haute école
neuchâtel berne jura

arc ingénierie
saint-imier le locle porrentruy

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences
Western Switzerland

Utilisation DLL

Joaquim Stähli

Professeur
Cédric Bilat

1 Utilisation DLL

1.1 Introduction

Un projet qui import ou export des dll utilise des techniques différentes en fonction de l'OS (Window, Linux) et du compilateur (GCC, MinGW, Visual, Intel).

Linux :

Exporter dll :

```
__attribute__ ((visibility ("default"))) void afficher();
```

Importer dll :

```
__attribute__ ((visibility ("default"))) void afficher();
```

Window :

Exporter dll :

```
declspec(dllexport) void afficher();
```

Importer dll :

```
declspec(dllimport) void afficher();
```

Objectif :

Un seul code qui import ou export des DLL fonctionnant dans touts les cas de figures (Window et Linux).

1.1.1 Solution Intermédiaire

Utiliser des variables utilisés par le préprocesseur.

Export :

```
#if defined _WIN32
  #define DLL_EXPORT __declspec(dllexport)
#else
  #define define DLL_EXPORT __attribute__ ((visibility ("default")))
#endif
```

dllExport.h

Import :

```
#if defined _WIN32
  #define DLL_IMPORT __declspec(dllimport)
#else
  #define define DLL_IMPORT __attribute__ ((visibility ("default")))
#endif
```

dllImport.h

Exemple d'utilisation :

Dans un projet il y a pour chaque header (.h) une version d'importation et d'exportation.

HelloWorld Importation :

```
#include <dllImport.h>
DLL_IMPORT sayHellWorld() ;
```

HelloWorld Exportation :

```
#include <dllExport.h>
DLL_EXPORT sayHellWorld() ;
```

Problème :

Il y a toujours 2 fichier .h a maintenir ce qui entraîne des erreurs.

Objectif :

Un unique fichier .h avec une seule variable qui fera soit de l'importation ou soit de l'exportation.

1.1.2 Solution final

La solution au final sera composée de 2 parties, une partie générique qui est la même de projet en projet et une partie spécifique au projet.

(E1) Partie générique

Le fichier générique qui est le même de projet en projet s'occupe de définir la variable d'importation/exportation selon l'OS et le compilateur.

```
ifndef IMPORT_EXPORT_H
#define IMPORT_EXPORT_H
// Generic helper definitions for shared library support
#if defined _WIN32 || defined __CYGWIN__
  #define HELPER_DLL_IMPORT __declspec(dllimport)
  #define HELPER_DLL_EXPORT __declspec(dllexport)
  #define HELPER_DLL_LOCAL
#else
  #if __GNUC__ >= 4
    #define HELPER_DLL_IMPORT __attribute__ ((visibility ("default")))
    #define HELPER_DLL_EXPORT __attribute__ ((visibility ("default")))
    #define HELPER_DLL_LOCAL  __attribute__ ((visibility ("hidden")))
  #else
    #define HELPER_DLL_IMPORT
    #define HELPER_DLL_EXPORT
    #define HELPER_DLL_LOCAL
  #endif
#endif
```

importExport.h

Note :

Ce fichier est dépendant des variables d'environnements, _WIN32 et __GNUC__ qui sont des variables défini par le système et non pas l'utilisateur.

(E2) Partie spécifique au projet

```
#ifdef PROJECTA_WITH_DLL
  #ifdef PROJECTA_DLL_EXPORT
    #define PROJECTA HELPER_DLL_EXPORT
  #else
    #define PROJECTA HELPER_DLL_IMPORT
  #endif
  #define PROJECTA_LOCAL HELPER_DLL_LOCAL
#else
  #define PROJECTA
  #define PROJECTA_LOCAL
#endif
```

envProjetA.h.

Le travail du développeur est de définir correctement les 4 variables nécessaires au bon fonctionnement de ce header (*envProjetA.h*).

Exporter du code dans une DLL

Le développeur doit définir 2 variables :

- PROJECTA_WITH_DLL
- PROJECTA_DLL_EXPORT

Il doit les définir à l'appel du compilateur.

Options compilateur :

```
gcc –D PROJECTA_WITH_DLL -D PROJECTA_DLL_EXPORT …
```

Utilisation avec une classe :

```
#include "envProjetA.h"
class PROJECTA HelloWorld
{
      public :
            HelloWorld() ;
            virutal ~HelloWorld() ;
            void sayHello() ;
}
```

HelloWorld.h

Utilisation avec des procédures :

```
#include "envProjetA.h"
PROJECTA void say(char* something);
```

say.h

Importation du code depuis une dll

Le développeur doit définir la variable **PROJECTA_WITH_DLL** !

Exemple 1 : *définie dans le code*

```
#define PROJECTA_DLL_WITH_DLL
#include "say.h"
int main(void)
      {
      Say("HelloWorld !");
      }
```

Exemple 2 : *option compilation*

```
#include "say.h"
int main(void)
      {
      Say("HelloWorld !");
      }
```

PROJECTA_LOCAL

Window :

Si on veut qu'une déclaration (fonction, classe, etc..) ne soit pas dans la dll, il suffit de ne rien mettre devant son prototype. Ce qui explique que dans le fichier *importExport.h*, HELPER_DLL_LOCAL est vide.

Linux :

Sous linux par défaut toutes les déclarations (fonction, classe, etc..) sont exportées dans la dll. On peut utiliser HELPER_DLL_LOCAL pour ne pas l'exporter. En pratique on conseil de désactivé cette fonctionnalités avec *-fvisibility=hidden*.

Commande :

```
g++ -fvisisibitly=hidden ….
```

On conseil donc ne jamais utiliser le PROJECTA_LOCAL !

1.2 Exemple

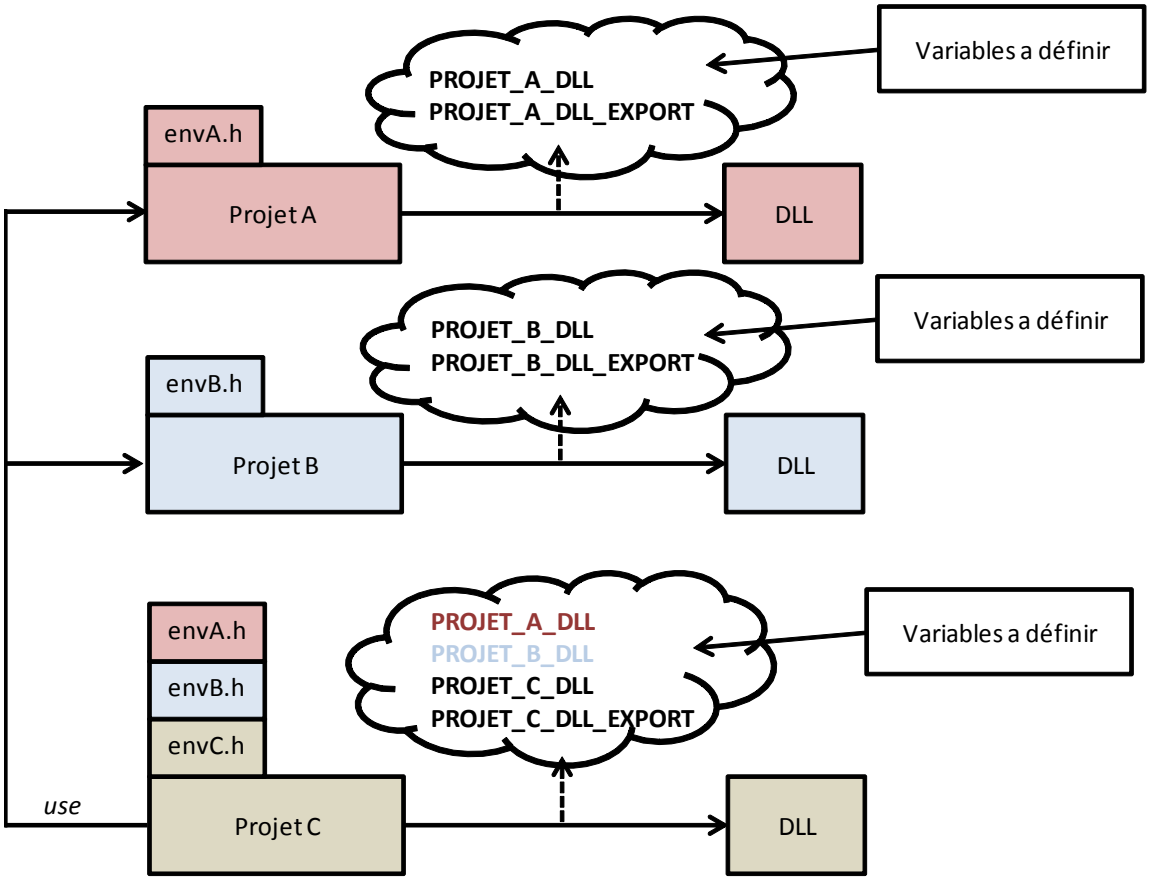
Objectif :

A export DLL

B export DLL

C export DLL, import A-DLL, B-DLL

Solution :



2