESTRICTED))
;    ("SWITZERLAND" "SWEDEN" "NORWAY" "AUSTRIA" "GERMANY" "DENMARK") ((("W I T H" "B53-SEAT SIDE ROLLS"))
))
;   (((CLASS** "TRIM LEVEL 51") (BODY** "3 DOOR SALOON") (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICT
ED)
;     (PERFORMANCE** NO-RESTRICTED) (DRIVE** "LEFT HAND      STEERING") (TRANSMISION** NO-RESTRICTED))
;    ("SWITZERLAND" "SWEDEN" "NORWAY" "AUSTRIA" "GERMANY" "DENMARK") ((("W I T H" "B53-SEAT SIDE ROLLS"))
))))
; ((("B51L+B44R+B69D") ("B51L+B44R-B69D"))
;  ((((CLASS** "TRIM LEVEL 6") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;     (PERFORMANCE** "MULTI POINT INJ STD PERFORM") (DRIVE** "LEFT HAND      STEERING") (TRANSMISION** NO
-RESTRICTED))
;    ("SWITZERLAND" "SWEDEN" "NORWAY" "AUSTRIA" "GERMANY" "DENMARK") ((("W I T H" "B53-SEAT SIDE ROLLS"))
))))
; ((("B51K+B44B"))
;  ((((CLASS** "TRIM LEVEL 3") (BODY** NO-RESTRICTED) (FAMILY** NO-RESTRICTED) (CAPACITY** NO-RESTRICTED)
;     (PERFORMANCE** NO-RESTRICTED) (DRIVE** NO-RESTRICTED) (TRANSMISION** NO-RESTRICTED))
;    ("SOUTH AFRICA" "PAKISTAN" "SRI LANKA" "NEPAL" "INDIA" "GUYANA" "FIJI" "BANGLADESH" "ST. KITTS NEVIS
" "SINGAPORE" "MALAYSIA"
;     "JAMAICA" "HONG KONG" "CYPRUS" "BERMUDA" "BARBADOS" "INDONESIA" "MALAWI" "ZAMBIA" "SEYCHELLES" "MAU
RITIUS" "KENYA"
;     "GRENADA" "BRUNEI" "JAPAN" "ISRAEL" "AUSTRALIA" "SAUDI ARABIA" "QATAR" "OMAN" "KUWAIT" "DUBAI" "BAH
RAIN" "ABU DHABI" "USA"
;     "CANADA" "CALIFORNIA" "SWITZERLAND" "SWEDEN" "NORWAY" "FINLAND" "AUSTRIA" "GERMANY" "SPAIN" "PORTUG
AL" "GREECE" "GIBRALTAR"
;     "CANARY ISLES" "TUNISIA" "TAHITI" "REUNION" "NEW CALEDONIA" "MARTINIQUE" "GUADELOUPE" "FRENCH GUIAN
A" "FRANCE" "YUGOSLAVIA"
;     "ITALY" "TAIWAN" "NETHERLANDS" "ICELAND" "DENMARK" "BELGIUM" "NEW ZEALAND" "MALTA" "GIBRALTAR-RHD"
"GERMANY-RHD"
;     "UNITED KINGDOM" "IRISH REPUBLIC")
;    ((("W I T H" "B53-SEAT SIDE ROLLS")))))))
;Command:



;;;;************************************************************************************************
******************

(defun specify-for-each-specific-feature-with-a-pseudo-NIL-TO_DRAW (a-feature-reference)
```

```
;  a-feature-reference    =    ("B51F"
;                              (("B51F" "Z" (#<Entity-Handle of FEATURE-CHART-ENTRY 83/1 400163706>) NIL)
;                               ("B51F" "Y" (#<Entity-Handle of FEATURE-CHART-ENTRY 83/2 400163520>) NIL)))
  (let* ((help (cadr (all-trims-common-with-RESTRICTIONS a-feature-reference nil))) (result nil)
         (PLUS-OR-MINUS  (car help)))   ;;;;(give-the-PLUS-OR-MINUS-afterwards  a-feature-reference)))
;;;;;;  help    ======
;;'((NIL NIL)       ;;;;  The reference from the Pseudo nil  Feature I manually  put  !!!
; ("B51F"
;  (("Y" (#<Entity-Handle of FEATURE-CHART-ENTRY 83/2 400163520>)) ("Z" (#<Entity-Handle of FEATURE-CHART
-ENTRY 83/1 400163706>)))
;  ))
    ;;;(print help)
    (let ((the-reference-of-MASTER-Feature
            (create-the-logic-references help)))
      (loop for MASTER-POSSIBILITY in the-reference-of-MASTER-Feature
            do
        (let* ((split-of-Master (part-apart-the-list MASTER-POSSIBILITY))
               (Master-Base-Features (car split-of-Master))
               (Master-Restrictions (cadr split-of-Master))
               (GROUPS-AVAILABLE-FOR-Master (GROUP-RESTRICTIONS-collect
                                             Master-Restrictions))
               (TERRITORIES-AVAILABLE-FOR-Master (TERRITORY-RESTRICTIONS-collect
                                                  Master-Restrictions))
               (COMBINATIONS-or-COMMENTS-of-Master
                 (COMBINATIONS-or-COMMENTS-RESTRICTIONS-collect Master-Restrictions))
               (final-the-COUNTRIES-AVAILABLE-for-MASTER
                  (set-difference
                           GROUPS-AVAILABLE-FOR-Master TERRITORIES-AVAILABLE-FOR-Master
                           :test #'string-equal))
               (BASE-FEATURES-to-be-compared
                 (reverse       ;;BECAUSE IT comes in OPPOSITE direction
                   (NO-opposite  Master-Base-Features '(" " " " " " " " " " " " " ")))) ;;;; a pseudo BAS
E FEATURES
                           ;;;; in order to keep the same format with the algorithm

               )
;          (print Master-Base-Features )(print GROUPS-AVAILABLE-FOR-Master) (print TERRITORIES-AVAILABLE-F
OR-Master)
;          (print COMBINATIONS-or-COMMENTS-of-Master) (print final-the-COUNTRIES-AVAILABLE-for-MASTER)
           (push  (list
                    BASE-FEATURES-to-be-compared final-the-COUNTRIES-AVAILABLE-for-MASTER COMBINATIONS-or-
COMMENTS-of-Master)
                  result))))
    (list  ;;;;;;  reference ***** and *******    ...In order to keep the same format with the old *COMMO
N-GROUND-RECORD*
      PLUS-OR-MINUS
      (car a-feature-reference)
      ""
      (remove-duplicates result))))


;;*****************************************************************************************************
*****************

(define-ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT-command
  (com-show-HOW-NOT--- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-3)))
     (send stream :clear-history)
     (create-others-and-root_FOR-NOT stream)))



;Command: (collect-all-nothings-for-draw)
;(("NOTHING" "B51L" "B44B" "NOTHING") ("NOTHING" "B51R" "B44B" "NOTHING") ("NOTHING" "B51K" "B44R" "NOTHI
NG")
; ("NOTHING" "B51K" "B69D" "NOTHING"))
;Command:

(defun collect-all-nothings-for-draw ()
  (let ((result nil))
    (loop for record in *pure-RUNS*
          do
      (let ((combinations (second record)))
        (loop for i in combinations
              do
          (cond ((string-equal "nothing" (car i))
```

```
                    (push i result))))))
     (loop for record in *HOW*
           do
       (let ((combinations (fourth record)))
         (loop for i in combinations
               do
           (cond ((string-equal "nothing" (car i))
                    (push i result))))))
     (remove-duplicates result :test #'equal)))


(defflavor  how-not
         (
          (feature-combinations  nil))
         (Feature-combination)
  :writable-instance-variables
  :initable-instance-variables)



(defmethod  (sys:print-self   how-not)    (stream  &rest   ignore)
  (print-node  self  stream))



(defmethod  (print-node   how-not) (stream)
  (format  stream    "THE BELOW FEATURES DON'T HAVE COMMON GROUND IN THE BAFCs OF THE MODEL ~A" *model* )
)

(defmethod  (node-inferiors   how-not)   ()
  feature-combinations)


(defun  graph-tree-ROOT-for-NOT  (root &optional   (stream  *standard-output*))
            (fresh-line  stream)
  (format-graph-from-root    root
                             #'princ
                             #'node-inferiors
                             :dont-draw-duplicates t
                             :orientation   :horizontal
                             :stream  stream
;                            :row-spacing 10
;                            :within-row-spacing 20
))


(defun create-others-and-root_FOR-NOT (&optional (stream *standard-output*))
  (setq *feature-combinations-for-DRAW* nil)
  (let* ((help1 (collect-all-nothings-for-draw)) (help2 nil)
;(("NOTHING" "B51L" "B44B" "NOTHING") ("NOTHING" "B51R" "B44B" "NOTHING") ("NOTHING" "B51K" "B44R" "NOTHI
NG")
; ("NOTHING" "B51K" "B69D" "NOTHING"))
         (help nil))
    (loop for i in help1
          do
              (push (list (string-append (second i) "+" (third i))) help2))

    (setq *ADD-FEAT-COMBINATIONs* nil)
      (loop for i in help2
            do
        (make-instance 'ADD-FEAT-COMBINATION :name (car i) :feature-combinations nil))
      (let ((root (make-instance 'how-not :feature-combinations *ADD-FEAT-COMBINATIONs*)))
        (graph-tree-ROOT-for-NOT root stream)
        (let ((variable (ADD-FEAT-COMBINATION-name  (accept 'ADD-FEAT-COMBINATION
                                                           :prompt "ENTER THE FEAT-COMBINATION YOU WANT TO see HO
W its TRUTH MAINTENACE TREE wouldn't match OR Return TO EXIT"))))
           ;;;;;;(print variable stream) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
          (cond ((check-if-tree-of-a-double-combination variable)
                  (do-the-draw-tree-for-a-double-combination variable stream)))))

)
```

```
;;; -*- Syntax: Common-lisp; Package: austin-rover; Base: 10; Mode: LISP -*-



(define-ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT-command
  (com-show-HOW-NOT--- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-3)))
     (send stream :clear-history)
     (create-others-and-root_FOR-NOT stream)))



;Command: (collect-all-nothings-for-draw)
;(("NOTHING" "B51L" "B44B" "NOTHING") ("NOTHING" "B51R" "B44B" "NOTHING") ("NOTHING" "B51K" "B44R" "NOTHI
NG")
; ("NOTHING" "B51K" "B69D" "NOTHING"))
;Command:

(defun collect-all-nothings-for-draw ()
  (let ((result nil))
    (loop for record in *pure-RUNS*
          do
      (let ((combinations (second record)))
        (loop for i in combinations
              do
          (cond ((string-equal "nothing" (car i))
                    (push i result))))))
    (loop for record in *HOW*
          do
      (let ((combinations (fourth record)))
        (loop for i in combinations
              do
          (cond ((string-equal "nothing" (car i))
                    (push i result))))))
    (remove-duplicates result :test #'equal)))


(defflavor  how-not
        (
         (feature-combinations  nil))
        (Feature-combination)
  :writable-instance-variables
  :initable-instance-variables)



(defmethod  (sys:print-self   how-not)   (stream  &rest  ignore)
  (print-node  self  stream))



(defmethod  (print-node   how-not) (stream)
  (format  stream    "THE BELOW FEATURES DON'T HAVE COMMON GROUND IN THE BAFCs OF THE MODEL -A" *model* )
)

(defmethod  (node-inferiors   how-not)  ()
  feature-combinations)


(defun  graph-tree-ROOT-for-NOT  (root &optional   (stream  *standard-output*))
            (fresh-line  stream)
  (format-graph-from-root   root
                            #'princ
                            #'node-inferiors
                            :dont-draw-duplicates t
                            :orientation   :horizontal
                            :stream  stream
;                           :row-spacing 10
;                           :within-row-spacing 20
))


(defun create-others-and-root_FOR-NOT (&optional (stream *standard-output*))
  (setq *feature-combinations-for-DRAW* nil)
```

```
  (let* ((help1 (collect-all-nothings-for-draw)) (help2 nil)
;(("NOTHING" "B51L" "B44B" "NOTHING") ("NOTHING" "B51R" "B44B" "NOTHING") ("NOTHING" "B51K" "B44R" "NOTHI
NG")
; ("NOTHING" "B51K" "B69D" "NOTHING"))
        (help nil))
    (loop for i in help1
          do
              (push (list (string-append (second i) "+" (third i))) help2))

    (setq *ADD-FEAT-COMBINATIONs* nil)
      (loop for i in help2
            do
        (make-instance 'ADD-FEAT-COMBINATION :name (car i) :feature-combinations nil))
      (let ((root (make-instance 'how-not :feature-combinations *ADD-FEAT-COMBINATIONs*)))
        (graph-tree-ROOT-for-NOT root stream)
        (let ((variable (ADD-FEAT-COMBINATION-name  (accept 'ADD-FEAT-COMBINATION))))
           ;;;;;;(print variable stream) ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
          (cond ((check-if-tree-of-a-double-combination variable)
                      (do-the-draw-tree-for-a-double-combination variable stream))))))

|

;
;                (t
;          (let* ((variable-which-keeps-both-flavor-names-and-tree-structure
;                     (create-the-tree-of-the-string variable))    ;;;;;;;;reference   ####
;                   (flavor-names (car variable-which-keeps-both-flavor-names-and-tree-structure))
;                   (tree-structure (cadr variable-which-keeps-both-flavor-names-and-tree-structure)))
;             (let ((table-of-flavor-names-with-their-equivalent-lists nil) (actual-flavors+lists nil))
;               (loop for i in flavor-names
;                        do
;                 (push (list (make-names-appending-strings i) i) table-of-flavor-names-with-their-equivale
nt-lists))
;; (("B51L" "B51L") ("B44R" "B44R") ("B69D" "B69D") ("B51L,B44R,B69D" ("B51L" "B44R" "B69D")) ("B51L,B44R
" ("B51L" "B44R"))
;; ("B51L,B69D" ("B51L" "B69D")))
;             (loop for i in table-of-flavor-names-with-their-equivalent-lists
;                     do
;               (push (list (make-instance 'Usage-statement :name (car i))
;                           (cond ((atom (cadr i)) (list (cadr i)))
;                                  (t (cadr i))))
;                     actual-flavors+lists))
;             ;;;;;;(print actual-flavors+lists)  ;;;;;reference &&&&&&&
;             (loop for i in tree-structure  ;;  i === (("B51L" "B44R") ("B51L") ("B44R"))
;                     do
;               (setf (Usage-statement-feature-combinations (return-the-flavor-of-the-list (car i) actual-f
lavors+lists))
;                        (list (return-the-flavor-of-the-list (second i) actual-flavors+lists)
;                              (return-the-flavor-of-the-list (third i) actual-flavors+lists))))
;             (let* ((final (loop for i in tree-structure
;                                 collect (list  (car i) (length (car i)))))
;                    (root (return-the-flavor-of-the-list
;                            (caar (sort final #'> :key #'second)) actual-flavors+lists)))
;
;               (graph-tree-for-FEATURE-COMBINATIONS root stream)
;               (loop for char = (accept '(or ;;;(dw:member-sequence (#\return))
;                                            (character)
;                                            (Usage-statement)))
;
;
;                    until (dw:compare-char-for-accept char #\return)
;                    do
;                 (cond ((instancep char)
;                        (let* ((feature-string (Usage-statement-name char))  ;;;;"B51L,B44R,B69D"
;                               (list-of-the-feature-combinations (get-words-from-string-comma feature-stri
ng)))
;                               ;;("B51L" "B44R" "B69D")
;                          (let ((FEATURE-COMBINATIONS-and-restrictions-to-be-drawn
;                                 (BLOCK RESTRICTIONS
;                                   (loop for i in *table*
;                                         do
;                                     (let ((features (only-atoms (car i))))
;                                       (let ((list-of-features-in-lists
;                                              (loop for i in features
;                                                    collect
;                                                      (get-words-from-string-other i))))
```

```
;                                        (cond ((or
;                                                 (member list-of-the-feature-combinations list-of-featur
es-in-lists
;                                                         :test #'equal)
;                                                 (check-if-any-of-them-is-a-member list-of-the-feature-c
ombinations
;                                                                                   list-of-features-in-l
ists))
;                                               (return-from RESTRICTIONS i)))))))
;                                   nil)))
;
;                            (cond ((equal (string-length feature-string) 4)
;                                     (get-the-string-of-only-one-feature-combination-in-STATICE feature-s
tring stream))
;                                  (t
;
;                                   (menu-tree feature-string FEATURE-COMBINATIONS-and-restrictions-to-be-d
rawn stream)))
;
;                  ;;;;;  before it was ONLY (menu-tree feature-string FEATURE-COMBINATIONS-and-restrictio
ns-to-be-drawn stream)
;
;                          ))))
;
;         );;;;loop
;           )
;
;           )
;)
;))   ;;;t  AND  COND
;)))))
```

```
;;; -*- Syntax: Common-lisp; Package: austin-rover; Base: 10; Mode: LISP -*-


(defparameter *FRAMEWORK-DISPLAY-margin-components*
              'dw:((margin-ragged-borders)
                   (margin-scroll-bar  :history-noun "display")
                   (margin-scroll-bar  :history-noun "display"
                                       :margin :bottom)
                   (margin-white-borders  :thickness 2)))


(DW:DEFINE-PROGRAM-FRAMEWORK ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT
  :SELECT-KEY
  #\square
  :COMMAND-DEFINER
  T
  :COMMAND-TABLE
  (:INHERIT-FROM '("colon full command" "standard arguments" "input editor compatibility")
   :KBD-ACCELERATOR-P 'NIL)
  :STATE-VARIABLES
  NIL
  :PANES
  ((PANE-2 :TITLE :HEIGHT-IN-LINES 1 :REDISPLAY-AFTER-COMMANDS NIL)
   (PANE-3 :DISPLAY
           :margin-components *FRAMEWORK-DISPLAY-margin-components*)

   (PANE-4 :COMMAND-MENU :MENU-LEVEL :TOP-LEVEL) (PANE-5 :INTERACTOR))
                                                          ::margin-components *FRAMEWORK-DISPLAY-margin-co
mponents*
                                                          ;:HEIGHT-IN-LINES 4))
  :CONFIGURATIONS
  '((DW::MAIN (:LAYOUT (DW::MAIN :COLUMN PANE-2 PANE-3 PANE-4 PANE-5))
     (:SIZES
      (DW::MAIN (PANE-2 1 :LINES)  ;;;(PANE-4 2 :lines)
                (PANE-4 :ASK-WINDOW SELF :SIZE-FOR-PANE PANE-4)
       (PANE-5 1 :LINES) :THEN (PANE-3 :EVEN))))))


(define-ENGINEERING-SPEC-------GRAPHING-THE-AUDIT-RESULT-command
  (com-show-HOW--- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-3)))
     (send stream :clear-history)
     (menu-tree "b51K"  stream)))
     ;(draw-the-binary-tree stream)))


(defun nested-formatting-2-NEGATIVES (tree &optional (stream *standard-output*))
  (fresh-line)
  (format-graph-from-root tree
                          #'(lambda (node stream)
                              (if (consp (first node))
                                  (with-character-size (:large stream :bind-line-height t)
                                    (princ "FEATURES-NEGATIVE-TO-THE-PART" stream))
                                  (format-item-list node
                                                    :stream stream
                                                    :optimal-number-of-rows 2
                                                    :fresh-line nil
                                                    :additional-indentation 0)))
                          #'(lambda (node)
                              (and (consp (first node)) node))
                          :stream stream))


(defun menu-tree (feature-combination &optional (stream *standard-output*))
  ;;;;(let ((stream (dw:get-program-pane 'pane-3)))
;  (let ((answers nil))
;    (push
      ;;(multiple-value-list
        (dw:accepting-values (stream :own-window t
                                     :near-mode '(:point 100 50)
                                     :label (format nil "DETAILED INFORMATION FOR THE FEATURE COMBINATION
: ~S"
                                                    feature-combination
                                                    :DISPLAY-EXIT-BOXES t))

          (nested-formatting-2-more-detailed-data   (simplify-countries paul);;;;;;;;'((a b) ((r t y) (j i
```

```
 o)) (y u i) (y u i) (j i o) (r t y) ((r t y) (j i o)))
                                        stream)))

;          (values
;            (accept '(alist-member :alist ,*models*) :prompt "ENTER THE MODEL YOU WANT TO SPECIFY THE par
t FOR"
;                     :stream stream )
;
;          )
;        )
;      answers)
;    answers
;  )
;))


(defun simplify-countries (ls)  ;;;;;paul
  (let ((result nil))
    (loop for i in (cadr ls)   ;;;restrictions
          do
      (let ((countries-which-goes (second i)))
        (cond ((equal (length countries-which-goes) 69)   ;;;; countries
               (push (list (car i)
                           (list "TO-ALL-COUNTRIES")
                           (list (third i))) result))
              (t (push i result)))
        ))
    result))


(defun nested-formatting-2-more-detailed-data (tree &optional (stream *standard-output*))
  (fresh-line)
  (format-graph-from-root  tree
                           #'(lambda (node stream)
                               (if (consp (first node))
                                   (with-character-size (:large stream :bind-line-height t)
                                     (princ "*" stream))
                                   (format-item-list node
                                                     :stream stream
                                                     :optimal-number-of-rows 2
                                                     :fresh-line nil
                                                     :additional-indentation 2
                                                     )))
                           #'(lambda (node)
                               (and (consp (first node)) node))
                           :stream stream
                           :orientation :horizontal
                           ))


(defflavor  Feature-combination
        () () )


(defflavor  Usage-statement
        (name
         (feature-combinations  nil))
        (Feature-combination)
  :writable-instance-variables
  :initable-instance-variables)


(defvar  *feature-combinations-for-DRAW*  nil)


(defmethod  (sys:print-self   Usage-statement)   (stream  &rest  ignore)
  (print-node  self  stream))

(defmethod (make-instance  Usage-statement)   (&rest  ignore)
           (push self *feature-combinations-for-DRAW*))


(defmethod  (print-node   Usage-statement) (stream)
  (format  stream  "FEATURE-COMBINATION ~A"  name))
```

```
(defmethod  (node-inferiors   Usage-statement)   ()
  feature-combinations)


(defun  graph-tree-for-FEATURE-COMBINATIONS  (Usage-statement &optional   (stream  *standard-output*))
            (fresh-line  stream)
  (format-graph-from-root   Usage-statement
                            #'princ
                            #'node-inferiors
                            :dont-draw-duplicates t
                            :orientation  :vertical     ;;;;;horizontal
                            :stream  stream))




;(defvar *btrees* nil)
;
;(defmethod (make-instance btree) (&rest ignore)
;  (push self *btrees*))
;
;
;
;(defflavor btree ((node 0) (left nil) (right nil)) ()
;  :initable-instance-variables
;  :readable-instance-variables
;  :writable-instance-variables)
;
;
;(defmethod (add btree) (n)
;  (cond ((= n node) t)
;        ((< n node)
;         (if (null left)
;             (setq left (make-instance 'btree :node n))
;             (add left n)))
;        (t (if (null right)
;             (setq right (make-instance 'btree :node n))
;             (add right n)))))
;
;(defmethod (add btree) (n)
;  (cond ((= n node) t)
;        ((< n node)
;         (if (null left)
;             (setq left (make-instance 'btree :node n))
;             (add left n)))
;        (t (if (null right)
;             (setq right (make-instance 'btree :node n))
;             (add right n)))))
;
;
;
;(defun draw-the-binary-tree (tree &optional (stream *standard-output*))
;(graphics:with-room-for-graphics (stream)
;  (format-graph-from-root tree
;                          #'(lambda (o str) (princ (btree-node o) str))
;                          #'(lambda (o)
;                              (append (if (btree-left o)
;                                          (list (btree-left o)))
;                                      (if (btree-right o)
;                                          (list (btree-right o)))))
;                          :stream stream)
;  )
;)
```

```
;;;; -*- Syntax: Common-Lisp; Package: austin-rover; Base: 10; Mode: LISP -*-









(DW:DEFINE-PROGRAM-FRAMEWORK ENGINEERING-SPEC-------SYSTEM-MAINTENANCE
  :SELECT-KEY
  #\triangle
  :COMMAND-DEFINER
  T
  :COMMAND-TABLE
  (:INHERIT-FROM '("colon full command" "standard arguments" "input editor compatibility")
   :KBD-ACCELERATOR-P 'NIL)
  :STATE-VARIABLES
  NIL
  :PANES
  ((PANE-2 :TITLE :HEIGHT-IN-LINES 1 :REDISPLAY-AFTER-COMMANDS NIL)
   (PANE-3 :DISPLAY)
           ;;;;;:margin-components *FRAMEWORK-DISPLAY-margin-components*]

   (PANE-4 :COMMAND-MENU :MENU-LEVEL :TOP-LEVEL) (PANE-5 :INTERACTOR
                                                         :margin-components *FRAMEWORK-DISPLAY-margin-com
ponents*

                                                         :HEIGHT-IN-LINES 4
                                                          ))
  :CONFIGURATIONS
  '((DW::MAIN (:LAYOUT (DW::MAIN :COLUMN PANE-2 PANE-3 PANE-4 PANE-5))
     (:SIZES
      (DW::MAIN (PANE-2 1 :LINES)  (PANE-4 7 :lines)      ;;(PANE-4 4 :lines)
                ;; (PANE-4 :ASK-WINDOW SELF :SIZE-FOR-PANE PANE-4)
                   (PANE-5 40 :LINES) ;;;;;     (PANE-5 43 :LINES)
       :THEN (PANE-3 :EVEN)
)))))




;;*****************************************************************************




(define-presentation-type  feature-group-string (())
   :history t
   :printer ((object stream)
             (format stream "~a" object))
   :parser ((stream)
            (dw:with-accept-help ((:subhelp "The specification of a FEATURE GROUP string must BE as follo
ws : first a character, then 2 numbers then a dash followed by the Description of the Group in English.")
)
            (let ((string "")(record -1))
              (loop for char = (dw:read-char-for-accept  stream)
                    until (equal record 41)
                    do
                (setq record (+ 1 record))

                (when (and (equal record 0)
                           (not (alpha-char-p char)))
                  (sys:parse-ferror "THE FIRST LETTER MUST BE A CHARACTER !"))
                (when (and (equal record 1)
                           (alpha-char-p char))
                  (sys:parse-ferror "THE SECOND LETTER MUST BE A NUMBER !"))
                (when (and (equal record 2)
```

```
                          (alpha-char-p char))
                    (sys:parse-ferror "THE THIRD LETTER MUST BE A NUMBER !"))
                  (when (and (equal record 3)
                          (not (dw:compare-char-for-accept char #\-)))
                    (sys:parse-ferror "THE FOURTH LETTER MUST BE A DUSH !"))
                  (when (and (equal record 4)
                           (or (dw:compare-char-for-accept char #\space)
                               (not (alpha-char-p char))))
                   (sys:parse-ferror "THE FIFTH LETTER MUST BE FILLED WITH A CHARACTER BECAUSE FROM NOW I
T STARTS THE DESCRIPTION  OF THE FEATURE GROUP !"))
                  (when (characterp char)
                    (setq string (string-append string  (string char))))
                  finally (return string))))))


(define-presentation-type  feature-string-KNOWING-THE-FEATURE-GROUP ((a-string))  ;;;;"Z99BSERVICE ITEMS
                                                                                                  "
   :history t
   :printer ((object stream)
             (format stream "~a" object))
   :parser ((stream)
            (dw:with-accept-help ((:subhelp "The specification of a FEATURE string must BE as follows : f
irst a character, then 2 numbers then a character followed by the Description of the Feature in English."
))
            (let ((string "")(record -1))
              (loop for char = (dw:read-char-for-accept  stream)
                    until (equal record 41)
                    do
                  (setq record (+ 1 record))

                  (when (and (equal record 0)
                           (not (alpha-char-p char)))
                    (sys:parse-ferror "THE FIRST LETTER MUST BE A CHARACTER !"))
                  (when (and (equal record 0)
                           (not (char-equal  char (aref a-string 0))))
                    (sys:parse-ferror "THE FIRST LETTER OF THE FEATURE DESCRIPTION IT DOES NOT MATCH WITH T
HE FIRST LETTER IN THE DESCRION OF THE FEATURE GROUP  !!!"))
                  (when (and (equal record 1)
                          (alpha-char-p char))
                    (sys:parse-ferror "THE SECOND LETTER MUST BE A NUMBER !"))
                  (when (and (equal record 1)
                           (not (char-equal  char (aref a-string 1))))
                    (sys:parse-ferror "THE SECOND LETTER OF THE FEATURE DESCRIPTION IT DOES NOT MATCH WITH
THE SECOND LETTER IN THE DESCRION OF THE FEATURE GROUP  !!!"))
                  (when (and (equal record 2)
                          (alpha-char-p char))
                    (sys:parse-ferror "THE THIRD LETTER MUST BE A NUMBER !"))
                  (when (and (equal record 2)
                           (not (char-equal  char (aref a-string 2))))
                    (sys:parse-ferror "THE THIRD LETTER OF THE FEATURE DESCRIPTION IT DOES NOT MATCH WITH T
HE THIRD LETTER IN THE DESCRION OF THE FEATURE GROUP  !!!"))
                  (when (and (equal record 3)
                          (not (alpha-char-p char)))
                    (sys:parse-ferror "THE FOURTH LETTER MUST BE A CHARACTER !"))
                   (when (and (equal record 4)
                           (or (dw:compare-char-for-accept char #\space)
                               (not (alpha-char-p char))))
                   (sys:parse-ferror "THE FIFTH LETTER MUST BE FILLED WITH A CHARACTER BECAUSE FROM NOW I
T STARTS THE FEATURE DESCRIPTION !"))
                  (when (dw:compare-char-for-accept char #\')
                    (sys:parse-ferror "BACKUOTES ARE NOT ALLOWED IN  THE DESCRIPTION OF A FEATURE !"))
                  (when (characterp char)
                    (setq string (string-append string  (string char))))
                  finally (return string))))))


(define-presentation-type  feature-string (())  ;;;;"Z99BSERVICE ITEMS                                  "
   :history t
   :printer ((object stream)
             (format stream "~a" object))
   :parser ((stream)
            (dw:with-accept-help ((:subhelp "The specification of a FEATURE string must BE as follows : f
irst a character, then 2 numbers then a character followed by the Description of the Feature in English."
))
```

```
            (let ((string "")(record -1))
              (loop for char = (dw:read-char-for-accept  stream)
                    until (or (equal record 41)
                              (dw:compare-char-for-accept char #\return)) ;;; When you RETURN it makes th
e string by itself !
                    do
                 (setq record (+ 1 record))

                 (when (and (equal record 0)
                            (not (alpha-char-p char)))
                   (sys:parse-ferror "THE FIRST LETTER MUST BE A CHARACTER !"))

                 (when (and (equal record 1)
                            (alpha-char-p char))
                   (sys:parse-ferror "THE SECOND LETTER MUST BE A NUMBER !"))
                 (when (and (equal record 2)
                            (alpha-char-p char))
                   (sys:parse-ferror "THE THIRD LETTER MUST BE A NUMBER !"))
                 (when (and (equal record 3)
                            (not (alpha-char-p char)))
                   (sys:parse-ferror "THE FOURTH LETTER MUST BE A CHARACTER !"))
                  (when (and (equal record 4)
                             (or (dw:compare-char-for-accept char #\space)
                                 (not (alpha-char-p char))))
                    (sys:parse-ferror "THE FIFTH LETTER MUST BE FILLED WITH A CHARACTER BECAUSE FROM NOW I
T STARTS THE FEATURE DESCRIPTION !"))
                 (when (dw:compare-char-for-accept char #\`)
                   (sys:parse-ferror "BACKUOTES ARE NOT ALLOWED IN  THE DESCRIPTION OF A FEATURE !"))
                 (when (characterp char)
                   (setq string (string-append string  (string char))))
                 finally (cond ((equal (string-length string) 42)
                                (return string))
                               (t (let ((temp (reverse string)))
                                    (loop for i from 1 to (- 42 (string-length string))
                                          do
                                      (setq temp (string-append #\space temp)))
                                    (return (reverse temp)))))
)))))


;*****************************************************************************
;;;;;;;;;;;;;;;;       Pseudo  Commands
(define-ENGINEERING-SPEC-------SYSTEM-MAINTENANCE-command
  (com-INSERT-NEW-PART-DESCRIPTIONS--- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-5)))
     (send stream :clear-history)))

(define-ENGINEERING-SPEC-------SYSTEM-MAINTENANCE-command
  (com-INSERT-MUST-and-OPTIONAL-FEATURES-for-PART-DESCRIPTIONS--- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-5)))
     (send stream :clear-history)))

(define-ENGINEERING-SPEC-------SYSTEM-MAINTENANCE-command
  (com-DELETE-OLD-Statice-DATA-BASEs-ENTER-NEW  :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-5)))
     (send stream :clear-history)))

(define-ENGINEERING-SPEC-------SYSTEM-MAINTENANCE-command
  (com-DELETE-OLD-EJA18V-data-ENTER-NEW  :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-5)))
     (send stream :clear-history)))

(define-ENGINEERING-SPEC-------SYSTEM-MAINTENANCE-command
  (com-QUERY-FOR-INFORMATION-OF-FEATURES-THE-SYSTEM-GENERATED-for-FAMILIES  :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-5)))
     (send stream :clear-history)))

;;;;;;;;;;;;;;;;;;
(define-ENGINEERING-SPEC-------SYSTEM-MAINTENANCE-command
  (com-CLEAR-the-DISPLAY-window--- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-5)))
     (send stream :clear-history)))


;(push the-feature  *headings*)
;       (push (list the-feature (+ 1 (find-the-maximum-of-list *EACH-FEATURE-WITH-NUMBER*))) *EACH-FEATUR
```

```
E-WITH-NUMBER*)


(define-ENGINEERING-SPEC-------SYSTEM-MAINTENANCE-command
  (Com-ENTER-NEW-FEATURES-of-AN-EXISTING-GROUP--- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-5)))  ;;; ;IT was pane-3
     (print  (format nil "PLEASE TYPE THE FEATURE  DESCRIPTIONS ACCORDING TO THE SYNTAX SPECIFIED IN  K87
 CHART.") stream)
     (terpri stream)
     (let ((the-features-of-GROUP
         (prog  (features-of-group flag times)
                (setq flag t)
                (setq times 0)
            NEXT
                (cond ((not flag) (return features-of-group ))
                      (t (cond ((equal times 0)    ;;;;;; IT IS THE FIRST   TIME    SO WE DEFINATELLY KNO
W THAT there are more
                                (push (the-enter-of-a-single-feature stream) features-of-GROUP)
                                (setq times 1)
                                (go NEXT))
                               (t
                                (cond ((ARE-THERE-MORE?) (push (the-enter-of-a-single-feature stream)
                                                                features-of-GROUP)
                                       (go NEXT))
                                      (t (setq flag nil)
                                         (go NEXT))))
                          ))))))
       (loop for feature in the-features-of-GROUP
             do
         (push feature *headings*))
       (loop for feature in the-features-of-GROUP
             do
         (push (list feature (+ 1 (find-the-maximum-of-list *EACH-FEATURE-WITH-NUMBER*))) *EACH-FEATURE-W
ITH-NUMBER*))
)))  ;;;;;;;;;        (print the-features-of-GROUP stream))))


(defun the-enter-of-a-single-feature (stream)
     (let ((the-feature
            (prog (feature  flag)
               LOOP
                  (cond (flag (return feature))
                        (t
                         (setq feature  (accept 'feature-string))))
                  (print (format nil "FEATURE  DESCRIPTION : " ) stream) (prinl  feature stream)
                  (terpri stream)
                  (cond  ((menu-QUERY-for-agreement-for-the-VPG-group-string)
                          (setq flag t)
                          (go LOOP))
                         (t (go LOOP))))  ))
the-feature
))


(define-ENGINEERING-SPEC-------SYSTEM-MAINTENANCE-command
  (Com-ENTER-A-NEW-FEATURE-GROUP--- :menu-accelerator t) ()
   (let ((stream (dw:get-program-pane 'pane-5)))  ;;; ;IT was pane-3
     (print  (format nil "PLEASE TYPE THE FEATURE GROUP DESCRIPTION ACCORDING TO THE SYNTAX SPECIFIED IN
 K87 CHART.") stream)
     (terpri stream)
     (let ((the-feature-group
     (prog (feature-group flag)
           LOOP
              (cond (flag (return feature-group))
                    (t
                     (setq feature-group (accept 'feature-group-string))))
              (print (format nil "FEATURE GROUP DESCRIPTION : " ) stream) (prinl feature-group stream)
              (terpri stream)
              (cond  ((menu-QUERY-for-agreement-for-the-VPG-group-string)
                      (setq flag t)
                      (go LOOP))
                     (t (go LOOP))))  ))
       (print  (format nil "PLEASE TYPE NOW THE FEATURE DESCRIPTIONS OF THE FEATURE GROUP:  ~s ACCORDING
TO THE SYNTAX SPECIFIED IN  K87 CHART." the-feature-group) stream)
       (terpri stream)
```

```
        (let ((the-features-of-GROUP
          (prog  (features-of-group flag times)
                 (setq flag t)
                 (setq times 0)
              NEXT
                 (cond ((not flag) (return features-of-group ))
                       (t (cond ((equal times 0)    ;;;;;; IT IS THE FIRST   TIME    SO WE DEFINATELLY KNO
W THAT there are more
                                 (push (accept-a-feature-string (substring the-feature-group 0 3) stream)
features-of-GROUP)
                                 (setq times 1)
                                 (go NEXT))
                                (t
                        (cond ((ARE-THERE-MORE?) (push (accept-a-feature-string
                                                         (substring the-feature-group 0 3) stream)
                                                       features-of-GROUP)
                              (go NEXT))
                             (t (setq flag nil)
                                (go NEXT))))
                        ))))
                             ))
;         (print (list the-feature-group (+ 1 (find-the-maximum-of-list *FEATURE-GROUP-WITH-NUMBER*))) str
eam)
;         (loop for i in  the-features-of-GROUP
;               do
;           (print (list i (+ 1 (find-the-maximum-of-list *EACH-FEATURE-WITH-NUMBER*))) stream))
          (push (list (substring the-feature-group 0 3)
                      (+ 1 (find-the-maximum-of-list *FEATURE-GROUP-WITH-NUMBER*))) *FEATURE-GROUP-WITH-NU
MBER*)
          (loop for i in  the-features-of-GROUP
                do
            (push (list i (+ 1 (find-the-maximum-of-list *EACH-FEATURE-WITH-NUMBER*))) *EACH-FEATURE-WITH-
NUMBER*))
          (loop for i in  the-features-of-GROUP
                do
            (push i *headings*))
          (push the-feature-group *headings*)

))))


(defun ARE-THERE-MORE? ()  ;;;;;; YOU MUST NOT put different character format to this menu ie. BOLD , ita
lic etc.
  ;;;;;;;; only  nil.nil.nil
  (values
    (dw:menu-choose '(("Yes" . T) ("No" . nil)) :prompt (zl:format nil "ARE THERE MORE FEATURES  ? "))))


(defun accept-a-feature-string (string stream)
  (let ((the-feature
     (prog (feature  flag)
           LOOP
              (cond (flag (return feature))
                    (t
                     (setq feature (eval '(accept '(feature-string-KNOWING-THE-FEATURE-GROUP  ,string))))
))
              (print (format nil "FEATURE  DESCRIPTION : " ) stream) (prin1  feature stream)
              (terpri stream)
              (cond  ((menu-QUERY-for-agreement-for-the-VPG-group-string)
                      (setq flag t)
                      (go LOOP))
                      (t (go LOOP))))  ))
the-feature))


(defun menu-QUERY-for-agreement-for-the-VPG-group-string ()  ;;;;;; YOU MUST NOT put different character
format to this menu ie. BOLD , italic etc.
  ;;;;;;;; only  nil.nil.nil
  (values
    (dw:menu-choose '(("Yes" . T) ("No" . nil)) :prompt (zl:format nil "DO YOU AGREE WITH THIS  ? "))))
```

```
;;;*****************************************************************************************


;;Command: *EACH-FEATURE-WITH-NUMBER*
;(("Z99BSERVICE ITEMS                          " 2644) ("Z98AALL SPECIFIED OPTIONS AVAILBLE        " 2643)
; ("Z97ZSALES TRIM VALIDATION CODE             " 2642) ("Z97APACK 2                                " 2641)
; ("Z96CNIGERIAN POLICE VEHICLE       NIGPOL   " 2640) ("Z96BPOLICE VEHICLE                POLIS   " 2639)
; ("Z96AMETROPOLITAN POLICE SPEC     ..............................................................
..
;Command: *FEATURE-GROUP-WITH-NUMBER*                " 2638)
;(("Z99" 667) ("Z98" 666) ("Z96" 665) ("Z93" 664) ("Z91" 663) ("Z90" 662) ("Z89" 661) ("Z88" 660) ("Z86"
659) ("Z78" 658)
; ("Z75" 657) ("Z70" 656) ("Z69" 655) ("Z63" 654) ("Z61" 653) ("Z57" 652) ("Z56" 651) ("Z55" 650) ("Z54"
649) ("Z53" 648)
; ("Z52" 647) ...............................................


(defun find-the-maximum-of-list  (ls)  ;;;;;  *EACH-FEATURE-WITH-NUMBER*   OR   *FEATURE-GROUP-WITH-NUMBE
R*
  (let ((result 0))
    (loop for feature-with-number  in  ls
        do
      (cond ((> (second feature-with-number) result)
                 (setq result (second feature-with-number)))))
result))


;;*****************************************************************************
;;;;;*****************************************************************************************
********************


;Engineering Spec        System Maintenance command: Enter A New Feature Group
;
;"PLEASE TYPE THE FEATURE GROUP DESCRIPTION ACCORDING TO THE SYNTAX SPECIFIED IN  K87 CHART."
;Enter a feature group string [default q45-blabvsbbn.>JKdvshbdhm j' nsdBb.j>Lkbcd]: z10-its a pseudo feat
ure group for trial
;"FEATURE GROUP DESCRIPTION : " "z10-its a pseudo feature group for trial  "
;
;"PLEASE TYPE NOW THE FEATURE DESCRIPTIONS OF THE FEATURE GROUP:  \"z10-its a pseudo feature group for tr
ial  \" ACCORDING TO THE
; SYNTAX SPECIFIED IN  K87 CHART."
;Enter a feature string [default a35rgfkgvgjvcgjcvgjv hjglhjvjvc vjhvjh jvk]: z10atakis takis takis

;"FEATURE  DESCRIPTION : " "z10atakis takis takis                    "
;Enter a feature string [default z10atakis takis takis                    ]: z10ekaterina katerina kater
ina
;"FEATURE  DESCRIPTION : " "z10ekaterina katerina katerina            "
;Engineering Spec        System Maintenance command:
;
;;;;;;                          ***************************************************
;
;Command: *headings*
;("z10-its a pseudo feature group for trial  " "z10atakis takis takis                    "
; "z10ekaterina katerina katerina            " "A10-MARQUE                                 "
; "A10AAUSTIN                        A       " "A10HHONDA                          HONDA   "
; "A10MMORRIS                        MORRIS  " "A10RROVER                          ROVER   "
; "A10XAUSTIN ROVER                  AUSROV  " "A20-MODEL RANGE (AUSTIN ROVER)             "
; "A20AADO 73                        A73     " "A20BLC8                            LC8     "
; "A20CMAESTRO                       LM10    " "A20DACCLAIM                        ACL     "
; "A20EMONTEGO                       LM11  15" "A20FMETRO 2                        METRO202"
; "A20GAR6(FROZEN SPEC)              AR6   10" "A20HROVER 200                      SD3   07"
; "A20JSG4                           SG4   08" "A20MAR16                           AR16  13"
; "A20NMINI                          A20F    " "A20PR6(METRO3)                     METRO304"
; "A20RVHPD                          RALLY 03" "A20SXX(UK BUILD)                   XXUK  17"
; "A20TXX(JAPAN BUILD)               XXJAPN20" "A20VR9                             R9    01"
; "A20WR8                            AR8   11" "A20YPOWER TRAIN INDUSTRIAL SALES  PTSALE99"
; "A20ZAD071 PRINCESS 1981           A71F    " "A22-MODEL RANGE    (ROVER-TRIUMPH)         "
; "A22RS.D.1                         SD1     " "A25-MODEL RANGE LCV (CURRENT)              "
; "A26-MODEL RANGE    (LANDROVER)            " "A27-MODEL RANGE (JAGUAR/DAIMLER)           "[Abort]
;Back to Lisp Top Level in Dynamic Lisp Listener 1.
;
;Command: *FEATURE-GROUP-WITH-NUMBER*
;(("z10" 668) ("Z99" 667) ("Z98" 666) ("Z96" 665) ("Z93" 664) ("Z91" 663) ("Z90" 662)
```

```
; ("Z89" 661) ("Z88" 660) ("Z86" 659) ("Z78" 658) ("Z75" 657) ("Z70" 656) ("Z69" 655) ("Z63" 654) ("Z61"
653) ("Z57" 652)
; ("Z56" 651) ("Z55" 650) ("Z54" 649) ("Z53" 648) ("Z52" 647) ("Z51" 646) ("Z50" 645) ("Z48" 644) ("Z44"
643) ("Z43" 642)
; ("Z42" 641) ("Z41" 640) ("Z36" 639) ("Z35" 638) ("Z34" 637) ("Z33" 636) ("Z32" 635) ("Z31" 634) [Abort]
;Back to Lisp Top Level in Dynamic Lisp Listener 1.
;
;Command: *EACH-FEATURE-WITH-NUMBER*
;(("z10atakis takis takis                       " 2646) ("z10ekaterina katerina katerina           " 2645)
; ("Z99BSERVICE ITEMS                           " 2644) ("Z98AALL SPECIFIED OPTIONS AVAILBLE        " 2643)
; ("Z97ZSALES TRIM VALIDATION CODE              " 2642) ("Z97APACK 2                                " 2641)
; ("Z96CNIGERIAN POLICE VEHICLE       NIGPOL    " 2640) ("Z96BPOLICE VEHICLE                POLIS   " 2639)
; ("Z96AMETROPOLITAN POLICE SPEC                " 2638) ("Z95APACK 1                                " 2637)
; ("Z94ASUPPRESS TERRITORIES                    " 2636) ("Z93ZAUSTRALIAN VIN                        " 2635)
; ("Z93XPORTUGESE VIN                           " 2634) ("Z93WHOME MARKET VIN                       " 2633)
; ("Z93VGULF VIN                      GULFVN    " 2632) ("Z93UG.R.H.D. HOT CLIMATE                  " 2631)
;[Abort]
;Back to Lisp Top Level in Dynamic Lisp Listener 1.
;
;Command:

;;;;;;***********************************************************************************************
**************************
```

APPENDIX 9

SAMPLE OF THE INFORMATION CREATED DURING THE RUN OF THE VALIDATION ALGORITHM

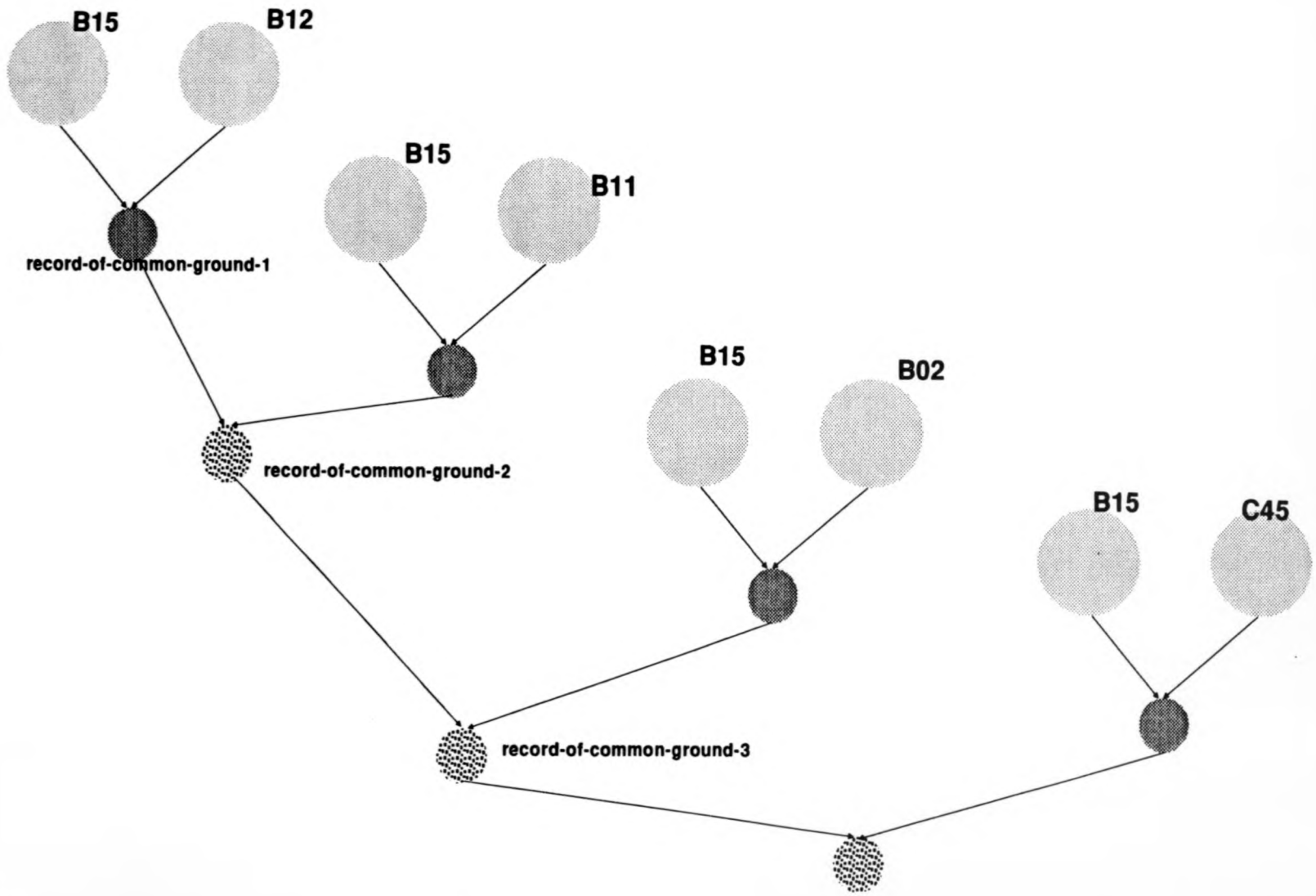


Figure A3b: The diagrammatic representation of the combination of the B51, B12, B11, B02, etc. feature groups.

[redacted] B15 → B12

```
(('B15P' 'B12C') ('B15P' 'B12A') ('B15Y' 'B12A') ('B15J' 'B12D') ('B15J' 'B12C') ('B15J' 'B12A') ('B15K' 'B12A')
 ('B15X' 'B12C') ('B15X' 'B12A'))
NIL
(('NOTHING' 'B15P' 'B12D' 'NOTHING')
 ('PLUS' 'B15P' 'B12C'
  (('M' * * * * * * * 'E' * * 'A')
    ('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56I' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
     'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57I' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
     'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55I' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
     'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
     'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
    (('R' 'R05A' '=' 'R05B') ('P' 'R05A' '=' 'R05B')))))
 ('PLUS' 'B15P' 'B12A'
  (('M' * * * * * * * * * * * *)
    ('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56I' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
     'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57I' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
     'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55I' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
     'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
     'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
   NIL)))
('NOTHING' 'B15Y' 'B12D' 'NOTHING') ('NOTHING' 'B15Y' 'B12C' 'NOTHING')
('PLUS' 'B15Y' 'B12A'



 ('PLUS' 'B15J' 'B12C'
  (('M' 'M' * * * * * * * * * *)
    ('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56I' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
     'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57I' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
     'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55I' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
     'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
     'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
    (('R' 'H50K' '=' 'H50P') ('P' 'H50K' '=' 'H50P')))))
 ('PLUS' 'B15J' 'B12A'
  (('M' 'M' * * * * * * * * * *)
    ('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56I' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
     'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57I' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
     'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55I' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
     'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
     'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
    NIL)
   (('M' 'M' * * * * * * * * * *)
    ('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56I' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
     'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57I' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
     'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55I' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
     'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
     'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
    NIL)
   (('M' 'M' * * * * * * * * * *) ('I60A') (('R' 'W57A' '=' 'Z96G' '=' 'R00E') ('P' 'R00E')))
   (('M' 'M' * * * * * * * * * *) ('I60A') (('R' 'W57A' '=' 'Z96B'))) (('M' 'M' * * * * * * * * *) ('I53N') NIL))
  )
('NOTHING' 'B15K' 'B12D' 'NOTHING') ('NOTHING' 'B15K' 'B12C' 'NOTHING')
('PLUS' 'B15K' 'B12A'
  (('M' 'M' * * * * * * * * * *)
    ('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56I' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
     'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57I' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
     'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55I' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
     'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
     'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
    NIL)
   (('M' 'M' * * * * * * * * * *)
    ('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56I' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
     'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57I' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
     'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55I' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
     'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
     'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
    NIL))
('NOTHING' 'B15X' 'B12D' 'NOTHING')


('PLUS' 'B15X' 'B12C'
  (('M' * * * * * * 'E' * * 'A')
    ('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56I' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
     'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57I' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
     'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55I' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
     'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
     'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
    (('R' 'R05A' '=' 'R05B') ('P' 'R05A' '=' 'R05B')))
  (('M' 'M' * * * * * * * * * *)
    ('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56I' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
     'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57I' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
     'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55I' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
     'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
     'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
    (('R' 'H50K' '=' 'H50P') ('P' 'H50K' '=' 'H50P')))
  (('X' * * * * * * * * * * *)
    ('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56I' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
     'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57I' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
     'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55I' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
     'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
     'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
    NIL)
   (('M' * * * * * * * * * * *)
    ('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56I' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
     'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57I' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
     'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55I' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
     'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
     'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
    NIL)))
('PLUS' 'B15X' 'B12A'
  (('M' * * * * * * * * * * *A')
    ('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56I' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
     'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57I' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
     'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55I' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
```

```
(('Y' * * * * * * * * * * * *)
 ('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I53J' 'I54R' 'I56I' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
  'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57T' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
  'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55I' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
  'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
  'I58R' 'I50X' 'I53A' 'I54N' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
 NIL)
(('H' 'H' * * * * * * * * * *)
 ('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I53J' 'I54R' 'I56I' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
  'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57T' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
  'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55I' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
  'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
  'I58R' 'I50X' 'I53A' 'I54N' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
 (('P' 'H50R' '=' 'H50P') ('P' 'H50R' '=' 'H50P'))))))))
```

815 → 811

```
LOOKUP 'LONDON GEOGRAPHICAL RECORDS'
PLUS' 'R15P' 'R11B'
(('R' 'N' * * * * * * * * * * *)
 ('T55A' 'T60A' 'T53R' 'T53X' 'T56G' 'T56Z' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' *
  'T56J' 'T56V' 'T57X' 'T59L' 'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T50R' 'T53C' 'T57E' 'T59E' *
  'T51R' 'T51V' 'T60E' 'T50A' [illegible] 'T56M' [illegible] 'T55L' 'T51N' 'T54A' *
  'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' [illegible] 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50X' 'T53A' *
  'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL NIL))))
PLUS' 'R15P' 'R11C'
(('R' 'N' * * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V' *
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C' [illegible] 'T51N' 'T54A' 'T55R' 'T56L' 'T58C' 'T60R' 'T56C' 'T54V' *
  'T51G' 'T52G' 'T54N' 'T55J' 'T56E' [illegible] 'T58N' 'T57J' 'T58J')
 (NIL NIL))))
PLUS' 'R15P' 'R11B' (((('D' 'N' * * * * * 'E' 'A' 'N' ('T60A') (NIL)))) ('NOTHING' 'R15P' 'R11C' 'NOTHING')
PLUS' 'R15J' 'R11B'
(('R' 'N' * * * * * * * * * * *)
 ('T55A' 'T60A' 'T53R' 'T53X' 'T56G' 'T56Z' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' *
  'T56J' 'T56V' 'T57X' 'T59L' 'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T50R' 'T53C' 'T57E' 'T59E' *
  'T51R' 'T51V' 'T60E' 'T50A' 'T50V' 'T52R' 'T55T' 'T57G' 'T57V' 'T57Z' 'T50M' 'T55C' 'T55L' 'T51N' 'T54A' *
  'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50X' 'T53A' *
  'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL NIL))
(('R' 'N' * * * * * * * * * * *) ('T60A') (NIL)) (('R' 'N' * * * * * * * * * *) ('T53R') (NIL NIL))))
PLUS' 'R15T' 'R11C'
(('R' 'N' * * * * * * * * * * *)
 ('T51N' 'T54A' 'T55R' 'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' *
  'T58R' 'T53A' 'T54N' 'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL NIL))
(('R' 'N' * * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V' *
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
 (NIL))))
PLUS' 'R15R' 'R11B'
(('R' 'N' * * * * * * * * * * *)
 ('T55A' 'T60A' 'T53R' 'T53X' 'T56G' 'T56Z' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' *
  'T56J' 'T56V' 'T57X' 'T59L' 'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T50R' 'T53C' 'T57E' 'T59E' *
  'T51R' 'T51V' 'T60E' 'T50A' 'T50V' 'T52R' 'T55T' 'T57G' 'T57V' 'T57Z' 'T50M' 'T55C' 'T55L' 'T51N' 'T54A' *
  'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50X' 'T53A' *
  'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL NIL))))
NOTHING' 'R15R' 'R11C' 'NOTHING')
PLUS' 'R15X' 'R11B'
('R' 'N' * * * * * * * * * 'A')
 ('T55A' 'T60A' 'T53R' 'T53X' 'T56G' 'T56Z' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' *
  'T56J' 'T56V' 'T57X' 'T59L' 'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T50R' 'T53C' 'T57E' 'T59E' *
  'T51R' 'T51V' 'T60E' 'T50A' 'T50V' 'T52R' 'T55T' 'T57G' 'T57V' 'T57Z' 'T50M' 'T55C' 'T55L' 'T51N' 'T54A' *
  'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50X' 'T53A' *
  'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL NIL))
(('R' 'N' * * * * * * * * * * *)
 ('T55A' 'T60A' 'T53R' 'T53X' 'T56G' 'T56Z' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' *
  'T56J' 'T56V' 'T57X' 'T59L' 'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T50R' 'T53C' 'T57E' 'T59E' *
  'T51R' 'T51V' 'T60E' 'T50A' 'T50V' 'T52R' 'T55T' 'T57G' 'T57V' 'T57Z' 'T50M' 'T55C' 'T55L' 'T51N' 'T54A' *
  'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50X' 'T53A' *
  'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL))))
PLUS' 'R15X' 'R11C'
('R' 'N' * * * * * * * * * 'A')
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V' *
  'T53V' 'T53Z' [illegible] 'T53N' 'T53C' 'T55C' 'T51N' 'T54A' 'T55R' 'T56L' 'T58C' 'T60R' 'T56C' 'T54V' *
  'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50X' 'T53A' 'T54N' 'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL))
('R' 'N' * * * * * * * * * * *)
 ('T55A' 'T60A' 'T53R' 'T53X' 'T56G' 'T56Z' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' *
  'T56J' 'T56V' [illegible] 'T59L' 'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T50R' 'T53C' 'T57E' 'T59E' *
  'T51R' 'T51V' 'T60E' 'T50A' 'T50V' 'T52R' 'T55T' 'T57G' 'T57V' 'T57Z' 'T50M' 'T55C' 'T55L' 'T51N' 'T54A' *
  'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50X' 'T53A' *
  'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL NIL))
('R' 'N' * * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V' *
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
 ('R' 'H50R' '=' 'H50P') ('P' 'H50R' '=' 'H50P')))
('X' * * * * * * * * * * *)
 ('T55A' 'T60A' 'T53R' 'T53X' 'T56G' 'T56Z' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' *
  'T56J' 'T56V' 'T57X' 'T59L' 'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T50R' 'T53C' 'T57E' 'T59E' *
  'T51R' 'T51V' 'T60E' 'T50A' 'T50V' 'T52R' 'T55T' 'T57G' 'T57V' 'T57Z' 'T50M' 'T55C' 'T55L' 'T51N' 'T54A' *
  'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50X' 'T53A' *
  'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL NIL))
('W' * * * * * * * * * * *)
 ('T55A' 'T60A' 'T53R' 'T53X' 'T56G' 'T56Z' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G'
```

B15 -> B11

```
MODE: (COMMON GROUND-RECORDS
PLUS' 'B15P' 'B11B'
(('H' 'N' * * * * * * * * * * * *)
 ('T55A' 'T60A' 'T53R' 'T53X' 'T56G' 'T56Z' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' *
  'T56J' 'T56V' 'T57X' 'T59L' 'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T50R' 'T53C' 'T57E' 'T59E' *
  'T51R' 'T51V' 'T60E' 'T50A' 'T50V' 'T52P' 'T55T' 'T57G' 'T57V' 'T57Z' 'T50N' 'T55C' 'T55L' 'T51H' 'T54A' *
  'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50Y' 'T53A' *
  'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL NIL))))
PLUS' 'B15P' 'B11C'
(('H' 'N' * * * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V' *
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C' 'T55C' 'T51H' 'T54A' 'T55R' 'T56L' 'T58C' 'T60R' 'T56C' 'T54V' *
  'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50Y' 'T53A' 'T54H' 'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL NIL))))
PLUS' 'B15Y' 'B11B' ((('D' 'N' * * * * * 'K' 'N' 'N') ('T60A') (NIL)))) ('NOTHING' 'B15Y' 'B11C' 'NOTHING')
PLUS' 'B15J' 'B11B'
(('H' 'N' * * * * * * * * * * * *)
 ('T55A' 'T60A' 'T53R' 'T53X' 'T56G' 'T56Z' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' *
  'T56J' 'T56V' 'T57X' 'T59L' 'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T50R' 'T53C' 'T57E' 'T59E' *
  'T51R' 'T51V' 'T60E' 'T50A' 'T50V' 'T52P' 'T55T' 'T57G' 'T57V' 'T57Z' 'T50N' 'T55C' 'T55L' 'T51H' 'T54A' *
  'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50Y' 'T53A' *
  'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL NIL))
(('D' 'N' * * * * * * * * * * * *) ('T60A') (NIL)) (('D' 'N' * * * * * * * * * * * *) ('T53P') (NIL NIL))))
PLUS' 'B15T' 'B11C'
(('H' 'N' * * * * * * * * * * * *)
 ('T51H' 'T54A' 'T55R' 'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' *
  'T50Y' 'T53A' 'T54H' 'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL NIL))
(('H' 'N' * * * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V' *
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
 (NIL))))
PLUS' 'B15K' 'B11B'
(('H' 'N' * * * * * * * * * * * *)
 ('T55A' 'T60A' 'T53R' 'T53X' 'T56G' 'T56Z' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' *
  'T56J' 'T56V' 'T57X' 'T59L' 'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T50R' 'T53C' 'T57E' 'T59E' *
  'T51R' 'T51V' 'T60E' 'T50A' 'T50V' 'T52P' 'T55T' 'T57G' 'T57V' 'T57Z' 'T50N' 'T55C' 'T55L' 'T51H' 'T54A' *
  'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50Y' 'T53A' *
  'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL NIL))))
NOTHING' 'B15K' 'B11C' 'NOTHING')
PLUS' 'B15X' 'B11B'
(('H' 'N' * * * * * * * * * * 'A')
 ('T55A' 'T60A' 'T53R' 'T53X' 'T56G' 'T56Z' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' *
  'T56J' 'T56V' 'T57X' 'T59L' 'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T50R' 'T53C' 'T57E' 'T59E' *
  'T51R' 'T51V' 'T60E' 'T50A' 'T50V' 'T52P' 'T55T' 'T57G' 'T57V' 'T57Z' 'T50N' 'T55C' 'T55L' 'T51H' 'T54A' *
  'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50Y' 'T53A' *
  'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL))
(('H' 'N' * * * * * * * * * * * *)
 ('T55A' 'T60A' 'T53R' 'T53X' 'T56G' 'T56Z' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' *
  'T56J' 'T56V' 'T57X' 'T59L' 'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T50R' 'T53C' 'T57E' 'T59E' *
  'T51R' 'T51V' 'T60E' 'T50A' 'T50V' 'T52P' 'T55T' 'T57G' 'T57V' 'T57Z' 'T50N' 'T55C' 'T55L' 'T51H' 'T54A' *
  'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50Y' 'T53A'
  'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL))))

PLUS' 'B15X' 'B11C'
(('H' 'N' * * * * * * * * * * 'A')
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V' *
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C' 'T55C' 'T51H' 'T54A' 'T55R' 'T56L' 'T58C' 'T60R' 'T56C' 'T54V' *
  'T51G' 'T52G' 'T54H' 'T55J 'T56E' 'T58G' 'T58R' 'T50X' 'T53A' 'T54H' 'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL))
(('Y' 'N' * * * * * * * * * * * *)
 ('T55A' 'T60A' 'T53R' 'T53X' 'T56G' 'T56Z' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' *
  'T56J' 'T56V' 'T57X' 'T59L' 'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T50R' 'T53C' 'T57E' 'T59E' *
  'T51R' 'T51V' 'T60E' 'T50A' 'T50V' 'T52P' 'T55T' 'T57G' 'T57V' 'T57Z' 'T50N' 'T55C' 'T55L' 'T51H' 'T54A' *
  'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50Y' 'T53A' *
  'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL NIL))
(('H' 'N' * * * * * * * * * * * *)
 ('T51H' 'T54A' 'T55R' 'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' *
  'T50Y' 'T53A' 'T54H' 'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL))
(('H' 'N' * * * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V' *
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
 (('R' 'H50K' '=' 'H50P') ('P' 'H50K' '=' 'H50P')))
(('X' * * * * * * * * * * * *)
 ('T55A' 'T60A' 'T53R' 'T53X' 'T56G' 'T56Z' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' *
  'T56J' 'T56V' 'T57X' 'T59L' 'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T50R' 'T53C' 'T57E' 'T59E' *
  'T51R' 'T51V' 'T60E' 'T50A' 'T50V' 'T52P' 'T55T' 'T57G' 'T57V' 'T57Z' 'T50N' 'T55C' 'T55L' 'T51H' 'T54A' *
  'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51G' 'T52G' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50Y' 'T53A' *
  'T54T' 'T56U' 'T58N' 'T57J' 'T58J')
 (NIL NIL))
(('N' * * * * * * * * * * * *)
 ('T55A' 'T60A' 'T53R' 'T53X' 'T56G' 'T56Z' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G'
```

```
[illegible] 'B15J' 'B12B') ('B15P' 'B12A') ('B15X' 'B11C,B12C') ('B15X' 'B11C,B12C') ('B15X' 'B11C,B12C')
B15X' 'B11C,B12A') ('B15X' 'B11C,B12C') ('B15X' 'B11C,B12A') ('B15X' 'B11C,B12A') ('B15X' 'B11C,B12C')
B15X' 'B11C,B12A') ('B15J' 'B11C,B12C') ('B15J' 'B11C,B12A') ('B15J' 'B11C,B12C') ('B15J' 'B11C,B12A')
B15P' 'B11C,B12C') ('B15P' 'B11C,B12A'))

PLUS' 'B15K' 'B12A'
('D' 'N' * * * * * * * * * *)
('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56T' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
 'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57T' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
 'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55T' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
 'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
 'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
NIL)
('D' 'W' * * * * * * * * * *)
('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56T' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
 'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57T' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
 'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55T' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
 'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
 'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
NIL)))
PLUS' 'B15J' 'B12B'
('D' 'S' * * * * * * * * * *)
('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56T' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
 'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57T' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
 'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55T' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
 'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
 'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
NIL)
('H' 'S' * * * * * * * * * *)
('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56T' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
 'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57T' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
 'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55T' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
 'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'




 'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
NIL)))
PLUS' 'B15Y' 'B12A'
('D' 'N' * * * * * 'E' 'A' 'A') ('I60A') (('R' 'Z96B'))) (('D' 'W' * * * * 'E' 'A' 'A') ('I60A') (('R' 'Z96B')))

PLUS,PLUS' 'B15X' 'B11C,B12C'
('W' * * * * * * * * * * * *)
('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56T' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
 'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57T' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
 'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55T' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
 'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
 'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
NIL)))
PLUS,PLUS' 'B15X' 'B11C,B12C'
('X' * * * * * * * * * * * *)
('I55A' 'I60A' 'I53R' 'I53X' 'I56G' 'I56Z' 'I51A' 'I52J' 'I54R' 'I56T' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G'
 'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V' 'I51X' 'I53V' 'I53Z' 'I57T' 'I58L' 'I53N' 'I50R' 'I53C' 'I57E'
 'I59E' 'I59G' 'I51R' 'I51V' 'I60E' 'I50A' 'I50V' 'I52R' 'I55T' 'I57G' 'I57V' 'I57Z' 'I50N' 'I55C' 'I55L'
 'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
 'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
NIL)))
PLUS,PLUS' 'B15X' 'B11C,B12C'
('H' 'N' * * * * * * * * * * *)
('I51A' 'I52J' 'I54R' 'I56T' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G' 'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V'
 'I51X' 'I53V' 'I53Z' 'I57T' 'I58L' 'I53N' 'I53C')
(('P' 'H50K' '=' 'H50P') ('P' 'H50K' '=' 'H50P')))))
PLUS,PLUS' 'B15X' 'B11C,B12A'
('H' 'N' * * * * * * * * * * *)
('I51A' 'I52J' 'I54R' 'I56T' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G' 'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V'
 'I51X' 'I53V' 'I53Z' 'I57T' 'I58L' 'I53N' 'I53C')
(('R' 'H50K' '=' 'H50P') ('P' 'H50K' '=' 'H50P')))))
[illegible] 'B11C,B12C'
 (('R' 'R05A' '=' 'R05B') ('P' 'R05A' '=' 'R05B')))))
('PLUS,PLUS' 'B15X' 'B11C,B12A'
 (((H' 'N' * * * * * * * * * * 'A')
   ('I51A' 'I52J' 'I54R' 'I56T' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G' 'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V'
    'I51X' 'I53V' 'I53Z' 'I57T' 'I58L' 'I53N' 'I53C' 'I55C' 'I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C'




    'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G' 'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N'
    'I57J' 'I58J')
   (('R' 'R05A' '=' 'R05B') ('P' 'R05A' '=' 'R05B')))))
('PLUS,PLUS' 'B15J' 'B11C,B12C'
 (((H' 'N' * * * * * * * * * * *)
   ('I51A' 'I52J' 'I54R' 'I56T' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G' 'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V'
    'I51X' 'I53V' 'I53Z' 'I57T' 'I58L' 'I53N' 'I53C')
   (('R' 'H50K' '=' 'H50P') ('P' 'H50K' '=' 'H50P')))))
('PLUS,PLUS' 'B15J' 'B11C,B12A'
 (((H' 'N' * * * * * * * * * * *)
   ('I51A' 'I52J' 'I54R' 'I56T' 'I59N' 'I55E' 'I60M' 'I53E' 'I53G' 'I54C' 'I56J' 'I56V' 'I57X' 'I59L' 'I59V'
    'I51X' 'I53V' 'I53Z' 'I57T' 'I58L' 'I53N' 'I53C')
   (('R' 'H50K' '=' 'H50P') ('P' 'H50K' '=' 'H50P')))))
('PLUS,PLUS' 'B15J' 'B11C,B12C'
 (((H' 'N' * * * * * * * * * * *)
   ('I51N' 'I54A' 'I55R' 'I56L' 'I58C' 'I60R' 'I56C' 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
    'I58R' 'I50X' 'I53A' 'I54H' 'I54I' 'I56U' 'I58N' 'I57J' 'I58J')
   (('R' 'H50K' '=' 'H50P') ('P' 'H50K' '=' 'H50P')))))
('PLUS,PLUS' 'B15J' 'B11C,B12A'
 (((H' 'N' * * * * * * * * * * *)
[illegible] 'I54V' 'I50Z' 'I51G' 'I52G' 'I54N' 'I55J' 'I56E' 'I58G'
```

```
('PLUS,PLUS' '015J' '011C,012A'
 (((''M' 'N' * * * * * * * * *)
   ('T51N' 'T54A' 'T55R' 'T56L' 'T58C' 'T60R' 'T56C' 'T54V' 'T50Z' 'T51Q' 'T52Q' 'T54N' 'T55J' 'T56E' 'T58G'
    'T58R' 'T50X' 'T53A' 'T54H' 'T54I' 'T56U' 'T50N' 'T57J' 'T58J')
  WHL)))
('PLUS,PLUS' '015P' '011C,012C'
 (((''M' 'N' * * * * 'E' * * 'A')
   ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53Q' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
    'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C' 'T55C' 'T51N' 'T54A' 'T55R' 'T56L' 'T58C' 'T60R' 'T56C'
    'T54V' 'T50Z' 'T51Q' 'T52Q' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50X' 'T53A' 'T54H' 'T54I' 'T56U' 'T50N'
    'T57J' 'T58J')
   (('R' 'R05A' '*' 'R05B') ('P' 'R05A' '*' 'R05B')))))
('PLUS,PLUS' '015P' '011C,012A'
 (((''M' 'N' * * * * * * * * *)
   ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53Q' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
    'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C' 'T55C' 'T51N' 'T54A' 'T55R' 'T56L' 'T58C' 'T60R' 'T56C'
    'T54V' 'T50Z' 'T51Q' 'T52Q' 'T54N' 'T55J' 'T56E' 'T58G' 'T58R' 'T50X' 'T53A' 'T54H' 'T54I' 'T56U' 'T50N'
    'T57J' 'T58J')
  WHL))))
```

```
[illegible] ('[illegible]' '[illegible]')
```

B15 → B02

```
[illegible]
[illegible]and: *COMMON-GROUND-RECORDS
*PLUS-AND-MINUS* 'B15P' 'B02B'
(('H' * * * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60H' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
 (NIL)))
*NOTHING* 'B15J' 'B02B' 'NOTHING')
*PLUS-AND-MINUS* 'B15J' 'B02B'
(('H' 'H' * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60H' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
 (NIL))
(('H' 'W' * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60H' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
 (NIL))
(('H' 'G' * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60H' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
 (NIL))
(('H' 'G' * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60H' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
 (NIL))))
*PLUS-AND-MINUS* 'B15K' 'B02B'
(('B' 'N' * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60H' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
 (NIL))
(('B' 'W' * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60H' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
 (NIL))))
*PLUS-AND-MINUS* 'B15X' 'B02B'
(('H' * * * * * * * * * 'A')
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60H' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
 ('B.ITD1P&TLBegin using 006 [illegible]:1 0 (NIL 0) (NIL :BOLD NIL) 'CPTFONTCR')R(2 0 (NIL 0) (NIL NIL NIL)
(('.' * * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60H' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
 (NIL))
(('H' 'N' * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60H' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
 ('IP2' 'H50K2' '1=2' 'H50P2') ('IP2' 'H50K2' '1=2' 'H50P2') ('P' 'H50P' '=' 'H50P') ('P' 'H50K' '=' '
(('.' * * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60H' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
 (NIL))
(('W' * * * * * * * * * * *)
 ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60H' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
  'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
 (NIL))))
[illegible]21:09 Process Terminal 1 Typeout wants to type out
[illegible]ect Terminal 1 (TELNET) -- TOM on INTERNET [Ambassador] by typing Function-0-S.]
[illegible]and:
```

```
mod: (begin '(('B15' 'B12') ('B15' 'B11') ( B15  B02 ))
RECORD-OF-COMMON-GROUND  31
('B15Y' 'B12A') ('B15X' 'B02B,B11C,B12C') ('B15X' 'B02B,B11C,B12C') ('B15X' 'B02B,B11C,B12C')
('B15X' 'B02B,B11C,B12A') ('B15X' 'B02B,B11C,B12A') ('B15X' 'B02B,B11C,B12C') ('B15X' 'B02B,B11C,B12C') ('B15X' 'B02B,B11C,B12A')
('B15K' 'B02B,B12A') ('B15K' 'B02B,B12A') ('B15J' 'B02B,B12D') ('B15J' 'B02B,B12D') ('B15J' 'B02B,B11C,B12C')
('B15J' 'B02B,B11C,B12A') ('B15P' 'B02B,B11C,B12C') ('B15P' 'B02B,B11C,B12A'))
IL
('PLUS' 'B15Y' 'B12A'
 (('D' 'N' * * * * * 'E' 'A' 'A') ('T60A') (('R' 'Z96B'))) (('D' 'W' * * * * 'E' 'A' 'A') ('T60A') (('R' 'Z96B')))
 )
('PLUS-AND-MINUS,PLUS,PLUS' 'B15X' 'B02B,B11C,B12C'
 (('W' * * * * * * * * * * *)
   ('T53R' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L'
    'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
   (('D,$TD1P$TIBegin using 006 escapes)(1 0 (NIL 0) (NIL :BOLD NIL) 'CPTFONTCR')R(2 0 (NIL 0) (NIL NIL NIL) 'CPTFC
('PLUS-AND-MINUS,PLUS,PLUS' 'B15X' 'B02B,B11C,B12C'
 (('X' * * * * * * * * * * *)
   ('T53R' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L'
    'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
   (('1R2' '1H50K2' '1=2' '1H50P2') ('1P2' '1H50K2' '1=2' '1H50P2')))))
('PLUS-AND-MINUS,PLUS,PLUS' 'B15X' 'B02B,B11C,B12C'
 (('H' 'N' * * * * * * * * * *)
   ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
    'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
   (('1R2' '1H50K2' '1=2' '1H50P2') ('1P2' '1H50K2' '1=2' '1H50P2')))))
('PLUS-AND-MINUS,PLUS,PLUS' 'B15X' 'B02B,B11C,B12A'
 (('H' 'N' * * * * * * * * * *)
   ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
    'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
   (('1R2' '1H50K2' '1=2' '1H50P2') ('1P2' '1H50K2' '1=2' '1H50P2')))))
('PLUS-AND-MINUS,PLUS,PLUS' 'B15X' 'B02B,B11C,B12A'
 (('Y' 'N' * * * * * * * * * *)
   ('T53R' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L'
    'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
   (('1R2' '1H50K2' '1=2' '1H50P2') ('1P2' '1H50K2' '1=2' '1H50P2')))))
('PLUS-AND-MINUS,PLUS,PLUS' 'B15X' 'B02B,B11C,B12C'
 (('M' 'N' * * * * 'E' * * 'A')
   ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
    'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
   (('1R2' '1H50K2' '1=2' '1H50P2') ('1P2' '1H50K2' '1=2' '1H50P2')))))
('PLUS-AND-MINUS,PLUS,PLUS' 'B15X' 'B02B,B11C,B12A'
 (('M' 'N' * * * * * * * * 'A')
   ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
    'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
   (('1R2' '1H50K2' '1=2' '1H50P2') ('1P2' '1H50K2' '1=2' '1H50P2')))))
('PLUS-AND-MINUS,PLUS' 'B15K' 'B02B,B12A'
 (('D' 'W' * * * * * * * * *)
   ('T53R' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L'
    'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
   (('1R2' '1H50K2' '1=2' '1H50P2') ('1P2' '1H50K2' '1=2' '1H50P2')))))
('PLUS-AND-MINUS,PLUS' 'B15K' 'B02B,B12A'
 (('D' 'N' * * * * * * * * *)
   ('T53R' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L'
    'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
   (('1R2' '1H50K2' '1=2' '1H50P2') ('1P2' '1H50K2' '1=2' '1H50P2')))))
('PLUS-AND-MINUS,PLUS' 'B15J' 'B02B,B12D'
 (('N' 'G' * * * * * * * * *)
   ('T53R' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L'
    'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
   (('1R2' '1H50K2' '1=2' '1H50P2') ('1P2' '1H50K2' '1=2' '1H50P2')))))
('PLUS-AND-MINUS,PLUS' 'B15J' 'B02B,B12D'
 (('D' 'G' * * * * * * * * *)
   ('T53R' 'T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L'
    'T59V' 'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
   (('1R2' '1H50K2' '1=2' '1H50P2') ('1P2' '1H50K2' '1=2' '1H50P2')))))
('PLUS-AND-MINUS,PLUS,PLUS' 'B15J' 'B02B,B11C,B12C'
 (('N' 'N' * * * * * * * * *)
   ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
    'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
   (('1R2' '1H50K2' '1=2' '1H50P2') ('1P2' '1H50K2' '1=2' '1H50P2') ('R' 'H50K' '=' 'H50P') ('P' 'H50K' '=' 'H50P
('PLUS-AND-MINUS,PLUS,PLUS' 'B15J' 'B02B,B11C,B12A'
 (('N' 'N' * * * * * * * * *)
   ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
    'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
   (('1R2' '1H50K2' '1=2' '1H50P2') ('1P2' '1H50K2' '1=2' '1H50P2') ('R' 'H50K' '=' 'H50P') ('P' 'H50K' '=' 'H50P
('PLUS-AND-MINUS,PLUS,PLUS' 'B15P' 'B02B,B11C,B12C'
 (('M' 'N' * * * * 'E' * * 'A')
   ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
    'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
   (('1R2' '1H50K2' '1=2' '1H50P2') ('1P2' '1H50K2' '1=2' '1H50P2') ('R' 'R05A' '=' 'R05B') ('P' 'R05A' '=' 'R05B
('PLUS-AND-MINUS,PLUS,PLUS' 'B15P' 'B02B,B11C,B12A'
 (('M' 'N' * * * * * * * * *)
   ('T51A' 'T52J' 'T54R' 'T56T' 'T59N' 'T55E' 'T60M' 'T53E' 'T53G' 'T54C' 'T56J' 'T56V' 'T57X' 'T59L' 'T59V'
    'T51X' 'T53V' 'T53Z' 'T57T' 'T58L' 'T53N' 'T53C')
   (('1R2' '1H50K2' '1=2' '1H50P2') ('1P2' '1H50K2' '1=2' '1H50P2')))))))
RECORD-OF-COMMON-GROUND  31
```

AUSTIN ROVER — AL IONAL FEATURE CHART — MODEL RANGE METRO 2

PRODUCT ENGINEERING — B11- REAR QUARTER LIGHTS — DATE 30 Nov 88

REC NO.	FEAT CODE	FEATURE DESCRIPTION (FEATURE WORD)	AVAIL. SUB AUTO	MODEL RG/CH	ENG	USAGE CONDITION: CLASS	BODY	FAMILY	CAPACITY	FUEL	TRANS	RIVET	QUALITY	-COMBINATIONS- RESTRICTIONS	GROUPS (DEF)	TERRITORY RESTRICTIONS	DESIGN EFFECT POINT IN OUT	REL STG IN OUT	AUTHORITY REFERENCE IN OUT
400	B11B	REAR QTR LIGHT FIXED (RQFIX)	B	XF	D	N	-	-	-	-	+	-	-				B9543	P	M/16P/87
410			P	XF	H	N	-	-	-	-	-	-	-				XF4C	P	M/10/83
420			B	XF	M	N	-	-	-	-	-	-	-				XF4C	P	M/10/83
400	B11C	REAR QTR LIGHT HINGED (HINRQ)	M	XF	H	N	-	-	-	-	-	-	-		+ M36* M37* M38* M39*		XF4C	P	M/10/83
410			O	XF	H	N	-	-	-	-	-	-	-	R H50K=H50R P H50K=H50P	+ M32*		JX054	P	M/5P/88
420				XF	M	N	-	-	-	-	-	-	-		+ M32* M35I M36* M37* M38* M39*		XF4G	P	M/34/83
430			S	XF	Y	N	-	-	-	-	-	-	-				XF4C	P	M/4/84
440			S	XF	W	-	-	-	-	-	-	-	-				XF4C	P	M/10/83
450			S	XF	X	-	-	-	-	-	-	-	-				XF4C	P	M/10/83

REV NO	REV DATE	REVISION DESCRIPTION	
5	30 NOV 88	CHART REISSUED TO SHOW 89MY SPEC. RECORDS BEFORE THIS ARE NOT SHOWN.	ALL

(+) MUST HAVE	(7) IF	(,) AND
(-) MUST NOT HAVE	(/) IF NOT	(<) AND NOT
(R) RESTRICTED TO		(=) OR
(P) PACKAGED WITH		
(I) INCLUDES	(C) "COMMENTS"	

END

AUSTIN ROVER

PRODUCT ENGINEERING

ADDITIONAL FEATURE CHART

D12- EXTERIOR MIRRORS FINISH

MODEL RANGE METRO 2

DATE 08 Dec 88

REC NO.	FEAT CODE	FEATURE DESCRIPTION (FEATURE WORD)	AVAIL - SLD / BEM / OMA / AHI	MODEL ENG/CH	USAGE CONDITION (CLAS DDY PAM CAP PEF RTV BAH / QUAI)	COMBINATIONS-RESTRICTIONS	TERRITORY GROUPS (B F F)	TERRITORY RESTRICTIONS	DESIGN EFFECT POINT IN OUT	REL SIG IN OUT	AUTHORITY REFERENCE IN OUT
400	D12A	BLACK MIRROR -STANDARD HEAD (BLAFIN)	S	XR	D H - - - - -				JX054	P	M/5P/88 S1
410			S	XF	D W				JX054	P	M/5P/88 S1
420			D	XF	H N - - - -				JX054	P	M/5P/88 S1
430			B	XF	H W - - - - -				JX054	P	M/5P/88 S1
440			D	XF	M - - - - - -				JX054	P	M/5P/88 S1
450			S	XF	Y - - - - - -				JX054	P	M/5P/88 S1
400	D12C	BODY COL MIRROR -STANDARD HEAD (PANFIN)		XF	K - - - - - -				XF4C	P	M/57/83
410			S	XF	W - - - - - -				XF6A	P	M/21P/88
420			O	XF	H H - - - - -	R HSDK=HGOR, P HSDK=HSUP			JX054	P	M/5P/88 S1
430			O'	XF	M - - - E - A	R RUSA=RUSD, P RUSA=RUSB			JX125	P	M/27P/88
400	D12B	BLACK MIRROR -LARGE HEAD (BIGBLK)	S	XF	- G - - - - -				JX054	P	M/5P/88 S1

REV NO	REV DATE	R E V I S I O N D E S C R I P T I O N
9	06 DEC 88	PPL M/27P/88---RECORD D12C,430 RAISED TO 'P' LEVEL.

(+) MUST HAVE	(?) IF	(,) AND
( ) MUST NOT HAVE	(/) IF NOT	(<) AND NOT
(R) RESTRICTED TO		(-) OR
(P) PACKAGED WITH		
(I) INCLUDES	(C) "COMMENTS"	

PAGE 1 END

GRP D12

AUSTIN ROVER

PRODUCT ENGINEERING

ADDITIONAL FEATURE CHART

B15- EXTERIOR MIRRORS

MODEL RANGE METRO 2

DATE 06 Dec 88

REC NO.	FEAT CODE	FEATURE DESCRIPTION (FEATURE WORD)	AVAIL - USAGE CONDITION S B O M A M	MODEL RG/CH	ENG C L A S O D Y M A A M F E F I R V A B H	-COMBINATIONS- RESTRICTIONS	-- TERRITORY -- GROUPS	RESTRICTIONS	DESIGN EFFECT POINT IN OUT	REL SIG IN OUT	AUTHORITY REFERENCE IN OUT
400	B15J	DOOR MIRRORS -HAND SET (2HSMIR)	S	XF	- G - - - - -				B9278	P	M/39P/85
410			O	XF	D H - - - - -	R W57A=W57A	+ M20U	M20U +GB	B9278	P	M/39P/85
420			O	XF	D W - - - - -	R W57A=Z96B	+ M20U	M20U +GB	B9278	P	M/39P/85
430			M	XF	D H - - - - -		+ M20E	M20E +DER	B9278	P	M/39P/85
440			M	XF	D W - - - - -		+ M20E	M20E +DER	B9278	P	M/39P/85
450			O	XF	H H - - - - -				B9553	P	M/5P/88
460			S	XF	H W - - - - -				B9278	P	M/39P/85
400	B15K	DRIVER'S DOOR MIRROR (DRDMIR)	O	XF	D H - - - - -				XF4C	P	M/10/83 S1
410			O	XF	D W - - - - -				XF5A	P	M/5P/85
400	B16P	DRV MIRR REMOTE BYPASS FIXED (DRCPFX)	O	XF	M - - - - - -				XF4C	P	M/8/84
400	B15A	DOOR MIRRORS -REMOTE CONTROL (2RCDMR)	S	XF	W - - - - - -				XF5A	P	M/32P/85
410			S	XF	X - - - - - -				XF5A	P	M/14P/85
420			O	XF	H H - - - - -	R H50K=H50P P H50K=H50P			JX054	P	M/5P/88 S1
430			S	XF	Y - - - - - -				JX054	P	M/5P/88 S1
440			O	XF	M - - - - A	R R05A=R05B P R05A=R05B			JX125	P	M/27P/88

REV NO	REV DATE	REVISION DESCRIPTION
9	06 NOV 88	PPL M/27P/88---RECORD B15X,440 RAISED TO 'P' LEVEL.

(+) MUST HAVE	(?) IF	(,) AND
(-) MUST NOT HAVE	(/) IF NOT	(<) AND NOT
(R) RESTRICTED TO		(=) OR
(P) PACKAGED WITH		
(I) INCLUDES	(C) "COMMENTS"	

AUSTIN ROVER — ADL ONAL FEATURE CHART — MODEL RANGE METRO 2

PRODUCT ENGINEERING — B15- EXTERIOR MIRRORS — DATE 06 Dec 88

REC NO.	FEAT CODE	FEATURE DESCRIPTION (FEATURE WORD)	AVAIL - USAGE CONDITION: S B O A / L E M S / MODEL RG/CH / ENG: C O F C P R / B D A A E I / D Y M P F V / T R A N / Q U A L	-COMBINATIONS- RESTRICTIONS	GROUPS D E F	-- TERRITORY -- RESTRICTIONS	DESIGN EFFECT POINT IN OUT	REL STG IN OUT	AUTHORITY REFERENCE IN OUT
400	B15Y	DRIVERS MIRROR -REMOTE CONTROL (DRCMIR)	O XF D N - - E A A R	Z96B	+ M20U	M20U +GB	XF4C	P	M/4/84
410			O XF D W - - E A A R	Z96B	+ M20U	M20U +GB	XF4C	P	M/4/84

REV NO	REV DATE	REVISION DESCRIPTION
8	06 NOV 88	PPL M/27P/88---RECORD B15X,440 RAISED TO 'P' LEVEL.

(+) MUST HAVE	(?) IF	(,) AND
(-) MUST NOT HAVE	(/) IF NOT	(<) AND NOT
(R) RESTRICTED TO		(=) OR
(P) PACKAGED WITH		
(I) INCLUDES	(C) "COMMENTS"	

APPENDIX 10

SAMPLE RUN OF POOVESP

Figure A4: A simplified "seat-front complete" assembly, both Right and Left Hand side.

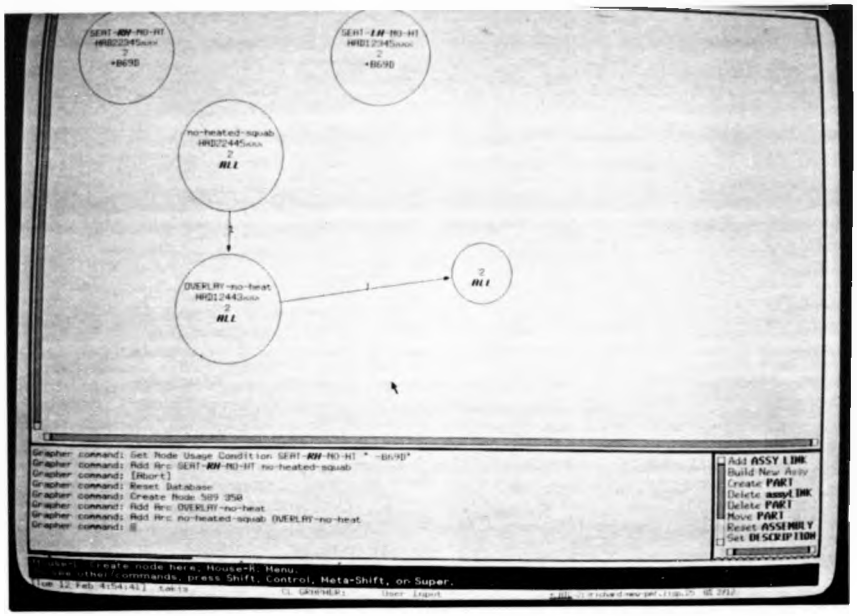


Figure A5: We link the "seat-LH-NO-HT" part with the "no-heated-squab" part. The change in the usage condition (ie. "+B69D") travels all the way down to the assembly hierarchy.

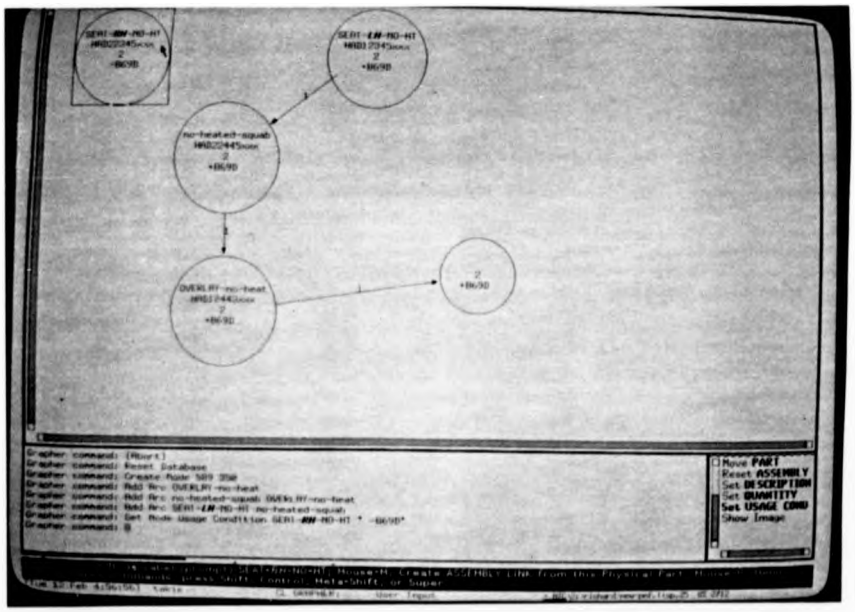


Figure A6: The "-B69D" cancels out the "+B69D" resulting in "ALL" (ie. offered to ALL vehicles) and travels all the way down to the assembly hierarchy.

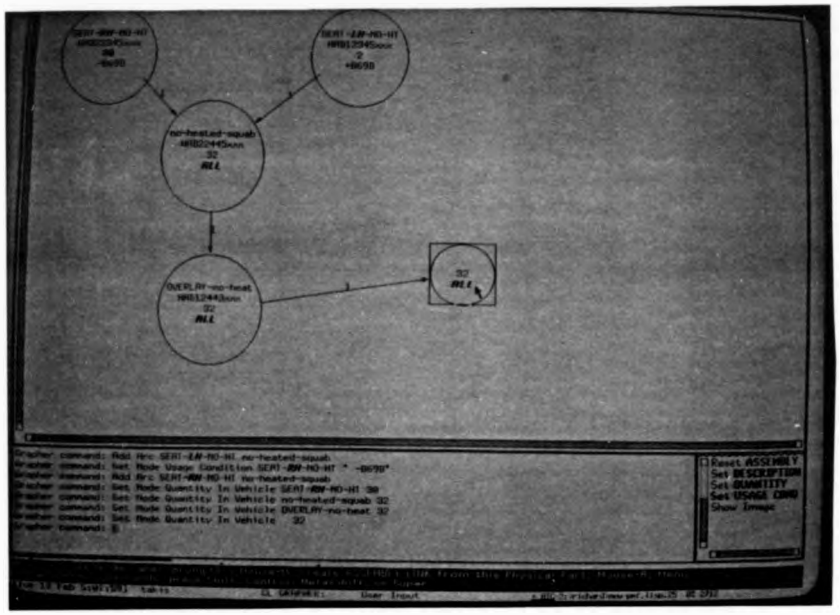


Figure A7: When an assembly link (similarly, a part) is deleted, the Network *intelegently* update its state at the Usage Conditions level. In a similar way it updates its state to the Quantity level

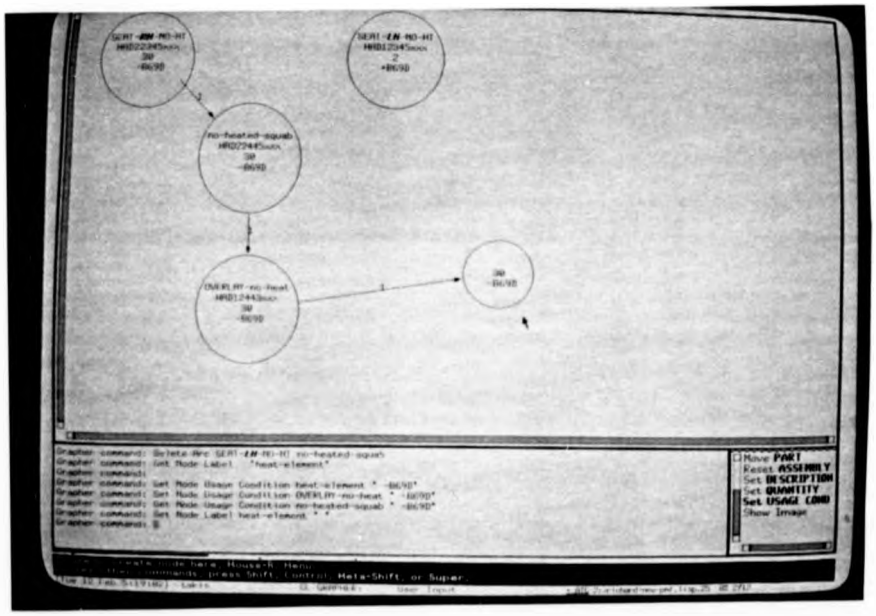


Figure A8: Part images stored in Statice can be displayed in the screen.

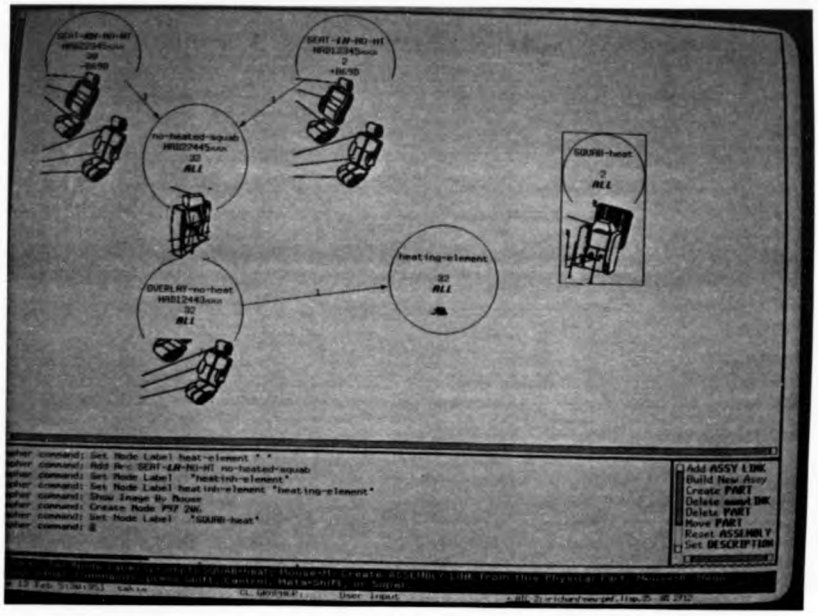


Figure A9: A closer look to an image of a part stored in Statice.

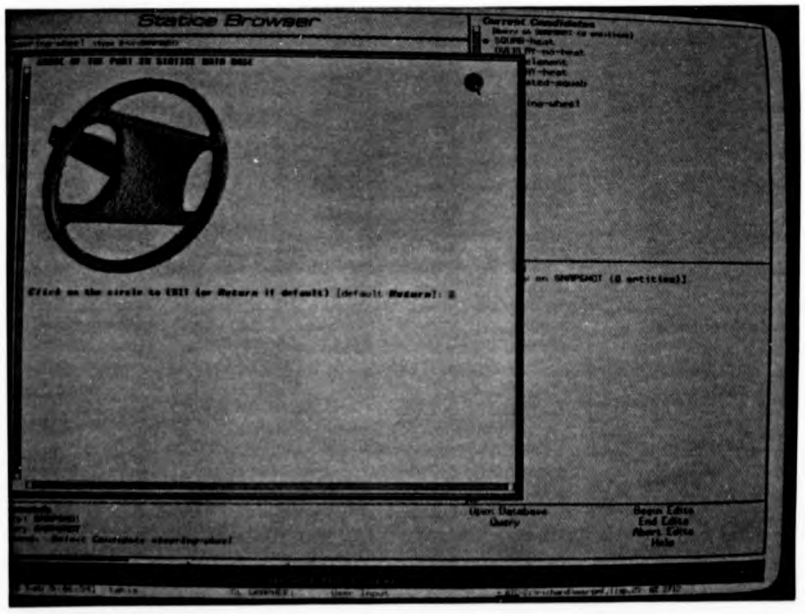
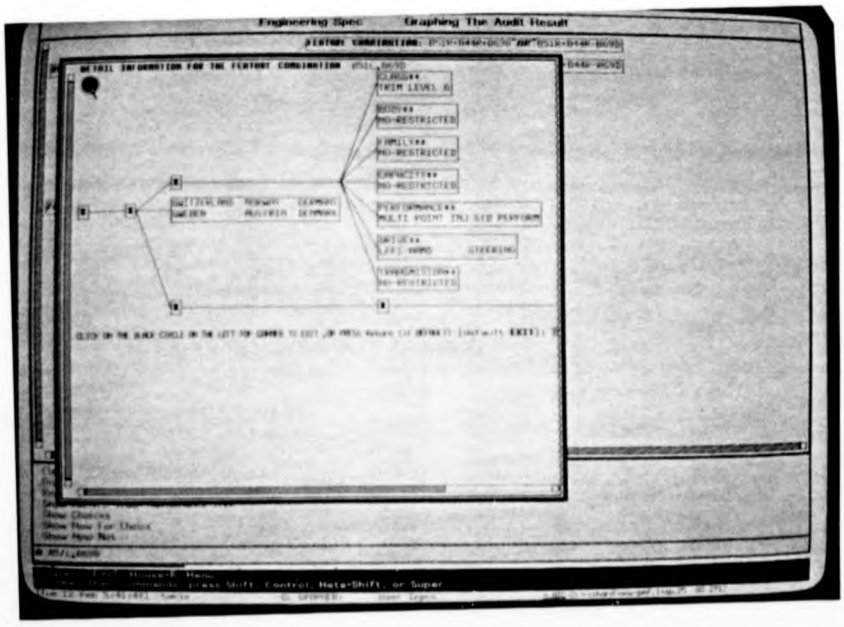


Figure A10: The truth maintenance tree shows how the validation algorithm achieved the combination "B51L + B44B +/- B69D".

Figure A11: By clicking the mouse in a node of the truth maintenance tree (ie. "B51L + B44B +/- B69D" a view window of the restrictions which govern the combination, pop up in the screen



The next photos A11, A12, A13 show respectively the explosion windows of the "B51L + B44B +/- B69D", "B51L + B44R" and "B51L +/- B69D".

Figure A12.

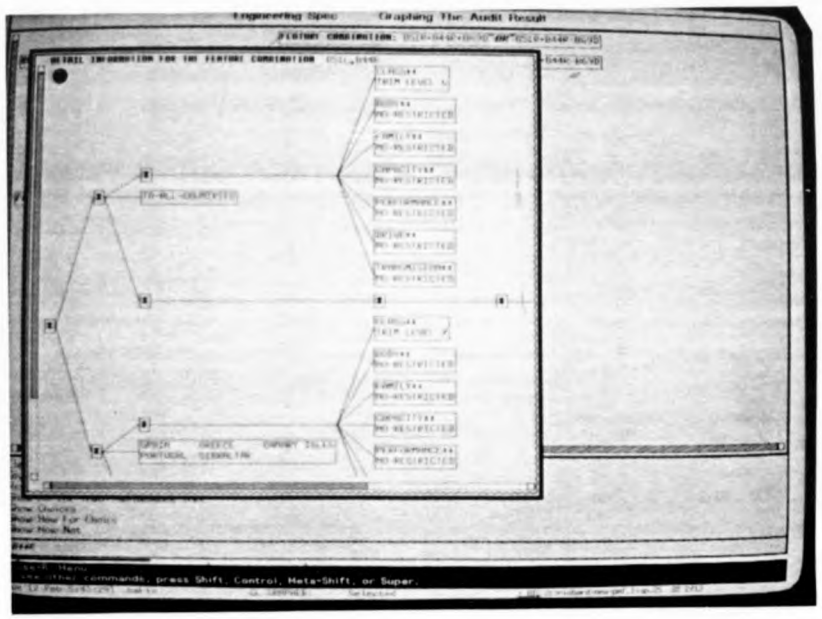


Figure A13.

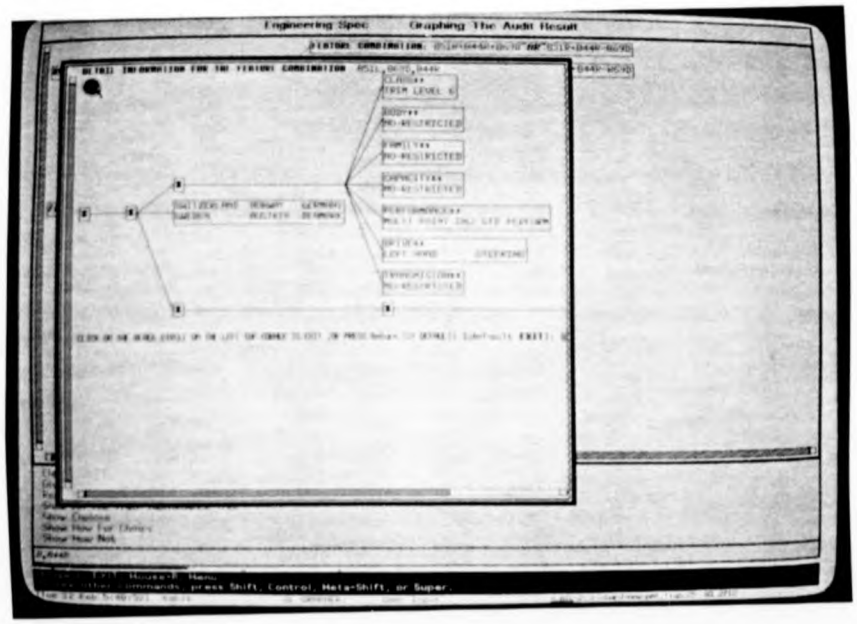


Figure A14: Statice objects representing part description from the APN catalogue are linked by "identity", ie. #entity-handle.. (see volume 1, pg. 318).

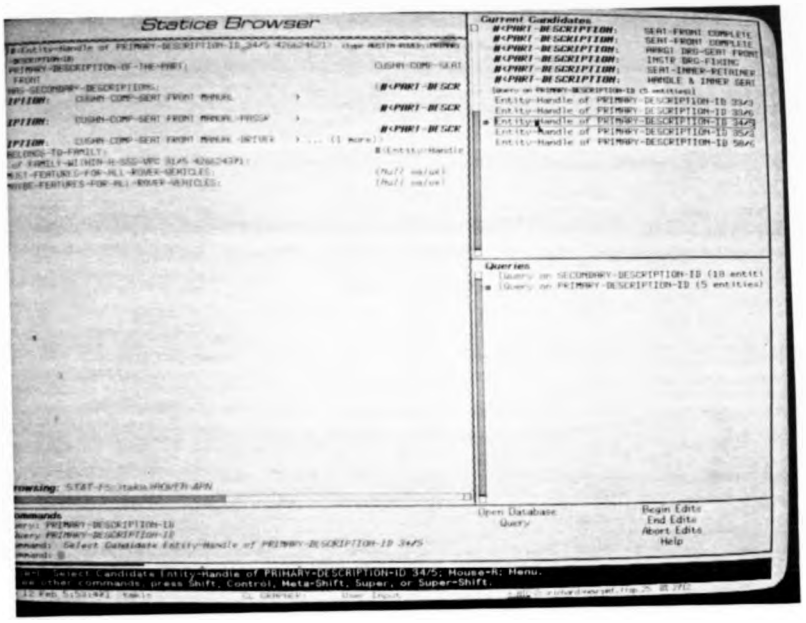


Figure A15: A diagramatic view of the hierarchical way in which APN is represented in Statice.

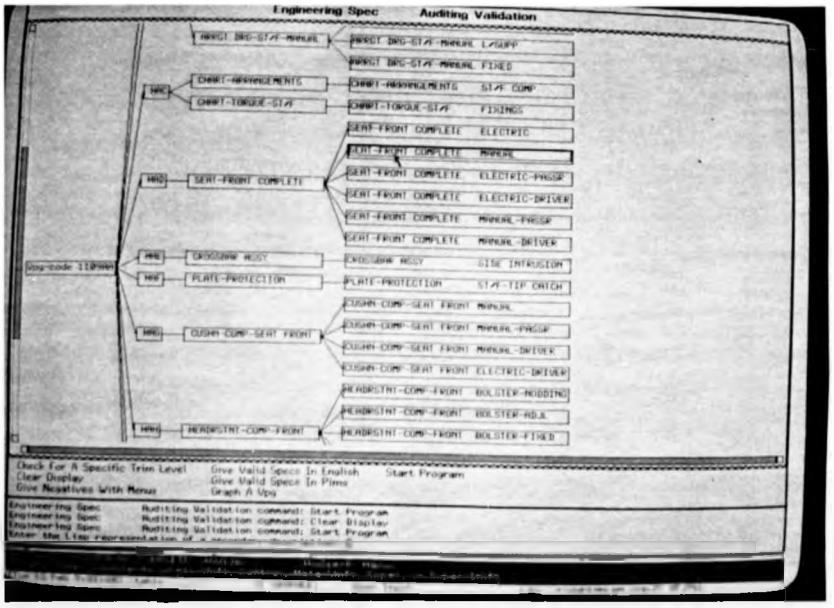


Figure A16: Learning by experience; the system remembers the negative features for the part description under consideration and blacks them out.

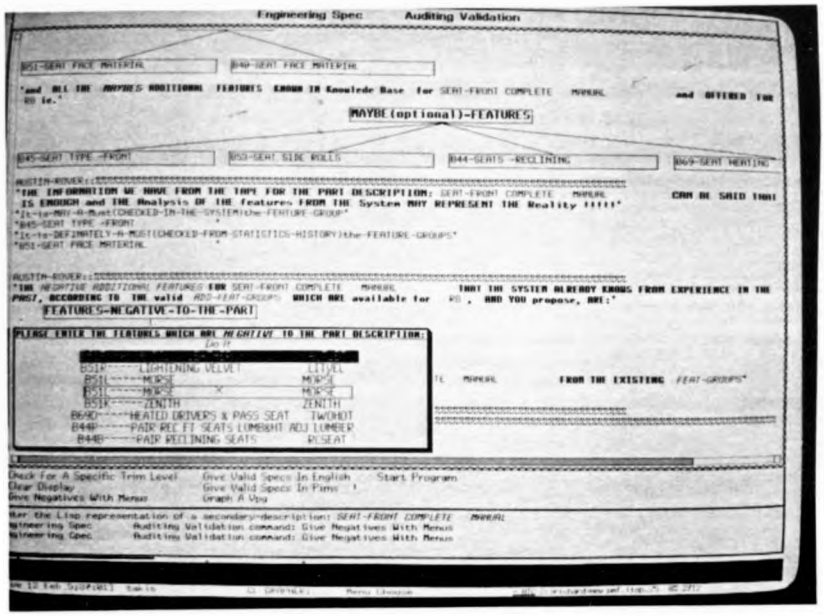
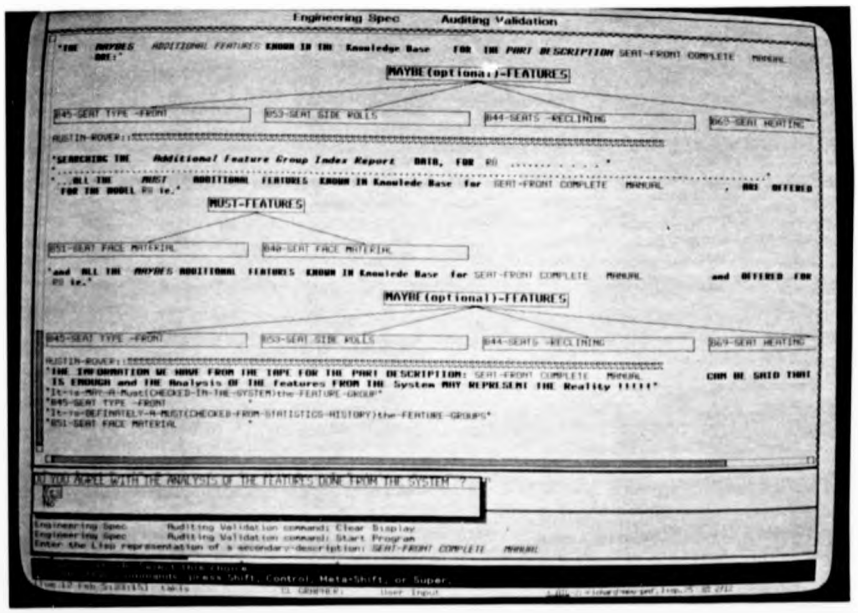


Figure A17: When the system runs it informs the user of the information resulting from the statistical analysis. ie. "B45-seat type front ...can be a *Must* feature group".



APPENDIX 11

INSTRUCTIONS TO RUN ROOVESP

Engineering Spec Auditing Validation Programs - User Commands

Login In.

Command : **SELECT L**
Command : **Login** <Password> **Takis** (type)

Load The Engineering Audit System.

Command: **SELECT L**

load <> **F** <space bar> ile **>atc-0:>takis>statice>*.bin.***

(system takes approximately 5 mins to load)

Running Eng. Spec Program

Command: **SELECT O** (Circle) to get into Eng. Spec System

Select **Start Program** -> from the menu, click LHB

****

Question : Do you wish to enter the Part Description using the VPG code it belongs Yes/No?

Answer **Yes.**

Displays Rover's VPG Codes available in Statice Data Base.

SELECT **VPG Group**, Click on a desc
(select any one by clicking LHB then press Done button.)
(another option can be selected by moving mouse to it and clicking LHB)

Enter the Lisp representation of a second description[default SEAT-FRONT COMPLETE ELECTRICAL-PASSR]:
(**select secondary description** from screen and click LHB).
(then click Done Button).

Enter the model you want to specify the part for: METRO R8 MAESTRO MINI
again **select one** by clicking LHB then click Done button.

System subsequently displays all MUST and MAYBES for the Additional Features
(**press space bar** to continue to display further detail).

System displays Base and Additional Features of the Model for all Available Feature Groups.

Enter the **MUST feature groups** in your opinion from a list of options
(click with LHB - more than one may be selected - then press Done option).

Enter the **MAYBE feature groups** (again as done for above)

Choose the one you want to specify with ?

****

==========

Question : Do you wish to enter the Part Description using the VPG code it belongs Yes/No?

Answer: No

Enter **Part Description** [default SEAT FRONT COMPLETE ELECTRICAL] from the APN catalogue, it then goes back to previous method for picking car model.

Do you agree with the conbimation of the features done from the system Yes/No ?
Answer : No

Do you want to enter your own feature group from scratch Yes/No ?
Answer : No

Select **MUST** features representing correct spec -
click LHB, select more than one, then click Done button.

Select **MAYBES** features used for the spec of part description
- click LHB, select more than one, then click Done button.

Select features which do not match - just select Done button do not select any features.

Wait - system processing.

==========

Give Valid Specs in English
May have to press space bar to show more than one screen of information.

Give Valid Specs in PIMS.
May have to press space bar to show more than one screen of information. Information layout
Pims Stmts, Base Features, Countries, Combinations, Restrictions

Graph a VPG.
Enter the **VPG code** your part Description belongs 1110DA, 1110EA, 1109AA.
select by LHB.

Give Negatives with Menus.
Enter the **features which are negative** to the part description e.g. B517 Autolux leather

<u>Note</u>
Do not alter any negatives - since this has an effect on the links provided in the database- could have disasterous consequences.

Clear Display.

Simply clears the screen of any information.

Check for a Specification Level.

Enter the **trim** you want to search through e.g. M S W X Y Z.
select by LHB, message bar also displays the numeric value for that trim level.

Graphing of the Feature Groups

SELECT Square -> for graphing
RESET

Clear the Display
Simply clears the display.

Reset
Reset the graphing capability.

Show How Not
Enter the Feature Combination you want to examine in more detail.
select from main screen - feature comb. click LHB

Enter the Feature Group you want to examine in detail, click on second.
Displays additional window
Click in black circle to exit additional window.

Show How For Choice.

Show all Truth Maintenance Tree.
Displays the Feature Combinations for particular model.

IMAGING AND ASSEMBLY DESIGN

SELECT L

Command : login <> Takis - only if login procedure has not been previously performed within the session.
SELECT F dont need to do a return
Tree Edit Home Dir
Select lisp-lore -> LHB
grapher -> LHB

go to grapher system.lisp.8 click on RHMB
go to edit file LHB
wait
go to defpackage grapher (in bold type)
click in middle button - either at beginning or end of function
(entire function will be underlined)
Meta W - saves it
go to lisp SELECT L
Control Y -> bring it back (if no new line created then press return)
then press return to execute
say yes to question - only if a question comes up.

Command : SELECT F
go to nodes-and-arcs-trail-for-quantities.bin.1. press RHSMB
then load it -> LHSMB
(enter y to any questions - default yes)
go to display-frame-trial-for-quantities.bin.1-> RHSMB
then load

go to Presentation-types.bin.3 (loading compiled version) RHS and load

loaded system
(puts presentation program in editor for use or can continue with anything else

Load the images

SELECT F
load the images

go to takis > images
LHS
go to symbolic 3-the network-example
LHS
go to atc-ingres lisp 2
select read-atc-images.lisp.2 RHMB

edit file LHS
compile file, **META X** to compile it

Command: Comp buffer press return twice

Either

click on middle button -> on related image
Meta W to save it

or

type **(reload-images)**

SELECT L
Command : (Control Y to bring it)

Print Path name (and press return to accept)

** reloades images

images now loaded

SELECT Network

if too many images - stack overflow the ABORT

RESET Assembly

SELECT F
select lisp lore
grapher
select lisp version of nodes and arcs trial for quantities click RHS

edit file

look for function -PRESENT-SELF-NODE
SCROLL

Need to comment out Present-Self-Node in the Grapher function - draw image in above function

SELECT E (dit)
CONTROL X CONTROL B - gives you a list of edit buffer.
go to nodes-and-arcs-for-quantities.lisp
click LHB
go to top of file
Press SCROLL button twice.
Not the ones that are commented out completly.

Inside Present -Self-Node
go to front of function
click Middle Mouse Button to highlight entire function
SCROLL down to find (graphics:draw-image
move mouse to front of that
click Middle Button to select from graphic:draw-image down to end of function
Control X Control; to comment out that portion of the code
Meta Scroll go up to top of function
Move mouse to start of function
Click Middle Button - to highlight all the function
Click RHB -> to select the menu
Select Compile Region from menu, click LB

<u>Note</u>
A message will be displayed showing if the function has been compiled successfully. Make sure it comes up, otherwise try compiling again.

SELECT Network -> then do RESET assembly LHB
-> no images displayed just circles.

To show images

Select Ed
Middle
Control - Control X Control ; - takes out remark statements

Imaging Operations

Add Ass. Link

click LHB, add arc (a Physical part) either takes the systems default or click on one of the screen objects.
Prompt returns to select another Physical Part
Then press return.

Build New Assy

Create Part
click LHB, then creates a new node (screen object)

Delete Assy Link

Click LHB, then select one physical screen object - default always given.
click on end physical part, press return.
May have to type a character to refresh screen.

Delete Part.

Click LHB, then click on Physical Part to delete,
press return,
then prompts do you really want to delete
click on message to confirm

Move Part
Click on LHB, click in physical part to move it,
press return
then click somewhere else on the screen.

Reset Assembly
click LHB, resets the assembly structure.

Set Description

Note
1. First time may be a stack overflow - question may be asked if you want to increase stack -> click y, then it continues -> do a RESET assembly the OK

2. If problem loading images then select Skip redisplay of pane and then Reset. This problem is as a result of a node not being found in the database

SELECT NETWORK.

Additional Commands include :
Left Mouse Button - will create a new objects
Middle Mouse Button - will create a link between to objects.
Control Left Mouse Button - deletes a link between two objects.
Control + Middle Button deletes an object
Mouse Left Button -changes the description.
Shift + Mouse Left - to move an object around the screen.
Shift + Middle Mouse Button - show image from Statice Object Data Base.

Statice Objects Browser

Select L
Command: Click Right Hand Button on Mouse
Select System Menu Option (LHB)
Select Statice Browser for Program Menu (LHB)
Select Open DataBase (LHB)
Enter pathname -> stat-fs:>takis>rover-apn
Select Query (LHB)
Query on entity QUERY: se<press COMPLETE key)
Click on attribute - the ANY bit.
This any bit becomes blank
Enter a query -> string-search "seat" <press return>
Select <end> use these values

end of query
******

Click on one of the Current Conditions to display for the Statice Browser its values

******

QUERY:pri<COMPLETE>
Rub out any
Enter string-search

******

Opening another DataBase

Select Open Database
Query: stat-fs>takis>image-text
Select Query from Menu
Query:snapshot
Select End
select one of the choices by selecting via the mouse.

****
Example statice entities to use include :

Prim
Secondary-Descriptions

Note

Demo the system first before going into statice browser. If statice database has been opened make sure its the Rover APN database that's open for running.

Maintenace

SELECT Triangle -> to do system maintenance

Note
Do not alter maintenance features, since this will effect the links between objects set up in statice.

Logging Out

Command : logout

Note
When logging out do not save any material to buffer. It may alter links set up in the Object Base. It may also fill up the internal memory of the machine - with the result that files may have to be deleted. Answer No to any questions.

System Specific Commands

Before starting program :

Command :**set package AUSTIN ROVER** - for the main engineering auditing program.
Command: **set package GRAPHER** - for the Imaging part of the program.

Meta Control Y - returns your last command to the command prompt.

In the imaging part if the system aborts you can select Skip Redisplay from the menu options displayed, then select Reset system

Any problems use **ABORT** always if that doesn't work then use **CONTROL ABORT**