Martin Burckhardt

# Un très court dll tutorial

traduction française: Pierre Gatoux

## Introduction

Le WDL  est un langage simple et sympa (tout comme le C).
Le C++, si vous vous y êtes faits – est un langage vraiment sympa. Le SDK, toujours, si vous vous y êtes faits, est un outil bien sympa.

Dans ce cas, il devrait être facile d'écrire une *dll. Pourtant ce n'est pas le cas. Pourquoi ?

Simplement parce qu'il est compliqué de toujours penser en termes d'interface. Imaginez que parler consisterait en la traduction simultanée :

Français ->  Anglais -> Français

Cela semble facile si vous avez un diagramme mais pensez aux mots qui ne peuvent être traduits comme « shallow = peu profond »
Cela se révèle très fastidieux et c'est encore pire lorsqu'on réalise que tout le travail n'est pas effectué qu'une seule fois mais nécessite un effort triple.

La meilleure façon d'aborder l'écriture de dlls est de tirer un trait sur le WDL. Alors faites comme le conquistador Cortez qui decida de brûler ses navires lorsqu'il accosta la terre des Aztèques.

## I. Comment accéder à une *dll ?

Qu'est-ce qu'une dll ? C'est un programme supplémentaire, par conséquent il doit y avoir une référence qui pointe vers elle. C'est ce qui est fait dans le WDL. Tout d'abord, vous avez besoin d'une clé (handle) qui doit être de type nosave, sinon la sauvegarde pourrait créer quelques problèmes lors du prochain chargement)

```
var_nsave  my_dll;
```

Ensuite vous devez ouvrir la dll (je ferais cela dans la boucle principale)

```
my_dll  =  dll_xpen(«MyDll.dll»);
```

Si vous ne faites tourner qu'une seule dll, c'est facile. Vous ouvrez la dll au début de la session et la gardez ouverte. S'il y a plus d'une dll, vous devez prendre garde à faire référence à la dll appropriée, ce qui est un peu plus délicat.

Bien, me direz-vous, j'ai le chauffeur, mais j'ai pas la voiture. Et si vous essayez de lancer votre projet de 3D Game Studio, vous aurez exactement la même réponse.

## 2. Comment écrire une dll ?

OK, constuisons la voiture. Gros avantage, je connais son nom : MyDll.dll.

Cela doit convenir au chauffeur, par conséquent, vous devez choisir la dll appropriée : si vous avez Visual C++, j'espère que c'est le cas., il y a deux types de *dll

- une qui est créée avec l'aide de MFC (qui n'est pas très sympa, désolé)
- une autre appelée simplement appelée « win32dynamicLinkLibrary »

C'est cette dernière qu'il nous faut. Créez un nouveau projet appelé « MyDll », copiez y tous vos fichiers SDK, ajoutez les fichiers *.cpp et créez une dll. Maintenant, vous devriez trouver un fichier « MyDll.dll » dans votre répertoire debug.

La seule chose que vous avez à faire est de la copier dans votre projet de 3D Game Studio.

Et que se passe-t'il ? 3D GameStudio ne se plaint plus qu'il ne trouve plus le fichier… c'est sympa.
Silence.

## 3. Comment aller d'ici à là.

Oui, c'est compliqué puisque nous allons commencer de la logique DoubleTalk et DoubleStandard.
J'essaierai donc de réduire la contribution d'un côté .

Réduisons donc la partie WDL au minimum. Mais qu'est-ce que le minimum ?

En fait vous vous rendrez compte que le minimum n'est autre que la syntaxe complète du WDL. Je vais vous montrer comment cela pourrait marcher mais je voudrais insister sur l'explication du mécanisme de communication WDL-dll.

Puisque Conitec a eu la gentillesse d'inclure quelques commandes, vous avez déjà quelques fonctions. Si vous voulez utiliser l'une d'entre elles dans une wdl, vous devez la déclarer. Vous devez donc écrire quelque chose comme :

```
dllfunction PainEntitiesRed(void_var);
```

Vous pourrez ensuite l'exécuter dans le wdl, en écrivant simplement :

```
PaintEntitiesRed(0);
```

Mais que se passe- t'il lorsque votre wdl exécute cette commande ?

1. Il accède à la dll
2. Il cherche la fonction « PaintEntitiesRed(0) »

En assumant que j'ai une banale fonction C++ du même nom, le moteur de 3D GameStudio ne la trouvera pas
Pourquoi ?
Parce qu'il ne recherche que les fonctions du type DLLFUNC.

Si vous avez donc une fonction comme celle-ci :

```
DLLFUNC fixed PaintEntitiesRed(void)
```

Cela fonctionnera, autrement non.
DLLFUNC est donc le mot clé et toute communication entre wdl et dll doit emprunter cette voie.

Mais qu'est-ce que DLLFUNC ?
Cela ne fait pas du tout partie du C++, c'est le type de donnée interface que Johann C. Lotter a défini pour nous, tout comme le type fixed, qui mesure 32 bits et équivalent au type long.

Faisons un tour de l'autre côté. Assumons que nous avons une dll propre et nette, pouvons nous être autonomes ?
Non, nous aurons toujours besoin de commandes wdl et plus.

Nous avons besoins d'actions (C++ ne connaît pas le type « action »). En fait, C++ ne connaît rien du moteur de 3D GameStudio. Voici quelques exemples de fonctions inconnues de C++ :

```
A4_ENTITY;
A4_STRING;
A4_TEX;
A4_BITMAP;
A4_TEXT;
A4_STRING;
...
```

Jetez un oeil à a5dll.h, vous y trouverez tout ce qui manque.

## 4. Désolé, je ne suis pas encore là. Je cherche encore un point de départ, un pointeur.

Mais je suis têtu, je veux utiliser toutes les choses de WDL et qui sont faciles à manier. Ou, dit autrement, je veux l'équivalent de cette simple ligne :

```
dllfunction PaintEntitiesRed(0);
```

Il y a une solution toute prête pour cela. Ecrivez juste toutes les commandes et importez ce set dans la dll.

C'est ce que vous devriez faire lorsque vous démarrez votre programme, placez le à la fin de votre fonction main.
Ce qui devrait ressembler à cela :

Partie WDL

```
dllfunction function_import(void_var);
```

Partie dll

Supposons que nous voulions utiliser l'objet player.

Première chose que nous devrions savoir :

Nous ne pouvons *jamais* utiliser un objet dll tel quel, nous utilisons toujours l'adresse (appelé un pointeur dans la syntaxe C++)
Nous devons donc créer un tel pointeur et lui assigner l'adresse de l'objet WDL.
Voici comment j'ai déclaré un pointeur vers le joueur dans la dll. Pour des raisons de simplicité, je lui ai donné le même nom que dans le WDL.

```
A4_ENTITY  *player;
```

Voici donc le pointeur, qui peut recevoir l'adresse du 'vrai' objet player du WDL

```
DLLFUNC  fixed  function_import(void)
{
player          = (A4_ENTITY *)a5dll_getwdlobj("player");       //
}
```

Ce pointeur n'est en aucun cas une entité mineure, il hérite de tous les attributs que nous connaissons de la syntaxe WDL
Si vous tapez le signe du pointeur (->) ;
player ->
Votre compilateur montrera tous les attributs disponibles comme alpha, red, green, blue, etc…

Cette opération doit être effectuée pour tous les objets WDL, dans l'idéal, pour toute la syntaxe WDL
Il est donc recommandé de construire tous les pointeurs et de les importer ensuite à la fin de la fonction main.
Il est nécessaire de le faire une fois seulement, il y a deux raisons à cela :

· Si vous ré-exécutez le processus d'import, vous perdrez du temps. : « a5dll_getwdlobj » est une fonction qui prend du temps

- Si vous voulez créer une application où tout est effectué par la dll, vous devriez : transférer la syntaxe en une fois. Ceci vous évitera, par la suite, d'avoir une communication bidirectionnelle qui pourrait occasionner quelques problèmes.

Nous pouvons donc étendre notre déclaration et import de fonction un peu plus loin :

```
A4_ENTITY  *player;
A4_ENTITY  *YOU;
A4_TEXT    *msg;
A4_STRING  *empty_str;
fixed      *time;          // Voici comment vous déclarez une variable

wdlfunc1   ang;           // une fonction avec 1 argument
```

| | | | |
|---|---|---|---|
| wdlfunc2 | vec_set; | // | 2 arguments |
| wdlfunc3 | vec_diff; | // | 3 arguments |

Vous verrez, ce n'est composé que de pointeurs, à part les fonctions.

Notre fonction d'import pourrait ressembler à cela

```
DLLFUNC fixed function_import(void)
{
player          = (A4_ENTITY *)a5dll_getwdlobj("player");       //
YOU             = (A4_ENTITY *)a5dll_getwdlobj("you");
msg             = (A4_TEXT *)a5dll_getwdlobj("msg");
empty_str = (A4_STRING *)a5dll_getwdlobj("empty_str");
game_panel= (A4_PANEL *)a5dll_getwdlobj("game_panel");
wdltime         = (fixed *)a5dll_getwdlobj("time");
vec_set         = (wdlfunc2)a5dll_getwdlfunc("vec_set");
}
```

## 5. La dll prend les commandes, mais comment ?

Supposons que vous ayez copié l'intégralité de la syntaxe – tout au moins ce dont vous avez besoin. Première chose que vous voudriez voir est comment manipuler l'entité, changer son alpha, sa valeur… OK, supposons que nous avons une action WDL appelée

```
action my_dummy
{
while(1)
     {
     // please call the dll
     wait(1);
     }
 }
```

C'est l'architecture minimale nécessaire pour permettre à notre dll de prendre le relais. Premièrement, nous déclarons une nouvelle fonction dll qui déclenche notre objet :
dllfunction call_my_dummy(ent);

Dans la dll, nous ajoutons:

```
DLLFUNC fixed call_my_dummy(long entity)
{
/////// L'action serait ici
return 0;
}
```

Un peu plus loin, nous ajoutons un pointeur vers l'objet:
A4_ENTITY *dummy;

Maintenant, je peux déclencher la dll et modifier la boucle while

```
while(1)
      {
      call_my_dummy(my);
      wait(1);
      }
```

Analysons ce qui se passe quand la boucle appelle call_my_dummy(my);

Elle transfère l'objet vers la dll. Là, il est reçu par la fonction, mais de nouveau pas tel quel mais comme une adresse.

Il doit donc y avoir une conversion pour changer la longue entité entre parenthèses en un pointeur A4_ENTITY

C'est ce qui peut être fait par cette commande

```
DLLFUNC fixed call_my_dummy(long entity)
{
dummy = (A4_ENTITY *)entity;
return 0;
}
```

Bien, çà marche.

Récapitulons ce que nous avons fait jusqu'à présent :

Nous avons une action WDL qui déclenche une fonction dll, et cette fonction reçoit l'adresse de cette action.

Maintenant, nous pourrions commencer à manipuler notre objet via la dll.

Comme ceci :

```
DLLFUNC fixed call_my_dummy(long entity)

dummy = (A4_ENTITY *)entity;
return dummy->pan += FLOAT2FIX(0.1);
}
```

Notre objet tournerait.

Jetons un œil à la nouvelle ligne. Pourquoi cette étrange ligne FLOAT2FIX(0.1), pourquoi ne pas écrire :

return dummy-> pan += 0.1;

La raison est que WDL n'accepte que le type de données fixed. Si vous voulez donc renvoyer des valeurs au moteur, vous devez les reconvertir en fixed. Il y a quelques macros qui peuvent faire cela :

```
INT2FIX(1);          // convertit un integer en fixed
FLOAT2FIX(1);   // convertit un float en fixed
```

Dans le sens contraire

```
FIX2INT(1);
FIX2FLOAT(1);
```

Les conversions deviennent une routine si vous prévoyez d'utiliser une dll comme plate-forme principale de votre jeu. Au départ cela vous paraitra gênant car vous n'aurait pas un simple mais un triple processus. Si vous avez une simple variable, vous devrez la convertir puis la reconvertir.

C'est toujours WDL à DLL à WDL

Vous pouvez vous demander ce que cela signifie : la réponse est que vous pouvez construire des classes. Vous ne vous préoccuperez pas des 48 skills, vous travaillerez dans un environnement orienté objet.

## 6. Construire une classe

Disons que nous voulons quelque chose de complexe, pas un objet ombre d'une entité WDL, nous voulons une classe comme :

```
class smart_dummy
{
public:
int MillionSkills[1000][1000];

A4_ENTITY *ent;

A4_TEXT     *text;
A4_PANEL   *panel;

} SmartGuy;
```

Au lieu de créer un pointeur global, nous l'avons 'encapsulé' dans une classe. Mieux encore, nous avons ajouté des pointeurs. Et pour vous montrer comment cela est facile, il a maintenant un tableau à 2 dimensions avec de la place pour de nombreuses valeurs.

Aussi facilement que nous avons déclenché notre 'dummy' global nous allons déclencher notre Smart dummy :

```
DLLFUNC fixed call_my_dummy(long entity)
{
SmartGuy.ent = (A4_ENTITY *)entity;
return SmartGuty.ent->pan += FLOAT2FIX(0.1);
}
```

Vous voyez comme cela est facile. Il serait pratique de pouvoir déclencher une fonction main pour cet objet classe qui doit faire partie de la déclaration de la classe

```
class smart_dummy
{
public:
int MillionSkills[1000][1000];

main();
```

```
A4_ENTITY *ent;

A4_TEXT     *text;
A4_PANEL   *panel;

} SmartGuy;

smart_dummy::main()
{
ent->pan += FLOAT2FIX(0.1);   // Ici, la rotation est déclenchée de l'intérieur…
}
```

Ce que nous avons fait est créer une communication DLL complète. Chaque fois que la boucle WDL appelle la dllfonction, l'objet est déclenché et redirigé vers la fonction principale de la classe.

Vous n'avez donc pas à vous préoccuper des appels de WDL.

## 7. Utiliser les fonctions WDL

Supposons que vous vouliez utiliser des fonctions de WDL. C'est un peu délicat parce que le code semble très élaboré.

D'abord : nous supposons que vous avez déclaré la fonction au début et importé celle ci dans la dll (voir précédemment)

Commençons. Déclarons un tableau à trois entrées qui recevra les coordonnées du joueur.

```
class  smart_dummy
{
...

public:
..
/// C'est ce que nous rajouterions…
fixed  temp[3];

execute_wdl();
...
}


smart_dummy::execute_wdl()
{
temp[0]  =  INT2FIX(1000);    // Rappelez-vous nous devons utiliser le type fixed…
temp[1]  =  INT2FIX(200);
temp[2]  =  FLOAST2FIX(0.2);

(*vec_set)((long)&(ent->x),(long)temp);
}
```

Je pense que les premières lignes ne demandent aucune explication. Si nous utilisons une fonction WDL, nous devons utiliser le type fixed, d'où les conversions.

Cependant, la dernière ligne est un peu difficile, attardons nous y un peu.

Pourquoi (*vec_set) ?

Parce que nous n'avons pas un accès direct à la fonction. Notre copie de « vec_set » dans la dll est un pointeur et puisque nous ne voulons pas transférer l'adresse mais la valeur, nous devons le déréférencer.

Si nous voulons utiliser un paramètre, nous devons le convertir en type long.

Regardez la déclaration dans un fichier en-tête du a5dll.

```
typedef fixed (*wdlfunc1)(long);
typedef fixed (*wdlfunc2)(long,long);
typedef fixed (*wdlfunc3)(long,long,long);
typedef fixed (*wdlfunc4)(long,long,long,long);
```

Nous devons ajuster notre appel pour réaliser notre déclaration.

Nous devons donc reconvertir notre tableau temp fixed en long, comme ceci :

```
(long) temp ;
```

Dans le cas du premier paramètre, nous n'accédons pas à un tableau[3] fixed mais juste l'adresse de départ, tableau[0]. Nous écrivons donc :

```
&(ent->x)
```

Ce qui signifie : prendre l'adresse et lire le segment de données long

Pour conclure :

Effectuer les conversions et les reconversions peut être un travail pénible mais gardez en mémoire que vous bénéficierez de l'avantage de la programmation orientée objet. Vous pouvez gonfler votre action avec la complexité que vous voulez, le jeu en vaut la chandelle.

## 8. Prise en compte des évènements

L'architecture du WDL serait incomplète sans l'intégration de la prise en compte d'évènements.

Une fois de plus, donc, nous devons revenir à notre action WDL et ajouter quelques lignes.

Tout d 'abord, déclarez une fonction dll qui exécutera le coté dll de la prise en compte d'évènements :

```
dllfunction call_my_dummyevent(eventNr) ;
```

Ensuite, vous devez écrire une fonction événement qui déclenchera la dll :

```
function dummy_event()
{
```

```
        call_my_dummyevent(event_type, my); // Ceci déclenche l'évènement dll
    }

    action my_dummy
    {
    my.enable_click = ON;
                        // Ajoutez tous les évènements que vous voulez utiliser
    my.enable_shoot = ON;

    my.event = dummy_event;                  // Ceci déclenche l'évènement

    while(1)
              {
              call_my_dummy(my);
              wait(1);
              }
    }
```

D'accord, c'est un beau parcours. Action appelle la fonction event du WDL qui appelle la fonction event dll

Vous seriez en droit de vous demander ce qui se passe pour la variable « event_type », passée en argument.

Je me suis également posé la question, jusqu'à ce que je me rende compte que event_type est synonyme de nombre.

Donc, si event_type est click, il ne passera pas la chaine « click » mais un nombre, 16, dans ce cas.

Chaque type a son numéro correspondant. Nous le verrons plus loin.

Nous avons maintenant déclaré une fonction dll dans WDL, mais pas dans la dll, c'est ce que nous allons faire maintenant.

```
    DLLFUNC fixed call_my_dummyevent(fixed event_type, long entity)
    {
    // Maintenant, nous devrions déclencher notre SmartDummy
    return 0;
    }
```

Avec cette fonction la communication est établie. Chaque fois qu'un événement apparaît, la dll est déclenchée.

OK, jusqu'à maintenant, rien ne se passe, nous devons assigner l'entité et ensuite ajouter une autre fonction événement à notre classe SmartDummy.

Commençons par cela :

```
    class smart_dummy
    {
    ...

    public:
```

```
..
// Ceci est ce que nous ajouterions

call_event(int number);  // rappelez-vous «event type» est juste un nombre
...
}


smart_dummy::call_event(int  number)
{
if (event == 5)                                     // Impact
     {
     }

if (event == 9)                                     // Shoot
     {
     }

if (event == 16)                                    // Touch
     {
     }

if (event == 18)                                    // Click
     {
     }


if (event == 19)                                    // Rightclick
     {
     }

if (event == 8)                                     // Scan
     {
     }

if (event == 1)                                     // Block
     {
     }

if (event == 3)                                     // Stuck
     {
     }

if (event == 2)                                     // Entity
     {
     }

if (event == 4)                                     // Push
     {
```

```
        }

    if (event == 17)                                        //   release
        {
        }

    if (event == 7)                                         // detect
        {
        }

    if (event == 10)                                        // trigger
        {
        }

    if (event == 11)                                        // Sonar
        {
        }


    if (event == 24)                                        // Disconnect
        {
        }

    }
```

D'accord, cela aurait été plus élégant avec un switch. Mais libre à vous d'améliorer ce code.
L'important est de bien transmettre les valeurs correctes aux évènements.
Cependant, il est nécessaire de modifier la fonction interface.

```
DLLFUNC fixed call_my_dummyevent(fixed event_type, long entity)
{
SmartGuy.ent = (A4_ENTITY *)entity;
int no = FIX2INT(event_type);          // le convertit en int
SmartGuy.call_event(no);
return 0;
}
```

Voilà donc, vous pouvez maintenant vous concentrer sur la programmation en C++.

traduction française: Pierre Gatoux