

## Table of Contents for the (pre)compiler

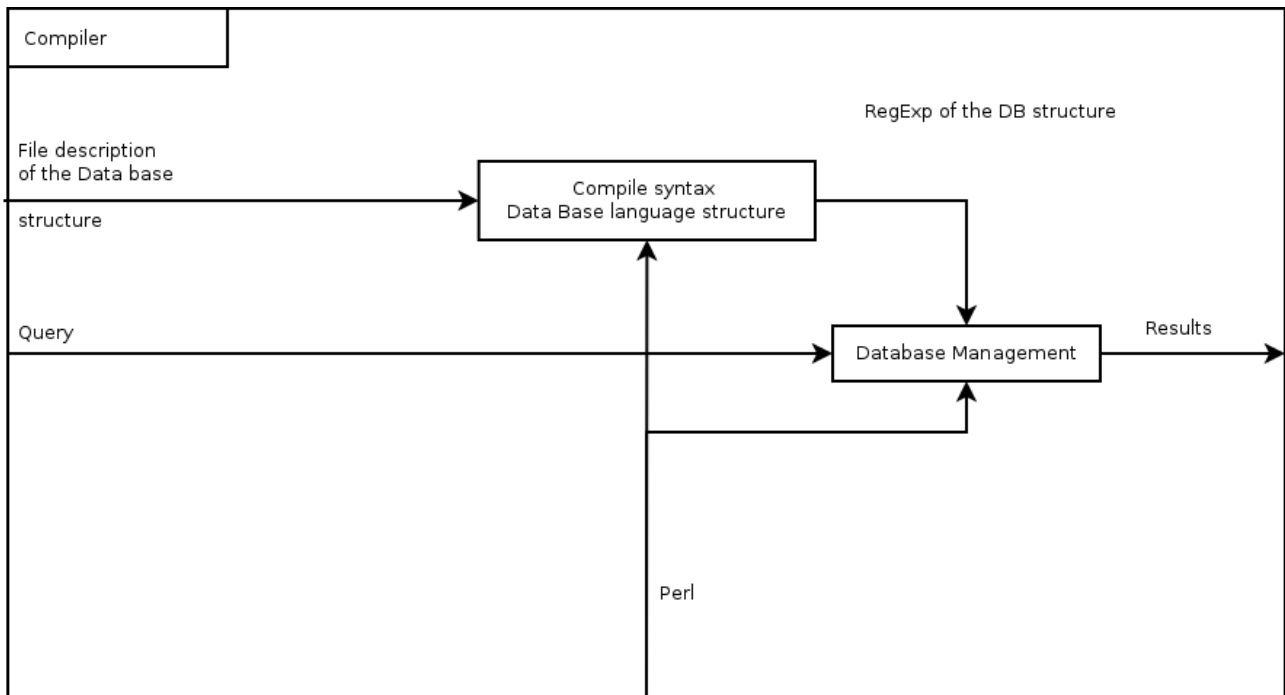
Why do we need to implement a program that analyze a file descriptor for a compiler.....	3
Description of the basic idea.....	4
Introduction.....	4
Description of dictionaries.....	4
- Description of the syntax.....	4
- Description of the grammar.....	4
- Description of the words.....	4
The architecture description.....	4
Structures of the dictionaries.....	5
Definition of the features of this parser.....	6
Basic mechanism.....	6
Security.....	7
Feature of the data base structure definition compiler.....	8
- Operator to comment.....	8
- Operator to define a rule.....	8
- Name a rule to define.....	8
- Name of the rule.....	8
- To define a leaf.....	8
- Parenthesis .....	8
- Logical operator: .....	8
Pre-compilation for the query.....	9
The algorithm.....	10
What is the syntax allowed by the (pre)compiler.....	12
- Token definition.....	12
- Cardinality.....	12
Basics:.....	12
Advanced:.....	12
Automata definitions.....	13
Sketches.....	13
Lower case character automaton.....	15
Definition.....	15
Conway diagram.....	15
Upper case character automaton.....	15
Definition.....	15
Conway diagram.....	15
Upper and lower case character automaton.....	15
Definition.....	15
Conway diagram.....	16
Number the character automaton.....	16
Definition.....	16
Conway diagram.....	16
Definition of the underlines automaton.....	16
Definition.....	16
Conway diagram.....	16
The token name automaton.....	16
Definition .....	16
Conway diagram.....	17
Definition of a token name rule automaton.....	17

Definition.....	17
Conway diagram.....	17
Definition of a token name definition automaton.....	17
Definition .....	17
Conway diagram.....	17
Alphabetical indexes.....	18
Illustration indexes.....	19

Why do we need to implement a program that analyze a file descriptor for a compiler

We want to write a program that do Data Base management. We want to set the compiler as we want with the syntax that we want. For ease of use and, modification we describe the syntax, grammar, constraints in a single file.

Hence this file will be parsed and analyzed in order to compare it with the user's syntax. If the user's syntax is not correct then there will be error of compilation. Nothin' much...



*Illustration 1: Structure of the compiler*

This drawing describe the mecanism of the compiler.

## Description of the basic idea

### **Introduction**

The aim of this document is to describe a way to optimize the development of a compiler for implementing a data base (see below).



*Illustration 2: Basics with the SHELL*

### **Description of dictionaries**

#### **- Description of the syntax**

We define a set of rules that define a syntax.  
The definition of the syntax is not yet defined see later.

#### **- Description of the grammar**

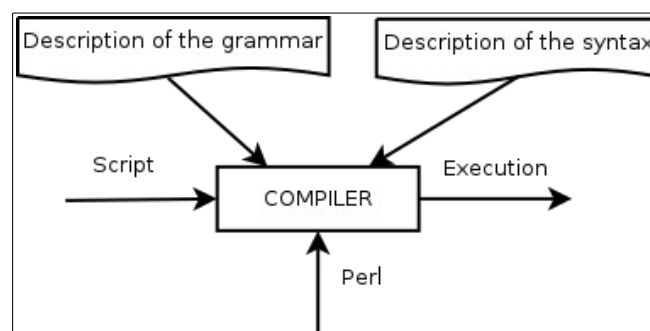
We define a set of rules that define the grammar.

#### **- Description of the words**

We define a dictionary that define the root words that can be used. The notion of a word is not yet defined.

### **The architecture description**

Architecture the basic compiler is composed of file descriptors. That file descriptors can be seen as a dictionary that will be read and analyzed by the compiler.



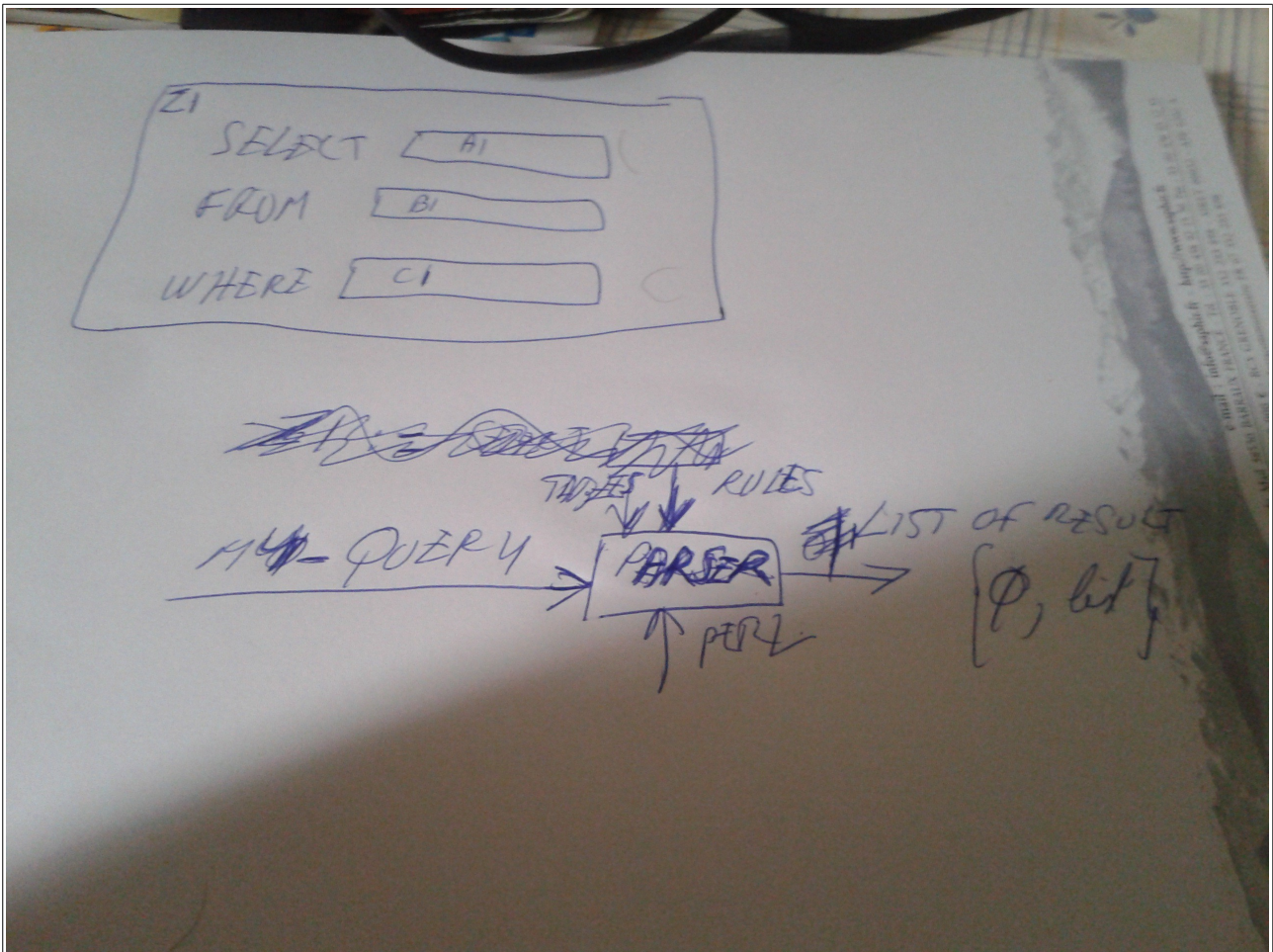
*Illustration 3: Architecture of the basics*

This will avoid to rewrite the rules of the compiler each time.

We need to define what are the basics to be coded. Need to defines the verbs, actions,...  
This syntax then will refer to a file that contains the set of words. All together it will define a language.

### **Structures of the dictionaries**

The structure is not yet defined but the format will probably be XML.



*Illustration 4: Basic information related to the file that defines structures*

Now here we have the file that do the definition.

```

# Definition of the syntax
# -----
#
# Definition of syntax must start with DEF_ prefix followed by :=
DEF_STRUCTURE_QUERY:=SELECT <STRUCTURE_COLUMN_NAMES_FOR_RESULT> FROM <STRUCTURE_TABLE_LISTS> WHERE <STRUCTURE_FOR_QUERIES> ;
DEF_STRUCTURE_COLUMN_NAMES_FOR_RESULT:=<STRUCTURE_FOR_A_COLUMN_NAME> | <STRUCTURE_FOR_SEPARATOR_COLUMN_NAMES_FOR_RESULT> <STRUCTURE_FOR_A_COLUMN_NAME>
DEF_STRUCTURE_TABLE_LISTS:=<STRUCTURE_FOR_A_TABLE_NAME> | <STRUCTURE_FOR_SEPARATOR_TABLE_NAMES> <STRUCTURE_FOR_A_TABLE_NAME>
DEF_STRUCTURE_FOR_A_COLUMN_NAME:=[a-zA-Z]
DEF_STRUCTURE_FOR_A_TABLE_NAME:=[a-zA-Z]
DEF_STRUCTURE_FOR_SEPARATOR_COLUMN_NAMES_FOR_RESULT:= ','
DEF_STRUCTURE_FOR_SEPARATOR_TABLE_NAMES_FOR_RESULT:= ','

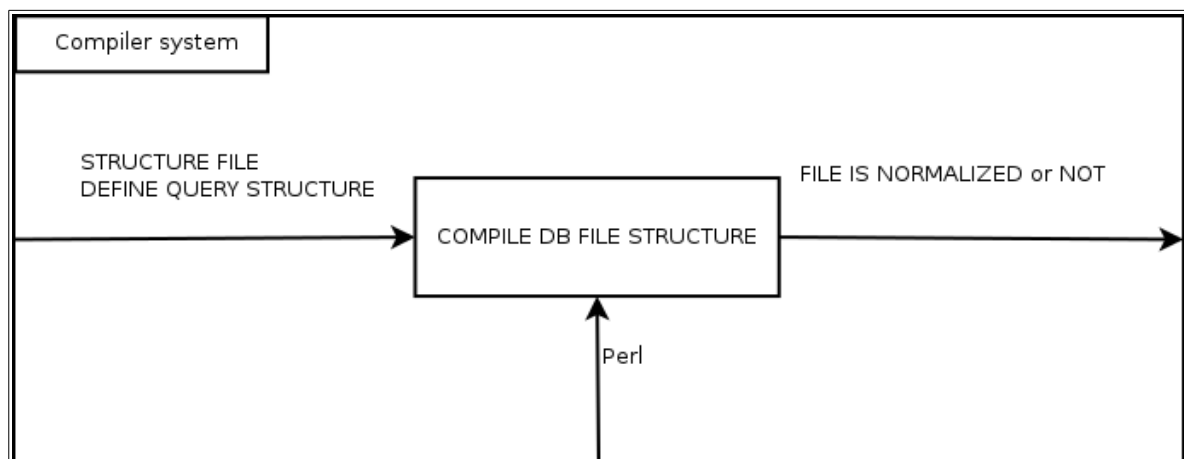
```

*Illustration 5: Basic definition of a syntax structure of a query no queries defined yet*

From this file a parser need to be constructed. This is only a sample. Definitions need to be optimised.

### **Definition of the features of this parser**

To parse this file, to analyze it, to check it a Perl script will be built. It will check any errors in the syntax, if no errors are found then it will store all the syntax information in the memories. Then it will be used to parse the query defined by the user. Its kinda compiler see schema below.



*Illustration 6: Basic description of the system that analyze file that describeData Base query syntax structures*

### **Basic mechanism**

We do macro compilation. Go to the leaf definition. We replace to the definition into the father definition. Then do the replacement to the father of the father and so on and so fourth. At the end we must have the first definition in the file that will

be replace by its basic definition. Then this can be match with the final definition(the user one). The recursive definition will shows up as \_ in the root definition (to redefine not well formulated) but in the final version this will be removed. In the definition of the format of the query only the syntax in the definition will be check, syntax too:

```
DEFINITION_NAME <def> ROOT
DEFINITION_NAME1 <def> NAME
DEFINITION_NAME2 <def> NAME NAME1
DEFINITION_NAME2 <def> NAME _
```

Example of file that define the format of a query will be digested by the compiler with OR but not the operator

instead of the next definition which is much more concise:

```
DEFINITION_NAME <def> ROOT
DEFINITION_NAME1 <def> NAME
DEFINITION_NAME2 <def> NAME NAME1
                    | NAME _
```

Example of file that define the format of a query will be digested by the compiler with OR operator '|' used

Recursivity will be defined with the \_ operator. We can suppose or guess that in the final version of this script this will be optimized. No cardinality is defined yet.

The OR operator is not yet defined. A good guess will be enumeration within the file (not good solution). We only have to check the syntax. The syntax then will define the format of the query. This is where we will compare the user definition with the syntax of the given query. To group elements parenthesis can be used. It is not defined yet.

## **Security**

Dictionaries can be encrypted. Algorithm not defined yet.

## Feature of the data base structure definition compiler

### - **Operator to comment**

Must have comment operator. This operator is a # followed by text. This character can be everywhere.

### - **Operator to define a rule**

To define a rule it must have ':='. It must contain a **DEF** at the left followed by the name of the rule. The format of the name of the rule will be defined later.

### - **Name a rule to define**

It must contain a **DEF** and then as it is defined in the definition of the name of the rule in the next paragraph but no '<' '>' characters is required.

### - **Name of the rule**

Must start with a CAPITAL LETTER than can have any character from the alphabet (CAPITAL LETTER TOO) and NUMBER from 0-9 as much as needed. It can contain the \_ character anywhere in the rule except at the first character place in the rule. The rule when use in a definition must be between '<', '>' characters.

### - **To define a leaf**

We declare it a name of a rule to define then the operator to define a rule and then definition. That's the atom. It cannot be divided as a subrule.

A set of characters can be defined as [a-c] meaning *from character a to character c*.

A character can be defined as a

Cardinality is + or \*. The + means one or more but \* mean 0 or more than 0.

### - **Parenthesis**

They can be used to group rules.

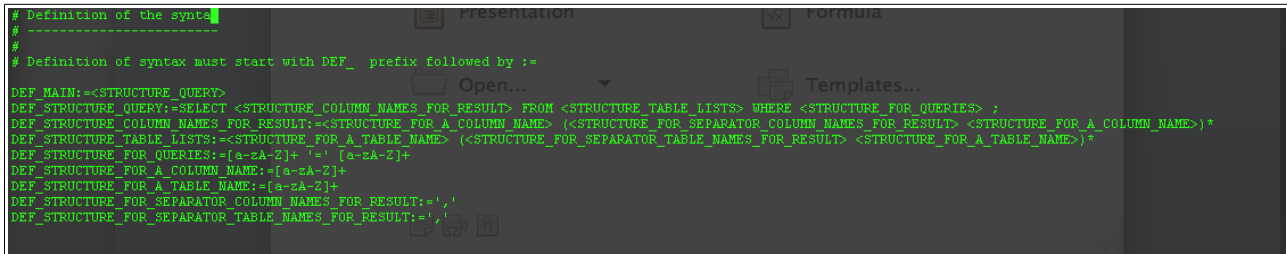
### - **Logical operator:**

The character | is or operator.



## Pre-compilation for the query

The query of a given user must be checked. To do that a syntax pre-compiler is necessary. First in a given file we describe the syntax. Then we create from this syntax a regular expression that will parse a given query.

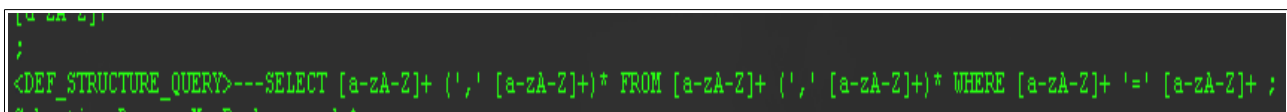


```
# Definition of the syntax
#
# Definition of syntax must start with DEF_ prefix followed by :=
DEF_MAIN:=<STRUCTURE_QUERY>
DEF_STRUCTURE_QUERY:=SELECT <STRUCTURE_COLUMN_NAMES_FOR_RESULT> FROM <STRUCTURE_TABLE_LISTS> WHERE <STRUCTURE_FOR_QUERIES> ;
DEF_STRUCTURE_COLUMN_NAMES_FOR_RESULT:=<STRUCTURE_FOR_A_COLUMN_NAME> (<STRUCTURE_FOR_SEPARATOR_COLUMN_NAMES_FOR_RESULT> <STRUCTURE_FOR_A_COLUMN_NAME>)*
DEF_STRUCTURE_TABLE_LISTS:=<STRUCTURE_FOR_A_TABLE_NAME> (<STRUCTURE_FOR_SEPARATOR_TABLE_NAMES_FOR_RESULT> <STRUCTURE_FOR_A_TABLE_NAME>)*
DEF_STRUCTURE_FOR_QUERIES:=<[a-zA-Z]+> '=' <[a-zA-Z]+>
DEF_STRUCTURE_FOR_A_COLUMN_NAME:=<[a-zA-Z]+>
DEF_STRUCTURE_FOR_A_TABLE_NAME:=<[a-zA-Z]+>
DEF_STRUCTURE_FOR_SEPARATOR_COLUMN_NAMES_FOR_RESULT:=','
DEF_STRUCTURE_FOR_SEPARATOR_TABLE_NAMES_FOR_RESULT:=','
```

### Full syntax

The given upper file screen copy , describe the syntax allowed.

To start reading it is necessary to have a DEF\_MAIN flag. This is where we start reading the file that describe the syntax for the query.

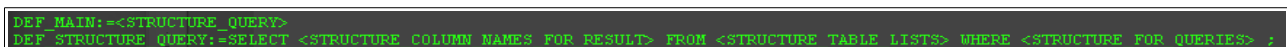


```
[a-zA-Z]+
;
<DEF_STRUCTURE_QUERY>---SELECT [a-zA-Z]+ (' [a-zA-Z]+)* FROM [a-zA-Z]+ (' [a-zA-Z]+)* WHERE [a-zA-Z]+ '=' [a-zA-Z]+ ;
```

### Example of transformation

Left side there is the tag and right side there is the query transformed as a regular expression.

Below is the description at the beginning before the transformation:



```
DEF_MAIN:=<STRUCTURE_QUERY>
DEF_STRUCTURE_QUERY:=SELECT <STRUCTURE_COLUMN_NAMES_FOR_RESULT> FROM <STRUCTURE_TABLE_LISTS> WHERE <STRUCTURE_FOR_QUERIES> ;
```

### Description of the query

The transformation works fine but some more tests need to be done.

## The algorithm

### - Introduction

We defined a file that contain an automata that fits to the definition of the regular expression of the structure of a query to manage a data base. This automata must be transformed when the algorithm is finished to a regular expression.

### - On 20121110

Presentation of the algorithm used.

First we store in a hash table all token(name and definitions). We replace each token in the main definition b its corresponding value.

```
DEF_MAIN:=<STRUCTURE_QUERY>
DEF_STRUCTURE_QUERY:=SELECT <SPACE>+
<STRUCTURE_COLUMN_NAMES_FOR_RESULT> FROM <STRUCTURE_TABLE_LISTS>
WHERE <STRUCTURE_FOR_QUERIES>;
DEF_STRUCTURE_COLUMN_NAMES_FOR_RESULT:=<STRUCTURE_FOR_A_COLUMN_NAME>
(<STRUCTURE_FOR_SEPARATOR_COLUMN_NAMES_FOR_RESULT>
<STRUCTURE_FOR_A_COLUMN_NAME>)*
DEF_STRUCTURE_TABLE_LISTS:=<STRUCTURE_FOR_A_TABLE_NAME>
(<STRUCTURE_FOR_SEPARATOR_TABLE_NAMES_FOR_RESULT>
<STRUCTURE_FOR_A_TABLE_NAME>)*
DEF_STRUCTURE_FOR_QUERIES:=[a-zA-Z]+=[a-zA-Z]+
DEF_STRUCTURE_FOR_A_COLUMN_NAME:=[a-zA-Z]+
DEF_STRUCTURE_FOR_A_TABLE_NAME:=[a-zA-Z]+
DEF_STRUCTURE_FOR_SEPARATOR_COLUMN_NAMES_FOR_RESULT:=', '
DEF_STRUCTURE_FOR_SEPARATOR_TABLE_NAMES_FOR_RESULT:=', '
DEF_SPACE:=\
```

Example of automaton

DEF\_ prefix means that is the definition name and := means that the expression after this is the definition.

When there if <NAME> this means that it has its corresponding definition <DEF\_NAME>.

### Problem

When we do this algorithm we have problem with extra spaces. See below:

```
SELECT <SPACE>+ <STRUCTURE_COLUMN_NAMES_FOR_RESULT>
FROM <STRUCTURE_TABLE_LISTS> WHERE
<STRUCTURE_FOR_QUERIES>;
```

See spaces between each token

### Solution

Do not put white spaces between each token.

-> will not be easy to read.  
May be there is another solution.

What is the syntax allowed by the (pre)compiler

to enhance later

### **- Token definition**

TOKEN: is the name of the tag. It can be defined or used as a tag.  
The token can be seen as a syntactic sugar.

:= : at left side is the token to define right side is the token  
already defined so it must be between <>.

A:=<B>

The syntax must be well defined. Look for automaton...

### **- Cardinality**

#### **Basics:**

\*: match 0 or more time the character or group of character tht is before.

+: match character 1 or more time the character or group of character that is before (if left most is used).

(): is the sequence of characters that group an expression.

#### **Advanced:**

A..C: character in the interval from A to C.

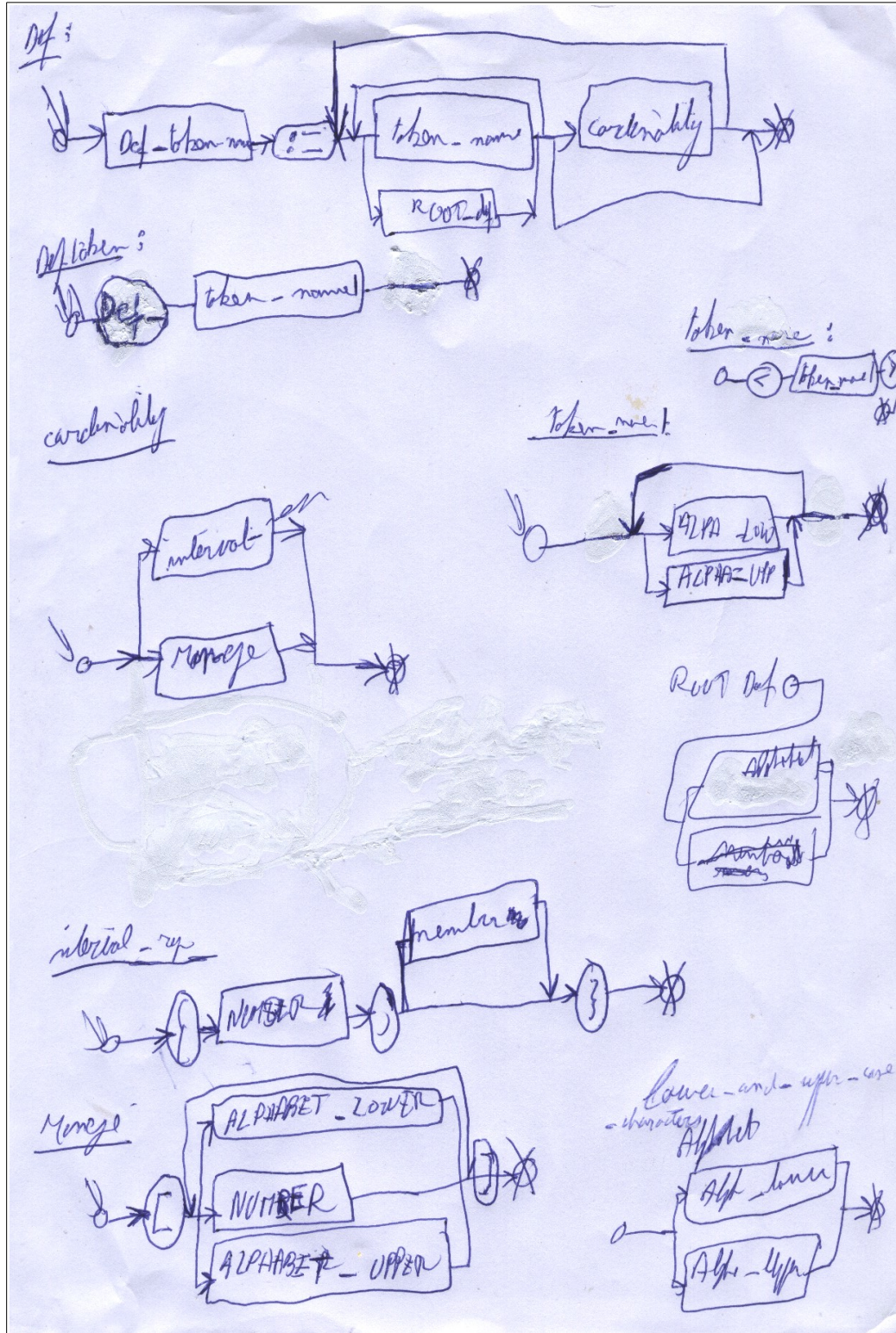
{a,b}: define the cardinality or the number of time that the character or group of character between () just before {} can be repeated to match an expression. The a value belongs to  $R^*$  and, b must be greater than or equal to a. If no b value is set this means infinite time.

[AC]: one of the character A or C

## Automata definitions

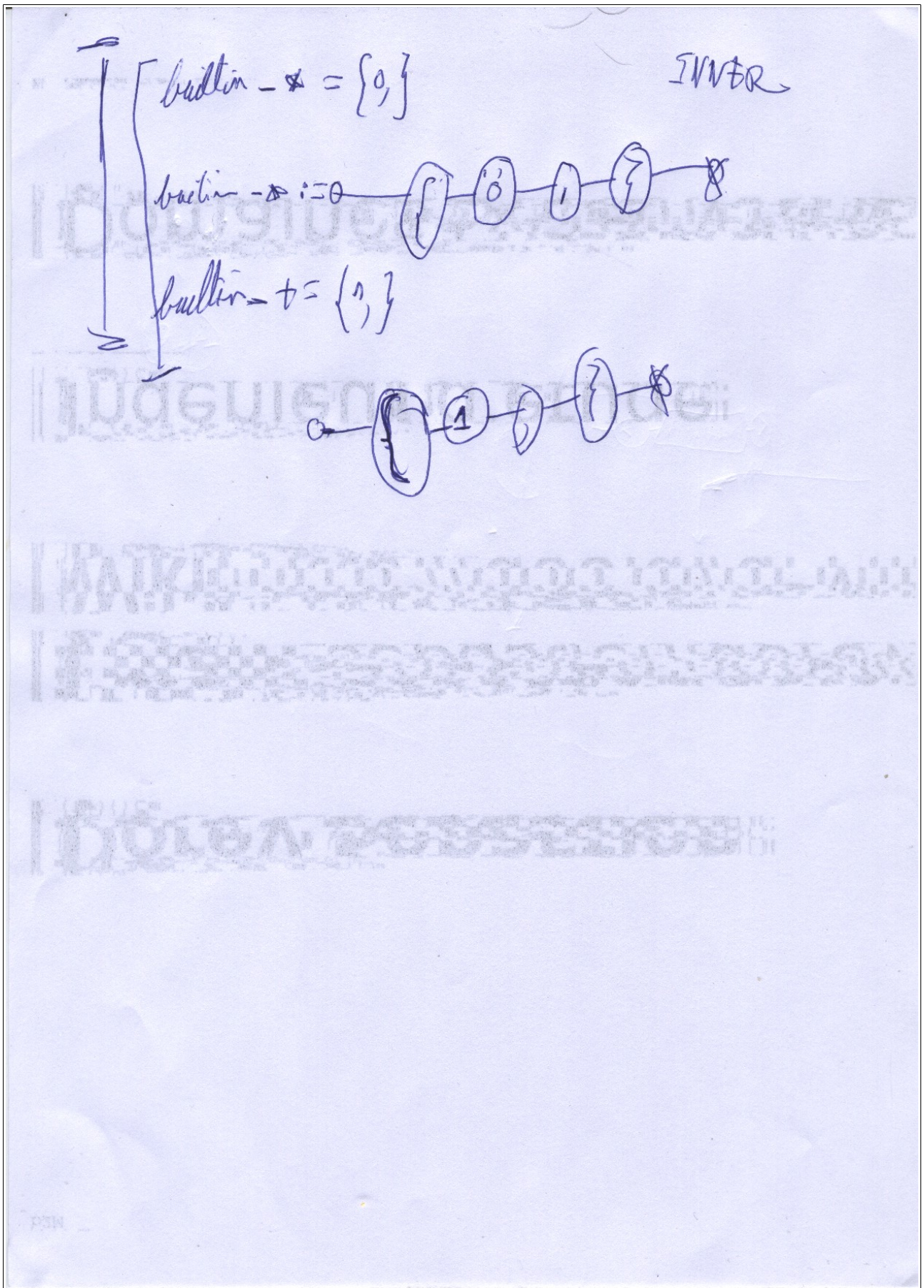
### Sketches

Before programming sketches are necessary. At a first glance it is not right but it can give a good sight of what's going on.



Sketches of description of syntax





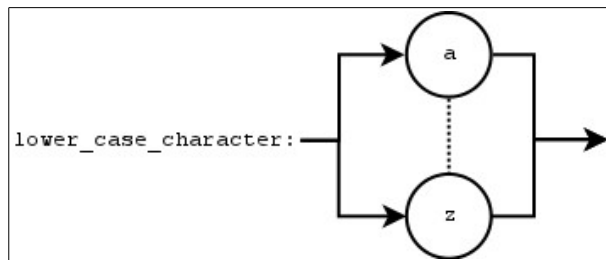
Sketches of definition of cardinality

### ***Lower case character automaton***

#### **Definition**

The alphabet is from 'a' to 'z'. A word must contain one lower case character.

#### **Conway diagram**



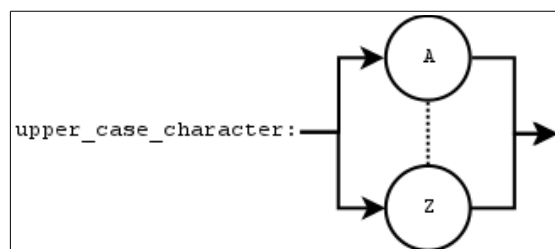
Definition of lower\_case\_character

### ***Upper case character automaton***

#### **Definition**

An alphabet is from 'A' to 'Z'. One word must contain one upper case character.

#### **Conway diagram**



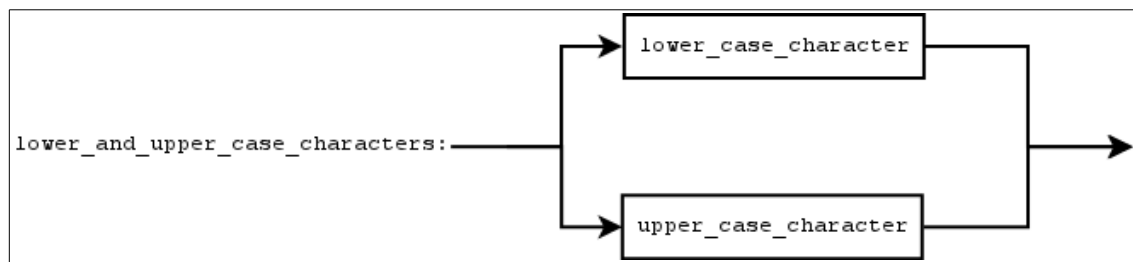
Definition of upper\_case\_character

### ***Upper and lower case character automaton***

#### **Definition**

The alphabet must contain a letter from a to z or from A to Z: upper\_case\_character, lower\_case\_character. It is the agregation of the two previous automaton with a fork on each alphabet.

Conway diagram



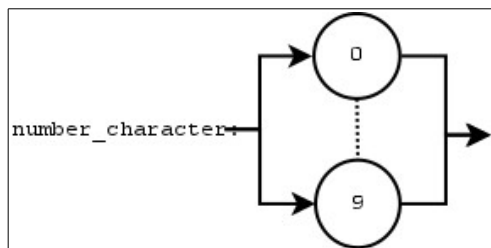
Definition of lower\_and\_upper\_case\_character

*Number the character automaton*

Definition

The alphabet must contain one character from 0 to 9. It as to be taken as a character.

Conway diagram



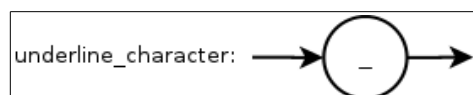
Definition of number\_character

*Definition of the underlines automaton*

Definition

An alphabet is '\_'. One word must contain one underline character.

Conway diagram



Definition of underline\_character

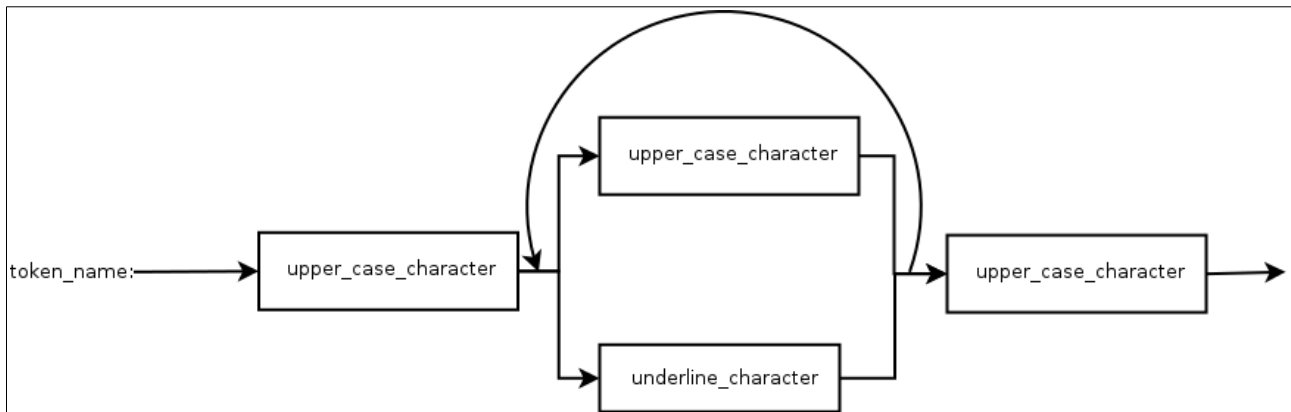
*The token name automaton*

Definition

It is a word that must start with an uppercase\_character followed n times by an underline\_character or an uppercase character. The word must end with an uppercase\_character.



### Conway diagram



Definition of token\_name

### *Definition of a token name rule automaton*

#### Definition

This starts with '<' character followed by the token\_name and followed by '>' character.

### Conway diagram



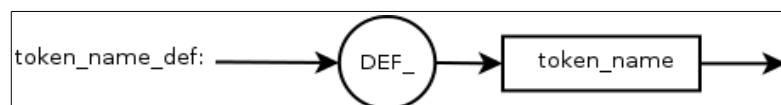
Definition of token\_name\_rule

### *Definition of a token name definition automaton*

#### Definition

To define a token name this must start with the prefix DEF\_ then it must be followed by the token name itself. A token name is a set of upper\_case character repeated n times.

### Conway diagram



Definition of token\_name\_def

## Alphabetical indexes

### Alphabetical Index

Automaton.....	
Description of the query.....	9
Example of automaton.....	10
Example of file that define the format of a query.....	7
Example of transformation.....	9
Full syntax.....	9
See spaces between each token.....	10
Compilation.....	
cardinality.....	12
Cardinality.....	12
syntactic sugar.....	12
token.....	12
Token.....	12
TOKEN.....	12
Conway diagrams.....	
Definition of lower_and_upper_case_character.....	16
Definition of lower_case_character.....	15
Definition of number_character.....	16
Definition of token_name.....	17
Definition of token_name_def.....	17
Definition of token_name_rule.....	17
Definition of underline_character.....	16
Definition of upper_case_character.....	15
Sketches of Conway diagrams.....	
Sketches of definition of cardinality.....	14
Sketches of description of syntax.....	13

## Illustration indexes

### Illustration Index

Illustration 1: Structure of the compiler.....	3
Illustration 2: Basics with the SHELL.....	4
Illustration 3: Architecture of the basics.....	4
Illustration 4: Basic information related to the file that defines structures.....	5
Illustration 5: Basic definition of a syntax structure of a query no queries defined yet.....	6
Illustration 6: Basic description of the system that analyze file that describeData Base query syntax structures.....	6