RENAULT

Software quality rules for programming of automated systems

EB03.07.010 /B

Standard

Status **Binding**

Important Note : This document has been translated from the French. In the event of any dispute, only the French version is referred to as the reference text and is binding on the parties.

Purpose	To lay down the main rules to be complied with to ensure a good level of control system programs (controller, numerical control, robots, PC...).
Scope	Renault Group.
Issuer	65940 - Industrial information systems and bodywork automation
Confidentiality	Non-confidential

Approved by	*Function*	*Signature*	*Date of application*
J.COANT	Department 65940 Manager		03/2001

Document history

Version	*Update*	*Purpose of the main modifications*	*Author*
A	02/1999	Creation	(1) C.PRUVOST
B	03/2001	Adaptation of document for Mech. Eng.	(1) J-P.FARRUGIA

Replaces	EB03.07.010. of 02/1999
Available from	Department 65931 - Normalisation des Biens d'Equipement (*Capital Goods Standardisation*) Fax: 01 34 95 81 79 Tel: 01 34 95 82 10 E-mail: norminfo.moyens@renault.com

Documents quoted

Regulations	:	
International	:	CEI 61131-3.
European	:	
English	:	
CNOMO	:	
Renault	:	
Other in-house doc.	:	
Other external doc.	:	

Codification ICS: 25.040.01; 03.120.99

Class E03

Keywords Automate programmable, automatisme, logiciel, programmation, programmable controller, automation, software, programming.,

Language English

(1) *Assisted in authoring the document*

Mgt.	Dept.	Name	Mgt.	Dept.	Name
DDIV/DPSI	65940	Gérard Bardou	DDIV/DPSI	65940	Giuseppe Lionetti
DDIV/DPSI	65940	Patrick Chemla	DDIV/DPSI	65940	Bruno Panel
DDIV/DPSI	65940	Fabien Delaveau	DDIV/DPSI	65940	Pascal Pottiez
DDIV/DPSI	65940	Jean Claude Gérard	DDIV/DPSI	65940	Claude Pruvost
DDIV/DPSI	65941	Doïc Brochon	DDIV/DPSI	65940	Pierre Nicolas
DDIV/DPSI	65941	Alain Chaillou	DDIV/DPSI	65941	Jean-Louis Ragois
DDIV/DPSI	65941	Gérard Daclon	DDIV/DPSI	65941	Bernard Trelcat

Contents

Page

Foreword ... **4**

1 General ... **4**

2 Modularity rule (division into programme blocks) ... **5**

3 Hierarchy rule ... **5**

4 Language rule ... **6**

5 Structure rule ... **6**

5.1 Writing order ... 7
5.2 Standard creation of a movement equation ... 8
5.3 Combining variables ... 8
5.4 Format of equations ... 8
5.5 Breakdown of equations ... 8
5.6 Interlocking of the controls ... 9
5.7 Check to ensure the limit switches function correctly ... 10
5.8 Activation/deactivation of the variables ... 10
5.9 Stored variables ... 10
5.10 Intermediate variables ... 10
5.11 Independence from the hardware ... 10
5.12 Cases specific to Structured Text language ... 11

6 Abstraction rule ... **12**

7 Variable designation rule ... **13**

7.1 General ... 13
7.2 Complex functios ... 14

8 Organisation rule ... **14**

9 Documentation ... **15**

9.1 Programme Documentation ... 15
9.2 Functional analysis ... 17
9.3 Organic analysis ... 17
9.4 Functional boxes or complex functions ... 17

10 List of cited documents ... **17**

11 Glossary ... **17**

Foreword

Software quality is a major factor in achieving:

- a rapid increase in the work rate of installations,
- a long-term rate of performance in compliance with the technical specifications,
- rapid familiarisation of the people concerned with automation.

It is useful to recall that the software is only a reflection of a requirement expressed by the customer, namely Renault. It is therefore important for Renault to take the necessary measures to formalise its requirements. The requirements are defined in a detailed booklet: "functional analysis". This analysis may, where necessary, be imposed by Renault or carried out by the supplier. In both cases, it must be subject to a written agreement between Renault and the supplier, prior to writing the programme.

However, no matter how specific the requirements, this may in no way prejudice the required quality level of the software.

Robot programming must comply with the principles for integration of robots into the plc's, as described in the professions guide and manuals. The rules set out in this document are complementary to same.

1 General

This document defines the rules necessary to achieve the expected level of software quality. These quality rules, defined in this document facilitate control throughout the writing of the software and therefore are a fundamental guarantee of the level of quality. They are based on simple principles which help to develop the essential skills which are the only guarantee of quality software.

These are:

- **the capacity** to manage the installation in compliance with the requirements expressed.

  It depends directly on the quality of expression of the requirements, the functional analysis and the rigour of the tests and checks carried out as part of the acceptance procedures. These individual elements are not covered in this specification.
- software **reliability**

  This is the ability of the programme to accomplish its task without failure and its durability. It is an essential factor to guarantee the contractual rates of performance.
- **homogeneity**

  This consists in writing the programme whilst respecting the same logic (writing structure), thereby improving its readability.
- **readability**

  This is dependent on how easily the software can be decoded. Good readability improves maintainability. Programming should enable browsing affording up-tracking to the source of a fault. This should be taken into consideration in the case of use of instantiation.
- **maintainability**

  This is the essential complement for long lasting performance. Maintainability is a fundamental condition for carrying out efficient maintenance operations and thus for guaranteeing performance.

  Furthermore, it is a means of rapid adaptation and low costs for future installation developments.

All these factors, which are essential to obtaining quality software shall be supplemented by detailed, rigorous and exhaustive documentation. The documentation consists of two parts:

- the first part covers the content of the software, the variables, equations, functional blocks, etc.,
- the second covers the essential complements for a good understanding of the complex functions.

The rules apply to the control system programs (plc's, numerical control, robots, PC,...).

These general rules can be supplemented by rules which are specific to each job. In this case, the associated documents shall be clearly specified in the relevant technical specifications.

In particular, this includes:

- the use of job-related functional boxes,
- flow management,
- recycling, for the relevant installations,
- fault management.

2 Modularity rule (division into programme blocks)

Modularity involves breaking a system, regardless of its complexity, down into a set of components deemed to be **individually single.** This division **complies with the functional analysis** and leads to components which combine a set of facts or actions which are **logically linked**.

In most cases, a module only includes basic functions.

E.g.:

A loading station (voluntarily single)

Module	⇒	A loading station (global functionality).
The basic functions	⇒	- the part presence inspection function, - the part assembly/disassembly function, - the part movement function (Forwards/Backwards).

It is impossible for a module to be linked to a physical sub-assembly, i.e. the management of the installation start modes or safety zones, etc..

A module can have just one function if it adds to a better understanding.

3 Hierarchy rule

The hierarchy rules give rise to a single and homogeneous assembly (module). This facilitates design control and ensures the software programme is easy to maintain.

The module obtained should enable, apart from writing independent from the other modules, all operations, such as development of the application, backup, duplication, destruction, etc.

In a module:

- Unless absolutely necessary, the **input variables** (physical inputs ⇒ sensors or information from an "intelligent" device via an intermediary) cannot be used by another module.
- The **intermediate variables** specific to one module cannot be used by another module.
- More generally, the variables used in a module cannot be used in another module, except for the variables dedicated to this purpose (exchange variables) and general variables of the "operating mode" or "component variable" type, etc..
- The **output and exchange variables** can only be activated/deactivated in this module.
- A variable, regardless of its type, can only be activated/deactivated in this module.
- Exchanges between modules are restricted to a bare minimum.

4 Language rule

Unless otherwise necessary or impossible, the programming language must be the "LD" contact language (ladder). The language is particularly specific to machine animation.

Where not, only the following languages are permitted:

— Structured literal text "ST" (Structured Text) in compliance with standard **CEI 61131-3**: it is reserved for calculation functions and functional boxes.

— Programming language "SFC" (Sequential Function Chart (programming using "grafcet")) : Renault only authorises SFC language after consent by the Automated Control manager, for machines as follows:

- simples (1)
- single-station,
- standardised,
- synchronous (single branch graph),
- linked on basis of strong supplier standards.

Renault forbids the use of "pseudo-grafcet" programming which does not use the SFC language supplied by the softshop.

(1) not considered as simple machines are those for which it is difficult to quickly return to the cycle resumption position (in under 5 minutes). These are notably:

— Machines with at least 2 input flows (parts flow, information flow):

examples : a machine for mounting externally worked parts on the main flow assemblies,

a "marrying up" station (for example, machine for fitting the cylinder head to the engine bottom end)

a machine with information flows not related to the physical flow (virtual labelling).

— Machines with "anticipation" :

example : a machine for fitting externally worked parts with forward reading of the part to be fitted and the cycle to be executed, and outsourced part currently being fitted at the workstation.

— Machines which have at least 2 asynchronous operating positions:

example : a robotised unit.

5 Structure rule

In the examples below in article 5, the mnemonics are taken directly from the corresponding heading.

5.1 Writing order

For reasons of homogeneity, Renault would like the order of writing of the programme within a module to be in conformity with the following model:

— the equations relating to the module's necessary conditions are:

  — the general safety equations,

  — the operating mode equations.

— the first movement equations:

  — the safety equations,

  — the operating mode equations,

  — the movement command equations:

    - Forwards, Upwards, Tightening, etc. equations,
    - Backwards, Downwards, Loosening, etc. equations,
    - high speed equations (if they exist),
    - low speed equations (if they exist),

  — the viewing equations, (1)

  — the equations associated with a fault monitoring tool, particularly the activation of default bits (if this function exists) (1).

— the second movement equations:

I
I

— the final movement equations:

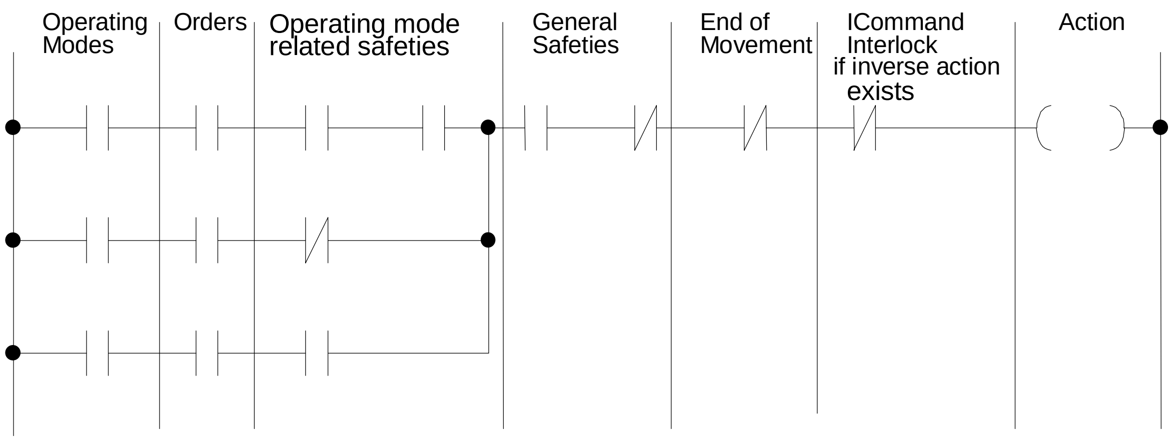
— the viewing equations (2).

— the equations associated with a fault monitoring tool, particularly the activation of default bits (if this function exists) (2).

— the equations relating to exchanges with the other modules.

Note: The position of the equations relating to fault viewing and monitoring (in italics) is, unless otherwise specified in the technical specifications, left to the supplier's initiative. However, in order to facilitate the use of the software, mixing of situations (1) and (2) is not authorised.

5.2 Standard creation of a movement equation


Operating Modes
Orders
Operating mode related safeties
General Safeties
End of Movement
ICommand Interlock if inverse action exists
Action

There are as many parallel branches as start modes. In cases where it is necessary to break the equation down (see paragraph 5.5), the final movement and interlocking information, if they exist, shall be included in the final equation (the one which activates the working variable).

5.3 Combining variables

To make the programme easier to read when the same group of variables is used in several equations of the same model, this group of variables is replaced by a meaningful intermediate variable.

However, there shall be no mixing of the various types (Start mode, Commands, Safety, etc.), see paragraph 5.2.

5.4 Format of equations

The format of the equations shall correspond to the characteristics of the associated viewing system.

Any of the equations shall be viewed on one screen page and be able to be used in its entirety without zooming.

To do this, the number of series variables and parallel branches shall be compatible with the display characteristics of the viewing system.

5.5 Breakdown of equations

When the equations do not respect the format (see paragraph 5.4), it is necessary to break them down. This operation shall follow a logic which does not generate equations.

The breakdown respects the different constituent items of the equation (see paragraph 5.2) and the programmer has to break the equation down as best as possible.

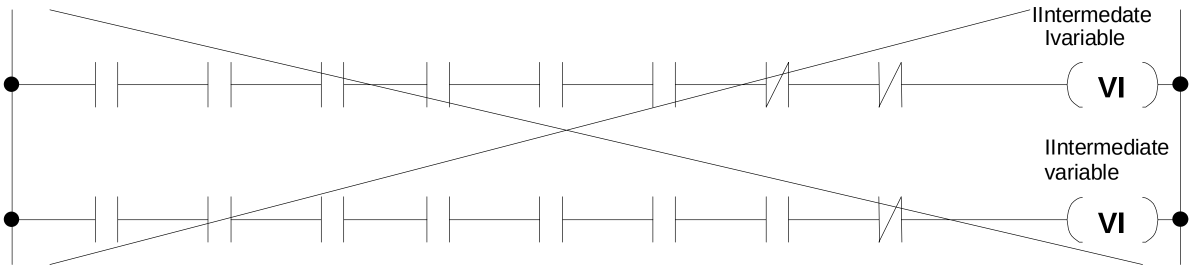
To remain homogeneous, all start modes shall be part of one particular equation if it proves necessary to create a particular equation for a particular start mode.

The equations obtained in such a way are written in the following order:

- general safety equation,
- start mode equations,
- action control equations.

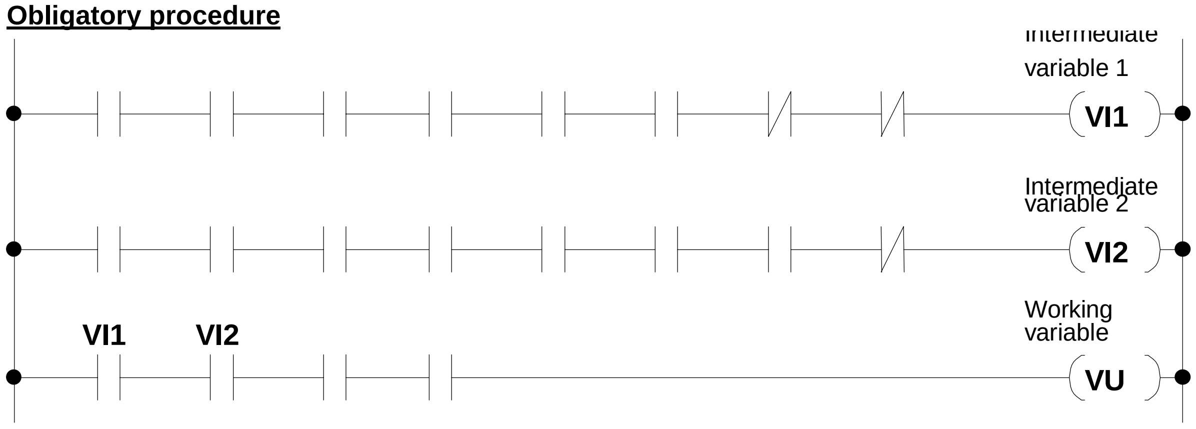
If one of these equations is itself broken down, the resulting equations shall be consecutive. The sum variable shall be placed at the end. Each equation activates its own variable.

Non-authorised procedure



This procedure, which uses a reduced number of variables, generates dynamic viewing diagnosis errors.

Obligatory procedure

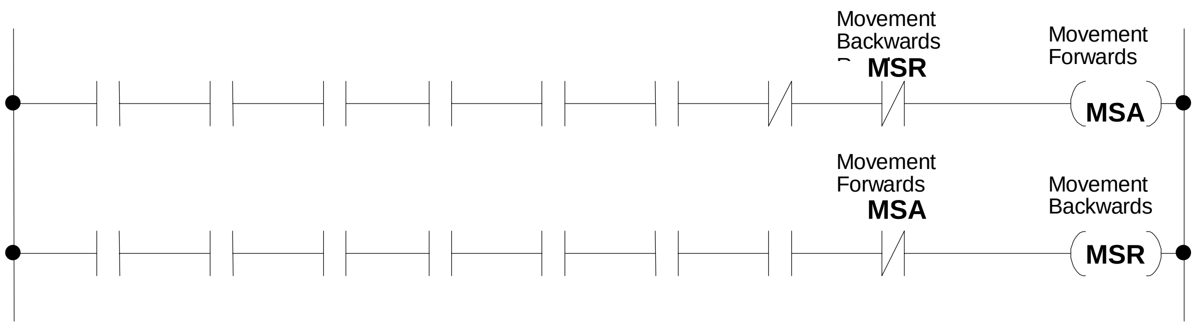


The VU variable is used in the equation of the corresponding movement.

5.6 Interlocking of the controls

The movement controls with several operating directions must be interlocked so that the associated actuators cannot be activated at the same time.

E.g.: 2 movements "Backwards and Forwards".



5.7 Check to ensure the limit switches function correctly

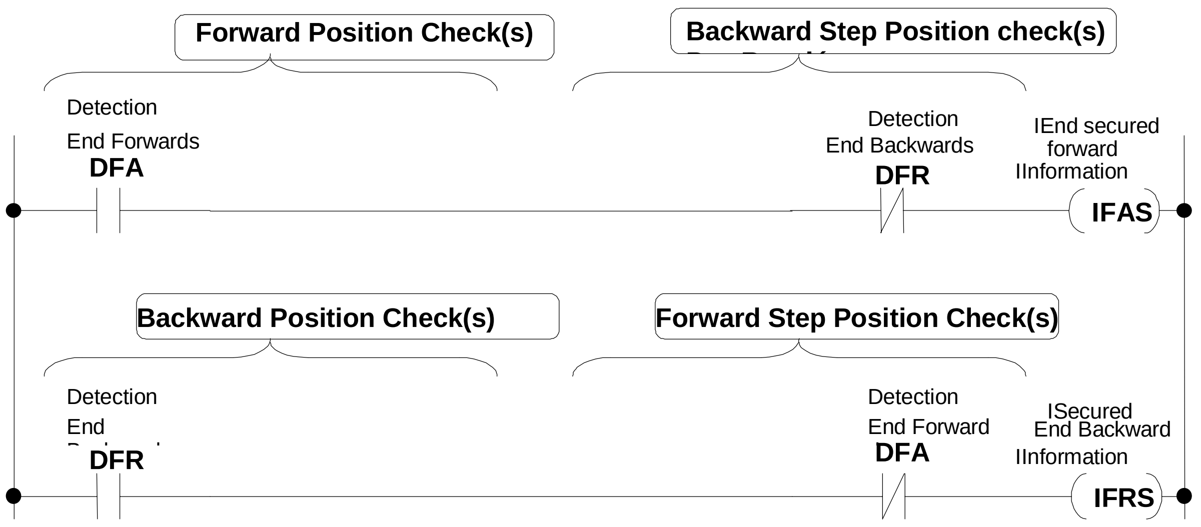
In order to avoid one of the sensors from causing a mechanical fault, it is necessary carry out a check to ensure the limit switches function correctly.

This inspection involves checking, during the most suitable machine cycle phase, the reverse status of the sensor of the machine with which it is to be used.

Checking functioning of the limit switches can be carried out in different ways.

For the movements and for reasons of homogeneity for the operator, Renault prefers the solution which consists in generating, for example, secured end of movement variables (IFAS and IFRS in the example below) which must be used whenever necessary.

Example of secured variables for a 2-position movement: "Backwards and Forwards".


Forward Position Check(s)
Backward Step Position check(s)
Detection
End Forwards
DFA
Detection
End Backwards
DFR
IEnd secured forward IInformation
IFAS
Backward Position Check(s)
Forward Step Position Check(s)
Detection
End
DFR
Detection
End Forward
DFA
ISecured End Backward IInformation
IFRS

NOTE: the secured Forwards limit information (FLI) and Backward limit information (BLI) shall not under any circumstances, be used to disable the corresponding actuators. Use the corresponding end of movement detector(s).

5.8 Activation/deactivation of the variables

The variables can only be activated/deactivated in one single equation.

5.9 Stored variables

The conditions for setting at 1 and 0 are:

- systematically positioned in the same module,
- positioned consecutively,
- exclusive.

For certain particular cases or for process reasons only, if the rule cannot be respected, it may be ignored under the following conditions:

- exclusivity respected,
- specify the position of the additional equation or programme line for each equation or programme line which modifies the status of the memory .

5.10 Intermediate variables

They are positioned upstream and nearest the user equations.

5.11 Independence from the hardware

Any movement control, shall be maintained until the end of the movement independent of the actuator technology.

5.12 Cases specific to Structured Text language

Generally, the structure rules defined for the contact languages apply to the structured text language.

5.12.1 Stored variables

The programme lines which set the logical status to 1 and 0, shall wherever possible, be grouped together and positioned in the same module.

5.12.2 Presentation

To improve readability, it is necessary to respect a certain presentation. In particular, the write entry for each nesting level is assigned to an indent. It is nevertheless necessary to manage correctly the length of the programme lines to avoid moving onto another page both for data entry and viewing.

The example below is not exclusive, if the solution adopted by the supplier offers the same readability.

For example:
(the example is for illustration purposes only).

```
(*Comment*)
IF EMIA OR CFOPEQT OR CFOPEQMN
  THEN
(*Comment*)
IF EQIPPL < (EQXPPL - HMPTEQP)
THEN EQEGL := TRUE; (*Comment*) Where necessary
        EQEPPL := FALSE; (*Comment*) Where necessary
    ELSE
    IF CLIPMH
      THEN
      (*Comment*)
      IF EQIPPL > (EQXPPL + HMPTEQP)
        THEN    EQEGPL := FALSE; (*Comment*) Where necessary
                EQEPPL := TRUE; (*Comment*) Where necessary
        ELSE
        (*Comment*)
        IF EQIPPL <= EQXPPL
          THEN    EQEPPL := FALSE; (*Comment*) Where necessary
        END_IF;
      END_IF;
    END_IF;
  END_IF;
END_IF;

(*Comment*)
EQIPPLMN := EQDPPL AND (EQIPPL >= HMPPEQIP) AND (EQIPPL <= (HMPPEQIP + HMPTEQP));

(*Comment*)
EQIPPLD := EQDPPL AND (EQIPPL >= HMPPEQIP) AND (EQIPPL <= (HMPPEQIP + HMPTEQP));
```

```
(*Comment*)
TON_1 (
IN := EQDPPL AND (EQIPPL >= (HMPPEQTP - HMPTEQP)) AND (EQIPPL <= (HMPPEQTP +
HMPTEQP));
PT := TIME # 5s);
```

Note: the carriage returns in the TON_1 equation are not obligatory. There are however strongly recommended when they improve readability.

This note is valid for all programme writing.

6 Abstraction rule

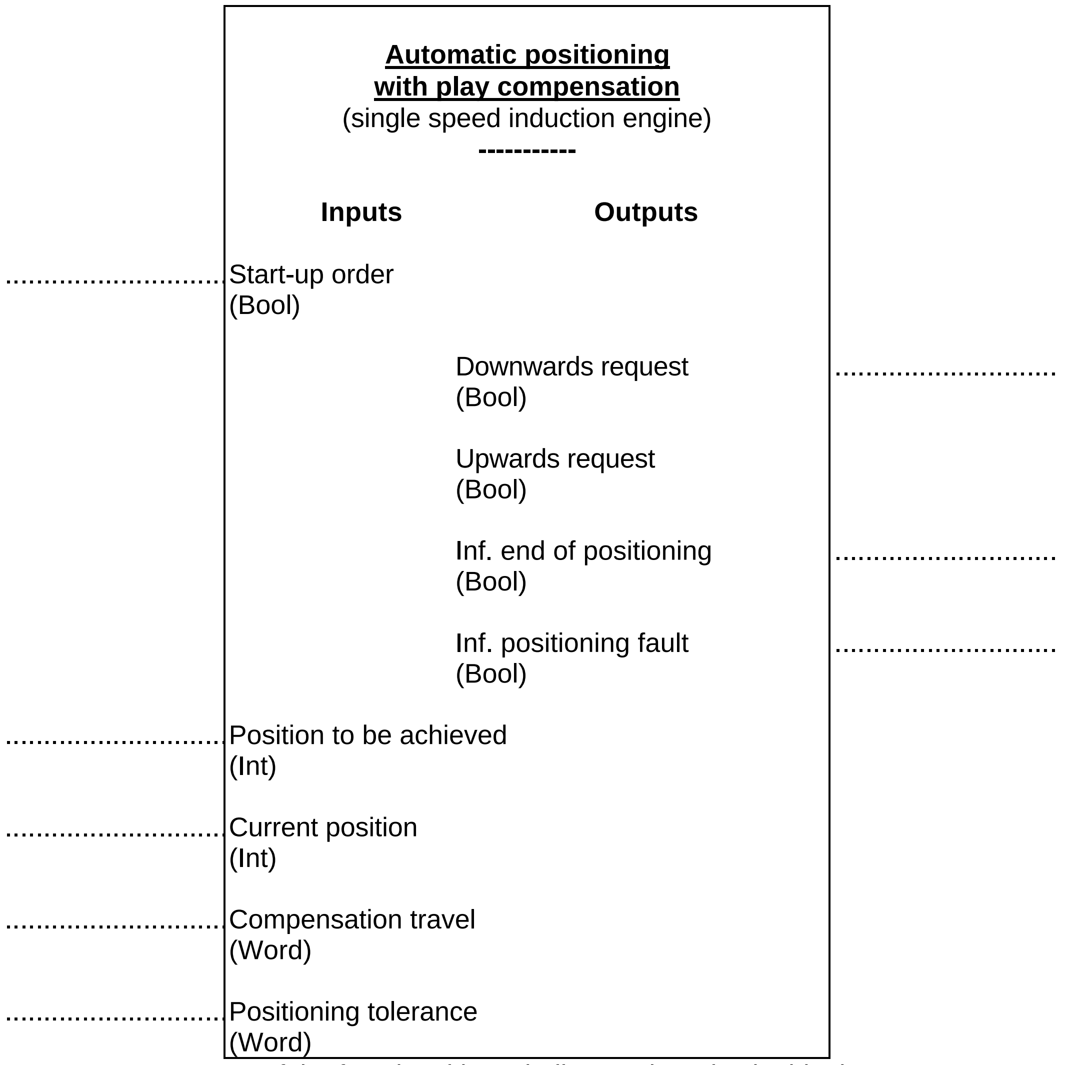
The abstraction consists in masking the components of the programme which are identical or which do not make for better understanding. It also contributes to better readability.

In programming, the abstraction shall apply:

- for all functions which are used more than once,
- for all complex functions.

The corresponding programme components are actually functional boxes.

E.g.: automatic positioning with play compensation for a single speed induction engine.


Automatic positioning
with play compensation
(single speed induction engine)

Inputs
Outputs
Start-up order (Bool)
Downwards request (Bool)
Upwards request (Bool)
Inf. end of positioning (Bool)
Inf. positioning fault (Bool)
Position to be achieved (Int)
Current position (Int)
Compensation travel (Word)
Positioning tolerance (Word)

The programme component of the functional box shall meet the rules in this document.

To be used correctly and understood by the operators, the use of the functional boxes shall be accompanied by:

- an explicit description of its function,
- the details of the input/output interfaces.

NOTE: Renault recommends using its functional boxes. The special functions developed by the supplier have to be accompanied by the necessary documents validated by Renault for it to be used correctly, including the details of the programme component.

7 Variable designation rule

7.1 General

In order to facilitate the design and reading of programmes, it is necessary to have a mnemotechnical variable marking method.

In particular, this system should comply with:

- Rules easy to implement, which give a mnemonic which is as clear as possible,
- Rules for coherence with other documents constituting the machine documentation.

The comment corresponding to the mnemonic must be written in the language of the user country.

The mnemonic must provide information on the links with the machine physically (Workstation number, movement number, direction of movement) and/or the type of component it refers to (detector, button, lamp, etc..).

Example of mnemonic construction rule:

It provides information on:

a) the origin of the variable (module),

b) the type of information (sensor, actuator, intermediate variable, etc.).

c) the nature of the physical phenomenon discovered (Forwards, Backwards, Upwards, Clamping, etc.).

d) the function in question (loading, lift, etc.).

e) the additional necessary information.

Each mnemonic has a corresponding comment.

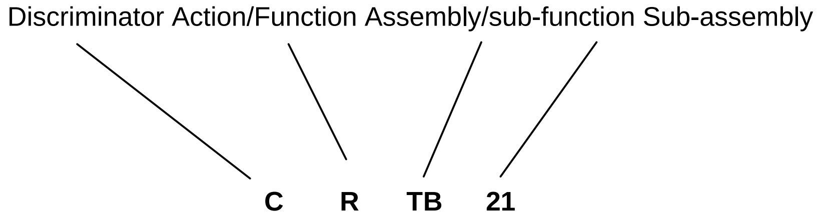
E.g.: the Forwards limit of the loading station (same example as in paragraph 2).

Origin (Module)	Info type	Nature	Function
LS (**L**oading **S**tation)	**D** (**D**etection)	**F** (**F**orwards)	**L** (**L**oader)

Mnemonic ⇒ **LS D F L**

Corresponding comment ⇒ Loading station Detection Forwards Loader

Another example: machining lathe unit 21 table return monitoring.



Mnemonic ⇒ **C R TB 21**

Corresponding comment ⇒ Unit 21 Table Return Monitoring.

7.2 Complex functios

For complex functions or functional boxes, it could prove difficult or even impossible to apply the mnemonics drafting rule. It is therefore recommended to associate the variables with a comment with the most meaning, with each word being separated by a "_" (under score) or a " " (space) to improve readability.

E.g.: the acceleration rate in a control loop.
"transfer_acceleration_rate" or "transfer acceleration rate"

It is better to use lower case letters with no accents in order to facilitate the reading and use of the keyboard during help operations.

8 Organisation rule

The modules are run in the same order as in the presentation in the functional analysis.

The run order of the modules shall not affect the operation of the programme.

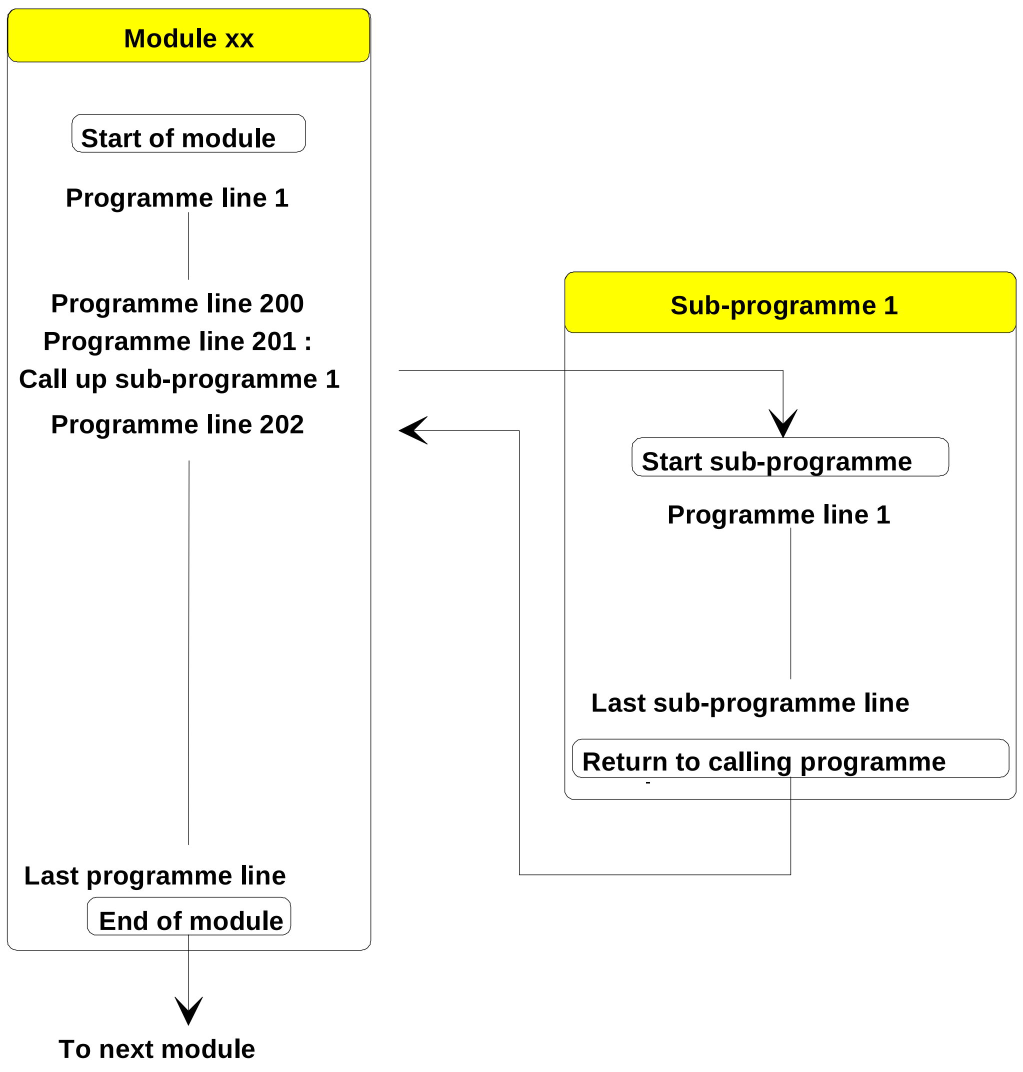
If the output variables are activated or deactivated when using interruption modules, the corresponding image memory shall be updated at the end of the interruption module. The interruption modules are as small and short as possible.

The programme drive lines are run line by line. Electronic approaches which implement the jumps according to the start modes or the installation status are forbidden.

Jumps to another module are forbidden.

A sub-programme must be called up via its input point. Similarly, the feedback to the calling component must be routed via its output point and only the programme line positioned after the calling line.

E.g.:



9 Documentation

9.1 Programme Documentation

9.1.1 General

— Each function is preceded by a meaningful comment (function = equations or programme lines which are necessary to control the movement).

— Each comment is preceded by a line break to improve its readability.

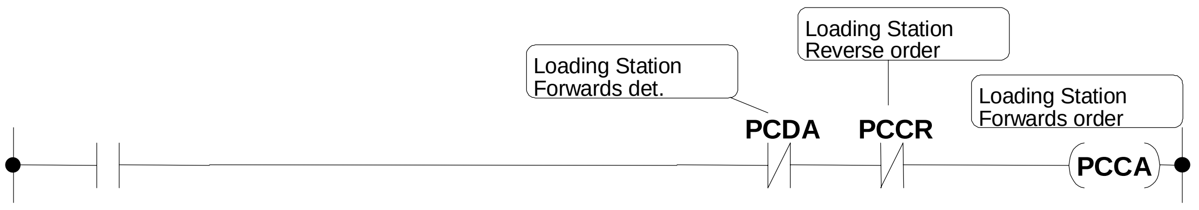
— Each equation or programme line is preceded by a meaningful comment.

— Each variable has a mnemonic which is associated to a corresponding comment. This variable shall be true to the mnemonic.

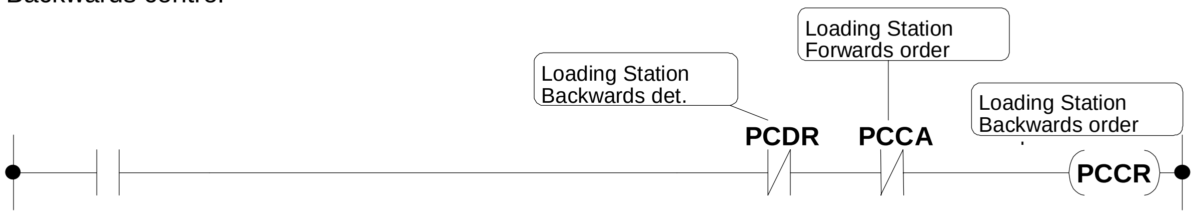
E.g.: the loading control (same example as in paragraph 2).

LOADING STATION

Forwards control


Loading Station Reverse order
Loading Station Forwards det.
Loading Station Forwards order
PCDA
PCCR
PCCA

Backwards control


Loading Station Forwards order
Loading Station Backwards det.
Loading Station Backwards order
PCDR
PCCA
PCCR

The wording corresponding to the mnemonics are only given to explain the example.

<u>Cases specific to the structured text language.</u>

The group of instructions which carry out a specific task are preceded by a comment. Within this group, in order to modify the readability, only meaningful programme lines are preceded by a comment.

9.1.2 Programme

In this we find:

- the application name,
- the author's name,
- the version number,
- the date of last update,
- the comment on the last update.

9.1.3 Functional boxes

The functionality of the functional boxes and mnemonics for the external environment associated variables must be sufficiently explicit. In this respect, the variables may, if necessary, be accompanied by a text in clear rather than a mnemonic (see paragraph 7.2.).

At the head of each functional box, we find :

- the functional box name,
- a description of the function of the functional box,
- the functional box version number (to be able to tie up with the documentation),
- the declaration of the variables handles in the functional box, in type order (inputs, outputs, internal),

9.2 Functional analysis

Regardless of the documents supplied as part of the consultation folder, the supplier is obliged to carry out a functional analysis going as far as the description of the cycle resumption cases to be provided. Other than the literal explanations, the supplier draws up, if necessary, explicit diagrams accompanied by functional diagrams, where applicable.

9.3 Organic analysis

The organic analysis is carried out on the basis of the functional analysis. It clearly demonstrates:

- the top level architecture of the system (the different constituent elements of the installation system) with its interconnections,
- the general architecture of the programme,
- the architecture of each of the programme modules.

9.4 Functional boxes or complex functions

They shall be accompanied by a detailed booklet which, as well as improving understanding, allows Renault to make modifications where necessary.

10 List of cited documents

NOTE: For undated documents, the last version in force applies.

CEI 61131-3 : Automates programmables. Partie 3 : Langages de programmation.

11 Glossary

Instantiation : multiple use of a programme entity (the model) without recopy of the code, with creation by the softshop of a data zone (memory context) relating to each use (instance).