

La gestion des inputs avec DirectX : DirectInput

par Cyril Doillon ([Page Perso](#))

Date de publication : 15/05/2008

Dernière mise à jour : 15/08/2008

DirectInput est un des modules indépendants de DirectX qui permet de simplifier l'utilisation des périphériques d'entrée et de sortie comme le clavier, la souris et autres joy pads. Ce tutoriel présentera comment bien débiter avec cette API qui peut être utilisée de façon complètement indépendante des autres modules de DirectX (DirectDraw, Direct3D, DirectAudio).

Introduction.....	3
II - Avant de commencer.....	4
II-A - Introduction à DirectInput.....	4
II-B - Installation.....	4
II-C - Création du projet.....	4
II-D - L'objet DirectInput.....	5
III - Le Clavier.....	6
III-A - L'objet DirectInput du périphérique.....	6
III-B - Configuration du mode d'acquisition.....	6
III-C - Etats du périphérique.....	8
III-D - Finaliser DirectInput.....	9
IV - La Souris.....	10
IV-A - L'objet DirectInput de la souris.....	10
IV-B - Configuration du format de données.....	10
IV-C - Etats du périphérique.....	10
V - Le Joypad.....	12
V-A - Enumération des joy pads et création de l'objet DirectInput pour un joy pad.....	12
V-B - Configuration du format de données.....	13
V-C - Etats du périphérique.....	14
VI - Pour aller plus loin.....	16
Remerciements.....	17

Introduction

Quand on parle de DirectX, les pensées collectives vont sur les performances graphiques, notamment en terme de 3D, de l'API de Microsoft. Mais DirectX n'est pas que cela... DirectX est composé de plusieurs modules qui peuvent être utilisés indépendamment : Direct3D, DirectDraw, DirectSound, DirectPlay, etc. et DirectInput. DirectInput est le module qui permet de créer une interface directe entre l'application que l'on est en train de créer et les drivers d'entrées / sorties de l'ordinateur. Les entrées supportées sont le clavier, la souris, les joypads ... et les sorties sont, par exemple, les retours de force.

L'indépendance des modules de DirectX permet d'utiliser DirectInput avec tout autre type de bibliothèque. Il est possible, par exemple, d'utiliser DirectInput dans une application qui utilise OpenGL comme interface graphique et FMOD comme moteur de son.

II - Avant de commencer...

II-A - Introduction à DirectInput

DirectInput est un module de DirectX qui permet à l'application créée de s'interfacer directement avec les drivers des périphériques d'entrée et de sortie de l'ordinateur. Il permet donc de connaître l'état de chaque bouton des périphériques à scruter. DirectInput n'est pas adapté à tous les types d'applications, car il omet tous les comportements ainsi que toutes les configurations de ces périphériques par Windows. Par exemple, le comportement de "frappe de texte" est complètement ignoré. Ainsi, les touches "Shift" et "Maj" sont considérées comme toutes les autres et les touches des lettres ne permettent pas de différencier majuscules et minuscules. DirectInput s'applique donc aux applications qui utilisent directement l'état des touches comme par exemple des jeux vidéo, des applications 3D utilisant Direct3D ou OpenGL, ...

La procédure d'utilisation de DirectInput peut suivre ce type de schéma :

- 1 Créer une instance de l'objet DirectInput
- 2 Enumérer les périphériques connectés (optionnel si on utilise la souris et le clavier dont les GUID sont connus)
- 3 Créer une instance de l'objet DirectInputDevice pour chaque périphérique à utiliser.
- 4 Configurer le mode d'acquisition de chaque périphérique
- 5 Démarrer l'acquisition de chacun des périphériques
- 6 Récupérer les états de chaque périphérique (boutons, axes ...)
- 7 Utiliser les nouveaux états récupérés
- 8 Stopper l'acquisition de chaque périphérique
- 9 Stopper l'utilisation de l'objet DirectInput

II-B - Installation


Pour utiliser DirectInput, ou DirectX en général, vous avez besoin d'un éditeur C++ comme Visual C++ Express Edition mais également du SDK de DirectX. Le dernier en date est celui de mars 2008, mais il en sort régulièrement tous les deux ou trois mois. Le SDK contient tous les fichiers (.lib, .h, etc.), exemples et documentations pour développer avec DirectX. Le SDK contient tous les éléments nécessaires au développement d'application utilisant DirectX dont les prototypes de fonctions et les libraires se trouvent respectivement (par défaut) dans les répertoires "**C:\Program Files Microsoft DirectX SDK (March 2008)\Include**" et "**C:\Program Files Microsoft DirectX SDK (March 2008)\Lib\x86**".

-  [Téléchargement de Visual C++ 2008 Express Edition](#)
-  [Téléchargement de DirectX SDK \(Mars 2008\)](#)

II-C - Création du projet

DirectInput est une bibliothèque complètement indépendante des autres bibliothèques de DirectX. A la création du projet, il est nécessaire d'ajouter les bibliothèques **dinput8.lib** et **dxguid.lib** comme bibliothèques additionnelles au linker.

Dans nos fichiers sources, il est nécessaire d'ajouter le header **dinput.h** pour l'utilisation de DirectInput. Si nous nous contentons de cet ajout, le compilateur va nous générer un warning qui précise qu'il va utiliser la version par défaut de DirectInput. La version par défaut est généralement la dernière.

 Pour rendre notre code compatible avec les futures versions de DirectInput, il est nécessaire de préciser la version à utiliser en définissant la macro **DIRECTINPUT_VERSION**. Ainsi, lors de la mise à jour de DirectX, il ne sera pas nécessaire de mettre à jour son code source avec les nouvelles versions.

Inclusion de la bibliothèque

```

// Utilisation de la version 8 de DirectInput
#define DIRECTINPUT_VERSION 0x0800
#include <dinput.h>
    
```

II-D - L'objet DirectInput

La première chose à faire pour utiliser DirectInput est de créer un pointeur sur une instance de l'objet DirectInput. Cette instance va nous permettre d'accéder à tous les périphériques connectés à l'ordinateur client de l'application. La fonction **DirectInput8Create** nous permet de créer cette instance et d'assigner notre pointeur. Elle prend comme paramètres :

- L'identificateur de notre application
- La version de DirectInput à utiliser
- L'identificateur unique de l'interface de l'objet à créer
- L'adresse de notre pointeur resultat
- Paramètre optionnel à utiliser en cas d'agrégation COM de l'interface IUnknown

La fonction retourne un résultat de type *DI_OK* en cas de réussite. Sinon, elle peut également retourner une erreur de type *DIERR_BETADIRECTINPUTVERSION*, *DIERR_INVALIDPARAM*, *DIERR_OLDDIRECTINPUTVERSION* ou *DIERR_OUTOFMEMORY* en cas d'échec.

Création de l'objet DirectInput

```

// Instance Windows de l'application
HINSTANCE instance;
//...création de l'instance...

// Valeur de retour des méthodes utilisées, permet la gestion des erreurs
HRESULT result;

// Pointeur sur l'objet DirectInput utilisé
LPDIRECTINPUT8 direct_input_object;

// Création de l'objet DirectInput
result = DirectInput8Create(
    instance,
    DIRECTINPUT_VERSION,
    IID_IDirectInput8,
    (void*)&direct_input_object,
    NULL
);


// Gestion des erreurs
if( FAILED( result ) )
{
    // Impossible de créer une instance de DirectInput
    // Erreurs possibles : DIERR_BETADIRECTINPUTVERSION, DIERR_INVALIDPARAM, DIERR_OLDDIRECTINPUTVERSION, DIERR_OUTO
    return 1;
}
    
```

III - Le Clavier

III-A - L'objet DirectInput du périphérique

Pour récupérer les données des états des touches du clavier, il est nécessaire de créer un objet pour ce périphérique. Cette instance se crée à partir de la méthode **CreateDevice** de l'objet DirectInput, créé au préalable. Cette méthode prend comme paramètres :

- Le GUID (**G**lobally **U**nique **I**Dentifier) du périphérique, son identificateur unique
- L'adresse de notre pointeur résultat
- Adresse optionnelle à utiliser en cas d'agrégation COM de l'interface IUnknown

 Le paramètre GUID du périphérique s'acquière soit par l'utilisation de la méthode **EnumDevices** de l'objet DirectInput, soit par les GUID prédéfinis. La méthode **EnumDevices** permet de récupérer tous les périphériques connectés. Les constantes prédéfinies **GUID_SysKeyboard** ou **GUID_SysMouse** correspondent respectivement au clavier et à la souris de l'ordinateur. La constante **GUID_Joystick** ne peut pas être utilisée, car elle ne correspond pas à une instance unique.

L'appel à cette méthode retourne un résultat de type **DI_OK** en cas de réussite. En cas d'échec, la méthode peut générer une erreur de type **DIERR_DEVICENOTREG**, **DIERR_INVALIDPARAM**, **DIERR_NOINTERFACE**, **DIERR_NOTINITIALIZED** ou **DIERR_OUTOFMEMORY**

Création de l'objet DirectInput pour le clavier


```
// Pointeur sur l'objet DirectInput du clavier
LPDIRECTINPUTDEVICE8 keyboard;

// Creation du périphérique
result = direct_input_object->CreateDevice(
    GUID_SysKeyboard,
    &keyboard,
    NULL
);

// Gestion des erreurs
if( FAILED( result ) )
{
    // Impossible de créer l'objet pour le clavier
    // Erreurs possibles : DIERR_DEVICENOTREG, DIERR_INVALIDPARAM, DIERR_NOINTERFACE, DIERR_NOTINITIALIZED, DIERR_OUTOFMEMORY
    Finalize();
    return 2;
}
```

III-B - Configuration du mode d'acquisition

Avant de commencer l'acquisition des états des touches du clavier, du périphérique en général, il est nécessaire de définir la façon dont ces données vont être récupérées. La première chose à faire est de définir le type des données à récupérer. Pour cela, on utilise la méthode **SetDataFormat** qui prend comme paramètre un pointeur sur une structure de type **DIDATAFORMAT**. Ce paramètre décrit le format de données du périphérique.

 Il existe cinq variables globales prédéfinies pour décrire des formats de données : **c_dfDIKeyboard**, **c_dfDIMouse**, **c_dfDIMouse2**, **c_dfDIJoystick**, **c_dfDIJoystick2**.

De la même manière que précédemment, la méthode retourne la valeur *DI_OK* en cas de réussite. Sinon, en cas d'échec, elle génère une erreur de type *DIERR_ACQUIRED*, *DIERR_INVALIDPARAM*, *DIERR_NOTINITIALIZED*.

Définition du format des données récupérées

```
// Définition du format de données utilisé
result = keyboard->SetDataFormat(
    &c_dfDIKeyboard
);

// Gestion des erreurs
if( FAILED( result ) )
{
    // Impossible d'initialiser le format de données
    // Erreurs possibles : DIERR_ACQUIRED, DIERR_INVALIDPARAM, DIERR_NOTINITIALIZED
    Finalize();
    return 3;
}
```


La seconde chose à configurer, avant de commencer l'acquisition, est le mode de coopération de DirectInput avec la fenêtre de notre application pour notre périphérique. On utilise ainsi la méthode **SetCooperativeLevel** qui fait partie de l'interface du périphérique à acquérir. Cette méthode prend, comme paramètres, d'une part, l'identificateur de la fenêtre et, d'autre part, le flag utilisé pour configurer le mode de coopération.

Plusieurs flags sont définis et peuvent être utilisés par paire pour le second paramètre. Par défaut, le mode d'acquisition est **DISCL_NONEXCLUSIVE | DISCL_BACKGROUND**, soit une utilisation autorisant d'autres applications à utiliser le périphérique, et l'acquisition peut se faire même quand la fenêtre n'est plus active.

Flag	Description
DISCL_EXCLUSIVE (0x01)	Quand le périphérique est configuré en mode exclusif, aucune autre application ne peut l'acquérir. Toutefois, l'acquisition en mode non-exclusif est toujours permise même si une autre application a obtenu l'accès exclusif.
DISCL_NONEXCLUSIVE (0x02)	L'accès non-exclusif ne modifie pas l'acquisition de ce même périphérique par d'autres applications
DISCL_FOREGROUND (0x04)	Le périphérique perd l'acquisition dès que la fenêtre associée n'est plus la fenêtre active
DISCL_BACKGROUND (0x08)	Le périphérique peut être acquis même si la fenêtre associée de l'application n'est pas la fenêtre active

 *Le mode d'acquisition **DISCL_EXCLUSIVE | DISCL_BACKGROUND** n'est pas valide pour le clavier et pour la souris.*

Si la configuration se passe bien, la méthode retourne *DI_OK* comme résultat. Dans le cas contraire, elle peut générer une erreur de type *DIERR_INVALIDPARAM*, *DIERR_NOTINITIALIZED* ou *E_HANDLE*

 *Un cinquième flag est défini : **DISCL_NOWINKEY** (0x10). Il s'utilise en combinaison avec **DISCL_NONEXCLUSIVE** pour désactiver l'action de la touche "Windows" pour éviter une interruption inattendue de l'application. En mode exclusif, la touche "Windows" est toujours désactivée.*

Définition du mode de coopération

```
// Identificateur de la fenêtre de l'application
HWND window_handle;
//...récupération de l'identificateur de la fenêtre de l'application...

// Définition du mode de coopération
result = keyboard->SetCooperativeLevel(
    window_handle,
```

Définition du mode de coopération

```
DISCL_FOREGROUND | DISCL_NONEXCLUSIVE
);

// Gestion des erreurs
if( FAILED( result ) )
{
    // Impossible d'initialiser le mode de coopération
    // Erreurs possibles : DIERR_INVALIDPARAM, DIERR_NOTINITIALIZED, E_HANDLE
    Finalize();
    return 4;
}
```

III-C - Etats du périphérique

Maintenant, tout est prêt pour lancer l'acquisition du périphérique. La méthode **Acquire** de l'interface des objets de périphériques de DirectInput permet de signaler que l'application va commencer l'acquisition des états des touches du périphérique. Cette méthode ne prend aucun paramètre. Toute la configuration du périphérique a été faite au préalable par les méthodes **SetDataFormat** et **SetCooperationLevel**.

Si la méthode réussie, elle retourne la valeur *DI_OK*. En cas de problèmes, elle peut retourner une valeur parmi *DIERR_INVALIDPARAM*, *DIERR_NOTINITIALIZED* et *DIERR_OTHERAPPASPRIO*.

Début de l'acquisition

```
// Marque le début de l'acquisition des données du clavier
if( keyboard )
{
    keyboard->Acquire();
}
```

La dernière étape est de récupérer les informations d'état des touches du clavier. Cela peut se faire par la méthode **GetDeviceState** de l'objet périphérique correspondant au clavier. Cette méthode prend, comme paramètres :

- La taille du buffer de données à récupérer
- Le pointeur vers le buffer de données

Dans le cas du clavier, qui contient environ 110 touches, nous pouvons créer un buffer de 256 octets soit un tableau de 256 caractères. Ce tableau correspond au type de données *c_dfDIKeyboard* défini lors de la définition du type de données par la méthode **SetDataFormat**. L'appel de cette méthode retourne la valeur *DI_OK* si l'extraction des données s'est bien passée. En cas d'échec, elle peut retourner une erreur de type *DIERR_INPUTLOST*, *DIERR_INVALIDPARAM*, *DIERR_NOTACQUIRED*, *DIERR_NOTINITIALIZED* ou *E_PENDING*. Ceci peut être dû à une perte de connexion avec le périphérique donc il est possible de réagir en conséquence en essayant, par exemple, de s'y reconnecter par l'appel de la méthode **Acquire**.

Chaque élément de notre tableau correspond à une touche du clavier. L'index de chaque touche du clavier est défini par DirectInput et peut être accédé par des constantes prédéfinies commençant par le préfixe **DIK_**. Par exemple, la constante *DIK_A* correspond à la touche "A", *DIK_B* à la touche "B", ... L'état de chaque touche est stocké sur le premier bit de chaque élément du tableau donc par un masque avec la valeur *0x80*. Ce bit est à "1" si la touche est enfoncée, à "0" sinon. On crée donc une macro permettant un accès simple à l'état d'une touche qui récupère la valeur de la touche et qui exécute ce masque.

Liste des touches du clavier DirectInput

i Toute cette opération doit être exécutée perpétuellement tout au long de l'application. Dans le cas d'un jeu vidéo ou d'une application 3D, il faut mettre à jour le buffer à chaque frame.

Récupération et traitement des données

```
// Macro définissant l'état d'une touche
#define KEYDOWN( buffer, key ) ( buffer[key] & 0x80 )

// Buffer contenant l'état temporaire des touches
char buffer[256];

// Récupération de l'état de touches
result = keyboard->GetDeviceState(
    sizeof(buffer),
    (LPVOID)&buffer
);

// Gestion des erreurs
if( FAILED( result ) )
{
    // Impossible des récupérer l'état des touches, périphérique perdu
    // Erreurs possibles : DIERR_INPUTLOST, DIERR_INVALIDPARAM, DIERR_NOTACQUIRED, DIERR_NOTINITIALIZED, E_PENDING
    return 5;
}

// Utilisation de l'état des touches
if( KEYDOWN( buffer, DIK_RIGHT ) )
{
    // Action en cas d'appui sur la touche "Flèche droite"
}
```

III-D - Finaliser DirectInput

A l'arrêt de l'application ou lors de la fin de l'utilisation de DirectInput, il est nécessaire de signaler que notre application n'en a plus l'utilité. Il faut ainsi signaler la fin de l'utilisation du ou des périphériques utilisés mais également de l'utilisation de DirectInput. On commence ainsi par libérer le périphérique "Clavier" par l'appel successif des méthodes **Unacquire** et **Release** qui, respectivement, arrête la récupération des données et libère le périphérique. On réinitialise également le pointeur de l'objet pour éviter tout nouvel appel. On finit par la libération de DirectInput par la méthode **Release** de cet objet et par la réinitialisation de son pointeur. On encapsule cette opération dans une méthode *Finalize* qui sera appelé à la fin de l'application et lors de chaque arrêt non-souhaité.

Finalisation de DirectInput

```
// Arrête l'utilisation de DirectInput sur un périphérique
void Finalize( void )
{
    if ( direct_input_object )
    {
        // Finalisation du périphérique
        if( keyboard )
        {
            keyboard->Unacquire();
            keyboard->Release();
            keyboard = NULL;
        }

        // Finalisation de DirectInput
        direct_input_object->Release();
        direct_input_object = NULL;
    }
}
```

IV - La Souris

IV-A - L'objet DirectInput de la souris

L'acquisition de l'état de la souris va se faire selon le même schéma que pour le clavier. On va donc commencer par créer ou utiliser l'objet DirectInput vu précédemment. Il faut donc commencer par créer l'objet du périphérique de la souris qui est, comme le clavier, de type `LPDIRECTINPUTDEVICE8`. Grâce à la méthode **CreateDevice** de l'objet DirectInput, on crée ce périphérique, en lui passant comme paramètre, le GUID prédéfini **GUID_SysMouse**

Création de l'objet DirectInput pour la souris

```
// Pointeur sur l'objet DirectInput de la souris
LPDIRECTINPUTDEVICE8 mouse;

// Création du périphérique
result = direct_input_object->CreateDevice(
    GUID_SysMouse,
    &mouse,
    NULL
);

// Gestion des erreurs
if( FAILED( result ) )
{
    // Impossible de créer l'objet pour le souris
    // Erreurs possibles : DIERR_DEVICENOTREG, DIERR_INVALIDPARAM, DIERR_NOINTERFACE, DIERR_NOTINITIALIZED, DIERR_OUTOFMEMORY
    Finalize();
    return 2;
}
```

IV-B - Configuration du format de données

Dans l'étape de configuration du mode d'acquisition de l'état du périphérique, la seule différence viens du type de données à récupérer. Pour cela, nous utilisons la constante prédéfinie `c_dfDIMouse` que l'on passe comme paramètre à la méthode **SetDataFormat** de l'objet du périphérique. Celle-ci définit le format défini par DirectInput pour le format d'état de la souris, de la même manière que le format de données du clavier était un tableau de 256 octets. On peut donc continuer par la configuration du niveau de coopération de la même manière que pour le clavier grâce à la méthode **SetCooperativeLevel**.

Définition du format des données récupérées

```
// Définition du format de données utilisé
result = mouse->SetDataFormat(
    &c_dfDIMouse
);

// Gestion des erreurs
if( FAILED( result ) )
{
    // Impossible d'initialiser le format de données
    // Erreurs possibles : DIERR_ACQUIRED, DIERR_INVALIDPARAM, DIERR_NOTINITIALIZED
    Finalize();
    return 3;
}
```

IV-C - Etats du périphérique

L'acquisition se fait de la même manière que pour le clavier. La différence principale vient du format des données récupérées. La structure **DIMOUSESTATE** prédéfinie par DirecX permet de décrire le format de données que va retourner DirectInput. Elle est composé des trois axes et de quatre boutons. Les deux premiers axes correspondent

aux déplacements en X et en Y de la souris, le troisième axe aux rotations de la molette. Les quatre boutons sont quatre octets définis de la même manière que les touches du clavier.

Format de données de la souris

```
typedef struct DIMOUSESTATE {  
    LONG lX;    // Axe X de la souris  
    LONG lY;    // Axe Y de la souris  
    LONG lZ;    // Axe Z de la souris, la molette  
    BYTE rgbButtons[4]; // Etats de 4 boutons de la souris  
} DIMOUSESTATE, *LPDIMOUSESTATE;
```

Après avoir signalé à DirectInput le début de l'acquisition par la méthode **Acquire**, on peut stocker, à chaque itération, l'état de la souris dans une structure de type *DIMOUSESTATE*. De la même manière que pour le clavier, on utilise la méthode **GetDeviceState** pour récupérer le déplacement et l'état des boutons de la souris. L'utilisation de l'état récupéré se fait de la même manière que pour le clavier en ce qui concerne les boutons. Quant aux valeurs X, Y et Z, elles indiquent les déplacements respectifs horizontal, vertical et de la molette depuis la dernière acquisition.

Récupération et traitement des données

```
// Etat de la souris  
DIMOUSESTATE state;  
  
// Récupération de l'état de la souris  
result = mouse->GetDeviceState(  
    sizeof(state),  
    (LPVOID)&state  
);  
  
// Gestion des erreurs  
if( FAILED( result ) )  
{  
    // Impossible des récupérer l'état des touches, périphérique perdu  
    // Erreurs possibles : DIERR_INPUTLOST, DIERR_INVALIDPARAM, DIERR_NOTACQUIRED, DIERR_NOTINITIALIZED, E_PENDING  
    return 5;  
}  
  
// Utilisation de l'état des boutons  
if( KEYDOWN( state.rgbButtons, 0 ) )  
{  
    // Action en cas d'appui sur le premier bouton  
}  
  
// Utilisation des axes  
if( state.lX > 0 )  
{  
    // Action en cas de déplacement de la souris vers la droite  
}
```

V - Le Joypad

V-A - Enumération des joy pads et création de l'objet DirectInput pour un joy pad

Le méthodologie d'acquisition des contrôleurs de jeu (joypad) est un peu différente, car nous ne connaissons pas leurs GUIDs. Le principe est assez simple car nous allons utiliser la méthode **EnumDevices** de l'objet DirectInput qui, comme son nom l'indique, va énumérer l'ensemble des périphériques connectés à l'ordinateur.

Cette méthode prend comme paramètre une fonction définie par l'utilisateur qui va permettre de traiter chacun des périphériques. Cette "callback" prend comme paramètres, l'instance, de type **DIDeviceInstance**, du périphérique, ainsi qu'un pointeur que nous n'allons pas utiliser. Le premier paramètre va être passé par la méthode d'énumération et va contenir de GUID du joy pad correspondant. On peut ainsi le récupérer pour créer l'objet du joy pad grâce à la méthode **CreateDevice** que nous avons déjà utilisée.

Pour signaler ce que l'on veut faire après l'énumération de chaque périphérique, il existe deux valeurs distinctes de retour que l'on peut utiliser. Ces valeurs sont *DIENUM_CONTINUE* et *DIENUM_STOP* qui vont signaler, respectivement, à la méthode d'énumération qu'il faut continuer ou s'arrêter. Dans notre cas, nous allons nous arrêter à la première création de périphérique qui a réussi. En cas d'échec, nous continuons sur le périphérique suivant (s'il y en a ...).

Callback de création de l'objet DirectInput pour le joy pad

```
// Pointeur sur l'objet DirectInput du joy pad
LPDIRECTINPUTDEVICE8 joypad;

// Callback d'énumération
BOOL CALLBACK CreateDeviceCallback(
    LPCDIDeviceInstance instance,
    LPVOID reference )
{
    HRESULT result;

    // Création du périphérique
    result = direct_input_object->CreateDevice( instance->guidInstance, &joypad, NULL);

    // Gestion des erreurs
    if(FAILED( result ))
    {
        // En cas d'échec ... on continue l'énumération
        return DIENUM_CONTINUE;
    }

    // En cas de réussite ... on arrête l'énumération
    return DIENUM_STOP;
}
```

Maintenant que notre callback est créée, nous pouvons l'utiliser dans la méthode d'énumération des périphériques de DirectInput. Cette méthode, **EnumDevice**, permet à DirectInput de parcourir l'ensemble des périphériques, connecté ou non, de l'ordinateur. Elle prend comme paramètres :

- Un filtre sur le type de périphérique à énumérer qui peut être choisi parmi les valeurs suivantes :

Valeurs	Description
DI8DEVCLASS_ALL	Tous les périphériques
DI8DEVCLASS_DEVICE	Tous les périphériques non inclus dans les autres classes
DI8DEVCLASS_GAMECTRL	Tous les contrôleurs de jeu
DI8DEVCLASS_KEYBOARD	Tous les claviers de type <i>DI8DEVTYPE_KEYBOARD</i>
DI8DEVCLASS_POINTER	Tous les périphériques de type <i>DI8DEVTYPE_MOUSE</i> et <i>DI8DEVTYPE_SCREENPOINTER</i> .

- La callback d'énumération appelée à chaque périphérique
- Une valeur référence de l'utilisateur qui va être passée comme deuxième paramètre de notre callback (peut être utilisée pour compter le nombre de périphériques par exemple)
- Un flag définissant le champ de l'énumération qui peut être composé des valeurs :

Flag	Description
DIEDFL_ALLDEVICES (0x00000000)	Tous les périphériques installés (valeur par défaut)
DIEDFL_ATTACHEDONLY (0x00000001)	Tous les périphériques installés et branchés
DIEDFL_FORCEFEEDBACK (0x00000100)	Tous les périphériques avec retour de force
DIEDFL_INCLUDEALIASSES (0x00010000)	Tous les périphériques qui sont des alias pour d'autres périphériques
DIEDFL_INCLUDEHIDDEN (0x00020000)	Tous les périphériques cachés
DIEDFL_INCLUDEPHANTOMS (0x00040000)	Tous les périphériques virtuels

Enumération des contrôleurs de jeux connectés

```
//Enumérer tous les périphériques
result = direct_input_object->EnumDevices(
    DI8DEVCLASS_GAMECTRL,
    &CreateDeviceCallback,
    NULL,
    DIEDFL_ATTACHEDONLY
);

// Gestion des erreurs
if( FAILED( result ) )
{
    // Impossible de créer l'objet pour le joypad
    // Erreurs possibles : DIERR_INVALIDPARAM, DIERR_NOTINITIALIZED
    Finalize();
    return 2;
}
```

V-B - Configuration du format de données

La configuration du format de données va se faire de la même manière que pour la souris ou le clavier. Cependant, le paramètre de type offre deux choix : *c_dfDIJoystick* et *c_dfDIJoystick2*. La seconde valeur correspond à un format de données supportant plus de composantes sur les joy pads. On trouvera ainsi des informations sur les vitesses ou accélérations des axes. Le paramètre choisi sera ainsi passé à la méthode **SetDataFormat** de l'objet créé pour le périphérique. Nous pouvons ainsi continuer par la configuration du niveau de coopération fait de la même manière que pour le clavier.

Définition du format des données récupérées

```
// Définition du format de données utilisé
```

Définition du format des données récupérées

```

result = joypad->SetDataFormat(
    &c_dfDIJoystick2
);

// Gestion des erreurs
if( FAILED( result ) )
{
    // Impossible d'initialiser le format de données
    // Erreurs possibles : DIERR_ACQUIRED, DIERR_INVALIDPARAM, DIERR_NOTINITIALIZED
    Finalize();
    return 3;
}
    
```

V-C - Etats du périphérique

Nous allons supposer que nous utilisons la deuxième structure de données plus complète pour acquérir un joypad. Le paramètre *c_dfDIJoystick2* correspond donc à une structure de type **DIJOYSTATE2**. Cette structure est composée des valeurs des 3 axes principaux, des rotations de ces axes, d'un contrôleur de "point de vue" et d'éventuellement 128 boutons. On peut également ajouter deux axes supplémentaires. On trouve enfin des valeurs de vitesse, accélération, force et couple de chacun des axes principaux.

Format de données du joypad

```

typedef struct DIJOYSTATE2 {
    LONG lX; // Axe X, généralement Gauche-Droite du Stick
    LONG lY; // Axe Y, généralement Haut-Bas du Stick
    LONG lZ; // Axe Z, troisième axe, les gaz pour certains joysticks
    LONG lRx; // Rotation de l'axe X
    LONG lRy; // Rotation de l'axe Y
    LONG lRz; // Rotation de l'axe Z
    LONG rgfSlider[2]; // Deux axes supplémentaires
    DWORD rgdwPOV[4]; // Contrôleur de point de vue
    BYTE rgbButtons[128]; // Etat des boutons
    LONG lVX; // Vitesse de l'axe X
    LONG lVY; // Vitesse de l'axe Y
    LONG lVZ; // Vitesse de l'axe Z
    LONG lVRx; // Vitesse de rotation de l'axe X
    LONG lVRy; // Vitesse de rotation de l'axe Y
    LONG lVRz; // Vitesse de rotation de l'axe Z
    LONG rgfVSlider[2]; // Vitesse des axes supplémentaires
    LONG lAX; // Accélération de l'axe X
    LONG lAY; // Accélération de l'axe Y
    LONG lAZ; // Accélération de l'axe Z
    LONG lARx; // Accélération de la rotation l'axe X
    LONG lARy; // Accélération de la rotation l'axe Y
    LONG lARz; // Accélération de la rotation l'axe Z
    LONG rgfASlider[2]; // Accélération des axes supplémentaires
    LONG lFX; // Force de l'axe X
    LONG lFY; // Force de l'axe Y
    LONG lFZ; // Force de l'axe Z
    LONG lFRx; // Couple de l'axe X
    LONG lFRy; // Couple de l'axe Y
    LONG lFRz; // Couple de l'axe Z
    LONG rgfFSlider[2]; // Force des axes supplémentaires
} DIJOYSTATE2, *LPDIJOYSTATE2;
    
```

Après avoir signalé le début de l'acquisition grâce à la méthode **Acquire** comme défini précédemment, nous pouvons commencer la récupération successive des états du périphérique. Tout va se passer de la même manière que pour les périphériques *clavier* et *souris*. La seule différence va venir dans le type de données extraite qui sera du type vu précédemment.

Récupération et traitement des données

```


// Etat du joypad
DIJOYSTATE2 state;
    
```

Récupération et traitement des données

```
// Récupération de l'état du joypad
result = joypad->GetDeviceState(
    sizeof(state),
    (LPVOID)&state
);

// Gestion des erreurs
if( FAILED( result ) )
{
    // Impossible de récupérer l'état des touches, périphérique perdu
    // Erreurs possibles : DIERR_INPUTLOST, DIERR_INVALIDPARAM, DIERR_NOTACQUIRED, DIERR_NOTINITIALIZED, E_PENDING
    return 5;
}
```

VI - Pour aller plus loin...

Ce tutoriel a présenté les bases et les premières choses pour acquérir des périphériques d'entrée pour votre application. DirectInput a de nombreuses autres possibilités qui seront abordées, probablement, dans un autre tutoriel. Pour continuer votre apprentissage de DirectInput, vous pouvez vous rendre sur la documentation officielle de DirectX ( **MSDN DirectX : DirectInput**) ou dans la documentation fournie dans le SDK DirectX. DirectInput propose notamment des fonctionnalités de bufferisation des états successifs des touches, de l'utilisation des retours de force, ...

Pour toutes questions, n'hésitez pas à demander directement sur le forum DirectX

Remerciements

Je tiens particulièrement à remercier **Loka** et **Melem** pour leur relecture et leur soutien.