Fabrication d'une librairie dynamique (DLL)

Nicolas Gazères

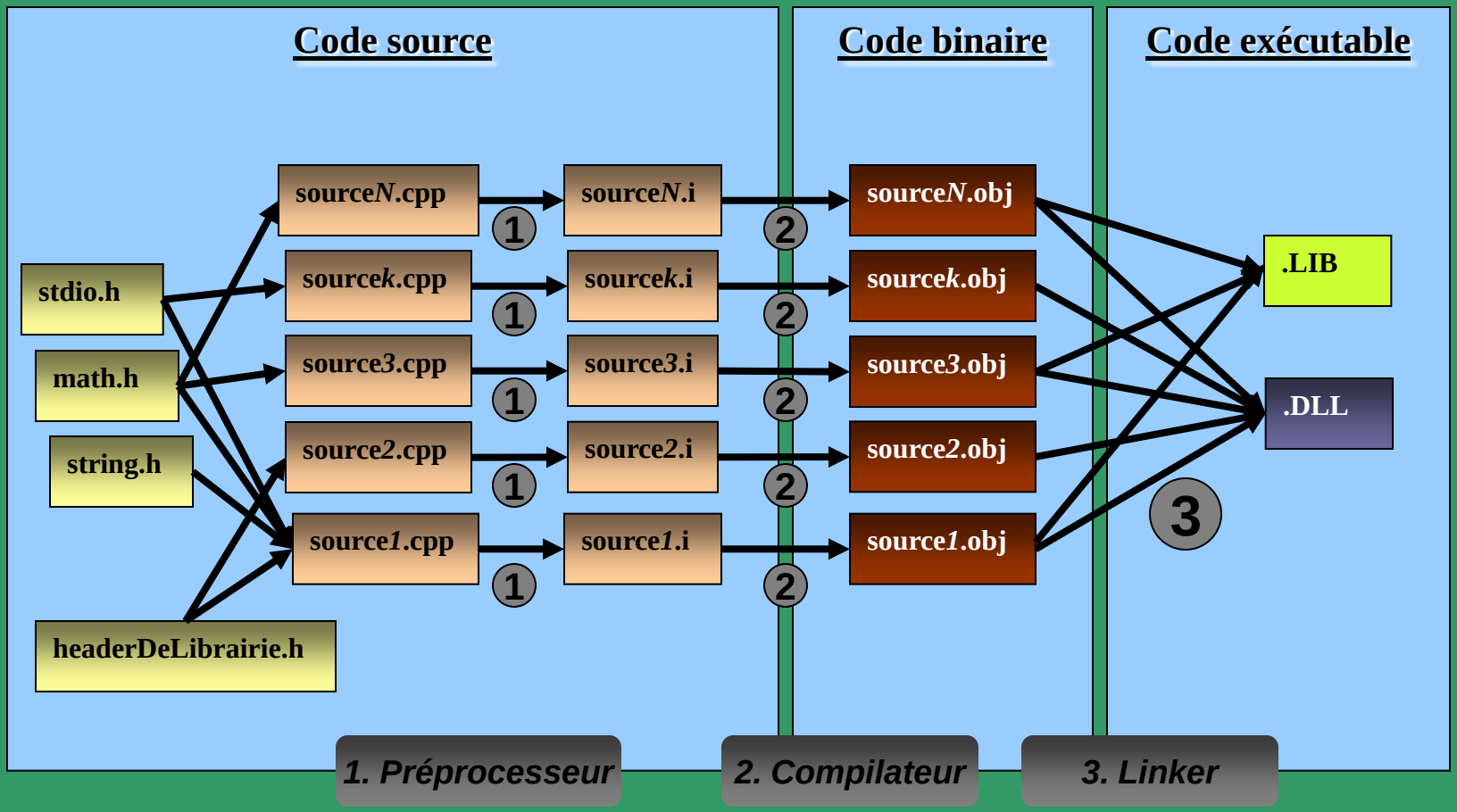
Dassault Systèmes
DS Research, Life Sciences
ngs@3ds.com

Qu'est-ce qu'une DLL ?

- Une DLL contient du code exécutable
  - Mais une DLL n'est pas exécutable.
- Une DLL exporte des symboles
  - variables, fonctions …
- Un exécutable peut utiliser les symboles exportés par une DLL et ainsi lui déléguer des traitements.
  - Lire les variables globales de la DLL
  - Appeler les fonctions (non *static*) de la DLL
- Avantages principaux
  - Une même DLL peut être *partagée* par plusieurs processus à un instant donné.
  - Une DLL est un format permettant de distribuer et de *maintenir* facilement un *composant*.

Fabrication d'une DLL

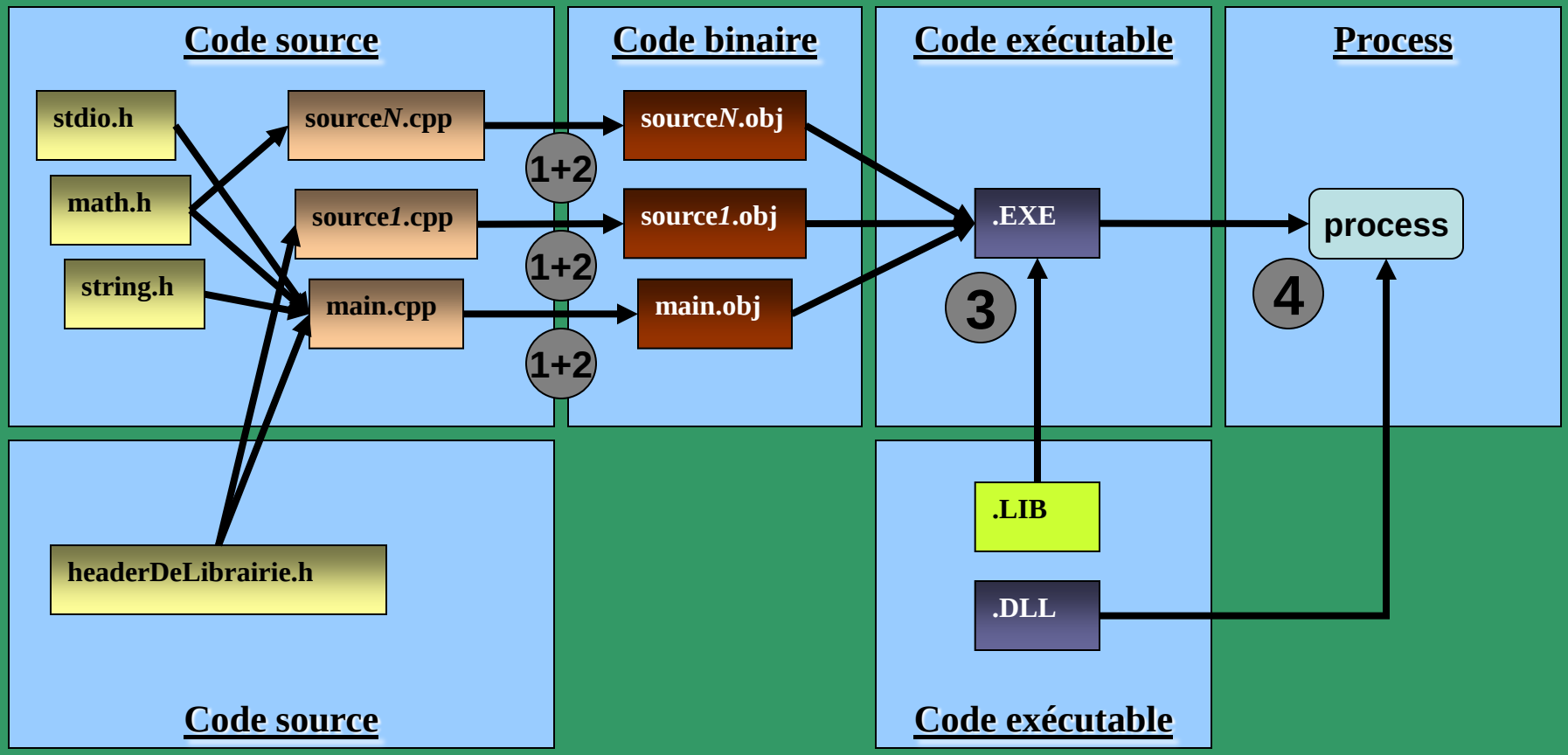
Schéma général



- Les produits du build contiennent le code binaire (*.DLL*) et la librairie d'import (*.LIB*).

Utilisation d'une DLL

Schéma général



Récapitulatif

- Pour utiliser un composant qui se présente sous forme de DLL, il faut disposer de:
  - Un (ou des) header (*.H*)
    - Contient la déclaration des symboles exportés
  - La librairie d'import (*.LIB*)
    - Pour que le programme sache que le code de la fonction pourra bien être trouvé dans la DLL en question.
  - La librairie dynamique (.DLL)
    - Contient le code proprement dit du composant.

- Le code qui utilise la DLL fait appel:
  - Au header: → à la compilation
  - À la librairie d'import: → à l'édition des liens
  - À la DLL: → au lancement du programme

Terminologie

- Librairie d'import (Import Library)
  - Fichier .LIB
- Créée à l'édition des liens d'une DLL
- On parle d'édition des liens implicites.
- Link-time
- Load-time
- Run-time
- Exporter un symbole, importer un symbole.

Des DLLs connues

- L'emplacement classique est:
  - C:\Program Files\Microsoft Visual Studio 8\VC\lib
- MSVCRT.dll
  - Le runtime C standard (printf, scanf, ...)
- MSVCRTD.dll
  - La même, compilée en mode Debug.
- KERNEL32.dll
  - Les appels Windows de bas niveau.
- USER32.dll
  - La librairie de gestion de la sécurité
- GDI32.dll
  - La librairie d'interface graphique.

Directive *dllexport*

- Principe
  - Cette directive à destination du linker permet d'exporter un symbole d'une DLL:
- Règle
  - Pour exporter des fonctions ou des variables globales, la directive doit apparaître à la gauche du symbole:

mon_header_qui_declare_un_symbole_exporte.h

```
__declspec(dllexport) void maFonction(void);
__declspec(dllexport) long maDonnee;
```

- Le symbole apparaît tagué « external » dans *dumpbin.exe*.

Directive *dllimport*

- Principe
  - Cette directive (à destination du linker) permet d'indiquer à une application cliente qu'elle pourra trouver le symbole dans une DLL.
- Exemple
  - Pour importer un symbole (fonction ou variable globale), la directive doit se situer à gauche du symbole.

mon_source_qui_importe_le_symbole.cpp

```
__declspec(dllimport) void maFonction(void);
__declspec(dllimport) long maDonnee;
```

Conséquence des directives d'import/export

- Le problème
  - On ne souhaite pas, pour les mêmes symboles, avoir à maintenir en cohérence:
    - un header avec *dllexport* pour compiler la DLL,
    - un header avec *dllimport* pour les clients de la DLL

- La solution
  - **Propriété fondamentale de Visual Studio**
    - Quand Visual Studio compile un projet de type "DLL" nommé **MaLib**, son préprocesseur définit automatiquement un symbole **MALIB_EXPORTS**.
    - Ce symbole n'est pas défini dans les projets de type "console application" qui sont clients de la DLL.
  - D'où la technique standard…

Header unifié (import/export)

- Une technique standard
  - pour maintenir une seule version du header
  - Valable à la fois pour la DLL et ses clients
- Technique

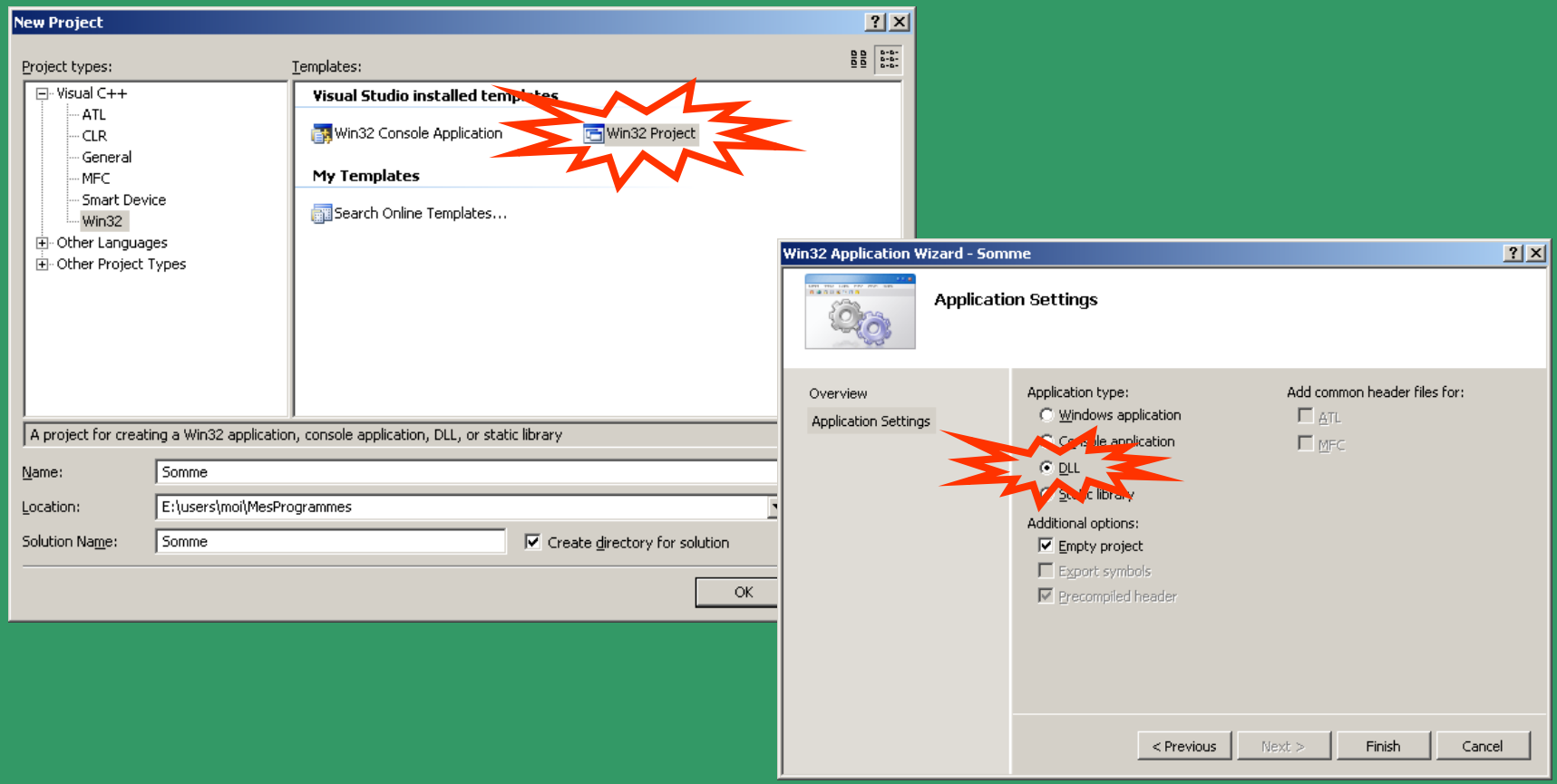
headerDeLaDLL.h

```
#ifdef MALIB_EXPORTS
#define ExportedBy __declspec(dllexport)
#else
#define ExportedBy __declspec(dllimport)
#endif

ExportedBy void maFonction(void);
ExportedBy long maDonnee;
```

Projet DLL

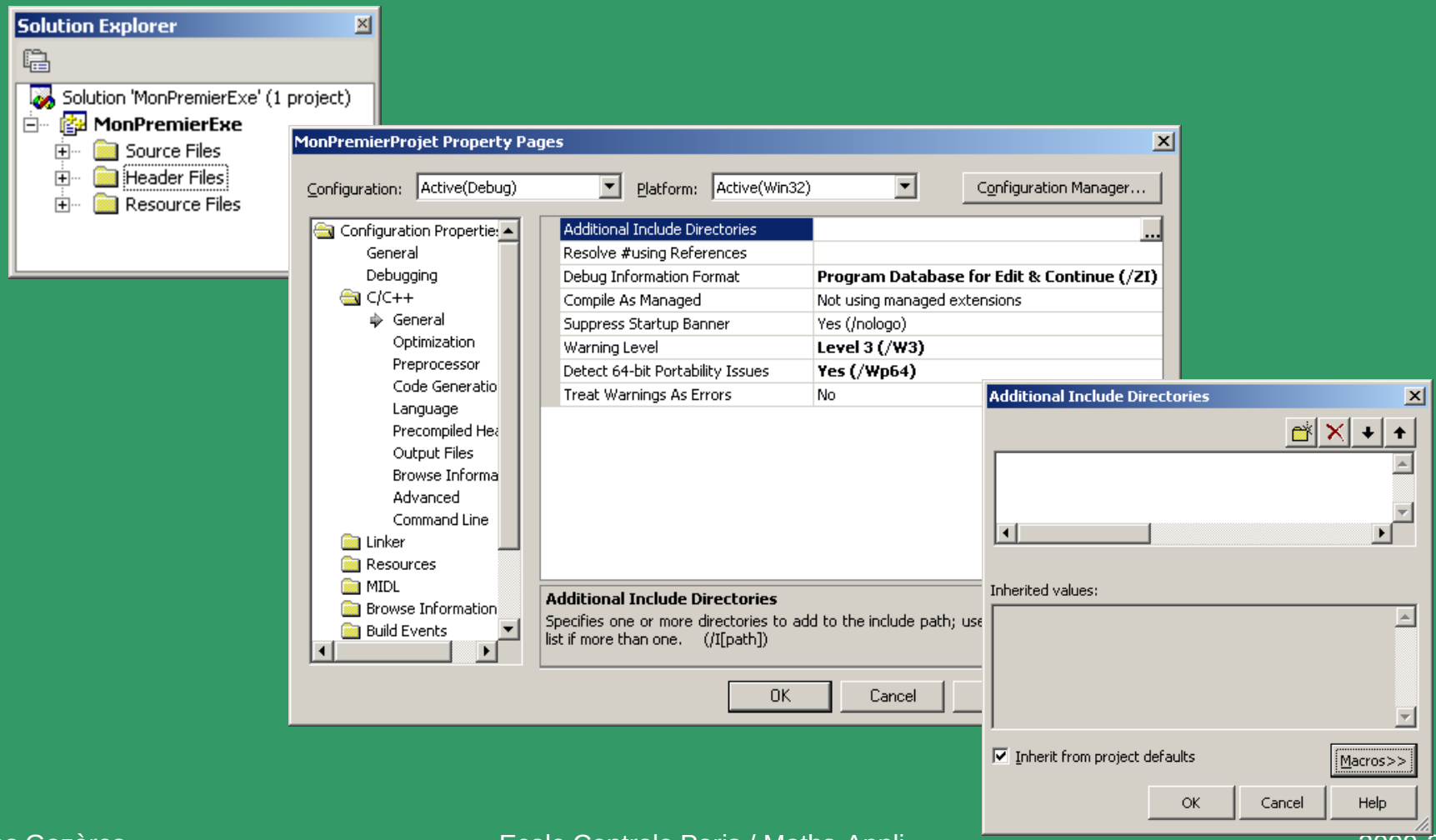
- File > New > Project… >
  - Sélectionner « Win32 Project » puis « DLL »



Dans un projet *utilisant* une DLL

Comment spécifier le chemin d'accès aux headers ?

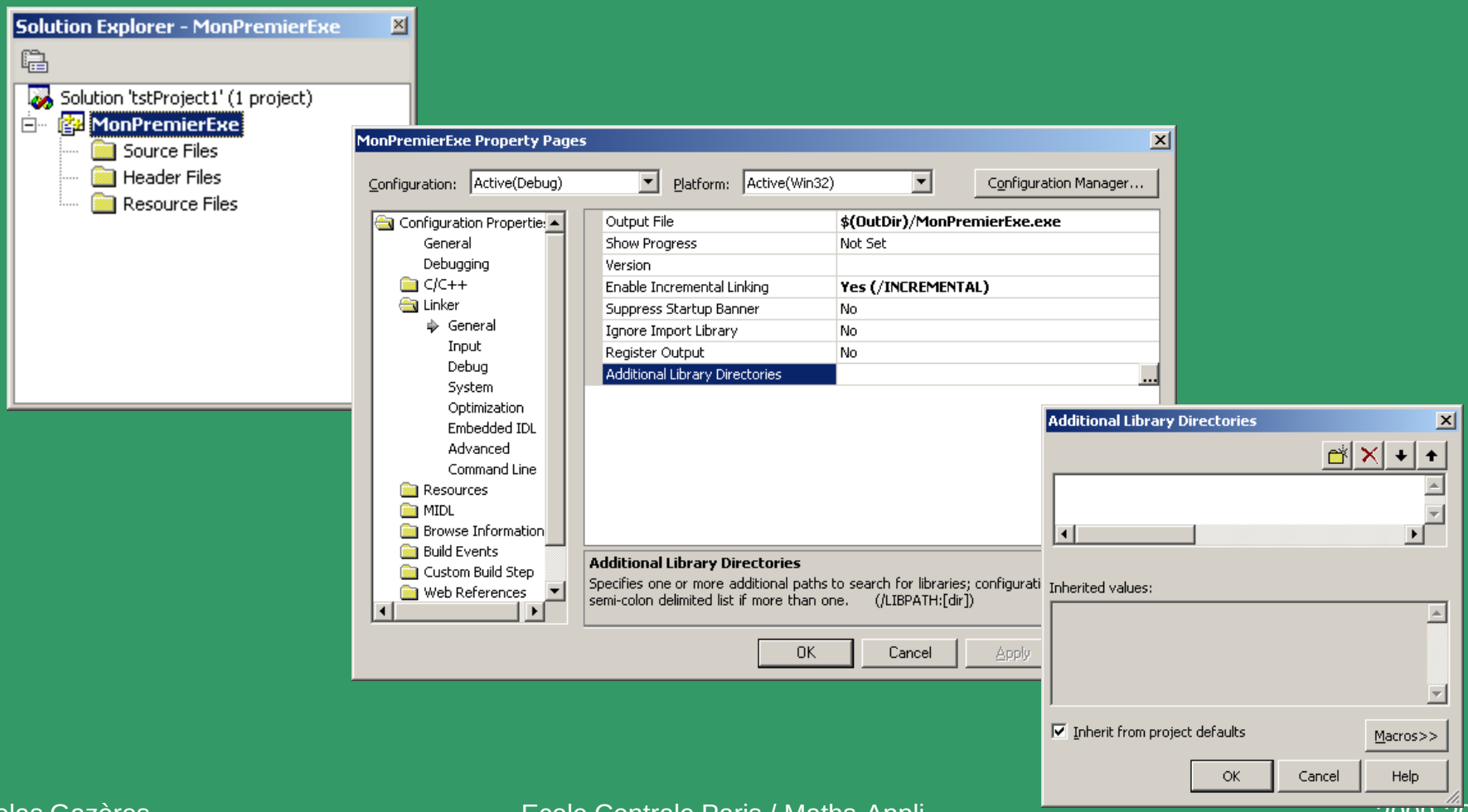
- **Project > Properties > Configuration Properties > C/C++ > Additional Include Directories**



Dans un projet *utilisant* une DLL

Comment spécifier le chemin d'accès au .LIB ?

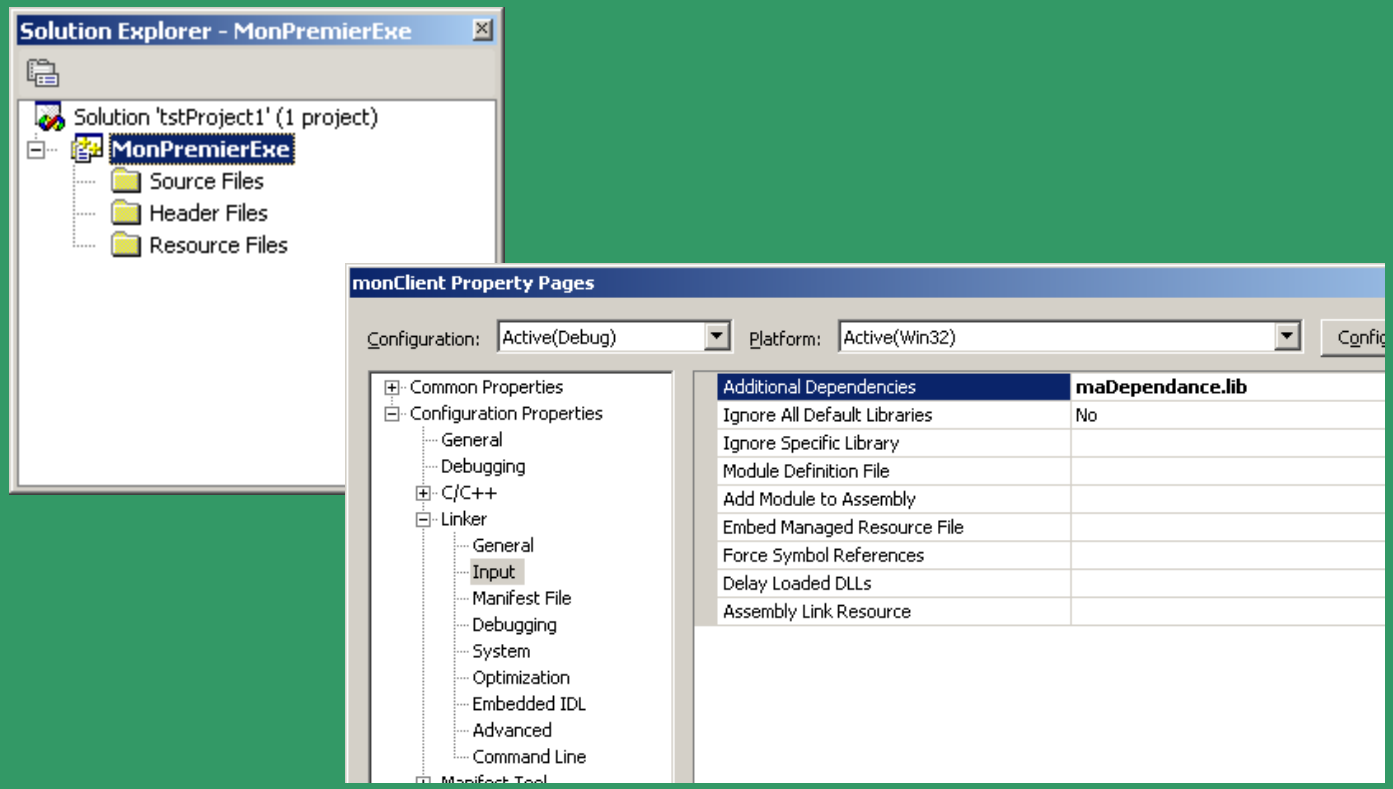
- **Project > Properties > Linker > General > Additional Libraries Directories**



Dans un projet *utilisant* une DLL

Comment spécifier les DLLs nécessaires?

- **Project > Properties > Linker > Input > Additional Dependencies**



Le Search Order

Comportement au *Load-time*

Au chargement d'un exécutable (load-time), Windows recherche les DLLs par ordre de préférence décroissante:

1. Le répertoire où l'exécutable se trouve.
   → C'est l'emplacement le plus naturel pour les petits projets.
2. Le répertoire depuis lequel l'exécutable est lancé.
3. Le répertoire système de Windows
   C:\Windows\System
4. Le répertoire Windows
   C:\Windows
5. Les répertoires figurant dans la variable d'environnement PATH
   → C'est ce répertoire que vous modifiez en général.

Avantages/Inconvénients des DLLs

- Avantages
  - Taille d'exécutable plus petite
  - Possibilité d'upgrader facilement une DLL
    - Si le code des fonctions exportées change sans que la signature change, il n'y a pas besoin de recompiler l'exécutable client.
  - Partage maximal du code de la DLL entre processus
    - L'OS factorise les pages de code
    - Moins de consommation-mémoire; moins de swap.
  - On peut appeler le code de la DLL d'un autre langage que C
    - Par exemple, Visual Basic…
- Inconvénients
  - L'application finale n'est pas auto-suffisante.
  - Temps de démarrage plus long
  - Risque de non-démarrage de l'appli.
  - Dépendances subtiles sur le Search Order

Fin