Documentation Script Thrustmaster Warthog

Maitrisez les scripts

30/12/2010
www.checksix-fr.com
Phoenix
Dimebug
Ghostrider

Contenu

Contenu des scripts

Contenu indispensable du fichier :

Le programme qui suit peut être lancé, mais n'exécutera aucune fonction. Vous devrez toujours mettre ces lignes dans votre fichier. Une fois lancé, tous les périphérique thrustmaster supportés seront regroupé au sein d'un contrôleur USB virtuel, mais rien ne se passera, car aucune fonction ou axes n'aura été associé.

```
Include "target,tmh" //Ici, nous faisons le lien entre le fichier actuel et le fichier qui contient les codes de fonction Thrustmaster.

Int main()
{
if(Init(&EventHandle)) return 1; // Déclare l'évènement return en cas d'erreur

//Les scripts et les fonctions sont à mettre ici et avant }
}

int EventHandle(int type, alias o, int x)
{
DefaultMapping(&o, x);
}
```

Cela ne vous semble pas clair, ne vous inquiétez pas, tout ce que vous avez à faire est de remplir la ligne "//les scripts et fonctions vont ici" avec les instructions fournies.

NOTE : Quand vous créez un profil, pensez aussi aux personnes qui le regarderont. L'ordre des fonctions est libre, mais évitez de mélanger différent genre de fonction, car ça en rend la lecture plus dure.

La boite à outil des fonctions de TARGET

Lier les axes avec une fonction de mappage d'axe.

La première chose à faire est de mapper les axes physiques du contrôleur aux axes du contrôleur virtuel. Direct X gère 8 axes. Nous pouvons les associés à n'importe quel axe ou contrôleur, en utilisant la commande MapAxis. Cette fonction lie un axe physique à un axe virtuel. Par défaut, aucun des axes physiques ne sont mappés. Nous verront plus loin dans le manuel le traitement de la réponse des axes.

Syntaxe :

```
MapAxis(&Périphérique utilisé, nom de l'axe physique, nom de l'axe direct X, option1, option2);
```

Tableau des noms d'axes

Nom DirectX	Nom des axes suivant les périphériques			
	Script Axis Name	HOTAS WARTHOG	HOTAS COUGAR	T-16000M
X	DX_X_AXIS	JOYX	JOYX	JOYX
Y	DX_Y_AXIS	JOYY	JOYY	JOYY
RZ	DX_ZROT_AXIS	THR_LEFT	RUDDER	RUDDER
Z	DX_Z_AXIS	THR_RIGHT	THROTTLE	
RX	DX_XROT_AXIS	SCX	RDR_X	
RY	DX_YROT_AXIS	SCY	RDR_Y	
Slider0	DX_SLIDER_AXIS	THR_FC	MAN_RNG	THROTTLE
Throttle	DX_THROTTLE_AXIS		ANT_ELEV	
MOUSE X	MOUSE_X_AXIS	MOUSE_X_AXIS	MOUSE_X_AXIS	MOUSE_X_AXIS
MOUSE Y	MOUSE_Y_AXIS	MOUSE_Y_AXIS	MOUSE_Y_AXIS	MOUSE_Y_AXIS

Commençons par lié l'axe X du Warthog à un axe X de Direct X.

Tout d'abord, nous devons spécifier un nom pour le contrôleur USB gérant l'axe.

MapAxis(&Joystick,

Ensuite, le nom de l'axe X du périphérique physique.

MapAxis(&Joystick, JOYX,

Au final, nous devons renseigner le nom de l'axe direct X à mapper.

MapAxis(&Joystick, JOYX, DX_X_AXIS);

Comme nous avons ouvert une parenthèse "(" nous devons la fermer par ")" et finir la ligne avec un point virgule ";"

Note : Le point virgule est indispensable pour cloturer une ligne appellant une fonction.

Occupons nous des autres axes du Warthog :

```
MapAxis(&Joystick, JOYY, DX_Y_AXIS);
MapAxis(&Throttle, THR_LEFT, DX_ZROT_AXIS);
MapAxis(&Throttle, THR_RIGHT, DX_Z_AXIS);
MapAxis(&Throttle, SCX, DX_XROT_AXIS);
MapAxis(&Throttle, SCY, DX_YROT_AXIS);
MapAxis(&Throttle, THR_FC, DX_SLIDER_AXIS);
```

Tous les axes sont maintenant liés à des axes Direct X, et une fois que vous aurez compilé et executé votre script, votre joystick et votre manette des gaz fonctionneront comme un seul périphérique.

Notre fichier ressemble à ce qui suit :

```
include "target.tmh" //here we link this file to the file that contain function code

int main()
{
if Init(&EventHandle)) return 1; // declare the event handler, return on error

//script and function functions goes here
MapAxis(&Joystick, JOYX, DX_X_AXIS);
MapAxis(&Joystick, JOYY, DX_Y_AXIS);
MapAxis(&Throttle, THR_LEFT, DX_ZROT_AXIS);
MapAxis(&Throttle, THR_RIGHT, DX_Z_AXIS);
MapAxis(&Throttle, SCX, DX_XROT_AXIS);
MapAxis(&Throttle, SCY, DX_YROT_AXIS);
MapAxis(&Throttle, THR_FC, DX_SLIDER_AXIS);

}

int EventHandle int type, alias o, int x)
{
DefaultMapping(&o, x);
}
```

Si vous executez le fchier et regardez ce qui se passe dans l'analyseur de périphérique (Device Analyser), vous constaterez que vos axes bougent. Si vous le souhaitez, vous pouvez appuyer sur un bouton du controleur, et vous constaterez que rien ne se produit, pour la simple raison qu'aucune fonction n'est associé aux boutons pour l'instant.

Option de la commande MapAxis

Dans le script précédent, nous n'avons utilisé que quelques paramêtres. La commande MapAxis gère aussi les options suivantes :

MapAxis(&Périphérique utilisé, nom de l'axe physique, nom de l'axe direct X, AXIS_NORMAL ou AXIS_REVERSED, MAP_ABSOLUTE ou MAP_RELATIVE);

- AXIS_NORMAL : l'axe utilise son mouvement par défaut.
- AXIS_REVERSED : inversion de l'axe.

- MAP_ABSOLUTE : fonctionnement par défaut de l'axe.

- MAP_RELATIVE : Il s'agit d'un mode particulier dans lequel la valeur de l'axe se fixera sur la valeur minimal ou maximal atteinte, jusqu'à ce que vous bougiez à nouveau. Ce paramêtre a été créé pour les axes pivotants, ou les microstick qui controlent le curseur d'un désignateur de cible. N'utilisez ce paramêtre que lorsque c'ets necessaire.

Note : Ce mode permet un controle réaliste du désignateur de cible sous Lockon Modern Air Combat, DCS Flaming Cliffs 1 (et 2 sans le patch).

Exemple :
MapAxis(&Throttle, SCY, DX_YROT_AXIS, AXIS_REVERSED, MAP_RELATIVE);

MapAxis(&Throttle, SCX, MOUSE_X_AXIS, AXIS_NORMAL, MAP_RELATIVE);

Exemple d'utilisation du stick du HOTAS Warthog avec la manette des gaz (TQS) du HOTAS Cougar.
Note : Vous remarquerez que l'axe des gaz du HOTAS Cougar est renseigné sous le périphérique servant de joystick. C'ets necessaire du fait que la manette de gaz et le joystick du Cougar sont reliés à l'ordinateur par un seul cable USB provenant de la base du joystick.

```
Include "target.tmh"
int main()
{
   if Init(&EventHandle)) return 1;
//axis mapping Warthog Joystick
   MapAxis(&Joystick, JOYX, DX_X_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);
   MapAxis(&Joystick, JOYY, DX_Y_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);
//Axis Mapping TQS
   MapAxis(&HCougar, THROTTLE, DX_Z_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);
   MapAxis(&HCougar, MAN_RNG, DX_SLIDER_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);
   MapAxis(&HCougar, ANT_ELEV, DX_THROTTLE_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);
   MapAxis(&HCougar, RDR_X, DX_XROT_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);
   MapAxis(&HCougar, RDR_Y, DX_YROT_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);
   MapAxis(&HCougar, RUDDER, DX_ZROT_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);

}
int EventHandle int type, alias o, int x)
{
   DefaultMapping(&o, x);
}
```

Maintenant que nous avons mappé nos axes, nous pouvons nous occuper des boutons. Nous verrons plus loins comment personnaliser les courbes des axes.

Note : Quand vous créez un profil, pensez aussi aux personnes qui le regarderont. L'ordre des fonctions est libre, mais évitez de mélanger différentes fonctions, car cela rend la lecture plus dure.

Souvenez vous que les axes Direct X seront toujours utilisables même s'ils ne sont liés à aucun axe physique.

Commande RotateDXAxis : Rotation d'Axe virtuel

La commande RotateDXAxis est un exemple de la puissance du script. Cette fonction a été créé pour corriger de manière logiciel une disposition physique particulière du stick. Cette fonction sera trés utile au concepteur de cockpit, qui ne manqueront pas de positionner physiquement leur joystick comme dans les avions réels.

Syntaxe :

RotateDXAxis(DirectX axis Name, Second Direct X axis name, twist angle value);

Exemples:

RotateDXAxis(DX_X_AXIS, DX_Y_AXIS, 5); //simule une rotation de 5° sur le côtés comme sur le stick du F-16

RotateDXAxis(DX_X_AXIS, DX_Y_AXIS, -15); //simule le centrage du stick du A-10 par une rotation de -15°.

RotateDXAxis(DX_X_AXIS, DX_Y_AXIS, 90); //transforme X en Y et Y en X

Note : Comme il s'agit d'un réglage important, nous vous suggérons de le placer avant la fonction MapAxis, afin que des personnes utilisant votre profil puissent le voir.

Programmer un bouton Direct X ou l'appui d'une touche via la commande MapKey

Nous allons utiliser une commande qui lie l'appui d'une touche à un bouton. Il s'agit de la commande :

MapKey

Nous allons utiliser MapKey avec un bouton, mais pour cela nous devons connaitre le nom de chaque bouton, mais aussi leur position.
Tous les boutons ont un nom, ainsi que les positions des interrupteurs, même ceux qui ne dépendent pas de Direct X.

Par exemple : Sur la manette des gaz du Warthog, au niveau des fonctions d'Autopilot, l'interrupteur LASTE offre 3 positions. Il n'a pas de déclaration lorsqu'il est centré, En position haute, il devient le bouton DX 27, et en position basse le bouton DX 28. Avec le script il est possible d'affecter une commande à la position centrale. Cela rend la programation plus simple. En

plus de ce manuel, vous trouverez un fichier PDF décrivant le nom des boutons pour les périphériques supportés.

Printed function	DX Button number	Script name
Alt	DX 27	APPAT
ALT/HDG	NA	APAH
ALT/HDG	DX 28	APALT

Règle sur la commande MapKey

La commande MapKey utilise toujours cette syntaxe :

MapKey(&périphérisue physique, nom du bouton, commande);

Example:
MapKey(&Joystick, TG1, 'a');

Neutraliser l'appui d'une touche

Si vous ne voulez produire aucun effet, c'est trés simple :

MapKey(&Joystick, TG1, 0);

"0" représente un évènement nul. Dans certains cas, vous voudrez neutraliser un bouton, ce qui se fait en executant "0".

Appui simple d'une touche

Tout d'abord nous devons définir le bouton que nous voulons programmer, par exemple Trigger1. Nous devons préciser à la commande **MapKey** que notre bouton se trouve sur le joystick du Warthog et qu'il se nomme TG1 :

MapKey(&Joystick, TG1,

Nous avons défini que nous voulions programmer le bouton Trigger 1 du joystick, mais il nous reste à renseigner l'effet produit. Programmons lui l'appui sur la touche "a" :

MapKey(&Joystick, TG1, 'a');

Vous remarquerez que la touche programmée se trouve entre deux symboles afin d'éviter tout mélange. Le signe apostrophe ' définit le fait que vous allez renseigner un appui sur une touche.

Tout ce que vous avez à faire lorsque vous utilisez le profil d'une autre personne est de vérifier la zone qui définit l'effet voulu.

Quelque fois, vous avez besoin d'utiliser une touche spéciale telle que Escape, la barre d'espace,... Comme il ne s'agit pas de lettre simple ou de chiffre, vous devez utiliser une syntaxe particulière. Avec ces touches, vous n'avez pas besoin de mettre des apostrophes avant et aprés la définition de la touche.

Exemple :

MapKey(&Joystick, S3, BSP); //quand j'appuie sur le bouton S3 du joystick, cela simule l'appuie sur retour arrière.

Voici la syntaxe des touches spécifiques

L_CTL
R_CTL
L_SHIFT
R_SHIFT
L_ALT
R_ALT
L_WIN
R_WIN
ESC
F1 to F12
PRNTSCRN
SCRLCK
BRK
BSP
TAB
CAPS
ENT
SPC
INS
HOME
PGUP
DEL
END
PGDN
UARROW
DARROW
LARROW
RARROW
NUML
KP0
KP1
KP2
KP3
KP4

KP5
KP6
KP7
KP8
KP9
KPENT

Vous êtes maintenant capable de programmer un bouton avec n'importe quelle touche.

Combinaison de touches

Parfois, dans un simulateur, vous aurez à utiliser une combinaison de touches faisant appel à plusieurs touches afin d'utiliser une commande particulière. Ces combinaisons utilisent en général les touches Control, Alt et Shift.

MapKey(&Joystick, S1, L_SHIFT+ 'b');//Lorsque l'on appuie sur le bouton S1 du joystick la combinaison "left shift b" est effectuée.

Commande "PULSE+"

Ajouter la commande **PULSE+** devant une touche simulera le maintien d'une touche préssée pendant un laps de temps très court. Par défaut, l'utilisation dela commande **PULSE+** simule une touche pressée pendant 25 millisecondes.

Exemple :

MapKey(&Joystick, H4P, PULSE+F1);

MapKey(&Throttle, SPDF, PULSE+L_ALT+'b');

Note : Si nous n'êtes pas familiarisé avec les boutons de type interrupteur, vous devez vous rappeler que ce ne sont pas de simple bouton, et qu'une fois basculés en position "**ON**", ils y restent. Si vous programmez l'appui d'une touche sur la position "**ON**", cela simulera une touche qui reste pressée jusqu'à ce que vous repositionnez l'interrupteur sur la position "**OFF**". Cela peut poser une problème, ainsi pour éviter la pression inutile d'une touche, vous pouvez utiliser la commande "PULSE+".

Commande "DOWN+" et "UP+"

Pour simuler le maintien d'une touche pressée, vous pouvez utiliser la commande "**DOWN+**". Méfiez vous de cette commande. Utiliser cette commande signifie que vous donnez l'ordre de simuler l'appuie maintenue d'une touche, jusqu'à e que vous donnier l'ordre de relacher la touche avec la commande "**UP+**". Vérifiez bien que vous ayez associé une commande "**UP+**" aprés une commande "**DOWN+**".

Exemple :

MapKey(&Joystick, H1U, DOWN+'a');
MapKey(&Joystick, H1D, UP+'a');

Faites une pause. Utilisez ces commandes, créez des profils et testez les. Une fois que tout vous semblera familier, vous pourrez aborder les chapitres suivants.

Simuler l'appuie d'une touche avec un code USB

Utilisez une simple touche ne représente pas la meilleur solution. En effet, la disposition des touches changeant d'un pays à l'autre, un "**a**" pourra être interpréter comme un "**q**" par le programme.

Syntaxe :
MapKey(&périphérique physique, nom du bouton, code usb);

MapKey(&Joystick, TG2, USB[0x07]);// code usb pour la touche "D"
Dans notre exemple, le Trigger 2 simule l'appuie sur la touche "**d**" : "**07**" est le code USB héxadécimal pour la touche "**d**". Vous trouverez ces codes dans le livret de référence. La syntaxe pour utiliser un code USB est : USB[0xXX]

Générer un bouton DirectX

Pour cela, nous allons utiliser à nouveau la commande MapKey, où nous remplacerons le code USB, ou la touches à simuler, par le numéro DirectX du bouton. N'oubliez pas que DirectX ne gère que 32 boutons et les 8 directions du chapeau chinois.

- Syntaxe des boutons DirectX : DX1, DX2, DX3,... DX32.
- Syntaxe du chapeau chinois sous DirectX : DXHATUP, DXHATUPRIGHT, DXHATRIGHT, DXHATDOWNRIGHT, DXHATDOWN, DXHATDOWNLEFT, DXHATLEFT, DXHATUPLEFT.

Exemple:
MapKeyIO(&Joystick, TG1, DX1);

Maintenant vous êtes capable de créer un profil équivalent à celui générer par l'interface graphique. Nous allons attaquer le niveau avancé.

Utilisation de fichier Macro

Qu'est ce qu'une macro ? On peut définr une macro comme étant le raccourci d'une combinaison de touches. Cela signifie que vous donnez un nom à l'appuie sur une touche spécifique dans le fichier macro, et que vous utiliserez ce nom dans le fichier principal pour y faire référence. Il y a un double avantage à utiliser une macro, C'ets plus simple à lire, et c'est aussi plus simple à adapter ,si vous utilisez un mapping personalisé de vos touches dans votre simulateur de vol.

Exemple:
Au lieu d'écrire :
MapKey(&Joystick, TG1, SPC);

Vous écrivez :
MapKey(&Joystick, TG1, Weapon_Fire);

Et vous trouverez dans le fichier de macro :

define Weapon_Fire USB[0x2C] //Weapon Fire

Le fichier utilisé pour renseigner les macros est un fichier *.ttm (pour Thrustmaster TARGET Macro).

Si vous voulez utilisez un fichier de macro avec votre fichier principal TMC, vous devrez y faire référence (le fichier contenant vos macros doit être stocké au même endroit que le fichier TMC de votre profil). Il vous suffit simplement de rajouter une ligne au début du fichier TMC en utilisant la commande "**include**".

Exemple:

```
include "target.tmh"
include "FC2_MIG_29C_Macros.ttm"

int main()
{
   if Init(&EventHandle)) return   ; // declare the event handler, return on error

MapAxis(&Joystick, JOYX, DX_X_AXIS);
MapAxis(&Joystick, JOYY, DX_Y_AXIS);
….
```

Exemple du contenu d'un fichier macro :

```
// Autopilot **********************

define  Autopilot_Attitude_Hold       L_ALT+USB[0x1E]       //Autopilot – Maintien d'altitude
define  Autopilot_Barometric_Altitude_Hold   L_ALT+USB[0x21]    //Autopilot – Maintien d'altitude barométrique
define  Autopilot       USB[0x04]       //Autopilot
define  Autopilot_Altitude_And_Roll_Hold   L_ALT+USB[0x1F]       //Autopilot - Altitude And Roll Hold
define  Autopilot_Barometric_Altitude_Hold_H       USB[0x0B]       //Autopilot -Maintien d'altitude barométrique 'H'
define  Autopilot_Transition_To_Level_Flight_Control  L_ALT+USB[0x20] //Autopilot - Transition vers le niveau de vol
define  Autopilot_Disengage L_ALT+USB[0x26]       // Désactiver l'autopilot
define  Autopilot_Radar_Altitude_Hold       L_ALT+USB[0x22]       //Autopilot - Radar Altitude Hold
define  Autopilot_Route_following    L_ALT+USB[0x23]       //Autopilot – 'Mode route'
```

Note :
Raphael Bodego a crée un générateur gratuit, de fichier de macro compatible avec TARGET, il couvre la plus part des simulateurs du marché, et vous permetra de générer plus rapidement votre fichier de macro. Ce programme s'appelle Sim2TARGET. Ce n'est pas un produit oifficiel Thrustmaster, mais c'est un outil trés utile. Vous le trouverez à cette URL :www.checksix-fr.com.

Possibilités multiples pour un bouton simple.

Commande TEMPO

Tempo est une sous fonction de MapKey. Elle est basée sur l'ergonomie réelle de l'avionique moderne. Tempo offre la posibilité au pilote d'affecter deux commandes à un seul bouton. Un appui court génèrera une commande tandis qu'un appui long en génèrera une autre. Cette possibilité est utilisée sur les avions de combat.

Syntaxe :
TEMPO(key1, key2, delay) delay est optionnel, 500 millisecondes est une bonne valeur.

Example:
```
MapKey(&Joystick, TG1, TEMPO('x', 'y'));          //appui court génère X, appui long génère Y
MapKey(&Joystick, TG1, TEMPO('x', 'y', 1000)); //si pressé plus d' 1 seconde
```

Layers / Profil

Une autre prossibilité de multiplier le nombre de bouton assignables à un bouton est d'utiliser des profils.

L'utilisation de profil s'apparente à l'utilisation de plusieurs programme tournant simultanément. Vous sélectionnez le profil désirer en agissant sur le bouton maitre. Le sélecteur de profil peut être un bouton ou plusieurs boutons en prenant garde de faire attention de l'état du ou des boutons. Ce qu'il faut garder à l'esprit, c'est qu'il faut actioner le bouton maitre pour accéder au profil. Pour une description complète des profils, nous vous invitons à consulter le paragraphe dédié dans la documentation du GUI.

Les utilisateur du HOTAS Cougar connaissent les intérrupteurs tel que /I/O/U/M/D

T.A.R.G.E.T est dédié aux périphériques de vol Thrustmaster. Il existe plusieurs possibilités de bouton pouvant servir de bouton maitre.

- Sur la manette de gaz du Cougar, nous vous conseillons d'utiliser le bouton à 3 positions "Dogfight", et le bouton S3 sur le manche.
- Sur la manette de gaz du Warthog, nous vous conseillons l'utilisation de l'interrupteur 3 positions "Boat", et le bouton S4 sur le manche.
- Sur le T16000, tous les boutons de la base peuvent être utilisés.
- Sur les MFD Cougar, les boutons GAIN, SYM, BRT et CON pourront être utilisés.

Note : Vous pouvez utilisez les boutons maitres de ces différents éléments.

Par défaut, l'accés au profil annexe est momentané. Le profil annexe est activé lorsque vous appuyez sur le bouton maitre. Si vous utilisez un interrupteur à positions pour sélectionner votre profil, les profils annexe U,M,D fonctionneront tant que votre interrupteur maitre sera dans la position donnée. Mais si vous utilisez un joystick T16000 qui ne possèdent que des boutons poussoirs, vous devrez appuyer sur plusieurs boutons simultanément pour accéder au profil souhaité.

Pour résoudre ce problème, le profil peut être défini comme temporaire, ou basculé.

- Temporaire : Vous devez maintenir appuyé un bouton pour accéder au profil.
- Basculé : Il vous suffit d'appuyer sur un bouton pour changer de profil.

Profils principaux

Les profils principaux sont appelés Up (Haut), Middle (Millieu) et Down (Bas). Par défaut vous programmez le profil Middle. Sur le HOTAS Cougar, les profils U, M, D sont habituellement liés à l'interrupteur Dogfight. Ce bouton est un interrupteur à 3 positions qui rempli parfaitement le role de sélecteur de profil. Comme il s'agit d'un interrupteur à position, le bouton reste dans la position du profil désiré.

Sous profil ou profil annex

Tout profil possède un sous profil que les utilisateurs de HOTAS Cougar connaissent en tant que In/Out. On l'utilise surtout pour un profil temporaire, que l'on active à partir d'un bouton agissant comme la touche "Shift". Vous pouvez utiliser ces sous profil pour activer des fonctions secondaires dans votre simulateur de vol, tel que les vues externes.

Dans notre fichier, il nous faut définir les interrupteurs qui controleront l'accés au profil annex. Nous avons plusieurs possibilités qui dépendront dépendra de ce que nous souhaitons, ou du type d'interrupteur voulu.

S4, PSF et PSB sont les noms des positions des boutons

SetShiftButton(&Joystick, S4, &Throttle, PSF, PSB); *// no toggle for U/M, usual HOTAS Cougar setting. All layers are momentary.*

SetShiftButton(&Joystick, S3, &Throttle, PSF, PSB, IOTOGGLE); *// toggle only for I/O button.*

SetShiftButton(&Joystick, S3, &Throttle, PSF, PSB, IOTOGGLE+UDTOGGLE); // toggle for I/O and U/M/D buttons.

SetShiftButton(&Joystick, S3, &Throttle, PSF, PSB, UDTOGGLE); *// toggle for U/D buttons.*

Note : Comme le profil Middle est celui par défaut, vous n'avez pas à le sélectionner ou spécifier une commande. Si vous n'êtes pas dans le profil U ou D, vous ne pouvez qu'être dans le profil Middle.

Il n'y a pas beaucoup de différence entre l'utilisation d'un profil, et la programmation d'un bouton.

Commande MapKeyUMD

Pour utiliser les profile UP, Middle et Down, vous devez utilisez la commande MapKeyUMD

Commande(&périphérique, bouton, effet Up, effet Middle, effet Down);

MapKeyUMD(&Joystick, S4, 'u', 'm', 'd'); //Quand vous appuyez sur le bouton S4 avec le bouton maitre UMD en position Up, un "u" sera généré, en position Middle, ce sera un "m", et en position Down, ce sera un "d"

Il n'y a aucune règle pour utiliser cette fontions, mais le bouton UMD permet de controler facilement 3 profils différents. Un pour le Air-Air, un pour la Navigation (atterrissage, etc...), et un pour le Air-Sol.

Commande MapKeyIO

Pour utiliser le sous profil In et Out, vous utiliserez la commande MapKeyIO

Commande(&périphérique, bouton, effet Out , effet In);

MapKeyIO(&Joystick, S1, L_SHIFT+ 'b', 'a');

En fonction de l'état du bouton sélecteur, lorsque vous appuierez sur le bouton S1, vous générerez un "B" ou un "a"

- En appuyant sur le bouton de sélection (in) : vous produirez "Lshift + b"
- Si le bouton selecteur n'est pas préssé (out) : vous produirez un "a"

Imaginons maintenant que vous souhaitiez garder une programmation Direct X sur le bouton TG1, et générer un "b" lorsque le bouton sélecteur est pressé.

MapKeyIO(&Joystick, TG1, DX1, 'b');

Commande MapKeyIOUMD

La commande MapkeyIOUMD vous permet d'affecter 6 effets différents à un seul bouton, le plus dure sera de vous rappeller ce que vous avez programmé.
Chaque profil U, M, D supporte un sous profil IO.

MapKeyIOUMD(&Joystick, S4, KP1, KP2, KP3, KP4, KP5, KP6);

Cette ligne n'est pas facile à lire, mais il est possible de la présenter différament.

```
MapKeyIOUMD(&Joystick, S3,
KP1,  // BSF button, if shift button In generate Keypad 1
KP2,  //BSF, if shift button Out generate Keypad 2
KP3,  // BSM, if shift button In generate Keypad 3
```

KP4, // BSM, if shift button Out generate Keypad 4
KP5, //BSB, if shift button In, generate Keypad 5
KP6); //BSB, if shift button Out, generate Keypad 6
De cette façon, il est plus simple de lire l'information.

Avec de la pratique, vous vous rendrez compte que presser une touche ne vous donne pas entière satisfaction dans le simulateur. C'est bien de pouvoir creer un effet en appuyant sur un bouton, mais il peut être interressant d'avoir une action lorsque l'on relache l'appui sur le bouton.

Commande MapKeyR : Produire un effet en relachant un bouton avec la

La commande MapKeyR fonctionne comme la commande Mapkey sauf qu'un effet se produit lorsque l'on relache le bouton pressé.

- *MapKeyR*

MapKeyR supporte aussi les sélecteurs de profil et sous profil In, Out, Up, Middle et Down :

- *MapKeyRIO*
- *MapKeyRUMD*
- *MapKeyRIOUMD*

La ligne de commande MapKeyR peut différer de la ligne de commande MapKey appliqué au même bouton. Vous pouvez avoir :

MapKeyUMD(&Joystick, S4, 'a', 'b', 'c');
MapKeyRIO(&Joystick, S4, 'y', 'z');

Note : Toutes les touches générées en relachant un bouton (utilisant la commande MapKeyR) utilisent la commande "Pulse" automatiquement, même si vous n'avez pas programmé la commande "Pulse + "touche"" afin d'éviter que la touche généré ne soit activé en continue. Vous pouvez changer ce fonctionnement en utilisant la commande "DOWN+".

Lorsque vous utilisez les sélecteur IO ou UMD, l'effet produit sera celui du bouton pressé utilisant la commande MapKeyIO, même si vous relachez le bouton de sélection de profil.

Commande CHAIN : Produire plusieurs effets en même temps avec la

CHAIN(PULSE+'a'*, PULSE+* 'b'*)* *//Cette commande produira un appui temporaire sur les touches a et b.*

Utilisons la commande CHAIN avec la commande MapKey.
Syntaxe :

MapKey(&périphérique, bouton, CHAIN(effet 1, effet 2,…)

Exemple :

MapKey(&Joystick, H3U, CHAIN(PULSE+'a', 'b'*));* // Lorsque j'utise la position haute du Hat 3, les touches "a" et "b" sout activés avec la commande pulse.

Si vous testez cette ligne de commande dans l'Event Tester, vous remarquerez :

- les touches "a" et "b" sont pressées simultanénent, et relaché ensemble. Cela signifie qu'il n'y a pas d'ordre d'appuie sur les touches. Pour éviter cela, utilisez la commande Delay.

- Si vous maintenez le Hat3 dans la position haute, la touche b"" sera maintenu jusqu'à ce que vous relachiez le bouton. Pour éviter le maintien de la touche, vous pouvez utiliser la commande "PULSE+"

Vous pouvez mettre un nombre illimité d'effet en utilisant la commande Chain, mais il y a une limitation lié au matériel. Un clavier ne supporte l'appuie simultané que de 5 à 6 touches, et comme nous avons vu que les touches sont préssées simultanément, si la commande CHAIN contient plus de 5 touches, les effets produits à partir de la sixième touche seront ignorés par windows. Il existe une solution pour éviter ce problème.

Nous savons que la durée de la commande "Pulse" est de 25 millisecondes. Nous allons donc attendre que notre première touche ait été pressé avant d'activer la suivante. Pour cela, nous allons utiliser la commande Delay.

Commande Delay : D() (parenthèse ouverte et parenthèse fermée)

Si vous insérez la commande D() dans votre commande CHAIN, vous utiliserez la durée par défaut de la commande Delay entre vos deux effets. Vous pouvez définir la durée, en reseignant une valeur exprimée en milliseconde, entre les parenthèses.

MapKey(&Joystick, H3U, CHAIN(PULSE+'a', *D(), PULSE+* 'b'*));*

En testant cette ligne de code dans l'Event Tester, vous remarquerez que : "a" est pressé puis relaché, et ensuite "b" est pressé puis relaché.

Comme la commande Delay laisse le temps de relacher la première touche, vous pouvez utilisez la commande Chain avec beaucoup de touches ou combinaison de touches. Vous devez garder à l'esprit que si vous voulez séparer les touches utilisant la commande Pulse, vous devez utiliser la commande Delay.

La durée par défaut de la commande Delay et Pulse peut être réglé par la commande :

SetKBRate(25, 33); // PULSE dure 25 ms, D() dure 33 ms

Exemple :

Vous pouvez utiliser la commande Chain pour gérer le lancement des Flarres et des Chaffs, ou les message radios les plus importants. Dans la plus part des simulateur, pour gérer les communications, vous devez appuyer sur une touche pour ouvrir un menu avec une liste de possibilité, qui ouvrira à son tour un sous menu pour enfin sélectionner l'action voulue.

Imaginons que pour appeller un ailier vous deviez appuyer sur la touche "w", et que lui ordonniez d'engager (F2) votre cible (F3).
Avec l'interrupteur radio de la manette des gaz du HOTAS Warthog, vous pouvez utiliser une des positions afin de créer un raccourci qui ordonnera à votre ailier d'attaquer votre cible. Au lieu d'appuyer sur 3 touches successivement, vous n'aurez qu'un interrupteur à actionner.

Pour obtenir cet effet, vous devrez écrire les lignes de code suivantes :

```
MapKey(&Throttle, MSD, CHAIN(
   PULSE+'w',    //Appel de l'ailié
   D(),
   PULSE+F2,    //Sélection de l'action Engager dans le menu de communication
   D(),
   PULSE+F3    //Ma cible
   ));
```

Ou de cette façon :

```
MapKey(&Throttle, MSD, CHAIN(PULSE+'w', D(),PULSE+F2, D(), PULSE+F3)); //Raccourci Ailié attaque ma cible.
```

Si l'appui généré des touches est trop rapide pour le programme (simulateur), il vous suffit d'augmenter le délais comme dans l'exemple qui suit :

```
MapKey(&Throttle, MSD, CHAIN(
   PULSE+'w',    //Appel de l'ailié
   D(50),
   PULSE+F2,    // Sélection de l'action Engager dans le menu de communication
  D(50),
  PULSE+F3    //Ma cible
  ));
```

Parfois vous devrez protéger votre commande Chain de tout parasitage du clavier. Tout appuie sur une touche avant que la séquence de la commande Chain soit fini (ou une partie spécifique de la commande Chain), devra attendre pour être executé.
Cette possibilité est géré par la commande **LOCK+**

Commande LOCK

La commande LOCK empèche tout interférence provoqué par l'appuie d'une touche, dans une séquence de touche en cours d'execution. De cette manière, votre commande Chain ne peut pas être interrompue.

Syntaxe :

Au début de la zone que vous désirez protéger : LOCK+ Touche
A la fin de la zone protégée : LOCK

Exemples :

```
MapKey(&Joystick, TG1, CHAIN(
        LOCK+           //Ouvre la zone protégée
        PULSE+'a',
        D(1000),        //Attente de 1 seconde
        PULSE+'b',
        D(1000),        //Attente de 1 seconde
        PULSE+'c',
        D(1000),        //Attente de 1 seconde
        LOCK            //Fermeture de la zone protégé, déverouillage
        ));
```

Cela revient à dire lancement de la commande Chain, ouverture de la zone protégée avec Lock, puis appuie sur la touche "a" avec une commande pulse, attente d'une seconde, appuie sur la touche "b" avec la commande pulse, attente d'une seconde, appuie sur la touche "c" avec la commande pulse, attente d'une seconde, fermeture de la zone protégée et déverouillage de l'execution d'une autre commande.

Note : Si votre commande Chain est trés longue, il est possible de créer plusieurs zones protégées ce qui permettra de pouvoir exécuter les commandes lancées par un autre bouton.

Exemple :

```
MapKey(&Joystick, TG1, CHAIN(
   LOCK+     //Activation de la zone protégée
   PULSE+'a',
   D(1000),   //Attente d'une seconde
   PULSE+'b',
   D(1000),   //Attente d'une seconde
   LOCK,      //Fermeture de la zone protégée, déverouillage
   D(33),     //Délais en millisecondes pour une action non prévue dans la séquence
   LOCK+   //Activation de la zone protégée
   PULSE+'c',
   D(1000),
   PULSE+'d',
   LOCK     //Fermeture de la zone protégé, déverouillage
   ));
```

Commande Sequences

La commande SEQ (pour sequence) est utilisée pour produire un effet différent à chaque fois que vous appuyer sur le même bouton. De cette manière vous pouvez gérer le défilement des vues externes, ou sélectionner votre armement, en appuyant sur le même bouton.

```
MapKey(&Joystick, S1, SEQ('a', 'b', 'c'));
```

Dans la ligne précédente, l'appuie sur le bouton TG1 produira un "a" jusqu'à ce que vous relachiez la détente (bouton TG1). Si vous re-pressez la détente, vous obtiendrez un "b", et si vous re-pressez la détente à nouveau vous obtiendrez un "c". Un nouvelle appuierelancera la séquence depuis le début en produisant un "a".

Si necessaire, il vous est possible d'utiliser la commande Pulse.

```
MapKey(&Joystick, S1, SEQ('a', PULSE+'b', 'c'));
```

Une commande **SEQ** peut être combiné avec les commandes **MapKey** et **MapKeyR**.

```
MapKeyIOUMD
   (&Joystick, H2U,     // hat2 du joystick en position Haute
   'a',
   SEQ(KP1, KP2, KP3, KP4),
   'c',
   'd',
   'e',
   'f');
```

Note : Il est possible d'utiliser une commande SEQ dans une commande SEQ, ce qui permet de programmer des commandes complexes. Voici un exemple de ce cas :

```
MapKeyR(&Joystick, TG1, SEQ(SEQ('1', '2'),  R_SHIFT+'s') );
```

Cette ligne de commande permmetra de généré la séquence suivante à chaque pression sur le bouton TG1 : 1,S, 2, S, 1, S ...

Il vous est possible d'utiliser la commande Chain :

```
MapKey(&Joystick, H2U, SEQ(
   CHAIN(PULSE+KP1, D(), PULSE+KP2),   //Première commande CHAIN
   CHAIN(PULSE+KP3, PULSE+KP4)         //Deuxième commande CHAIN
   ));
```

Les commandes Chain et Seq peuvent être combiné afin d'obtenir des effets plus complexes. Par exemple, vous pouvez affecter la touche "a" au bouton TG1 (génération d'un "a" lors de la pression du bouton TG1, puis génération d'un "a" lorsque l'on relache le bouton TG1), et la touche "b" aprés une commande de délais ayant une durée d'attente par défaut :

```
MapKey(&Joystick, TG1, CHAIN(SEQ(DOWN+'a', UP+'a'), D(), PULSE+'b'));
```

Les axes

Nous savons déjà comment assossié un axe physique à un axe Direct X. Affinons notre connaissance sur le comportement des axes. La première chose à faire est de définir le type d'axe que nous voulons modifier. Si l'axe possède un centrage mécanique comme les axes X et Y d'un joystick, nous utiliserons la commande SetSCurve, tandis que si nous avons un axe de type poussoir (Slider) comme une manette de gaz, un frein, nous utiliserons la commande SetJCurve.

Commande SetSCurve

Vous utiliserez la commande SetSCurve pour modifier les axes de votre joystick et de votre palonnier.

Syntaxe :

```
SetSCurve(&Device,  axis name,  left_deadzone, center_deadzone,  right_deadzone,  curve,  scale);
```

```
SetSCurve(&Throttle, SCX, 0, 30, 0, 0, -4);
```

Ou

```
SetSCurve(&Joystick, JOYX,
5,      //Zone neutre de 5% sur la gauche
2,      //Zone neutre de 2% au centre
5,      //Zone neutre de 5% sur la droite
3,      //Réglage de la courbe à 3
0      //Scale/zoom neutral
 );
```

Une zone neutre est une plage de valeur où rien ne se passe. Cette valeur est un pourcentage de la totalité du débatement de l'axe. Plus la valeur est grande et plus la zone neutre sera importante. Manipulez avec prudence les zones neutre, sous peine d edégradé le fonctionnement de votre controleur.

- **Zone neutre au centre** : Vous utilisez cette option lorsque vous souhaitez éviter les mouvements de faibles amplitudes au centre de l'axe. En général, les valeurs seront comprise entre 0 et 5%

- **Zone neutre à droite et à gauche** : En augmentant ces valeurs, vous raccourcissez l'amplitude du mouvement à effectuer pour atteindre la valeure maximal et minimale de l'axe physique. Cela a pour résultat d'augmenter la sensibilité du joystick (n'est pas utilisé habituellement).

Le paramêtre Curve permet d'ajuster la sensibilité du controleur. En utilisant cette option, vous choisissez de rendre votre stick :

- moins sensible dans la position centrale, mais plus réactif sur les extrémités de l'axe.

- plus sensible en position centrale en augmentant la précision sur les extrémités d'axes

Il y a une plage de 40 valeurs possiblesallant de -20 à +20. Les valeures négatives rendent votre joystick plus sensible autours de la position centrale, tandis que les valeurs positives offrent un meilleurs controle dans la position centrale.

L'option Scale est un nouveau paramêtre. Il s'agit d'une sorte de multiplicateur/diviseur.

- Avec une valeur négative, il limitera le déplacement de l'axe.

- Avec une valeur positive, il vous permettra d'atteindre les valeurs maximales et minimales de vos axes, sans que vous ayez à utiliser toute l'amplitude physique du débatement de l'axe.

L'utilisation de la fonction Scale est le meilleur moyen pour affiner le controle du microstick de la manette des gaz du HOTAS Warthog. Si vous le trouvez trop sensible dans le simulateur, indiquez une valeur négative. Plus la veleur sera grande, et plus l'effet de l'option Scale affectera l'axe (utilisez une valeur comprise entyre -20 et +20, zéro n'a aucun effet).

Exemple de paramêtre, pour ralentir le mouvement de la souris controlé par le microstick de la manette des gaz du HOTAS Warthog :

```
MapAxis(&Throttle, SCX, MOUSE_X_AXIS, AXIS_NORMAL, MAP_RELATIVE);
SetSCurve(&Throttle, SCX, 0, 10, 0, 0, -4);
 MapAxis(&Throttle, SCY, MOUSE_Y_AXIS, AXIS_REVERSED, MAP_RELATIVE);
SetSCurve(&Throttle, SCY, 0, 10, 0, 0, -4);
```

Commande SetJCurve

La commande SetJCurve est dédié au réglage de la sensibilité des axes servant pour les gaz, le mélange, le pas d'hélice, les freins. Vous définissez la valeur d'un axe Direct X qui sera émulé quand vous atteindrez physiquement cette valeur avec votre axe. La réponse linéaire se défini par deux paramêtres.

Exemple d'application :

- Vous pouvez rendre votre manette des gaz plus ou moins sensible lorsque vous atteignez de hautes valeur.
- Vous pouvez faire en sorte que le déclenchement logiciel de la post combustion corresponde à la détente physique du controleur.

Syntaxe :
SetJCurve(&périphérique, nom de l'axe physique, valeur de l'axe physique, valeur DirectX voulue);

Les valeurs de l'axe physique et Direct X sont exprimées en poucentage.

Exemple :
SetJCurve(&Throttle, THR_LEFT, 80, 95); //A 80% du débattement de l'axe physique, l'axe Direct X aura atteind 95% de la valeur de l'axe.

Commande SetCustomCurve

Parfois, vous aurez besoin de créer vos propres courbes, d'ajouter une zone où l'effet de l'axe réagit peu, ou une zone neutre. La commande SetCustomCurve vous offrira l'oportunité de faire ce que vous désirez. Pour cela, vous devrez utilisez la commande List qui vous permettra d'associer des positions à une valeur effective d'un axe DirectX. La courbe s'adapte pour corespondre à vos points. Les valeurs sont exprimées en pourcentage.

Note : Un état défini par la commande SetCustomCurve ne pourra pas être utilisé par la commande EXEC.

Syntaxe :
SetCustomCurve(&périphérique, nom de l'axe physique, LIST(Position 1 de l'axe physique, Valeur 1 effective de l'axe, Position 2 de l'axe physique, Valeur 2 effective de l'axe, …);

Exemple :

SetCustomCurve(&Joystick, JOYX, LIST(0,0, 25,25, 50,50, 75,75, 100,100)); // Crée une réponse linéaire parfaite

SetCustomCurve(&Joystick, JOYX, LIST(0,0, 45,50, 55,50, 100,100));//Crée une zone neutre au milieu de l'axe entre 45 et 55%

SetCustomCurve(&Joystick, JOYX, LIST(0,0, 25,50, 50,0, 75,50, 100,0)); //Crée un effet amusant mais inutile de l'axe.

Controler un axe avec des boutons

Vous pouvez gérer vos axes avec n'importe quel bouton. C'est une bonne solution alternative pour controler le mélange, ou une plage de paramêtres sans avoir à mapperl'axe physique du controleur.

Syntaxe :
AXIS(Nom de l'axe DirectX, incrément, delais avant répétition);

Exemple: souris virtuel en utilisant le Hat2
MapKey(&Joystick, H1U, AXIS(MOUSE_Y_AXIS, -80, 20));
MapKey(&Joystick, H1D, AXIS(MOUSE_Y_AXIS, 80, 20));
MapKey(&Joystick, H1L, AXIS(MOUSE_X_AXIS, -80, 20));

MapKey(&Joystick, H1R, AXIS(MOUSE_X_AXIS, 80, 20));

Trimmer un axe

La commande Trim est trés imlple à utiliser, mais pour la maitriser, vous devez connaitre la commande EXEC que nous décrirons plus tard.

La valeur de la commande Trim est une modification que l'on applique à la valeur réelle de l'axe. Vous lisez la valeur de l'axe, auquelle vous aditionnez ou soustrayez la valeur de la commande Trim, et vous transmettez le résultat à appliquer à DirectX.
La commande Trim est utile pour les axes du joystick et du palonnier. Nous l'utiliserons pour les surfaces de controle de l'avion, afin d'obtenir une action moins importante sur le joystick et le palonnier.

Note : T.A.R.G.E.T. Permet d'utiliser 2048 valeurs (+/- 1024) afin de couvrir la plage complète des axes.

La pluspart des simulateurs permmettent de trimmer. Vous êtes libre d'utiliser les outils de T.A.R.G.E.T. ou ceux de votre simulateur. L'avantage de l'utilisation des commandes du simulateur est la possibilitée de mesurer l'effet appliqué via un affichage graphique.
L'avantage d'utiliser T.A.R.G.E.T. est le control total ainsi que la possibilité de définir les valeurs de trim.

Avec T.A.R.G.E.T. Nous pouvons :

- Aditionner ou soustraire une valeur de trim. C'est la façon habtuelle de gérer un trim digital. Chaque pression sur un bouton modifie l'axe d'un valeur donnée. Vous pouvez définir la valeur de trim, ainsi que sa direction.

- Forcer la valeur du trim. Nous l'utiliserons pour remettre à zéro le trim en donnant la valeur zéro à la modification à appliquer.

- Lire la valeur d'un axe pour lui appliquer une valeur de trim, ou à un autre axe, et aussi une formule mathématique si necessaire.

Application pratique (ne vous inquiétez pas si vous ne comprenez pas ces ligne, car elles sont liées à la commande EXEC que nous décrirons plus loin, recopier ces lignes dans votre fichier) :

Additioner ou soustraire une valeur de trim

MapKey(&Joystick, H1U, EXEC("TrimDXAxis(DX_Y_AXIS, -10);"));
MapKey(&Joystick, H1D, EXEC("TrimDXAxis(DX_Y_AXIS, 10);"));
MapKey(&Joystick, H1L, EXEC("TrimDXAxis(DX_X_AXIS, -10);"));
MapKey(&Joystick, H1R, EXEC("TrimDXAxis(DX_X_AXIS, 10);"));

Dans ces lignes, à chaque fois que j'appuie sur un bouton du Hat1, les axes X ou Y du joystick sont modifiés de 10 points. Si je maintiens le bouton appuyé, cela n'aura aucun effet, car la valeur du

trim n'est EXECuter qu'une fois. Si je souhaite que la valeur de trim boucle jusqu'à ce que je relache le bouton, il faut que j'utilise la commande REXEC (nous verrons plus loin en détail cette commande).

MapKey(&Joystick, H1L, REXEC(0, 100, "TrimDXAxis(DX_X_AXIS, -5);"));
MapKey(&Joystick, H1R, REXEC(0, 100, "TrimDXAxis(DX_X_AXIS, 5);"));
MapKey(&Joystick, H1U, REXEC(1, 100, "TrimDXAxis(DX_Y_AXIS, -5);"));
MapKey(&Joystick, H1D, REXEC(1, 100, "TrimDXAxis(DX_Y_AXIS, 5);"));

Forcer la valeur de trim

MapKey(&Joystick, S4, EXEC("TrimDXAxis(DX_X_AXIS, SET(0));TrimDXAxis(DX_Y_AXIS, SET(0));"));

Nous forçons la valeur du trim à zéro ce qui consiste en une remise au neutre du trim. Dans notre exemple, quand nous appuyons sur le bonton S4 du joystick, nous réglons le trim appliqué aux axes X et Y à zéro.

Lire la valeur d'un axe et lui appliquer une valeur de trim

MapKey(&Joystick, S1, EXEC("TrimDXAxis(DX_X_AXIS, CURRENT);TrimDXAxis(DX_Y_AXIS, CURRENT);"));

Quand nous appuyons sur le bouton S1 du joystick, nous mémorisons la position des axes X et Y, et nous calculons la distance par raport au centre, afin de l'appliquer aux axes X et Y. Cela fonctionne de la même manière que le trim dans DCS Blackshark. Pour ceux qui ne connaissent pas le trim du Blackshark, il s'agit de mettre le joystick dans une position, puis d'apuyer sur un bouton, et de relacher le manche. Le bouton aura mémorisé la position physique du manche, et appliquera la même pression sur les axes concernées au moment de l'appuie sur le bouton de trim automatique. Dans ce cas de figure, il est fortement recommendé d'utiliser une commande permettant d'annuler la commande de trim précédente.

Le prochain cas est un peu plus complexe :
MapKey(&Joystick, S1, EXEC("TrimDXAxis(DX_Y_AXIS, SET(Throttle[THR_FC]/32));"));

Quand nous appuyons sur le bouton S1 du joystick, nous lisons la valeur de l'axe de friction de la manette des gaz que nous divisons par 32 pour appliquer le résulat. Pourquoi ? Mon axe possède 65536 valeurs, alors que mon trimme n'offre que 2048 possibilités pour couvrir toute la plage de valeur de l'axe. En faisant 65536 divisé par 32, nous obtenons 2048, ce qui nous permet d'appliquer une valeur de trim compatible avec l'échelle de mon axe.

Commande KeyAxis permettant de produire un effet à partir d'un axe

Parfois, vous aurez besoin d'utiliser un axe pour produire un effet tel que l'appuie sur une touche. Nous appellons cela un "axe digital". Si votre simulateur vous donne la possibilité de configurer un axe classique, ou un axe digital, choisissez l'axe classique afin de vous simlplifier la tache. L'utilisation d'axe digital est recommandé lorsque tout vos axes analogiques sont utilisés, ou lorsque vous souhaitez utiliser une fonction qui ne peut être affectée à un axe,.

Syntaxe :

KeyAxis(&périphérique, nom de l'axe physique, 'Profil concerné', Type d'axe digital programmé);

"Profil concerné"

Vous définissez quel profil sera affecté par la fonction.

Exemples :

KeyAxis(&Joystick, JOYX, '',... //sera appliqué à tous les profils
KeyAxis(&Joystick, JOYX, 0,... //sera appliqué à tous les profils
KeyAxis(&Joystick, JOYX, 'i',...//Sera appliqué lorsque le selecteur In sera activé
KeyAxis(&Joystick, JOYX, 'ud',...//sera appliqué qu'aux profils UP et Down

Note : Il est inutil d'inclure le selecteur "i" et "o" en même temps, car :
'ioud' = 'ud' ,

Il en est de même pour les profils dans le cas où ils sont tous actif.
'ioumd' = 0 = ''

La gestion de l'appuie des touches et des fonctions varie d'un simulateur à l'autre, et il peut y avoir plusieurs possibilités pour s'adapter à la logique de chaque simulateur de vol. Afin de répondre à tous les cas de figure, nous avaons créé deux façons de gérer les axes.

Note : Les possesseur de HOTAS ont la possibilité de choisir entre 5 types d'axe digital. Les differents types d'axe digital géré par le Cougar peuvent être reproduit avec T.A.R.G.E.T.

Commande AXMAP1

AXMAP1 ets la première commande. Dans le mode 1, c'est la "direction" de l'axe qui définit la fonction générée (par exeple l'appuie sur une touche).

Utilisez cette commande lorsque votre simulateur utilise une touche pour incrémenter une valeur, et une autre touche pour la décrémenter.

Note : La commande AXMAP2 permet de simuler les axes digitaux du HOTAS Cougar de type 1,5,6.

Exemple :
Vous ajustez la puissance du moteur avec les touche "+" et "-" du pavé numérique.

Syntaxe :
KeyAxis(&périphérique, Nom de l'axe physique, profil(s), AXMAP1(nombre de zones, Fonction en incrémentant, fonction en décrémentant, fonction optionelle pour le centre);

Exemple :

KeyAxis(&Joystick, JOYX, 0, AXMAP1(5, PULSE+'r', PULSE+'l'));

Ou

```
KeyAxis(&Joystick, JOYX, 0,
   AXMAP1(       //utilisation du mode 1 pour la commande AXMAP
   5,          //Divise la plage de l'axe en 5 zones égales
   PULSE+'r',    //quand la valeur de l'axe augmente appliquez la commande pulse à la touche "r" dans chaque zone
   PULSE+'l'   //quand la valeur de l'axe diminue, appliquez la commande pulse à la touche "l" dans chaque zone
   ));
```

En bougeant l'axe X du joystick du Warthog de la gauche à la droite, on produira un effet égal à cinq fois l'appuie sur la touche "r", ce qui se traduit par :

rrrrr

En allant de la droite à la gauche (en buté) puis à nouveau complètement à droite, on produira un effet égal à cinq fois l'appuie sur la touche "l", puis cinq fois l'appuie sur la touche "r" ce qui se traduira par :

lllllrrrrr

Note : Afin d'éviter que la touche soit maintenue, la commande "pulse+" est requise.

Commande AXMAP1

L'utilisation de la commande AXMAP1 est parfaite pour controler la portée d'un radar, d'une caméra, ou une commande de gaz actionnée via deux touches.

La commande AXEMAP1 offre la possibilité de programmer une fonction sur la position centrale de l'axe. Pour utiliser cette possibilité, il faut que vous sachiez que la position centrale n'est pas une zone. Si une zone se trouve sur la position centrale, la fonction ne sera pas executée. Cela signifie que cette fonction sera incompatible avec un nombre de zone impaire, car l'axe est divisé en zone égale.

```
KeyAxis(&Joystick, JOYX, 0,
   AXMAP1(     // utilisation du mode 1 pour la commande AXMAP
   2,          //Divise la plage de l'axe en 2 zones égales
   PULSE+'r',  //quand la valeur de l'axe augmente, appliquez la commande pulse à la touche "r"
dans chaque zone
   PULSE+'l',  //Quand la valeur diminue, appliquez la commande pulse à la touche "l" dans
chaque zone
   PULSE+'c',  //Quand l'axe est en position centrale, appliquez la commande pulse à la touche "c"
   ));
```

Si vous bougez votre joystick sur l'axe X d'une position plein gauche à plein droite vous produirez l'appuie sur la touche "r", puis "c", et "r"

```
KeyAxis(&Joystick, JOYX, 0,
   AXMAP1(     // utilisation du mode 1 pour la commande AXMAP
   3,          //Divise la plage de l'axe en 3 zones égales
   PULSE+'r',  //quand la valeur de l'axe augmente, appliquez la commande pulse à la touche "r"
dans chaque zone
   PULSE+'l',  //Quand la valeur diminue, appliquez la commande pulse à la touche "l" dans
chaque zone
   PULSE+'c',  //
   ));
```

Si nous bougeons l'axe des X de notre joystick d'une position plei gauche à plein droite, nous produirons trois fois l'appuie sur la touche "r". La fonction pour la position centrale sera ignoré, car le centre de notre axe se trouve sur une zone impaire.

Souvenez vous que vous pouvez utiliser un évènement nul "0" lorsque vous ne voulez pas produire un effet.

Dans les exemples précédents, nous avons utilisé une fonction simple, mais si vous le désirez, il est possible d'utiliser les commandes MapKey, CHAIN, SEQ, SEQ avec CHAIN, ...

```
 KeyAxis(&Joystick, JOYX,0,
   AXMAP1(
   2,
         CHAIN(
                LOCK+PULSE+'h',D(),
                 PULSE+'e',D(),
                 PULSE+'l',D(),
                 PULSE+'l',D(),
                 PULSE+'o',LOCK),
         CHAIN(
                LOCK+PULSE+'g',D(),
                 PULSE+'o',D(),
                 PULSE+'o',D(),
                 PULSE+'d',D(),
                 PULSE+'b',D(),
```

```
                PULSE+'y',D(),
                PULSE+'e', LOCK),
        CHAIN(
                LOCK+PULSE+'e',D(),
                PULSE+'c',D(),
                PULSE+'h',D(),
                PULSE+'o', LOCK),
  ));
```

Commande AXMAP2

La commande AXMAP2 est un deuxième mode pour les axes digitaux.

Note : La commande AXMAP2 permet de simuler les axes digitaux du HOTAS Cougar de type 2,3,4.

Exemple : Si la puissance du moteur est controlé par les touches 0,1,2,3,4,5,6,7,8,9.

Syntaxe :
KeyAxis(&périphérique, nom de l'axe physique, profil(s), AXMAP2(nombre de zones, effet1, effet2, effet3…);

Note : Le nombre de zone doit être égale au nombre d'effet voulus.

Exemple :
KeyAxis(&Throttle, THR_RIGHT, 'ioumd', AXMAP2(5, PULSE+KP5, PULSE+KP4, PULSE+KP3, PULSE+KP2, PULSE+KP1));

Ou

```
KeyAxis(&Throttle, THR_RIGHT,'ioumd',
    AXMAP2(
    5,
        PULSE+KP5,
        PULSE+KP4,
        PULSE+KP3,
        PULSE+KP2,
       PULSE+KP1
  ));
```

En bougeant la manette des gaz de droite du Warthog de la position zéro à plein poussé, on générera 2345 ; la première zone qui sera ignorée, car on commence l'action depuis cette derniere.

Note : Les touches générées possèdent l'attribut "pulse+" afin d'éviter un appuie continu.

Essayons de controler le curseur de désignation de cible du radar avec le bouton "Slew control" de la manette des gaz du Warthog. Le curseur simulé est controlé par les touches flêchées du clavier.

Note : Avec le Hotas Cougar, nous aurions utilisés un axe digital de type 3. Dans notre cas, nous allons utilisé la commande AXMAP2 pour reproduire la commande du Cougar.

Nous allons diviser chaque axe en 3 zones :

- Bas ou Gauche
- Centré (où l'on ne veut pas que le curseur bouge)
- Haut ou Droit

Comme nous ne voulons aucun effet au centre, nous utiliserons la commande "null" (0).

```
KeyAxis(&Throttle, SCX, 0, AXMAP2(3, LARROW, 0, RARROW));
KeyAxis(&Throttle, SCY, 0, AXMAP2(3, UARROW, 0, DARROW));
```

Nous n'avons pas utilisé la commande "PULSE+", car les touches fléchées doivent restées appuyées afin de déplacer le curseur.

Les commandes KeyAxis offrent une utilisation flexible, mais la solution de diviser la plage de l'axe en zones égales, ne conviendra pas à toutes les situations. Nous allons donc aborder la commande LIST.

Commande LIST

La commande LIST se substitue au "divisiseur de la plage de l'axe". La commande LIST permet de personnaliser la taille des zones créées, ce qui est utile pour le déclanchement d'une fonction à un instant précis. Pour utiliser la commande LIST, il vous suffit de remplacer le nombre de zone par la commande LIST (plage de zone exprimée en poucentage).

LIST(0,10,90,100) = 3 zones, une de 0 à 10%, une de 10 à 90% et une de 90 à 100%

Exemples:

```
KeyAxis(&Joystick, JOYX, 0,AXMAP2(LIST(0,10,90,100), PULSE+'l',0, PULSE+'r')); //quand l'axe du joystick atteind les derniers 10% à gauche, la touche "l" est activée. Quand on atteind les derniers 10% à droite, la touche "r" est activée.
```

Activons la post combustion (touche "b"), lorsque la manette des gaz gauche, passe le point physique de post combustion.

```
KeyAxis(&Throttle, THR_LEFT, 0, AXMAP1(LIST(0,33,100), 0, PULSE+'b'));
```

Vous remarquerez que la touche "b" est activée uniquement dans la plage de l'axe correspondant à la post combustion (de 33 à 100%). S'il y a une touche pour désactiver la post combustion, nous pouvons utiliser la commande Pulse, lorsque nous atteignons la première zone. Si "n" est la touche permettant de désactiver la post combustion, nous aurons :

KeyAxis(&Throttle, THR_LEFT, 0, AXMAP1(LIST(0,33,100), PULSE+'n', PULSE+'b'));

Commande LockAxis

La commande LockAxis est une fonction, qui permet de vérouiller, la valeur d'un axe défini. Vous pouvez l'utiliser dans votre fichier pricipal, ainsi qu'avec la commande EXEC (voir les fonctions avancées)
En utilisant différents profils et axes digitaux, vous pourrez utiliser différents axes physiques pour controler des axes digitaux, sans toucher au valeur de l'axe DirectX concerné.

Syntaxe:
"LockAxis(&Périphérique physique, Nom de l'axe, Etat);
LockAxis(&Throttle, THR_LEFT, 1); *//Verouillera la manette gauche de la manette des gaz du Warthog*
LockAxis(&Throttle, THR_LEFT, 0); *//Déverouillera la manette gauche de la manette des gaz du Warthog*

Fonctions avancées

Avant de commencer à utiliser le code suivant, soyez certain de bien maitriser l'ensemble des fonctions précédentes.
Jusqu'à présent, nous nous sommes contentés d'utiliser des fonctions préfabriquées. Nous allons maintenant commencer à découvrir la vrai flexibilité et puissance du Script.
Commençons avec le très puissant EXEC.

EXEC, ou la boite de pandore.

EXEC est une fonction très puissante, sont principe est simple, EXEC exécute une fonction ou du code. Il est possible d'utiliser EXEC depuis une des fonctions préconstruite de T.A.R.G.E.T et de pouvoir ainsi coder en script pure au sein d'une fonction préfabriqué.
Il est possible d'utiliser EXEC d'une multitude de façon, pour gérer des fonctions, des drapeaux logiques ou du code pure.

Gestion de fonction

EXEC peut être utilisé dans toutes les fonctions de la famille des MAPKEY, KEYAXIS, ect… ON peut utiliser un EXEC au sein d'une CHAIN, d'une SEQ, d'un TEMPO, comme un simple événement. Par se fait, l'utilisation d'EXEC est assez simple, car vous connaissez déjà l'utilisation de ses fonctions.

Dans l'exemple suivant nous utiliseront EXEC pour changer la courbe de réponse des axes Joystick X et Y quand l'utilisateur presse le bouton S4.

Syntax:
EXEC("……")

MapKey(&Joystick, S4, *EXEC("SetSCurve(&Joystick, JOYX, 0, 0, 0 ,5, 0); SetSCurve(&Joystick, JOYY, 0, 0, 0 ,5, 0);"));*

Ici il est intéressant de constater que nous nous contentions de réécrire le code qui définie le comportement des axes. Quand on exécute cette ligne, on réécrit tout simplement certaines lignes de notre configuration. Prêtez une attention toute particulière aux " utilisé au sein de l'EXEC. Avec ces symboles nous avons ouvert, puis fermé nos lignes de code. Pour amélioré la lisibilité, il est également possible de découper notre EXEC de la manière suivante :

```
MapKey(&Joystick, S4,
            EXEC(
            "SetSCurve(&Joystick, JOYX, 0, 0, 0 ,5, 0);"
            "SetSCurve(&Joystick, JOYY, 0, 0, 0 ,5, 0);"
 ));
```

C'est nettement plus confortable à lire. A noter qu'une fois Exécuté, notre nouvelle configuration de courbe restera active. Si nous voulons revenir a la configuration initiale, il nous suffis de créer une séquence qui alterne les 2 configurations. Un premier appui change les courbes de réponse. Un second appui rétabli les paramètres d'origine.

```
MapKey(&Joystick, S4,
                        SEQ( //open the sequence
                        EXEC( //open the first EXEC
                        "SetSCurve(&Joystick, JOYX, 0, 0, 0 ,5, 0);"
                         "SetSCurve(&Joystick, JOYY, 0, 0, 0 ,5, 0);"
                         ), //close the first EXEC
                        EXEC( //open the second EXEC
                        "SetSCurve(&Joystick, JOYX, 0, 0, 0 ,0, 0);"
                         "SetSCurve(&Joystick, JOYY, 0, 0, 0 ,0, 0);"
                         ) //close the second EXEC
        ) //close the Sequence
); //close the MapKey
```

Cet exemple est une petite démonstration des capacités d'EXEC. Il était également possible de faire la même chose en créant une chaine contenant les 2 EXEC (un par Axe), mais nous avons choisi la façon qui semble être la moins complexe à écrire.

De la même façon, vous pouver utiliser EXEC pour :

- Remapper ou swapper des axes (également inverser le sens de fonctionnement, utiliser le mode relatif ou absolu…).
- Changer les réglages d'axes (courbes, Deadzone, scale).
- Changer le comportement des Axes digitaux (Axemap…).
- Changer le comportement des boutons (EXEC permet de reprogrammer le mapkey associé à un bouton).
- Eviter d'abuser des drapeaux logique, bien souvent, EXEC permet de faire la même chose de manière plus simple, sans utiliser de drapeau logique.

EXEC permet tout simplement de réécrire une configuration complète, sur une simple pression de bouton.

Il est également possible d'utiliser EXEC comme un bouton de « Shift », par exemple pour créer une génération de caractère dynamique (dépendante d'autres éléments).

Par exemple dans Flaming Cliffs 2, les modes autopilote sont directement activé via des raccourcis clavier dédié. Un vrai pilot auto ne fonctionne pas de cette manière et si vous souhaitez profiter au mieux du panel autopilot de votre Warthog, il est possible de corriger ce défaut du simulateur.

La solution consiste à générer des codes clavier quand on presse le bouton APENG de la throttle du Warthog, mais le contenu de ses codes clavier dépendra de la position de l'interrupteur LASTE.

En fait nous ne programmerons pas le bouton APENG, c'est le bouton LASTE, qui, en fonction de sa position, programmera le bouton APENG grâce à la fonction EXEC.

```
MapKey(&Throttle, APPAT, EXEC("MapKey(&Throttle, APENG, L_ALT+'6');"));
MapKey(&Throttle, APAH, EXEC("MapKey(&Throttle, APENG, L_ALT+'2');"));
MapKey(&Throttle, APALT, EXEC("MapKey(&Throttle, APENG, L_ALT+'4');"));
```

Il n'est même pas nécessaire de créer une ligne de code pour APENG, l'EXEC s'en charge.

Cependant, il faudra que l'EXEC soit activé au moins une fois pour qu'APENG soit fonctionnel.
Il peut donc être intéressant de créer une ligne de code par défaut pour APENG et celle si sera remplacé des qu'un EXEC sur APENG sera généré par l'interrupteur LASTE.

Quand vous exécuter du code dédié a un autre bouton, il est important de se rappeler que ce code ne sera exécuté qu'uniquement lorsque que le bouton concerné sera pressé.

Afin de protéger les ressources processeur, l'utilisation de SEQ, CHAIN, EXEC, TEMPO, AXIS, LIST au sein d'un EXEC est interdite, l'inverse est cependant possible. Il est possible d'outre passer cette limitation en créant votre propre fonction et en l'appelant depuis un EXEC (voir un peu plus loin).

REXEC

REXEC est une fonction définie pour ré exécuté un EXEC. Cette dernière bouclera jusqu'à ce que vous lui demandiez d'arrêter. Il est possible de lancer plusieurs REXEC en même temps, c'est pourquoi, chaque REXEC doit être référencé, numéroté. Ce numéro est nommé « handle ». De cette manière, il devient très simple de stopper spécifiquement un REXEC.
Le second paramètre est un retard définis en millisecondes, on l'utilise pour définir la fréquence de répétions. Plus la valeur en millisecondes est faible, plus la boucle sera rapide.
REXEC est parfait pour trimmer un axe ou lancer une fonction qui se répète.

Syntax:
REXEC(Handle, Delay, "code goes here", optional RNOSTOP)

Par défaut, REXEC bouclera tant que vous presses sa commande. Il est également possible de lui ordonner de boucler en continu puis de l'arrêter avec un ordre spécifique.

Pour stopper un REXEC, il suffit d'utiliser la commande StopAutoRepeat suivie de la référence (handle) de notre REXEC.

```
EXEC("StopAutoRepeat(4);")  //stop the REXEC number 4
```
Example:

```
MapKey(&Joystick, H1L, REXEC(4, 100, "TrimDXAxis(MOUSE_X_AXIS, -10);", RNOSTOP));


MapKey(&Joystick, H1R, EXEC("StopAutoRepeat(4);"));
```

Note: si vous utiliser un événement ou une fonction qui utilise une longue durée, il est prudent de s'assurer de ne pas REXECuter la fonction avant que le déroulement de celle ci ne soit fini.

Nous allons maintenant trimmer les axes X et Y du joystick en utilisant REXEC. Etant donné que nous utilisons 4 REXEC, on pourrait penser qu'il faut utiliser 4 « handles » différents, mais dans ce ca, comme tous les boutons ne peuvent être actionnés simultanément, seule 2 « handles » suffisent.

```
MapKey(&Joystick, H1L, REXEC(0, 100, "TrimDXAxis(DX_X_AXIS, -5);"));
MapKey(&Joystick, H1R, REXEC(0, 100, "TrimDXAxis(DX_X_AXIS, 5);"));
MapKey(&Joystick, H1U, REXEC(1, 100, "TrimDXAxis(DX_Y_AXIS, -5);"));
MapKey(&Joystick, H1D, REXEC(1, 100, "TrimDXAxis(DX_Y_AXIS, 5);"));
```

DeferCall

DeferCall est une commande utilisé pour appeler une fonction après un une durée d'attente programmé, c'est une action a retardement. C'est l'outil parfait pour insérer des notions de retard dans un EXEC ou un REXEC. Le compte à rebours commence quand la ligne est exécutée. Par exemple, si vous souhaitez avoir 3 événements, espacé d'une seconde chacun.

- Pour le 1er événement, on n'utilisera pas DeferCall.
- Pour le deuxième événement on utilisera un DeferCall callé à 1000 millisecondes.
- Pour le troisième est dernier, on utilisera un DeferCall callé à 2000 Millisecondes.

Syntax:

```
DeferCall( Delay, code goes here);
```

Example:

```
MapKey(&Joystick, H4U, EXEC("ActKey(KEYON+PULSE+'a'); DeferCall(1000, &ActKey, KEYON+PULSE+'b');"));
```

Syntax du Script

Apres avoir commencé à faire connaissance avec les fonctions avancé, il est temps d'approfondir un peu plus. EXEC nous permet d'exécuter du code, encore faut il savoir coder. Le but de ce document n'est pas de vous apprendre à le faire, cela demande beaucoup de pratique et de temps, nous serions déjà rendus à la page 200. La vocation de ce document est plutôt d'illustrer et montrer quelques une des possibilités offertes par le script. Les personnes familières du langage C s'y retrouveront très facilement, les autres feront abstraction de ce langage barbare pour essayer de comprendre la logique de fonctionnement et se l'approprier.

Mots clef supportés :

char, byte, short, word, int, alias, float, struct, include, if - else, do - while, while, return, goto, break

Operateurs

Les operateurs sont les outils de comparaison, transformation et logique.

Reference: **&** (placed before a variable) – obtain the physical address of the variable
Double Reference: **&&** (placed before a variable) – obtain the physical address of the variable buffer
Logical NOT: **!**
Multiplication: *****
Division: **/**
Modulus: **%**
Addition: **+**
Substraction: **-**
Right shift: **>>**
Left Shift: **<<**
Greater than: **>**
Less than: **<**
Less than or equal: **<=**
Greater than or equal: **>=**
Equal to: **==**
Not equal to: **!=**
Bitwise AND: **&**
Bitwise XOR: **^**
Bitwise OR: **|**

Seules les personnes maîtrisant la programmation sont aptes à les utiliser, mais à l'aide de quelques exemples basique, il nous est possible d'en comprendre l'usage, et de les réemployer.

Générer des codes clavier en Script pure.

Ou comment écrire sans l'aide de MapKey.

Si l'on ne passe pas par une fonction de la famille de MapKey, il faut utiliser la « vraie » syntaxe du script pour générer des code clavier. Jusqu'à présent, la fonction MapKey se chargeait d'une partie de la gestion des codes clavier, ici, nous allons le faire « à la main ».
Pour presser une touche, nous allons faire appel à la fonction « ActKey ». Cette fonction est compatible avec les flag PULSE+, L_SHIFT.. ce qui en simplifie l'emploi.

```
ActKey(KEYON+'x');                    //génère et presse continuellement "x"

ActKey('x');                          //relâche la touche "x"
```

L'emploi de KEYON est donc obligatoire pour active une touche.

```
ActKey(PULSE+KEYON+'A');              // pulse la touche "a"

ActKey(PULSE+L_SHIFT+KEYON+'b'); //pulse L shift+ "b"

ActKey(PULSE+KEYON+autopilot);    //active la macro ou function "autopilot"
```

Exemple :

MapKey(&Joystick, S2, PULSE+'j');

=

MapKey(&Joystick, S2, EXEC("ActKey(PULSE+KEYON+'j');"));

La seule différence entre les 2 lignes précédentes et qu'au lieu d'utiliser une fonction préformaté, pour la seconde nous avons utilisé du code pur en passant par la fonction EXEC.

Dans l'essentiel des cas, nous utiliserons ActKey comme résultat d'une condition d'entée, d'action. Par exemple, a l'issue d'une opération logique.

Note : regarder bien comment, en code pure, on défini et écrit la condition d'entrée pour un bouton.

```
MapKey(&Joystick, S4, EXEC(
                        "if(Joystick[TG1]) ActKey(PULSE+KEYON+'a');"
                        ));
```

=

```
MapKey(&Joystick, S4, EXEC(
                        MapKey(&Joystick, TG1, PULSE+'a');
                        ));
```

On peut constater que l'emploi de MapKey, même au sein d'un EXEC reste plus simple à écrire que du code pur.

Note : Il est possible d'utiliser les codes clavier USB dans une ActKey.

Drapeaux Logiques

Un drapeau logique n'est autre qu'une mémoire dans laquelle on stock une valeur. A l'aide de simple requête de logique, il est possible de créer des événements dépendants de l'état de cette mémoire.

Avec T.A.R.G.E.T, il n'y a pas de limitation de nombre de drapeaux et chaque drapeau peut stocker toutes sortes de valeurs. Il est également nécessaire de définir le nom de chaque drapeaux, cela rends la manipulation bien plus simple qu'avec le Cougar.

Chaque drapeaux peut contenir des valeurs différentes, c'est à dire qu'un drapeau n'est pas limité a un état 0 ou 1, il peut contenir a, b, c, d… cela évite de cumuler un trop gros nombre de drapeau pour gérer une même fonction.

La première chose à faire pour utiliser un drapeau est de le déclarer. Cela se fait, au début de votre script, juste après la ligne définissant le fichier .tmh.
Il suffit de taper ; « char nom du drapeau ;»

« char » spécifie que la variable qui va suivre sera destinée a recevoir des caractères.
En fonction de l'usage que vous réservez au drapeau, il est judicieux de le nommer en conséquence.

Autopilot_Status
Master_arm_status…

Par default, au lancement de votre script, l'état d'un flag est nul. Il faut soit une action pour l'activer, soit le « forcer » en lui déclarant un état au démarrage.
Pour définir sont état, il suffit de créer une ligne de code forçant sont contenu avant vos fonctions MapKey :

Autopilot=0; // le drapeau nommé Autopilot est démarré avec l'etat 0.

Ou

Autopilot=1; // le drapeau "Autopilot" est créer et contient la valeur 1.

Dans l'exemple suivant, le bouton S4 du joystick du Warthog est utilisé pour sécuriser l'emploi du bouton S1. Si S4 n'est pas pressé pendant l'appui sur S1, S1 ne générera aucun code clavier.
Pour ce genre de cas, il aurait été plus simple d'utiliser une fonction EXEC sans flag, car on peut faire exactement la même chose, plus simplement.

Pour valider que S4 a bien été pressé, nous utiliseront donc un drapeau qui deviendra « vrai » quand S4 est pressé, et « faux » quand S4 est relâché.
Nous utiliseront la condition logique « if » (si) pour valider ou non l'envoi des keystrokes depuis S1.

```
include "target.tmh"

char flag1; // nous declarons le drapeau "flag1"
int main()
{
if Init(&EventHandle)) return 1;

flag1=0; //nous forçons le Flag1 a etre nul au démarrage du programme
MapKey(&Joystick, S4, EXEC("flag1=1;")); //donne la valeur 1 au flag1 quand S4 est pressé
MapKeyR(&Joystick, S4, EXEC("flag1=0;")); //donne la valeur 0 au flag1 quand S4 est relâché
MapKey(&Joystick, S1, EXEC("if(flag1) ActKey(PULSE+KEYON+'a');")); //si le Flag1 est vrai, générer un code clavier "a" quand j'appui sur S1.
}

int EventHandle int type, alias o, int x)
{
DefaultMapping(&o, x);
}
```

Dans ce dernier exemple, S1 ne peut générer ActKey que si le Flag1 est vrai. Maintenant imaginons que si le Flag1 est « faux », nous souhaitons générer un « b ».
Pour cela nous utiliserons l'operateur « Equal to » (égale à) qui s'écrit de la manière suivante ==.
Pour ce faire, il nous suffit de modifier la ligne dédié au bouton S1 :

```
MapKey(&Joystick, S1, EXEC(
                        "if(flag1) ActKey(PULSE+KEYON+'a');
                        if(flag1==0) ActKey(PULSE+KEYON+'b');"
));
```

Cela marchera parfaitement, mais il est possible de faire plus simple. Ici nous savons que nous avons que 2 « output » possible : « a » ou « b ». Si les conditions ne sont pas réunies pour l'un, on gênera donc forcément l'autre.

En utilisant un operateur logique « else » (sinon), voila du code qui donnerait exactement le même résultat :

```
MapKey(&Joystick, S1, EXEC(
                        "if(flag1) ActKey(PULSE+KEYON+'a');
                        else ActKey(PULSE+KEYON+'b');"
));
```

Pour des choses relativement simples comme celles-ci, n'oubliez pas qu'un simple EXEC sera souvent bien plus simple à utiliser et écrire. Gardez les drapeaux logiques quand vous avez réellement besoin d'une mémoire.

Illustration

Maintenant, nous allons utiliser un Flag comme mémoire.

Imaginons que nous avons un bouton commandant le pilote automatique.

- La première fois qu'on presse le bouton, cela active le pilote automatique en émulant un appuis bref sur la touche « a ».
- La seconde fois que le boutons est actionné, nous souhaitons qu'il désactive le pilote auto en envoyant la combinaison de touche « shift+ a ».

Jusqu'ici, pas de difficulté particulière, une simple SEQuence permet d'y arriver très simplement. Imaginons maintenant que ce bouton pilotant le pilote automatique se trouve en double sur les manettes, à 2 endroits différents (c'est le cas sur le A-10). Nous devons synchroniser nos boutons de manière à ce qu'a chaque fois que je presse n'importe lequel des boutons, le code clavier corresponde à la situation pour soit désactiver le pilote auto, ou l'activer.

Pour cette synchronisation, nous allons utiliser un drapeau logique (flag) que nous appellerons « autopilot ». Ce dernier sera utilisé pour savoir si le pilote auto est allumé ou éteint. En fonction de cet état, nous générerons le code clavier adéquat.
Voici la logique utilisée lors de l'appui sur le bouton :

Si le drapeau pilot auto est activé, envoyer une pulsation de touche « a » ; sinon envoyer une pulsation « shift + a ».

Ce qui donne en script :
"if(autopilot=1) ActKey(PULSE+KEYON+'a'); else ActKey(PULSE+KEYON+L_SHIFT+'a');"
Avec cette ligne, nous avons déterminé quel code clavier envoyer. Il nous reste cependant a changer l'état du drapeau logique à chaque fois que nous pressons l'un des boutons.

Si le drapeau pilote auto est activé il faut le désactiver, sinon il faut l'activer.

"if(autopilot=1)autopilot=0 else autopilot=1;"// if the autopilot =1 set it to 0 else set it to 1

Cette phrase peut être écrite plus simplement en utilisant l'operateur"!"

autopilot = !autopilot; // reverse the flag status

Comme nous avons 2 opérations logique à exécuter, dans un ordre précis (il serait dommage d'inverser l'état du drapeau logique trop tot), nous utiliserons une CHAIN pour manager les etapes.

```
MapKey(&Throttle, LTB,CHAIN(
                EXEC(
                   "if(autopilot) ActKey(PULSE+KEYON+'a'); else
ActKey(PULSE+KEYON+L_SHIFT+'a');"),
                EXEC("autopilot = !autopilot;")));
MapKey(&Throttle, LTB,CHAIN(
                EXEC(
                   "if(autopilot) ActKey(PULSE+KEYON+'a'); else
ActKey(PULSE+KEYON+L_SHIFT+'a');"),
                EXEC("autopilot = !autopilot;")));
```

Voila pour le bouton LTB, reste à faire APENG:

```
MapKey(&Throttle, APENG,CHAIN(
                EXEC(
                   "if(autopilot) ActKey(PULSE+KEYON+'a'); else
ActKey(PULSE+KEYON+L_SHIFT+'a');"),
                EXEC("autopilot = !autopilot;")));
```

Voila, ces 2 boutons partagent un même flag pour générer un code clavier dépendant de l'état du flag. Chaque action sur un des boutons provoque l'inversion de l'état du flag.

Au final nous avons :

```
include "target.tmh"

char autopilot; // we create the flag
```

```
int main()
{
if Init(&EventHandle)) return 1;

MapKey(&Throttle, LTB CHAIN(
                    EXEC(
                      "if(autopilot) ActKey(PULSE+KEYON+'a'); else
ActKey(PULSE+KEYON+L_SHIFT+'a');"),
                    EXEC("autopilot = !autopilot;")));
MapKey(&Throttle, APENG CHAIN(
                    EXEC(
                      "if(autopilot) ActKey(PULSE+KEYON+'a'); else
ActKey(PULSE+KEYON+L_SHIFT+'a');"),
                    EXEC("autopilot = !autopilot;")));
}

int EventHandle int type, alias o, int x)
{
DefaultMapping(&o, x);
}
```

Quand on regarde le code appliqué aux boutons, on se rend compte qu'il est à 80% identique. C'est un peu dommage de ce répéter de cette manière, d'autant plus que le Script permet d'éviter ce genre de chose.

Quelque ligne plus tôt, nous avons vu que la génération des touches clavier du pilote auto pourrait se résumer à une simple SEQuence. Nous allons donc partir de ce principe pour construire une routine ou simple fonction, qui sera appelé quand on presse l'un des boutons concerné.

Vous allez voir que de cette manière, nous allons économiser de l'énergie et rendre le code bien plus simple.

Créer ses propres fonctions.

Créer une fonction et une bonne méthode pour transformer du script compliqué ou répétitif en quelque chose de plus simple. C'est également un moyen de contourner certaine limite des MapKey et EXEC. Avant tout, il faut savoir exactement ce que notre fonction doit faire et quand.

Commençons par un exemple tres simple, vous avez écrit le code suivant, mais le compilateur le rejette :

```
MapKey(&Throttle, CSD, EXEC("SetCustomCurve(&Throttle, SCX, LIST(0,35, 45,50, 55,50, 100,65));"));
```

Le compilateur refuse le code car vous utilisez une fonction LIST dans un EXEC et c'est tout simplement interdit pour protéger les ressources processeur.

La solution consiste à créer une fonction qui contient votre LIST puis a appeler la fonction dans l'EXEC.

Pour l'exemple, nous allons utiliser 2 LIST pour changer la réponse d'un axe.

La première chose à faire est donc de créer les fonctions contenant les LIST., Nous les appellerons List1 et List2. Cela se fait en debut de fichier avec la commande « int » :

```
int list1, list2; //declaring our custom LIST
```

Maintenant, le compilateur sait qu'il va y avoir 2 fonctions, mais il ne connaît pas encore leur contenu. Ce contenu sera spécifié dans le corps du script.

```
list1 = LIST(0,30, 45,50, 55,50, 100,70);
list2 = LIST(0,35, 45,50, 55,50, 100,65);
```

Ensuite, il ne nous reste plus qu'appeler nos fonctions depuis un EXEC.

```
MapKey(&Throttle, CSD, EXEC("SetCustomCurve(&Throttle, SCX, list1);"));
MapKey(&Throttle, CSU, EXEC("SetCustomCurve(&Throttle, SCX, list2);"));
```

Voiçi le fichier complet, faite bien attention à la position des éléments et leur ordre.
Selon mon action sur CSD ou CSU, je change la sensibilité de l'Axe SCX.
Ce qui est intéressant ici, c'est que je peut appliquer mes LIST à n'importe quel axe. Si je la modifie la fonction, l'ensemble des axes l'utilisant seront affecté.

```
include "target.tmh"

int list1, list2; //declaring our custom LIST

int main()
{
   if Init(&EventHandle)) return 1; // declare the event handler, return on error
MapAxis(&Throttle, SCX, DX_XROT_AXIS); //mapping axis
MapAxis(&Throttle, SCY, DX_YROT_AXIS); //mapping axis

list1 = LIST(0,30, 45,50, 55,50, 100,70);//defining list1
list2 = LIST(0,35, 45,50, 55,50, 100,65);//defining list2

MapKey(&Throttle, CSU EXEC("SetCustomCurve(&Throttle, SCX, list1);
                            SetCustomCurve(&Throttle, SCY, list1);"));//changing axis
behaviour
MapKey(&Throttle, CSD EXEC("SetCustomCurve(&Throttle, SCX, list2);
                            SetCustomCurve(&Throttle, SCY, list2);"));    //changing axis behaviour

}

int EventHandle int type, alias o, int x) {
   DefaultMapping(&o, x);
}
```

Comme vous le constatez, cela n'a rien de compliqué et facilite grandement la lecture d'un script, car sa rédaction s'en trouve simplifié. Des que vous avez du code qui se répète plusieurs fois ou que vous souhaitez partager des commandes, pensez y.
Au départ, c'est surtout après avoir fini un script que vous vous direz, tiens, pour faire telle fonctionnalité, il était possible de faire plus simple. Apres un peu de pratique, votre approche changera et le recourt aux fonctions sera automatique.

Etudions maintenant un autre cas.
Dans Flaming Cliffs 2, il n'est pas possible de créer son propre programme de largage de leurres. En utilisant T.A.R.G.E.T cela devient possible, il suffit de programmer une suite d'événements clavier.

Nous voulons donc lancer 4 chaff et 4 flare avec un espacement de 400 milliseconde entre chaque. Nous relancerons éventuellement ce programme toutes les 4 secondes. Le HAT4 sera utilisé pour contrôler tout ca.

- H4U lancera le programme qui sera répété uniquement si le bouton reste pressé.
- H4D lancera le programme qui bouclera jusqu'à un ordre d'arrêt
- H4P stoppera l'exécution de la boucle du programme.

Içi, nous allons utiliser 2 fois le même programme, donc c'est le contenu de ce programme qui sera la « fonction ».

Apres il ne nous restera plus qu'a la faire looper avec un REXEC.

Pour générer la succession de chaff et flare, nous utiliserons une simple CHAIN incluant un délai de 400 millisecondes entre chaque événement clavier. Nous appellerons notre fonction Chaff_Flare_Program_1.

```
Chaff_Flare_program_1 = CHAIN(
                        PULSE+INS,D(400),
                        PULSE+INS,D(400),
                        PULSE+INS,D(400),
                        PULSE+INS,D(400),
                        PULSE+DEL,D(400),
                        PULSE+DEL,D(400),
                        PULSE+DEL,D(400),
                        PULSE+DEL);
```

Maintenant que notre fonction est crée, il nous reste à l'associer aux boutons.

Pour que la fonction boucle, il faut utiliser REXEC. Le programme dure en tout 2800 millisecondes (7 x400). Ensuite, nous voulons une pause de 4 secondes (soit 4000 millisecondes). Pour la boucle, nous devons donc REXECcuter notre fonction toute les 6800 secondes.

H4U lance la boucle une fois et la repete si le bouton reste pressé.

```
MapKey(&Joystick, H4U, REXEC(0, 6800, "ActKey(KEYON+Chaff_Flare_program_1);"));
```

H4D Lance la boucle.

MapKey(&Joystick, H4D, REXEC(1, 6800, "ActKey(KEYON+Chaff_Flare_program_1);", RNOSTOP));

H4P stop la boucle
MapKey(&Joystick, H4P, EXEC("StopAutoRepeat(0);"));

Il ne reste plus qu'a declarer la function avant le *int main ():*

***int** Chaff_Flare_program_1; //we declare our new function*

Et voici le script complet:

```
include "target.tmh" //here we link this file to the file that contain the default Thrustmaster function code

int Chaff_Flare_program_1; //we declare our new function

int main()
{
   if Init(&EventHandle)) return 1;

   MapKey(&Joystick, H4P, EXEC("StopAutoRepeat(1);")); //stop repeating chaff and flare program 1
   MapKey(&Joystick, H4U, REXEC(0, 6800, "ActKey(KEYON+Chaff_Flare_program_1);")); //execute and loop chaff and flare program 1 until i release the button
    MapKey(&Joystick, H4D, REXEC(1, 6800, "ActKey(KEYON+Chaff_Flare_program_1);", RNOSTOP)); //execute and loop chaff and flare program 1

   Chaff_Flare_program_1 = CHAIN(
                           PULSE+INS,D(400),
                           PULSE+INS,D(400),
                           PULSE+INS,D(400),
                           PULSE+INS,D(400),
                           PULSE+DEL,D(400),
                           PULSE+DEL,D(400),
                           PULSE+DEL,D(400),
                           PULSE+DEL); //content of the chaff and flare program 1
}

int EventHandle int type, alias o, int x)
{
   DefaultMapping(&o, x);
}
```

Cet exemple n'est qu'une petite application, souvenez vous simplement que quand le code se répète ou que la rédaction devient trop complexe, une fonction permet de déporter une partie du code afin de faciliter les manipulations et la lecture. Il n'existe pas de fonction ultime, il faut juste créer les outils qui vous semblent utile et pratique.

Maintenant retournons a notre pilote automatique sous Flaming Cliffs 2. Nous l'avons managé avec l'aide d'un drapeau logique, maintenant, essayons avec une fonction pour voir si l'on ne peut pas se simplifier la vie.

Le management du drapeau logique est effectué de la manière suivante :

```
MapKey(&Throttle, LTB CHAIN(
                    EXEC(
                      "if(autopilot) ActKey(PULSE+KEYON+'a'); else
ActKey(PULSE+KEYON+L_SHIFT+'a');"),
                    EXEC("autopilot = !autopilot;")));
MapKey(&Throttle, APENG CHAIN(
                    EXEC(
                      "if(autopilot) ActKey(PULSE+KEYON+'a'); else
ActKey(PULSE+KEYON+L_SHIFT+'a');"),
                    EXEC("autopilot = !autopilot;")));
```

Il est claire que le code est ultra répétitif, içi, une fonction serait bien plus adapté. Nous avons constaté que le control du pilote auto ne faisait que changer d'état entre on et off, donc ici une simple SEQuence de code clavier devrait suffire.

Voici le résultat final, vous constaterez que la rédaction des MapKey pour LTB et APENG s'en trouve très nettement simplifié :

```
include "target.tmh"

int autopilot; // we Declare the autopilot function

int main()
{
if Init(&EventHandle)) return 1;

autopilot = SEQ(   EXEC("ActKey(PULSE+KEYON+'a');"),
                    EXEC("ActKey(PULSE+KEYON+L_SHIFT+'a');")
                   ); //we defines content of the autopilot function

MapKey(&Throttle, LTB, autopilot); //we simply call our function inside the MapKey
MapKey(&Throttle, APENG, autopilot); //We simply call our function inside the MapKey
}
//and that's all !
int EventHandle int type, alias o, int x)
{
DefaultMapping(&o, x);
}
```

Ici, le contenu de la function est plutôt simple : une SEQuence avec 2 output. Une fonction peut également contenir du code, ce qui permet énormément de choses.

Continuons à développer l'exemple du pilote auto sous Flaming Cliffs 2.

Si nous voulons manager le pilote auto d'une façon réaliste et en profitant au mieux des possibilités offertes par la throttle du Warthog, il va falloir ruser un peu, car le fonctionnement du PA du simulateur est erroné.

Sur la Throttle Warthog, nous avons un switch 3 positions pour déterminer le mode de pilote auto que nous souhaitons utiliser.

- APATH
- APAH
- APPAT

Nous avons également 2 boutons commandant l'activation et la désactivation du PA.

- LTB
- APENG

Nous savons déjà synchroniser les 2 derniers boutons, reste qu'il nous faut, en fonction de la position du LASTE switch du panel autopilot de la Throttle, générer le code clavier de démarrage du pilote auto adéquat.

En effet, sous Flaming Cliff 2 selon la combinaison de touches pressé pour activer le Pilote Automatique, vous le démarrerez dans des modes différents.

Il nous faut donc :
Générer la combinaison de touche pour allumer le PA en tenant compte de la position du LASTE switch lors d'un appui sur LTB ou APENG

Puis l'appui suivant sur LTB ou APENG, désactiver le pilote Auto.

Il est possible de faire cela de plusieurs façon :

- En utilisant des drapeaux logiques, un pour le statut du pilote auto, un second contenant l'état du switch LASTE. Cela marchera parfaitement mais risque d'être assez long a écrire comme nous l'avons déjà constaté.
- En mixant, un drapeau logique et une fonction
- En utilisant 2 fonctions, une fonction gère le code généré par le LASTE Switch. Cette fonction sera elle-même appelé au sein de la fonction Autopilot telle que nous l'avons utilisé précédemment. Nous choisirons cette solution.

En pratique nous allons conserver une bonne partie de notre code existant, nous allons juste changer le 1er événement de notre SEQuence en le remplaçant par une nouvelle fonction.

Cette fonction sera baptisée ap_ON_output est essentiellement basé sur un filtre logique :

Si le switch LASTE est en position APPAT, générer une pulsation « Alt + 6 »,
Sinon si le switch LASTE est en position APALT, générer une pulsation « Alt + 4 »,
Sinon générer une pulsation « alt + 2 ».

On notera ici que la logique n'utilise comme filtre que 2 des 3 positions du LASTE switch. C'est tout simplement lié au fait que si les 2 premières conditions sont fausses, ca ne peut être que la 3eme, l'interrupteur n'ayant que 3 position possibles.

Au final, voiçi le code. Noter que la fonction AP_ON_ouput a été crée en fin de fichier, juste avant int EventHandle int type, alias o, int x)µ.

```
include "target.tmh"

int main()
{
if Init(&EventHandle)) return 1;

int autopilot = SEQ(EXEC("ap_ON_output();"), PULSE+L_ALT+'9');// defining the autopilot function

MapKey(&Throttle, APENG, autopilot);
MapKey(&Throttle, LTB, autopilot);
}
//notice that the next function is build out of the main program
int ap_ON_output()  //naming the function
{
if(Throttle[APPAT]) ActKey(KEYON+PULSE+L_ALT+'6');
else if(Throttle[APALT]) ActKey(KEYON+PULSE+L_ALT+'4');
else ActKey(KEYON+PULSE+L_ALT+'2'); //  if it's not the 2 others, it can only be APAH
}


int EventHandle int type, alias o, int x)
{
DefaultMapping(&o, x);
}
```

Rajoutons encore une variable à l'opération logique, la position du toggle switch RDR ALTM.
En fonction de sa position, le PA utilisera l'altitude barométrique ou la sonde radar. Pour ce faire, nul besoin de créer une nouvelle fonction, nous allons simplement tenir compte de se paramètre dans notre fonction ap_ON_output.

```
int ap_ON_output()
{
if Throttle[APPAT]) ActKey(KEYON+PULSE+L_ALT+'6');
else if(Throttle[APALT] & Throttle[RDRDIS]) ActKey(KEYON+PULSE+L_ALT+'4');
else if (Throttle[APALT] & Throttle[RDRNRM]) ActKey(KEYON+PULSE+L_ALT+'5');
else ActKey(KEYON+PULSE+L_ALT+'2');
 }
```

Cela parait un peu barbare mais si l'on utilise les macros, la lecture du filtre devient bien plus confortable :

```
int ap_ON_output()
{
if(Throttle[APPAT]) ActKey(KEYON+PULSE+Autopilot_Route_following);
else if(Throttle[APALT] & Throttle[RDRDIS])
ActKey(KEYON+PULSE+Autopilot_Barometric_Altitude_Hold);
else if (Throttle[APALT] & Throttle[RDRNRM])
ActKey(KEYON+PULSE+Autopilot_Radar_Altitude_Hold);
else ActKey(KEYON+PULSE+Autopilot_Altitude_And_Roll_Hold);
}
```

Nous en avons finis avec le pilote auto de Flaming Cliffs 2. Nous sommes maintenant en mesure de le manipuler de manière réaliste, ce qui n'était pas le cas avec une configuration simple n'utilisant pas de logique.

Allons un peu plus loin dans le code, avec un exemple un peu plus compliqué.

Nous disposons the 8 modes de gestion du vol d'un appareil. Nous voulons les sélectionner à l'aide de 2 boutons, l'un permet de passer au mode suivant, le second de retourner au mode précédent.

En utilisant une simple SEQuence, il est possible de tricher et d'arriver a quelque chose d'approchant ce mode de fonctionnement. En fait pour arriver à ce que nous voulons faire, il faudrait créer une séquence ou nous choisissons le sens d'incrémentation.

Pour cela, nous allons créer une fonction baptisé « listmode » qui contient une séquence ainsi qu'une fonction index. Nous utiliseront l'index pour « naviguer » au sein de la fonction.

Voiçi le script, nous détailleront plus loin son contenu.

```
include "target.tmh"
include "FC2_MIG_29C_Macros.ttm"

int listmode, index;  //master modes list function & index

int main()
{
   if(Init(&EventHandle)) return 1;

 listmode = SEQ(_8__Gunsight_Reticle_Switch,
            _7__Air_To_Ground_Mode,
           _6__Longitudinal_Missile_Aiming_Mode,
                    _5__Close_Air_Combat_HMD_Helmet_Mode,
           _4__Close_Air_Combat_Bore_Mode,
            _3__Close_Air_Combat_Vertical_Scan_Mode,
           _2__Beyond_Visual_Range_Mode,
           _1__Navigation_Modes);
```

```
//China Hat is used for Modes selection----------
MapKey(&Throttle, CHF, EXEC("index = (index+1)%8; ActKey(KEYON+PULSE+X(listmode,
index));")); // forward
MapKey(&Throttle, CHB EXEC("index = (index+7)%8; ActKey(KEYON+PULSE+X(listmode,
index));")); // 7 is 8-1 = backward
}

int EventHandle int type, alias o, int x)
{
   DefaultMapping(&o, x);
}
```

Regardons en détail le contenu de l'EXEC

Pour CHF

`index = (index+1)%8;` ici, nous définissons le mouvement de l'index. Nous avons 8 modes la SEQuence, ce que nous indiquons a l'index avec %8. Comme nous souhaitons passer au mode suivant, nous indiquons a l'index de se décaler d'un pas la séquence.

Nous avons donc sélectionné l'événement suivant de la séquence, il ne reste plus qu'à l'activer.

`ActKey(KEYON+PULSE+X(listmode, index));` sert donc a activer le contenu de l'index.

L'emploi de la fonction X(list,index) est compatible avec les séquences, les chaines et les LIST.

Pour CHB

On reprend les mêmes éléments, avec une petite subtilité. Ici, nous voulons que notre index se décale vers le mode précédent. Il semblerait logique de se contenter de lui dire index-1, mais que ce passera t'il quand l'index arrivera a la première macro de la séquence ? Et bien ca ne fonctionnera pas normalement. En fait notre séquence sait boucler, mais uniquement par la fin. Hors, si on utilise -1, a un moment, l'index aura une valeur zéro ce qui risque d'avoir des conséquences inattendues.

Comme nous savons que notre séquence boucle à la fin, nous allons, pour aller au mode précédent, faire un tour complet (soit 8 valeurs) – 1=7. Voila donc l'explication du (index+7)%8.