#Etude de l'interface IVRDriverInput

Cette interface permet de créer et mettre à jour des composants d'entrée (boutons, joysticks, ect...), décrite ici https://github.com/ValveSoftware/openvr/wiki/IVRDriverInput-Overview

Chaque composant disponible (genre bouton, gachette, D-pad...) est décrit par un chemin, ex:

`/input/trigger/value` pour une gachette (valeur analogique/continue) `/input/a/click` pour le bouton a d'une manette type Xbox par exemple (valeur binaire)

le composant [ou chemin] `/input/system/click` est spécial et correspond à la touche pour faire apparaître l'overlay steam en VR

un driver doit disposer d'un fichier de profil d'entrée décrit ici -> https://github.com/ValveSoftware/openvr/wiki/Input-Profiles Ce fichier au format .json [format assez courant dans OVR], donne le type d'appareil d'entrée utilisé (manette, controlleur vive... [on peut aussi mettre un nom customisé]) ainsi que d'autres informations [voir le fichier InputProf.md pour détails]

COMMANDES DE L'INTERFACE: FONCTIONS DE CREATION DE COMPOSANT:
//_____

```
EVRInputError CreateBooleanComponent( PropertyContainerHandle_t ulContainer, const char *pchName, VRInputComponentHandle_t *pHandle )
```

^ `EVRInputError` est le type de retour, ne l'oubliez pas!!! cela veut dire que si la fonction s'execute correctement, la fonction retournera VRInputError_None et

Un composant boolean décrit par cette fonction est par exemple un BOUTON (contrôle binaire);

arguments : `ulContainer` -> la propriété handle de l'appareil parent de ce composant (ce sera celle de la manette à laquelle le bouton est raccroché par exemple) `pchName` -> le NOM du composant, il doit commencer par la forme "/input//", des exemples de noms ont déjà été donnés plus haut `pHandle` -> POINTEUR vers la valeur "handle" à mettre à jour avec le handle du nouveau composant [~flou~]
//_____

```
EVRInputError CreateScalarComponent( PropertyContainerHandle_t ulContainer, const char *pchName, VRInputComponentHandle_t *pHandle, EVRScalarType eType, EVRScalarUnits eUnits )
```

Même chose que CreateBooleanComponent mais scalaire, autrement dit continu ou analogique, exemple: une GACHETTE.

arguments: `ulContainer` -> déjà vu `pchName` -> déjà vu `pHandle` -> déjà vu `eType` -> peut avoir deux valeurs: VRScalarType_Absolute [gachettes, joysticks] ou VRScalarType_Relative [souris, trackballs], cela dépend du comportement de la valeur `eUnits` -> donne l'unité utilisé: VRScalarUnits_NormalizedOneSided pour 0:1 [potentiomètre] OU VRScalarUnits_NormalizedTwoSided pour -1:1 [gyroscope?? enfin peut aller dans le negatif quoi]

//_____

_____

```
EVRInputError CreateHapticComponent( PropertyContainerHandle_t ulContainer, const char *pchName, VRInputComponentHandle_t *pHandle )
```

ça va être tout ce qui est retour haptique (vibrations surtout), rien de nouveau au niveau des arguments. On peut pas update celui-là, c'est automatique

//_____

_____

```
EVRInputError UpdateBooleanComponent( VRInputComponentHandle_t ulComponent, bool bNewValue, double fTimeOffset )
```

ça, c'est pour mettre à jour la valeur de notre composant booléen (bouton)! on va l'appeler dès que l'utilisateur appuie sur un bouton

arguments : `ulComponent` -> le handle du composant à mettre à jour `bNewValue` -> la nouvelle valeur booleenne du composant (0||1); `fTimeOffset` -> le temps depuis que l'utilisateur a pressé ce bouton, négatifs = passé positifs = futur, sert à jauger la latence du hardware [ignorable??]

//_____

_____

```
EVRInputError UpdateScalarComponent( VRInputComponentHandle_t ulComponent, float fNewValue, double fTimeOffset )
```

idem pour les scalaires/continus/analogiques, pas besoin de détails ici...

//_____

_____