1 Contexte des modules D.L.S

L'ensemble des programmes permettant d'agir sur l'environnement ou de réagir sur ce dernier sont appelés « Module D.L.S » ou encore « Plugin D.L.S ».

Ces programmes peuvent être édités via le menu « Admin->Plugin D.L.S » dans l'interface cliente.

1.1 Environnements des modules D.L.S

L'ensemble des différents modules D.L.S partagent la même mémoire d'information. Tous les bits internes sont accessibles en lecture et écriture par tous les modules D.L.S.

Cette mémoire d'information est composée d'un ensemble de « bits internes », ayant chacun une signification propre selon son type. Il existe des bits d'entrées, d'autres de sorties, ou encore des bits de temporisations.

1.2 Organisation interne d'un plugin

Chacun des plugins D.L.S est découpé en deux zones distinctes :

- la zone de description des ALIAS
- la zone de description du fonctionnement

1.2.1 Zone de description des ALIAS

L'ensemble des définitions d'alias de cette zone n'a aucun impact sur le contenu de la zone fonctionnelle, si ce n'est l'amélioration de la lisibilité, et la facilité du renommage des bits internes en cas de modification (un seul alias à modifier plutôt que l'ensemble des occurrences unitaires de la zone fonctionnelle).

Il s'agit d'une zone donc la syntaxe est la suivante :

- *ALIAS* ↔ *ID*;

Elle commence directement par une chaine de caractères représentant l'alias, puis une double flèche (un inférieur « < », un tiret « - », un supérieur « > »), un IDentificateur, et enfin un point virgule terminal.

Des options peuvent être affectées à cet alias, en utilisant la syntaxe suivante :

- *ALIAS* ↔ *ID(OPTIONS)*;

Chaque occurrence d' « *ALIAS* » de la zone fonctionnelle sera remplacé par « *ID(OPTIONS)* » .

Exemple de syntaxe complète :

- *VOLET_OUVERT ↔ _E01; /* VOLET_OUVERT sera remplacé par _E01 dans la suite */*

1.2.2 Zone de description fonctionnelle

C'est dans cette zone où toute l'intelligence du plugin va résider. C'est elle qui définie l'ensemble des comportements à avoir en fonction de l'environnement du module.

Cette zone est composée d'une suite de ligne (appelée plus tard « ligne D.L.S » dont le contenu d'une ligne suit la syntaxe suivante :

- - *EXPRESSION → LISTE_ACTIONS*;

Elle commence par un tiret « - », suivi d'une « *EXPRESSION* », suivi par une flèche (tiret « - » puis supérieur « > »), une liste d'actions séparées par des virgules et enfin, un point virgule terminal.

2 Grammaire et sémantique

2.1 Un IDentificateur

Chacune des zones mémoires interne peut être adressée sous la forme d'une chaine de caractères (un « **ID** », ou IDentificateur) commençant par le type de mémoire à adresser, suivi du numéro du bit en question.

Par exemple, pour récupérer la valeur du monostable numéro 01, il faudra utiliser la syntaxe DLS suivante : « _M23 ».

Pour récupérer la valeur de l'entrée TOR numéro 02, nous utiliserons « _E02 ».

2.2 Une ligne D.L.S

Une ligne D.L.S est de la forme suivante :

- *- EXPRESSION → LISTE_ACTIONS;*

« EXPRESSION » est un IDentificateur, ou un ensemble d'IDentificateurs liés entre eux par des opérateurs de base.

« LISTE_ACTIONS » est une liste d'un ou plusieurs d'IDentificateurs, séparés par des virgules.

Chacun des IDentificateurs, s'il est suivi par des parenthèses peut s'adjoindre une liste d'options, elles aussi séparées par des virgules.

Exemple de syntaxe complète :

- *- _E01 . (/_M01 + _TR01) → _I01(mode=0,couleur=rouge); /* Un exemple */*

Nous détaillons la grammaire dans chacun des paragraphes ci dessous.

2.3 Les commentaires

Dans tout le code D.L.S, il est possible d'intégrer des commentaires. Utiles pour augmenter le niveau de lisibilité du code, ils permettent aussi d'accélérer la compréhension du mode de fonctionnement.

Un commentaire commence par la chaine « /* » et se finit par la chaine « */ »

Exemple de syntaxe :

- *- a . b → c; /* Si a et b sont vrais alors nous positionnons c */*

2.4 Description des opérateurs de base

2.4.1 Le ET « . »

Dans une « *EXPRESSION* », le ET « . » permet d'opérer la fonction logique ET entre deux sous-expressions.

« a . b est vrai » si et seulement si « a est vrai » et « b est vrai ».

Exemple de syntaxe :

- *- a . b → c; /* Si a et b sont vrais alors nous positionnons c */*

Le ET n'a pas de sens dans une « *LISTE_ACTIONS* ».

2.4.2 Le OU « + »

Dans une « *EXPRESSION* », le OU « + » permet d'opérer la fonction logique OU entre deux sous-expressions.

« a + b est vrai » si « a est vrai » ou « b est vrai ».

Exemple de syntaxe :

- *- a + b → c; /* Si a ou b sont vrais alors nous positionnons c */*

Le OU n'a pas de sens dans une « *LISTE_ACTIONS* ».

2.4.3 Le complément « / »

Dans une « *EXPRESSION* » ou une « *LISTE_ACTIONS* », le complément « / » permet d'opérer la fonction logique NON sur l'expression suivante.

« /a est vrai » si « a est faux ».

Exemple de syntaxe :

- *- /a → c; /* Si a est faux alors nous positionnons c à 1 */*
- *- a . b → /c; /* Si a et b sont vrais alors nous positionnons c à 0 */*

2.4.4 Précédences et parenthèses

Les priorités d'opérations sont les suivantes, dans l'ordre décroissant de priorité :

- Le NON
- Le ET
- Le OU

Ce système de priorité peut être modifié en utilisant les parenthèses ouvrantes et fermantes.

Exemple de syntaxe:

- *- a + b . c → d; /* Si a est vrai, ou b et c sont vrais, alors nous positionnons d à 1 */*
- *- (a+b) . c → d; /* Si a ou b est vrai, et c est vrai alors nous positionnons d à 1 */*
- *- a . /(b+c) → /c; /* Si a est vrai, et que l'on a ni b ni c, alors nous positionnons c à 0 */*

2.5 Descriptions des bits internes.

2.5.1 Les monostables _M

Les monostables sont des bits internes accessibles par le tableau « _M ».

Par exemple, « _M23 » pour le 23ième bit internes monostable.

Chacun des monostables à pour valeur soit « 0 », soit « 1 » et peut être positionné dans une « *EXPRESSION* » ou une « *LISTE_ACTIONS* ».

Quand un monostable est utilisé dans la « *LISTE_ACTIONS* » d'une ligne D.L.S, sa valeur est positionnée à la valeur de l' « *EXPRESSION* ».

Exemple de syntaxe :

- *- a → _M23; /* Si a est vrai, _M23 = 1. Si a est faux, _M23 = 0 */*
- *- /_M42 → b; /* Si _M42 = 0, alors nous positionnons b à 0 */*

2.5.2 Les bistables _B

Les bistables sont des bits internes accessibles par le tableau « _B ».

Par exemple, « _B23 » pour le 23ième bit interne bistable

Chacun des bistables à pour valeur soit 0, soit 1 et peut être positionné dans une « *EXPRESSION* » ou une « *LISTE_ACTIONS* ».

Quand un bistable est utilisé dans la « *LISTE_ACTIONS* » d'une ligne D.L.S, sa valeur est égale à « 1 » si l'EXPRESSION est vraie.

Exemple de syntaxe :

- *- a → _B23; /* Si a est vrai, _B23 = 1. Sinon, _B23 n'est pas modifié */*
- *- b → /_B42; /* b est vrai, _B42 est positionné à 0. Sinon, _B42 n'est pas modifié */*

2.5.3 Les entrée TOR _E

Les entrées Tout Ou Rien (TOR) sont des bits internes accessibles par le tableau « _E ».

Par exemple, « _E23 » pour le 23ième bit interne d'entrée TOR

Chacune des entrées TOR à pour valeur soit 0, soit 1 et ne peut être positionné que dans une « *EXPRESSION* ».

Exemple de syntaxe :

- *- _E23 → _B23; /* Si _E23 = 1 alors _B23 = 1 */*
- *- /_E24 → /_B23; /* Si _E24 = 0 alors _B23 = 0 */*

2.5.4 Les sorties TOR _A

Les sorties Tout Ou Rien (TOR) sont des bits internes accessibles par le tableau « _A », comme actionneurs.

Par exemple, « _A23 » pour le 23ième bit interne de sortie TOR.

Chacune des sorties TOR à pour valeur soit 0, soit 1 et ne peut être positionné que dans une « *LISTE_ACTIONS* ».

Exemple de syntaxe :

- *- _E23 → _A23; /* Si _E23 = 1 alors _A23 = 1 */*
- *- /_E24 → /_A23; /* Si _E24 = 0 alors _A23 = 0 */*

2.5.5 Les Entrée ANA _EA

2.5.6 Les Sorties ANA _AA

2.5.7 Les compteurs horaires _CH

Les compteurs horaires sont des bits internes accessibles par le tableau « _CH ».

Par exemple, « _CH23 » pour le 23ième compteurs horaire.

Chacun des compteurs horaires à une valeur entière représentant le nombre de minutes ou le compteur est activé, depuis les origines des temps. Un compteur horaire peut être positionné dans une « *EXPRESSION* » ou une « *LISTE_ACTIONS* ».

Un compteur horaire autorise les options suivantes :

- reset=1 pour remettre le compteur horaire à zero.

Exemple de syntaxe :

- *- _CH01 > 25.0 → _A23; /* Si on dépasse 25 minutes alors _A23 = 1 */*
- *- _E01 → _CH01; /* Si _E01 = 1 alors le compteur continue à décompter. Sinon sa valeur reste inchangée */*
- *- _E02 → _CH01(reset=1); /* Si _E02 = 1 alors le compteur est remis à zero */*

2.5.8 Les compteurs d'impulsions _CI

2.5.9 Les temporisations retard _TR

Les temporisations Retard sont des bits internes accessibles par le tableau « _TR ».

Par exemple, « _TR23 » pour la 23ième temporisation

Chacune des temporisations retard à pour valeur soit 0, soit 1 et peut être positionné dans une « *LISTE_ACTIONS* » ou un « *EXPRESSION* ».

Lorsque la temporisation est dans une « *EXPRESSION* », et si cette « *EXPRESSION* » est vraie, alors la temporisation commence à décompter.
Si l'« *EXPRESSION* » devient fausse avant que le temporisation atteigne sa consigne, celle ci se remet à 0.
Si l'« *EXPRESSION* » reste vraie pendant après que la temporisation ait atteint sa consigne, celle ci devient vraie à son tour

Comportement spécial du complément :

La représentation « /_TR » n'est pas exactement identique au complément de la valeur de la temporisation. Cette notation à les valeurs suivante :

- 0 si la temporisation n'est pas en cours de décompte
- 1 si la temporisation est en cours de décompte

Exemple de syntaxe :

- *- _E01 → _TR23; /* Si _E01 = 1 alors la temporisation commence le décompte */*
- *- TR23 → _A01; /* Au bout du décompte, A01 est positionné à 1 */*

2.5.10 Les Messages _MSG

2.5.11 Les icones _I

2.5.12 Bits spéciaux.

- _B0 : change d'état tous les tours programme
- _B1 : toujours à « 0 »
- _B2 : toujours à « 1 »
- _B4 : change d'etat toutes les secondes
- _B5 : change d'état toutes les demi-secondes
- _B6 : change d'état tous les 3 dizièmes de secondes
- _I1: icône toujours en mode 0, couleur rouge