



Systeme de Simulation IOCards

Par Manuel Vélez
www.opencockpits.com

- Août 2004 -

Traduction : Philippe CHATAGNIER (Chamois)

0. SOMMAIRE

1. INTRODUCTION

- 1.1. Protocole IOCP
- 1.2. Structure du Système de simulation IOC
- 1.3. Système basé sur des événements
- 1.4. Serveur IOCP

2. SYSTÈME DE SIMULATION DES IOCARDS - SIOC -

- 2.1. Variables du système
- 2.2. Modules du SIOC. Types d'unions (links) aux variables du système
 - 2.2.1. Module FSUIPC
 - 2.2.2. Module client IOCP
 - 2.2.3. Module connexion aux IOCards
 - 2.2.3.1. Connexion SW IOCards
 - 2.2.3.2. Connexion Sortie IOCards
 - 2.2.3.3. Connexion Digits IOCards
 - 2.2.3.4. Connexion Encodeurs IOCards
- 2.3. Programme CONFIG-SIOC pour la définition de variables et scripts
 - 2.3.1. Options de fichiers
 - 2.3.2. Options d'édition
 - 2.3.3. Autres options
- 2.4. Programmation de scripts
 - 2.4.1. Variables locales
 - 2.4.2. Commande ASSIGNATION
 - 2.4.3. Commande FONCTION
 - 2.4.4. Commande CONDITION IF (Si la condition X se produit...)
 - 2.4.5. Commande CONDITION ELSE (Dans un autre cas...)
 - 2.4.6. La boucle sans fin
 - 2.4.7. Exemples de programmation
- 2.5. Programme SIOC. Serveur IOC et noyau de l'application.

1. INTRODUCTION

1.1. Protocole IOCP

Le protocole **IOCP** (IOCards Protocol) naît d'une nécessité d'intercommunication des plaques IOCards avec différents programmes dans différentes machines, utilisant pour cela le protocole de réseau TCP/IP, permettant l'interconnexion tant autour de réseaux locaux comme à travers internet.

Ce protocole présente des avancées remarquables face à d'autres protocoles utilisés pour l'interconnexion de modules ou de plaques électroniques au simulateur de vol Flight Simulator :

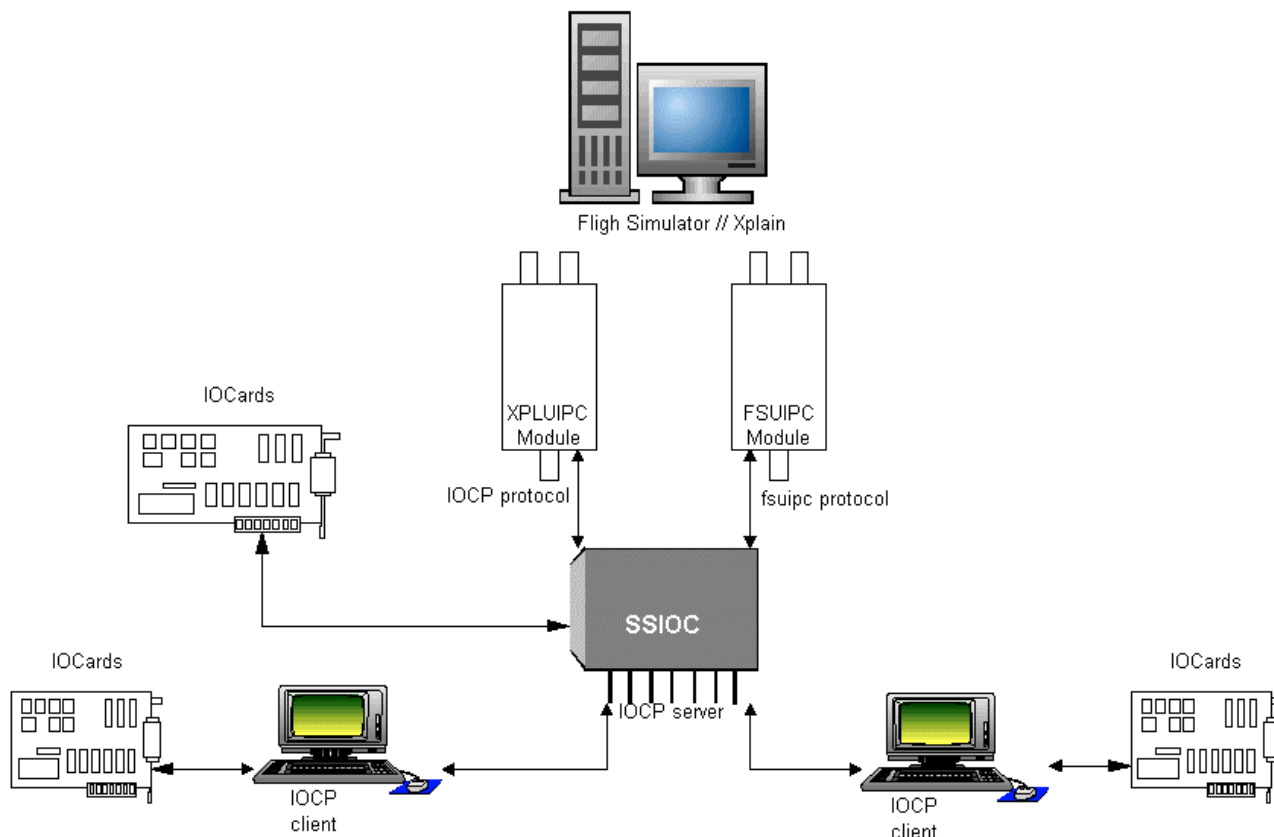
- C'est un protocole énormément plus rapide.
- Il consomme moins de ressources.
- Il est basé sur des événements.
- Il n'a pas besoin d'applications du type WideFS pour être interconnecté avec d'autres ordinateurs.
- Il est multiplateforme, pouvant être relié à FSimulator ou à X-plane.
- Il est totalement Freeware pour des usages non commerciaux.

1.2. Structure du Système de simulation IOC

Le Système de simulation IOC a été conçu pour couvrir des questions que le logiciel des IOCards ne couvre pas, étant donnés les besoins possibles qui surviennent en allant au-delà du fonctionnement électronique des IOCards.

Par ce système, il est possible de réaliser des simulations de tout type, qui sont définies par l'utilisateur, affectant pour cela non seulement la façon de fonctionner de l'électronique, mais évidemment aussi la façon d'opérer du simulateur de vol ou des modules associés à ce dernier.

Le Système utilise le protocole **IOCP** comme moyen de transport de l'information, et comme moyen d'accéder aussi aux variables internes du système.



Le noyau du **SS-IOC** est basé sur :

- Un **serveur IOCP** capable de donner une entrée et une sortie à toute variable définie dans ce dernier.
- Un **module client IOCP** pour accéder à d'autres systèmes (par exemple FSimulator en utilisant IOCPserver.dll, ou X-plane au moyen de XLUIPC).
- Un **module client FSUIPC** pour accéder à d'autres systèmes (FSimulator, Project Magenta, etc.).
- Un **module d'interconnexion au système IOCards**, pour pouvoir contrôler les plaques directement depuis le système.
- Un programmeur **de scripts** visuel qui permet un vaste champ opérationnel entre tous ces modules, avec des fonctions spéciales, formules mathématiques, conditionnelles, temporisateurs, etc..

1.3. Système basé sur des événements

Tout le système est basé sur **des événements**.

Un événement peut être défini comme un ensemble d'opérations qui s'effectuent quand il se passe quelque chose.

Dans notre système, les événements peuvent être lancés pour diverses raisons :

- 1) En cas de changement de l'état ou de la valeur d'une variable interne.
- 2) En étant lancé au moyen **d'un temporisateur**. C'est-à-dire au moyen d'un chronomètre qui, après un certain laps de temps, lance l'événement.

3) En cas de changement d'état dans l'électronique reçue.

Pour nous, tout cela se réduit toujours à la même chose puisque tout aboutit au paragraphe 1), un changement dans l'état ou la valeur d'une variable interne.

1.4. Serveur IOCP

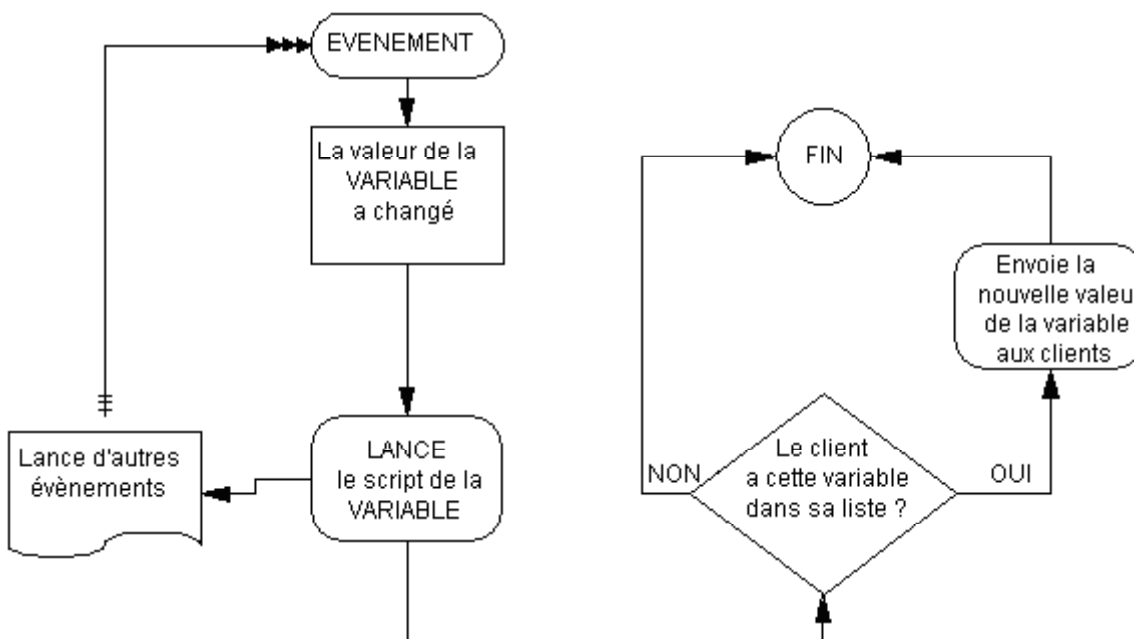
Tout le système **SIOC** tourne autour d'un serveur IOCP. Depuis le logiciel, il est possible de définir jusqu'à 10.000 variables IOCP, auxquelles peuvent être associés un script qui sera exécuté lors de l'activation d'un événement de certaines d'entre elles.

Ces variables qui sont définies par l'utilisateur, sont accessibles soit de manière interne, soit depuis toute application cliente reliée au serveur, ce client pouvant aussi bien provoquer un événement en changeant la valeur de l'une des variables du serveur, que recevoir les valeurs de toute variable dont l'état a changé.

Les événements peuvent être lancés de manière interne et récursive en exécutant un script, parce qu'au moyen **d'une commande d'assignation** on peut altérer la valeur de l'une de ces variables internes, laquelle exécutera à son tour le script associé à son événement.

La première chose que les clients reliés au serveur IOCP font est d'inscrire dans le serveur ces variables dont on veut avoir l'information quand un événement de modification de ces dernières se produira.

Le diagramme de fonctionnement est le suivant :



Par conséquent, s'il n'existe pas de changements dans les variables, il n'existe pas d'événement ni de transmission d'information (c'est l'une des raisons qui entraîne une augmentation de vitesse du protocole IOCP en ce qui concerne les FSUIPC, d'où un besoin de ressources moindre).

Pour qu'il y ait un événement, il doit exister un changement dans la variable associée.

2. SYSTÈME DE SIMULATION DES IOCARDS - SIOC -

2.1. Variables du système

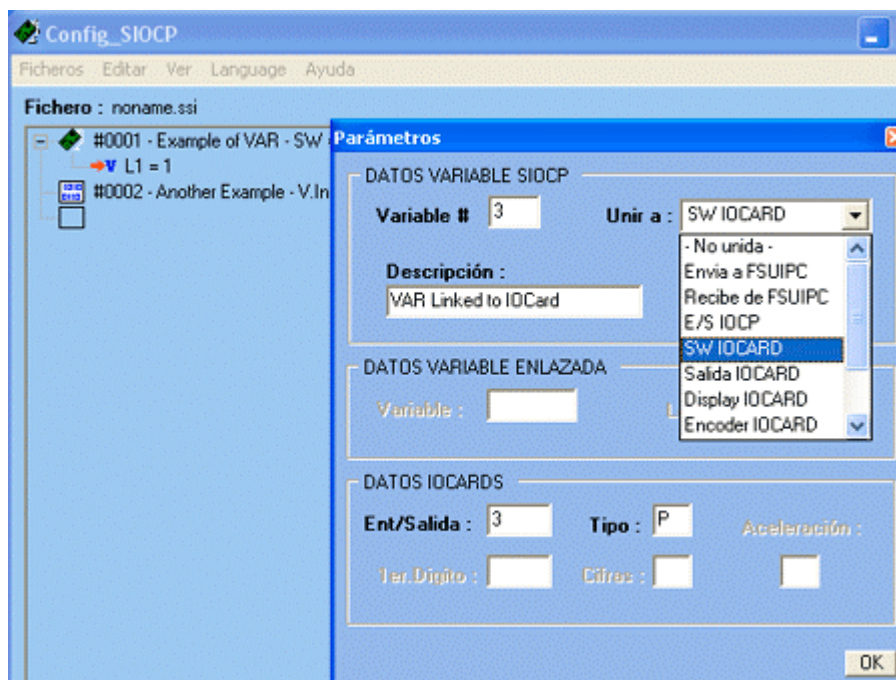
La programmation du système **SIOC** est basée sur la définition de ses variables et de ses scripts associés (événement).

Chacune des variables se distingue par son numéro qui peut aller de 0 jusqu'à 9999. De telle sorte que tout client relié au serveur **SIOC**, qui requiert une information de la variable #0134, fait référence à la variable que nous avons définie avec ce numéro. C'est pourquoi toute modification de la valeur de cette dernière entraînera la notification au client de ce changement. En outre, un ordre de modification de la variable par le client entraînera le lancement de l'événement correspondant et l'exécution de son script associé.

Les variables peuvent comporter, outre leur numéro d'identification, une description de celles-ci et une valeur initiale (facultatif). Cette valeur sera la première inscrite dans la variable au lancement du programme.

Pour pouvoir accéder au reste des modules que gère le **SIOC**, les variables ont une caractéristique spéciale : elles peuvent être **LIEES** ou **UNIES** à ces modules de telle sorte que la modification d'une variable entraînera une interaction avec le module auquel elle est unie, ou au contraire, une action de l'un des modules entraînera la modification et par conséquent le lancement de l'événement de la variable à laquelle il est associé.

Par conséquent la façon de contrôler *les modules du SIOC* se fait au moyen des variables que nous unissons au module.



Au moment que nous définissons un certain type d'union, le programme sollicite l'introduction des données du module auquel nous voulons nous lier. Dans cet exemple, nous pouvons observer qu'en unissant la variable au module de switches des IOCards, il est automatiquement demandé d'introduire le numéro d'entrée associée au switch et le type de ce dernier.

Les variables stockent toujours une valeur entière dans le rang -2147483648 à 2147483647, donc pour stocker des valeurs décimales nous devons toujours avoir une conversion de celles-ci.

Par exemple, si nous devons garder une valeur avec 4 décimales, par exemple 1,0345, alors nous stockerons toujours la valeur multipliée par 10.000, de tel sorte que serait stockée 10345 et, le cas échéant, en cas de besoin de la valeur réelle, nous diviserons toujours par 10.000 les opérations correspondantes. Ceci est dû aux spécificités propres du protocole IOCP.

2.2. Modules du SIOC. Types d'unions (links) aux variables du système

Il existe 3 modules de base auxquels nous pouvons unir nos variables : le module FSUIPC, le module IOCP et le module d'accès aux IOCards.

En outre, évidemment, nous pouvons ne pas unir notre variable à un module; pour cela nous choisirons cette possibilité dans le menu déroulant correspondant. C'est pourquoi cette variable fonctionnera avec le serveur sans que la modification de cette dernière n'affecte aucun module.

La formule adoptée pour l'interaction avec les modules est toujours la suivante : les variables prennent la valeur du résultat d'un événement lancé par un module (celui auquel on l'a uni) et à son tour on lance le script correspondant associé, ou à l'inverse, en modifiant la valeur d'une variable on lance l'événement et, au même moment, on envoie au module l'information du module qui dans chaque cas lui affectera celui-ci d'une manière ou d'une autre comme nous le verrons ultérieurement.

Par conséquent, nous devons différencier 3 types d'unions : celles qui sont de type **envoi ou sortie**, celles qui sont de type **réception ou d'entrée**, et celles qui sont de type bidirectionnel ou **d'entrée et de sortie**.

2.2.1. Module FSUIPC

Par ce module nous nous connectons directement à Flight Simulator (le module serveur FSUIPC doit fonctionner dans le simulateur), ou à l'utilitaire WideFS qui à son tour se chargera d'être relié avec le simulateur, auquel peuvent en outre être relié d'autres applications comme les modules de *Project Magenta* ou d'autres.

Pour ce module, les variables du SIOC pourront être unies de deux manières : en mode **ENVOI à FSUIPC**, ou en mode **RÉCEPTION de FSUIPC**.

Dans le mode d'envoi à FSUIPC, il est nécessaire d'indiquer l'offset de destination (avec un \$ devant, par exemple \$0B00), la longueur de la variable FSUIPC de destination (voir les tableaux du SDK des FSUIPC téléchargeables sur le site web de Peter Downson), et facultativement une valeur initiale qui sera envoyée en lançant notre application.

De cette manière, une fois exécutée l'événement avec son script, la valeur de la variable sera envoyée à l'offset correspondant.

Dans le mode de réception depuis FSUIPC, de la même manière, il est nécessaire d'indiquer l'offset d'origine et la longueur de ce dernier.

Le système effectue une lecture à intervalle régulier (cet intervalle est défini dans le fichier ini), et au cas où la valeur de l'offset diffère de la dernière valeur reprise par le système, on envoie un événement à la variable associée, prenant alors la valeur de la variable de l'offset FSUIPC qui extrait alors le script correspondant.

Il serait intéressant, l'information provenant des offsets étant souvent codifiée, d'introduire dans le script de la variable associée les commandes correspondantes qui décodent la valeur réelle qu'indique l'offset, transformant cette valeur en une autre variable pour son processus.

2.2.2. Module client IOCP

Nous ne devons pas confondre le module **client IOCP**, qui inclut **le SIOC**, avec le **serveur** propre **IOCP** du SIOC.

Nous parlons réellement de variables du **serveur d'IOCP du SIOC** comme variables propres **du SIOC**, mais quand nous parlons **du module client IOCP** nous parlons d'autres variables différentes des nôtres, qui sont dans un autre serveur et auxquelles nous accédons depuis notre module client.

Le SIOC peut par conséquent être relié à tout **serveur IOCP**, en indiquant seulement dans le fichier de configuration INI la direction IP et port TCP/IP où se trouve ce dernier. Ce serveur peut être par exemple le module **XPLUIPC** pour **X-plane**, permettant au **SIOC** d'accéder aux variables de ce simulateur, le module **IOCPServer.dll** pour **FlightSimulator**, ou tout autre **serveur IOCP** qui se trouve en réseau avec le nôtre (ou par internet), y compris un autre **système SIOC**.

L'union de ce module avec les variables est du type **bidirectionnel**, c'est-à-dire qu'une modification dans notre variable entraînera un envoi par le client au serveur de cette modification, ou une notification de modification de variable par le serveur au client entraînera de la même manière l'activation de l'événement de la variable associée au module.

Quand nous choisissons ce type d'union au moment de définir notre variable, nous devons indiquer le numéro d'identification de la variable de destination à laquelle nous voulons nous unir, ainsi que facultativement une valeur initiale à envoyer à la variable en lançant l'application.

2.2.3. Module connexion aux IOCards

Ce module permet que **le SIOC** soit relié de manière **directe** à un système **IOCards** relié à l'ordinateur.

Il est important de tenir compte que **ce n'est pas la seule méthode** pour relier à un système **IOCard**, parce que nous avons aussi la possibilité que le logiciel des **IOCards** par son propre **client IOCP** se connecte au **serveur IOCP du SIOC**, et par la configuration **des IOCards** et la programmation des scripts correspondants s'effectue l'interaction que souhaite l'utilisateur lui-même.

Selon le type de connexion que nous définissons avec les **IOCards**, la variable inter-agira de l'une ou l'autre manière.

De la même manière, nous avons des connexions avec **les IOCards d'envoi et de réception** suivant le type de circuit que nous relierons.

2.2.3.1. Connexion SW IOCards

Dans ce mode de connexion, la variable sera directement reliée aux entrées **de la plaque IOCard Master**. En définissant la variable, il est nécessaire d'introduire **le numéro de l'entrée** correspondant à celle à laquelle on fait référence et **le type** d'entrée (ces derniers ont été définis dans **le logiciel des IOCards**).

Le SIOC étant **une connexion de réception**, lancera alors l'événement de la variable correspondante quand il existera une modification de l'état de l'entrée en question, en mettant cette variable avec **la valeur 0** quand l'entrée se trouvera désactivée et avec **la valeur 1** si l'entrée est activée. Quand on lance l'événement et son **script** correspondant, l'utilisateur peut alors définir les commandes correspondantes pour les actions qu'il estime nécessaires.

2.2.3.2. Connexion Sortie IOCards

Tout comme dans l'option précédente, la variable est reliée directement aux sorties de la plaque **IOCard Master**. Comme paramètre on doit indiquer **le numéro de la sortie** correspondant à celle à laquelle on fait référence.

Dans ce cas la connexion est **de type envoi**, et le script sera lancé **juste avant** d'envoyer la commande d'activation/désactivation de la sortie correspondante.

Pour que le module comprenne qu'il doit désactiver la sortie, la valeur de la variable associée **doit être 0**, pour que le module active la sortie, la variable devra prendre toute valeur **différente de 0**.

2.2.3.3. Connexion Digits IOCards

Cette connexion effectuera le contrôle **des plaques IOCard Display**, au moyen de la variable associée. Il s'agit **d'une connexion d'envoi** où, outre le numéro du **premier digit** à utiliser, il sera nécessaire d'introduire **les chiffres** qui composent le nombre à afficher, ainsi que facultativement un paramètre **type**, pour définir des manières alternatives de fonctionnement.

La variable associée devra stocker le numéro à présenter, en exécutant d'abord le script associé, préalablement à l'envoi de l'ordre de présentation dans le display.

2.2.3.4. Connexion Encodeurs IOCards

Il s'agit **d'une connexion de réception**, dans laquelle la variable associée au moment où l'on enregistre un accroissement ou une diminution dans l'encodeur rotatif lance l'événement correspondant, de telle sorte que **la valeur de la variable** devient la quantité **d'accroissement produit** dans la rotation (elle dépend directement de l'accélération qu'on lui a déterminée). Sa valeur revenant **automatiquement à 0**.

La variable sera par conséquent **modifiée deux fois**, mais le script associé sera seulement lancé quand la valeur **de la variable ne sera pas 0**.

Pour cette configuration, il est nécessaire d'indiquer **l'entrée initiale** de la plaque **IOCard Master** à utiliser, **l'accélération** correspondante et **le type** d'encodeur à utiliser (mêmes paramètres que ceux utilisés dans le programme **des IOCards**).

2.3. Programme CONFIG-SIOC pour la définition des variables et des scripts.

Par ce programme, l'utilisateur pourra définir le mode de fonctionnement **du SIOC**, en introduisant les variables dont il aura besoin, les liens aux différents modules, ainsi que les paramètres à configurer pour chacune d'elles.

Nous pouvons en outre introduire les commandes associées à chaque variable, qui seront exécutées lors de l'activation de son événement correspondant.

À cet effet, le programme présente un écran en mode **arborescence avec différents niveaux**, où le premier niveau correspond toujours à la définition des variables, et ensuite les commandes dans l'ordre où elles seront exécutées, étant précisé qu'il est possible de créer différents niveaux dans les commandes de condition correspondantes. Ces dernières seront seulement exécutées si la condition est remplie.



L'édition des différentes variables et commandes pourra être effectuée à l'aide du **menu options** ou directement au moyen du menu déroulant qui apparaît en cliquant sur **le bouton droit de la souris**.

2.3.1. Options de fichiers

Entre les différentes options du menu, nous trouvons la possibilité de créer un **nouveau** fichier de configuration, en **ouvrir** un existant, **enregistrer** la configuration actuelle en pouvant changer le nom et la situation choisies et finalement une option pour **sortir** de l'application.

2.3.2. Options d'édition

Soit au moyen des options **du menu de l'application** soit avec le menu déroulant qui apparaît en cliquant sur **le bouton droit** de la souris, nous pouvons **créer**, **insérer**, ou **effacer** toutes nos variables et commandes.

En créant une nouvelle variable ou commande, le programme nous présente un **petit carré** à l'endroit où l'on doit définir la variable ou la commande.

En **cliquant avec la souris** sur ce petit carré, un objet apparaît, dans **une couleur plus sombre**.

En **double-cliquant** dans cet objet, un **formulaire** apparaît où nous pouvons choisir les différentes options. Dans ce formulaire sont **grisées** les options auxquelles **nous n'avons pas accès**, en devant compléter le reste de données.

Une fois que nous cliquons **OK** dans ce **formulaire**, notre petit carré prend une autre forme suivant la configuration qui a été donnée à la variable ou à la commande, permettant ainsi de distinguer simplement quel type d'objet a été défini.

Pour **insérer** une nouvelle **variable**, nous devons en sélectionner une déjà existante et choisir l'option **insérer variable**. La nouvelle variable apparaîtra juste avant celle que nous avons sélectionnée, sous forme de carré, tant que nous n'avons pas complété le formulaire.

Il en est de même avec **une commande**, en en choisissant une déjà existante et en choisissant l'option **insérer commande**.

En ayant sélectionné une **variable**, nous pouvons choisir l'option **de créer une commande**, de cette manière apparaîtra un **petit carré** mais dans une **branche accrochée à cette variable**. Ceci indique qu'il s'agit **d'une commande de script**

associé à cette variable. À partir de cette commande nous pouvons **insérer** ou **créer** des nouvelles commandes qui composeront **le script de la variable**, en prenant en considération que **l'ordre d'exécution** sera toujours le même que l'ordre de création, en commençant par la première commande **en haut** et en continuant jusqu'à la dernière.

Dans les **commandes de condition** qui apparaissent, on peut **créer une autre commande** (une fois choisie la condition) dépendant d'elle. Ainsi les commandes accrochées à cette condition seront exécutées au cas où cette **condition est certaine**, sinon elles ne seront pas exécutées et il sera continué avec la commande suivante du même niveau que la condition correspondante.

C'est-à-dire que les différents niveaux nous indiquent quelles commandes seront exécutées lorsque les conditions sont remplies. Et toujours en tenant compte du fait que l'exécution se fait en fonction de l'arborescence **du haut vers le bas**.

Pour **effacer** toute **commande** ou **variable**, nous devons seulement sélectionner l'option **éliminer**. Au cas où l'élimination entraîne l'effacement **d'un groupe de commandes**, le programme demandera **une confirmation**.

2.3.3. Autres options

Une autre des options disponibles est celle de **Développer** et **Réduire** l'information présentée sur l'écran, de telle sorte que les différents niveaux de commandes dépendant des variables apparaîtront **tous visibles**, ou en cas de sélection de l'option contracter, tous **occultes**.

Un **click de souris** dans **l'icône +** fait apparaître de nouveau la branche avec les commandes qui étaient maintenues occultes.

Une autre option disponible est celle de choisir **le langage de l'application**. Il est important de tenir compte qu'une fois qu'on aura choisi un nouveau langage, **il faut sortir puis lancer à nouveau l'application pour qu'apparaisse le nouveau langage**.

2.4. Programmation de scripts

Pour chaque variable nous pouvons définir une séquence de commandes, qui seront exécutées consécutivement lors de l'activation de l'événement correspondant à cette variable.

La programmation permet d'effectuer des assignations, d'utiliser des fonctions ou des commandes de condition.

On dispose aussi de différentes variables additionnelles que nous pouvons utiliser pour des valeurs temporaires.

2.4.1. Variables locales

En lançant un script on crée automatiquement plusieurs variables locales. Elles sont définies comme locales parce que leur valeur est seulement maintenue pendant l'exécution du script avec l'idée de stocker des valeurs temporaires. Il existe deux types de variables, les réelles, et les booléennes :

Les variables réelles stockent une valeur réelle, c'est-à-dire des valeurs décimales ou entières, positives ou négatives, dans le rang 5.0×10^{-324} à 1.7×10^{308} .

Il existe 3 variables réelles qui sont définies comme **L0**, **L1** et **L2**.

Les variables booléennes stockent la valeur d'une condition qui peut être **fausse** ou **vraie**. Elles sont fondamentalement utilisées pour former des conditions à partir de conditions simples puisqu'une seule condition est possible dans les commandes conditionnelles.

Il existe 3 variables booléennes qui sont définies comme **C0**, **C1** et **C2**.

Par exemple, pour définir une condition du type "Si $L0 > 5$ et $L0 < 10$ ", nous ferons une première assignation du type " $C0 = L0 > 5$, $C1 = L0 < 10$, $C2 = C0 \text{ AND } C1$ ". De tel manière que **C0** sera **faux** ou **vrai** en remplissant les conditions précédentes.

Par défaut, en lançant le script les variables réelles prennent **la valeur 0**, et les booléennes **la valeur faux**.

2.4.2. Commande ASSIGNATION

Cette commande sert à assigner à une variable la valeur d'une autre ou la valeur d'une opération entre 2 variables ou valeurs.

Le premier paramètre sera la variable réceptrice du calcul ou de l'assignation, qui pourra donc seulement être une variable (locale ou du serveur). Dans le cas d'une variable locale booléenne, les paramètres suivants pourront seulement être une condition entre d'autres variables ou valeurs, ou une assignation directe d'une autre variable locale booléenne.

Au cas où la variable réceptrice est l'une du serveur, la variable ne prendra cette valeur que si elle diffère de celle qu'elle avait avant, provoquant le lancement de l'événement associé à cette variable à ce stade du script, continuant l'exécution une fois fini le processus de ce nouvel événement.

Dans le menu déroulant on peut voir les variables disponibles pour cette opération, en prenant en considération que **dans une assignation dans le script d'une variable, cette variable ne peut jamais être réceptrice d'une assignation** sinon elle entrerait dans **une boucle infinie**.

Les paramètres suivants peuvent être également des variables locales, des variables du serveur, ou des valeurs constantes réelles (entières ou décimales, positives ou négatives).

La variable prendra dans ce cas le résultat de l'opération effectuée entre les paramètres.

Si la variable réceptrice est du serveur et la valeur du résultat à assigner n'est pas entière, **un arrondi sera fait automatiquement** et ce sera cette valeur qui sera assignée.

Entre les différents opérateurs nous avons +, -, *, /, **AND**, **OR**, ces derniers étant des opérateurs binaires devant des variables numériques.

S'agissant des conditions, nous pouvons utiliser >, >=, =, <=, <, **AND** et **OR** comme conditions "**ET**" et "**OU**".

2.4.3. Commande FONCTION

Cette commande opère sur une variable soit locale soit du serveur.

Il existe plusieurs fonctions définies pour chaque type de variable :

ROUND : Cette fonction effectue l'arrondi de la variable ou de la constante qu'elle a comme paramètre, en assignant le résultat à la variable à laquelle elle se réfère. Si cette variable est du serveur, cette fonction **peut provoquer un événement** à ce point de la variable affectée (Par exemple $L0 = \text{ROUND}(\#L1)$).

SETBIT : Cette fonction active le bit indiqué dans le paramètre (au travers d'une variable ou d'une constante entière). Si cette variable est du serveur, cette fonction **peut provoquer un événement** à ce point de la variable affectée (Par exemple $L0 = \text{SETBIT}(5)$, qui fait que le bit numéro 5 est mis à 1 et que le reste n'est pas modifié).

CLEARBIT : Cette fonction désactive le bit indiqué dans le paramètre (au travers d'une variable ou d'une constante entière). Si cette variable est du serveur, cette fonction **peut provoquer un événement** à ce point de la variable affectée (Par exemple $L0 = \text{CLEARBIT}(5)$, qui fait que le bit numéro 5 est mis à 0 et que le reste n'est pas modifié).

NOT : Cette fonction inverse la valeur d'une variable locale booléenne. C'est-à-dire que si la valeur était vraie, elle devient fausse et vice versa (Par exemple NOT(C0)).

TOGGLE : Cette fonction n'est pas encore mise en œuvre.

TIMER : Cette fonction n'est pas encore mise en œuvre.

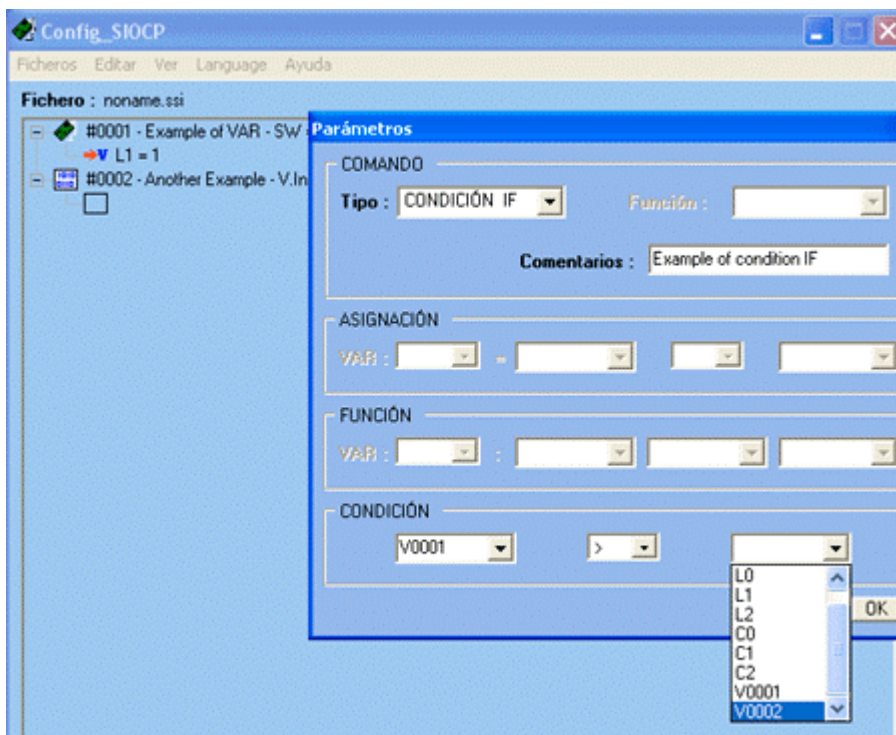
2.4.4. Commande CONDITION IF (Si la condition X se produit...)

Ce type de commande nous permet d'exécuter un ensemble de commandes dans le cas où une **condition est remplie**.

L'ensemble des commandes sera défini par les lignes de commandes **dépendantes accrochées** à cette **commande IF**.

Entre les paramètres de cette commande, nous pouvons avoir soit une condition entre variables et constantes, soit simplement une variable booléenne (qui sera vraie si la condition est remplie), ou comme unions avec **AND** ou **OR** de 2 variables booléennes (qui sera remplie si le résultat final de l'opération entre les deux variables est vrai).

Si la condition n'est pas remplie (elle est fausse), il sera alors poursuivi avec la commande suivante dans le script.



2.4.5. Commande CONDITION ELSE (Dans un autre cas...)

Cette commande aussi nous permet d'exécuter un ensemble de commandes.

Elle accompagne toujours une commande **de type CONDITION IF**, de telle sorte que dans le cas où la condition n'est pas remplie, ce serait cet autre bloc de commandes qui serait exécuté. Toutefois, si **la CONDITION IF** précédente est remplie, ce bloc de commandes **ne sera pas exécuté**.

2.4.6. La boucle sans fin

Un danger dans la programmation du script est **la boucle sans fin**.

La boucle sans fin est produite lorsqu'on lance des événements de manière circulaire de tel manière que l'exécution ne s'arrête jamais, parce que des événements en lancent d'autres, qui eux-mêmes lancent ceux d'origine, sans possibilité d'arrêter ou d'arrêter après **un overflow**.

2.4.7. Exemples de programmation

Avec la combinaison des différentes possibilités, variables, commandes, etc., nous pouvons effectuer des simulations complexes et des opérations.

Sur www.opencockpits.com apparaîtront différents tutoriaux où il sera expliqué pas à pas comme effectuer différentes opérations avec ce logiciel.

2.5. Programme SIOC.Serveur IOC et noyau de l'application.

L'application SIOC.exe est chargée de lire le fichier de configuration et de lancer le serveur IOC et les différents modules de l'application.

Au moyen du fichier SIOC.INI on configure les paramètres de base de l'application. Ce fichier explique l'utilisation de chaque paramètre.

L'application pourra être lancée sur la même machine où est le simulateur ou sur une autre, reliée par le biais **du client FSUIPC** ou **le client IOCP** (en indiquant dans ce cas les paramètres de la connexion TCP/IP).

Sur l'écran de l'application, on peut observer l'état du serveur IOCP ainsi que chacun des modules actifs dans l'application.