

Installer et utiliser Boost/Boost.TR1 avec Visual C++

par Aurélien Regat-Barrel ([Espace personnel](#))

Date de publication : 05/10/2006

Dernière mise à jour : 07/07/2007

Ce tutoriel explique comment installer la bibliothèque C++ Boost sous Windows au moyen de l'installateur développé par Boost Consulting, et comment configurer Visual C++ 2005 pour pouvoir l'utiliser. Des aspects avancés de son usage avec ce compilateur y sont aussi traités, comme la possibilité de l'utiliser sous forme statique ou dynamique (DLL). Il présente aussi Boost.TR1 et fournit un exemple d'utilisation de `std::tr1::regex` pour tester que tout est correctement installé. La version de Boost utilisée est la 1.34 (11 Mai 2007).



Introduction

Présentation de Boost.TR1

Installation de Boost

Configuration de Visual C++ 2005 pour utiliser Boost et Boost.TR1

Configuration du PATH de Windows

Programme d'exemple

CRT statique / dynamique (/MT[d], /MD[d])

Conclusion

Introduction

Boost est un vaste ensemble de bibliothèques C++ dont la plupart peuvent être utilisées directement sans avoir à être compilée. Il suffit simplement de les ajouter à l'INCLUDE_PATH du compilateur pour pouvoir s'en servir. Cependant, quelques unes des bibliothèques proposées nécessitent de l'être. Citons :

- date_time
- filesystem
- graph
- iostreams
- program_options
- python
- regex
- serialization
- signals
- test
- thread

La compilation de Boost s'effectue en ligne de commande au moyen d'un outil spécifique (Boost.Jam). Voilà de quoi effrayer ou désarmer un débutant peu habitué au maniement de la console sous Windows, ou même un programmeur expérimenté s'il est un peu paresseux :-)

Par chance, la société **Boost Consulting** met gracieusement à notre disposition un installeur pour Visual C++ 2003 (7.1) et Visual C++ 2005 (8). Ce dernier va télécharger une version déjà compilée des bibliothèques concernées vous épargnant ainsi de devoir les compiler vous même. Il ne reste plus ensuite qu'à configurer l'IDE de Visual C++ pour pouvoir les utiliser. C'est ce que nous allons voir dans cet article.

Présentation de Boost.TR1

Depuis Boost 1.34, la bibliothèque **Boost.TR1** est disponible officiellement. Cette bibliothèque est une implémentation partielle du **Technical Report 1** au moyen de ce qui existe déjà dans Boost. En fait, Boost.TR1 est un wrapper qui va tenter d'inclure l'implémentation du TR1 de votre compilateur si elle existe (ce n'est pas le cas avec Visual C++ 2005), et importer les bibliothèques Boost équivalentes dans le namespace std::tr1 en cas contraire.

Ainsi, grâce à Boost.TR1, il est possible de compiler des programmes se servant des ajouts du TR1 en utilisant en réalité des objets de Boost. Par exemple, le code suivant ne compilera qu'avec Boost.TR1, et générera une erreur avec une réelle implémentation du TR1 (comme celle de la libstdc++ de GCC 4 ou encore celle de **Dinkumware**).

```
#include <tr1/memory> // Boost.TR1

int main()
{
    std::tr1::shared_ptr<int> p1( new int );
    boost::shared_ptr<int> p2( new int ); // OK!
}
```

Ceci parce que Boost.TR1 effectue en fait un simple aliasage de boost::shared_ptr vers std::tr1::shared_ptr :

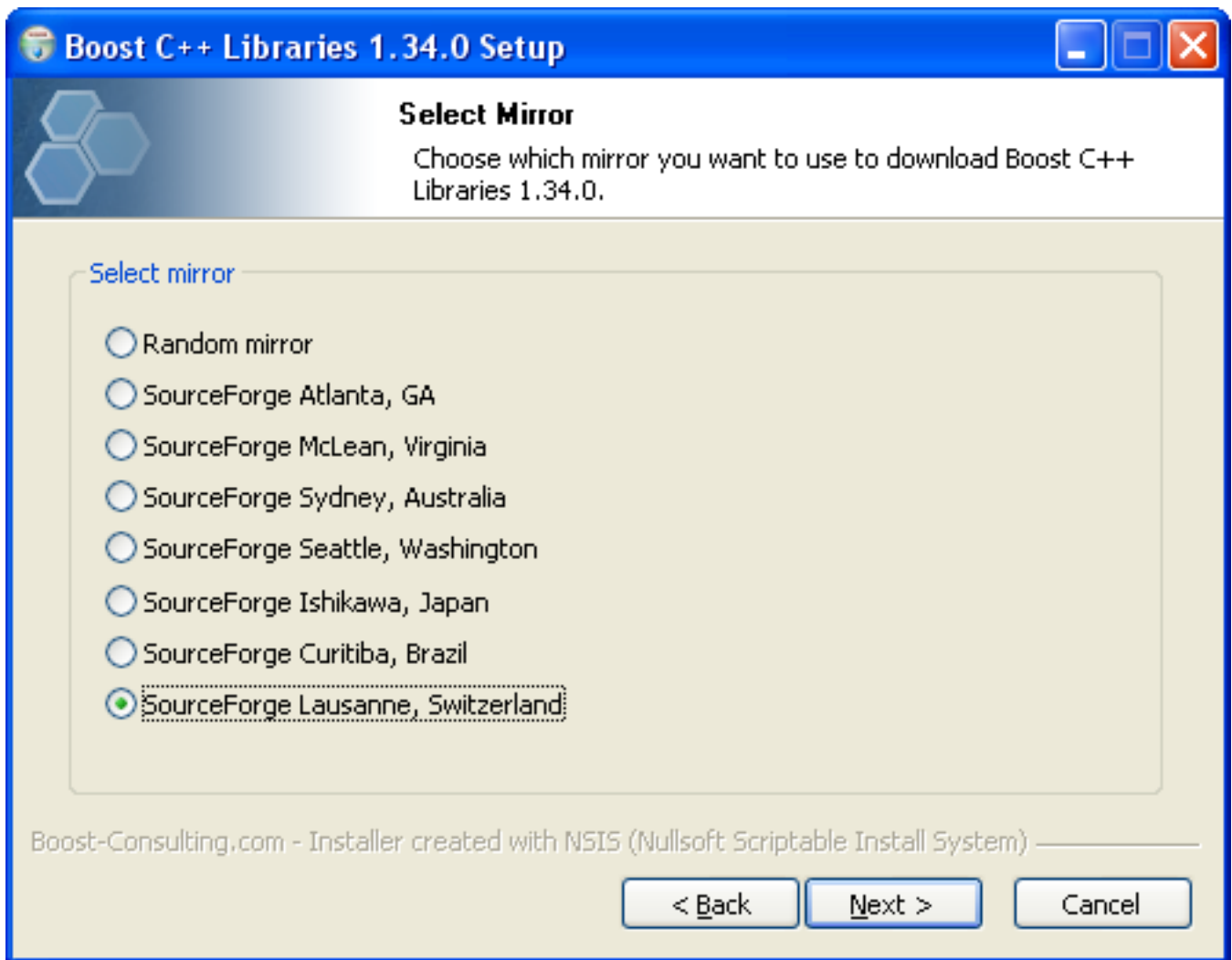
tr1/memory

```
#include <boost/shared_ptr.hpp>

namespace std
{
    namespace tr1
    {
        using ::boost::shared_ptr;
    }
}
```

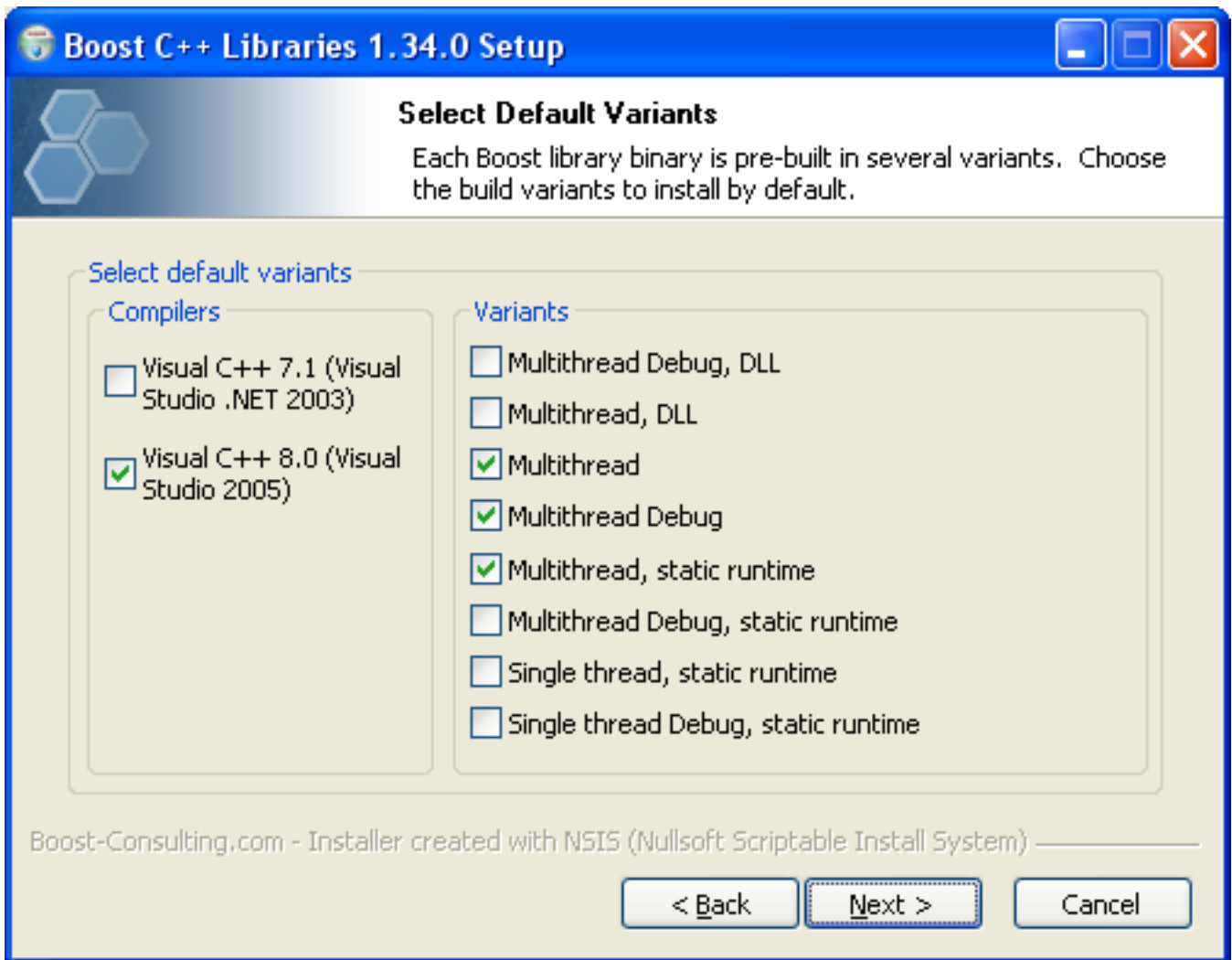
Installation de Boost

Tout d'abord, il convient de récupérer l'installateur développé par Boost Consulting. Celui-ci se trouve ici : **Downloading and Installing the Boost Libraries**. A l'heure où j'écris ce tutoriel, le dernier installateur proposé concerne la version **1.34.0** de Boost.



Sélection d'un serveur pour le téléchargement

Après avoir accepté les licences de l'installateur et de Boost, choisissez le serveur le plus proche de chez vous afin de bénéficier d'un débit de téléchargement correct (il y a de grosses différences!). L'installateur vous demande ensuite de sélectionner les versions des bibliothèques à télécharger par défaut :



Sélection des options par défaut

La signification de chaque option est la suivante :


- **Multithreaded Debug, DLL** : DLL Debug (/MDd) - Utilisé en Debug si BOOST_DYN_LINK est défini
- **Multithreaded DLL** : DLL Release (/MD) - Utilisé en Release si BOOST_DYN_LINK est défini
- **Multithreaded** : LIB statique Release (/MD) - **Utilisé par défaut en Release**
- **Multithreaded Debug** : LIB statique Debug (/MDd) - **Utilisé par défaut en Debug**
- **Multithreaded, static runtime** : LIB statique Release, CRT statique (/MT) - **Recommandé en Release**
- **Multithreaded Debug, static runtime** : LIB statique Debug, CRT statique (/MTd)
- **Single thread, static runtime** : LIB statique Release, CRT static singlethread (/ML), **VC++ 7.1 seulement**
- **Single thread Debug, static runtime** : LIB statique Debug, CRT static singlethread (/MLd), **VC++ 7.1 seulement**

Notez que depuis VC++ 8, il n'y a plus de distinction multithread / multithread : la CRT est obligatoirement multithread. Les deux dernières options ne concernent donc que Visual C++ 7.1 (voir **VC++ 7.1 C Run-Time Libraries** pour plus de détails).

Par défaut, les projets VC++ sont compilés en /MD[d] (CRT liée dynamiquement), et c'est donc la 3° et la 4° version qui sont respectivement utilisées si vous ne changez rien à vos options de compilation.

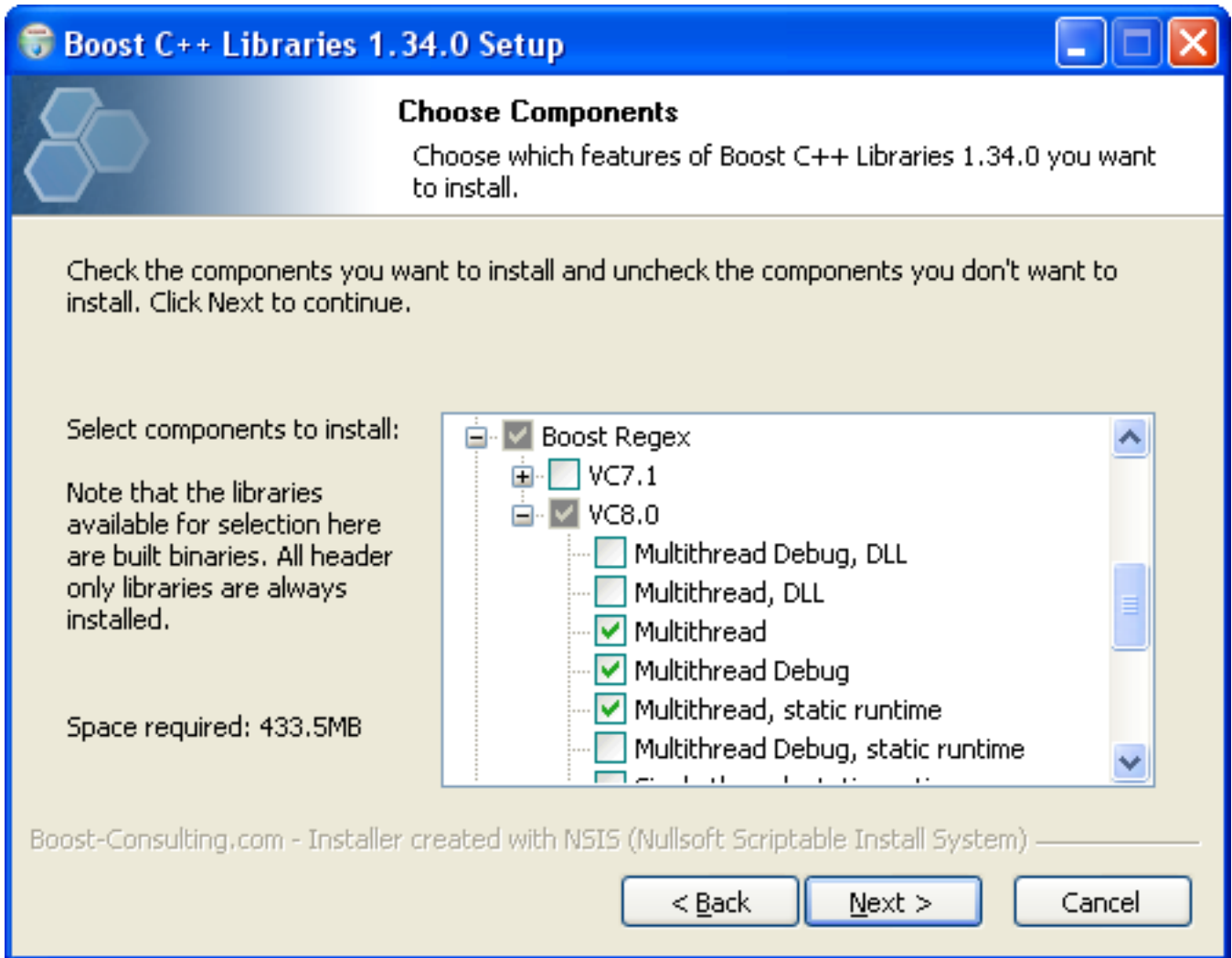
Si vous définissez BOOST_DYN_LINK, ce seront alors la 1° et la 2° version qui seront utilisées. Vous aurez besoin de ces versions si vous voulez utiliser Boost sous forme de DLL.

Si vous compilez en /MT[d] afin d'utiliser la version statique de la CRT, alors ce seront la 5° et la 6° version qui seront utilisées.

 *Je vous recommande personnellement de compiler votre version Release en /MT (plus d'informations en fin d'article), donc de cocher le 5° choix. Afin de ne pas avoir de problème pour compiler en conservant les options de projet par défaut, je vous recommande aussi de cocher au minimum les 3° et 4° choix proposés. Vous pouvez aussi tout cocher pour plus de tranquillité, mais soyez prévenu que chaque bibliothèque est assez volumineuse (433Mo juste avec ces 3 options, 677Mo pour les 6 premières).*

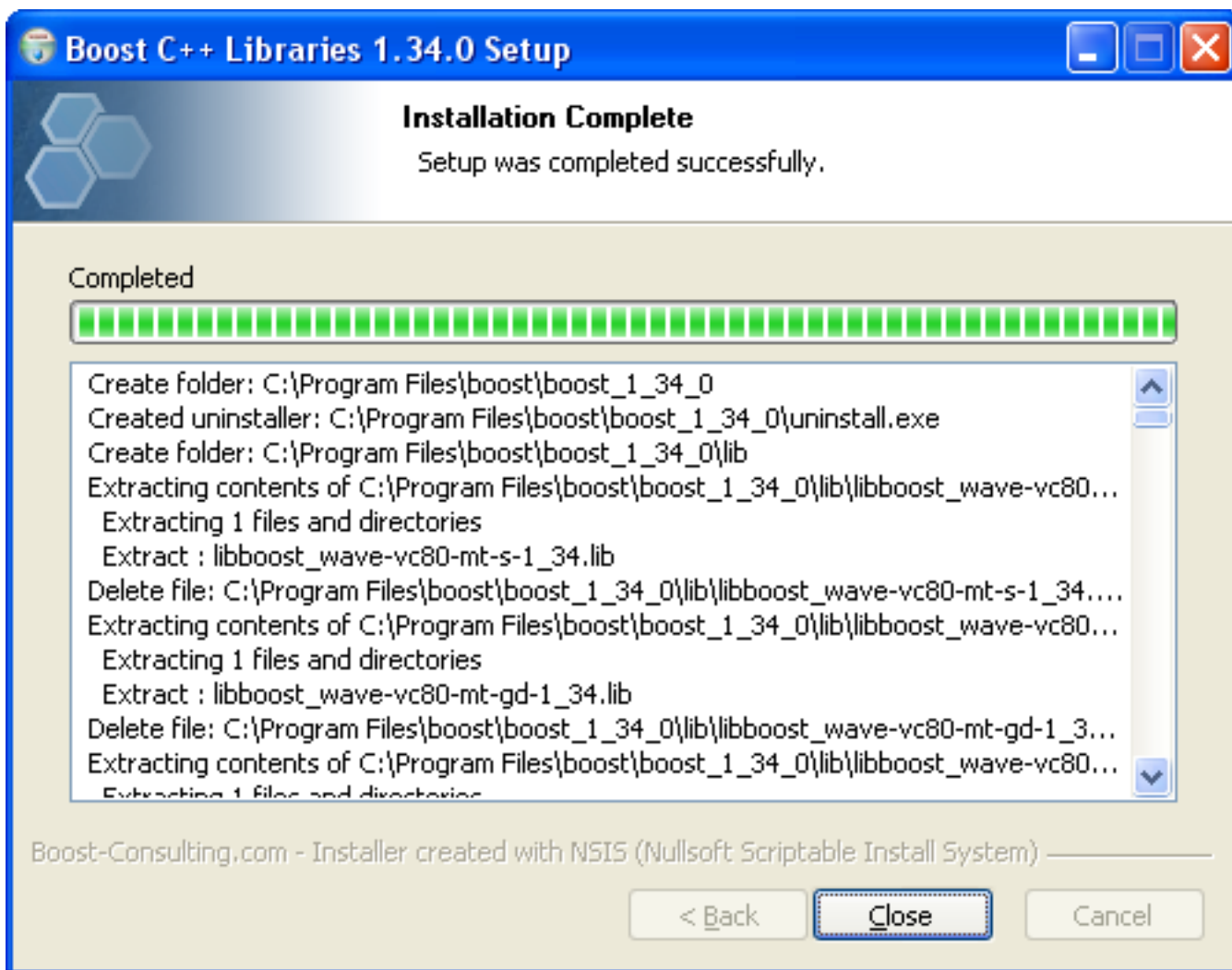
Notez qu'il n'existe volontairement pas de version combinant /MT[d] et BOOST_DYN_LINK (utilisation des bibliothèques Boost sous forme de dlls, mais avec la CRT en version statique). Pour plus de détails sur tout ceci, reportez vous au chapitre dédié en fin d'article.

Une fois votre choix effectué, vous pouvez ajuster cette configuration par défaut au niveau de chaque bibliothèque :




Ajustement des options au niveau de chaque bibliothèque

Une fois cette étape achevée, l'installateur vous demande où stocker les fichiers, puis effectue le téléchargement d'archives .zip qu'il décompresse ensuite dans le répertoire d'installation spécifié :



Téléchargement

 La version 1.31 de l'installeur semble être boguée dans la mesure où elle n'effectue pas la décompression des archives téléchargées si certaines options sont cochées (Boost header files (required) en particulier). Vous devez donc vous même décompresser les archives téléchargées si vous utilisez cette version.

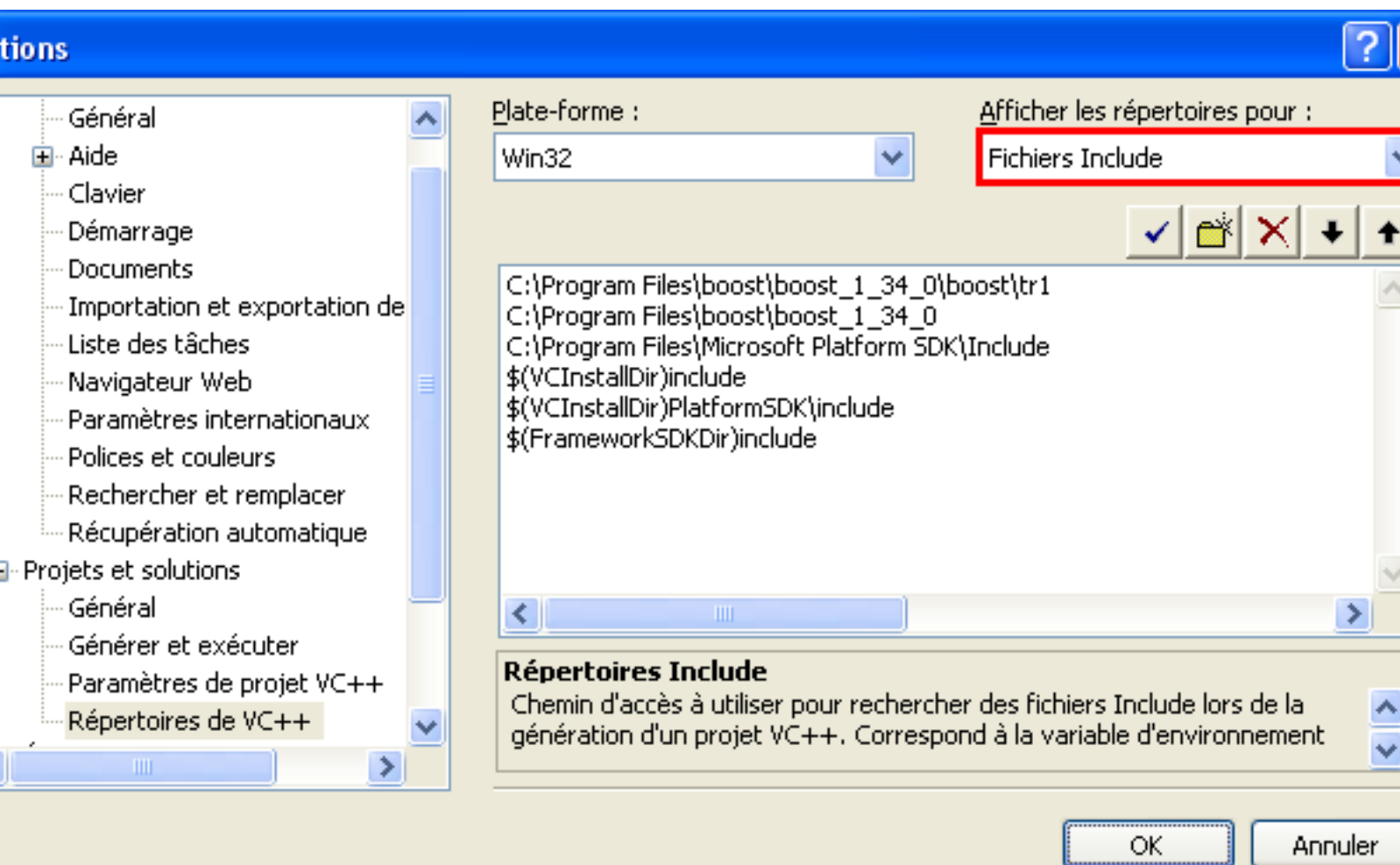
Vous devriez avoir au final dans le répertoire d'installation :

- un répertoire *boost/* contenant les fichiers d'en-têtes des bibliothèques téléchargées
- un répertoire *lib/* contenant les fichiers .lib des bibliothèques compilées ainsi que leurs dlls associées, s'il y en a
- éventuellement d'autres répertoires en fonction des options cochées, comme le répertoire *doc/* qui contient la documentation HTML de Boost (la même que celle qui est en ligne)

Configuration de Visual C++ 2005 pour utiliser Boost et Boost.TR1

En suivant les instructions précédentes, vous avez donc installé Boost ainsi que Boost.TR1. Mais Visual C++ n'en sait encore rien. Si vous tentez de compiler un programme les utilisant, l'opération échouera car Visual C++ ne trouvera pas les fichiers d'en-tête spécifiés. Nous allons donc configurer l'IDE pour qu'il sache où le trouver.

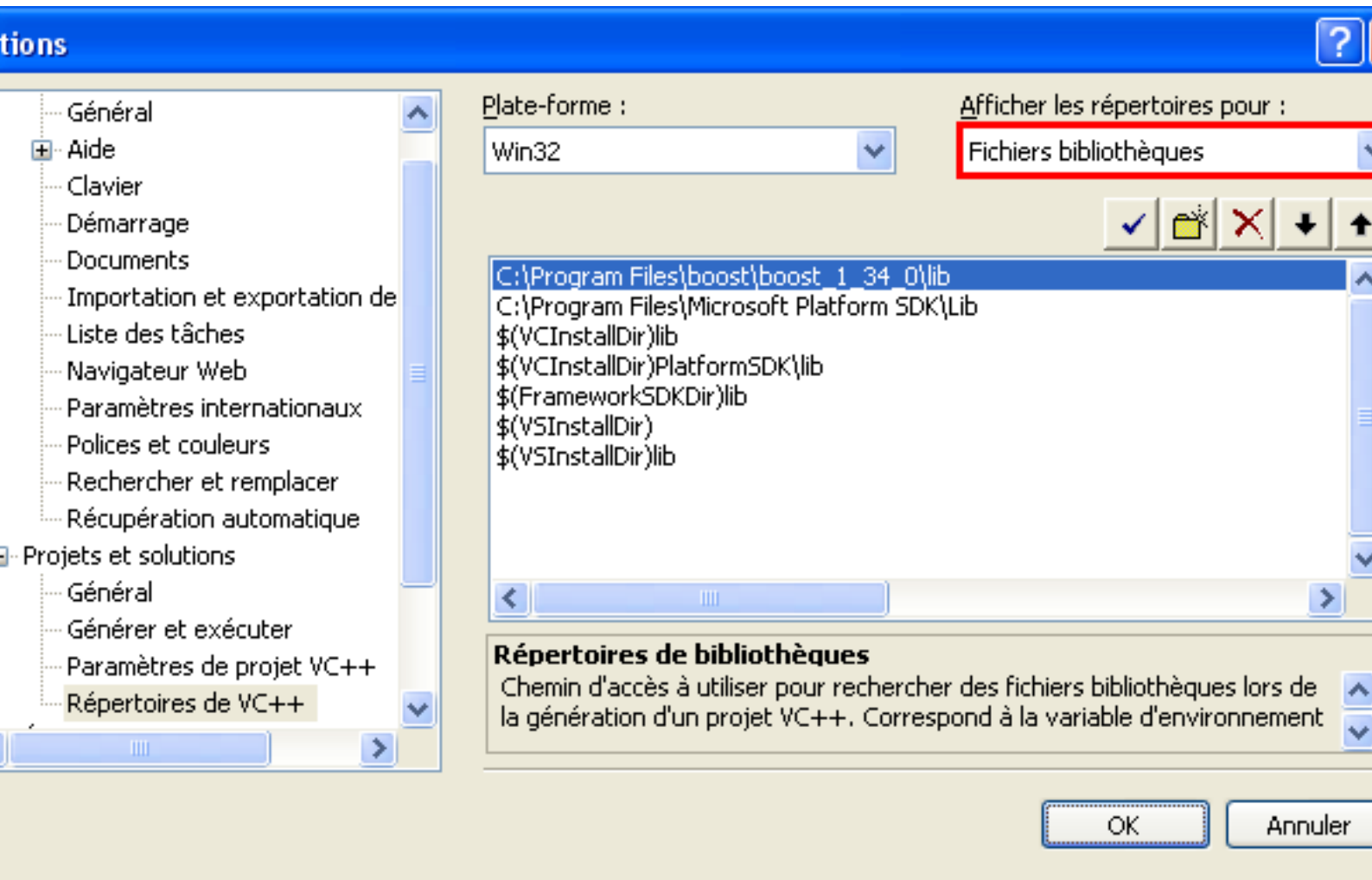
Dans le menu *Outils->Options...*, activez l'item *Projets et solutions->Répertoires de VC++*. Dans la boîte déroulante *Afficher les répertoires pour :*, choisissez *Fichiers Include* :



Configuration de l'INCLUDE_PATH

C'est ici que l'on peut configurer l'INCLUDE_PATH du compilateur (lorsqu'il est utilisé depuis l'IDE). Rajoutez donc le répertoire racine d'installation (*C:\Program Files\boost\boost_1_34_0* par défaut) et surtout pas *C:\Program Files\boost\boost_1_34_0\boost*, car le préfixe *boost/* est manuellement ajouté par le programmeur au niveau de la clause *#include* (par exemple, *#include <boost/shared_ptr.hpp>*). Pour pouvoir utiliser Boost.TR1, ajoutez aussi le sous-répertoire *boost/tr1* (*C:\Program Files\boost\boost_1_34_0\boost\tr1* par défaut).

Puis répétez l'opération avec le répertoire *C:\Program Files\boost\boost_1_34_0\lib* au niveau de la liste des *Fichiers bibliothèques*



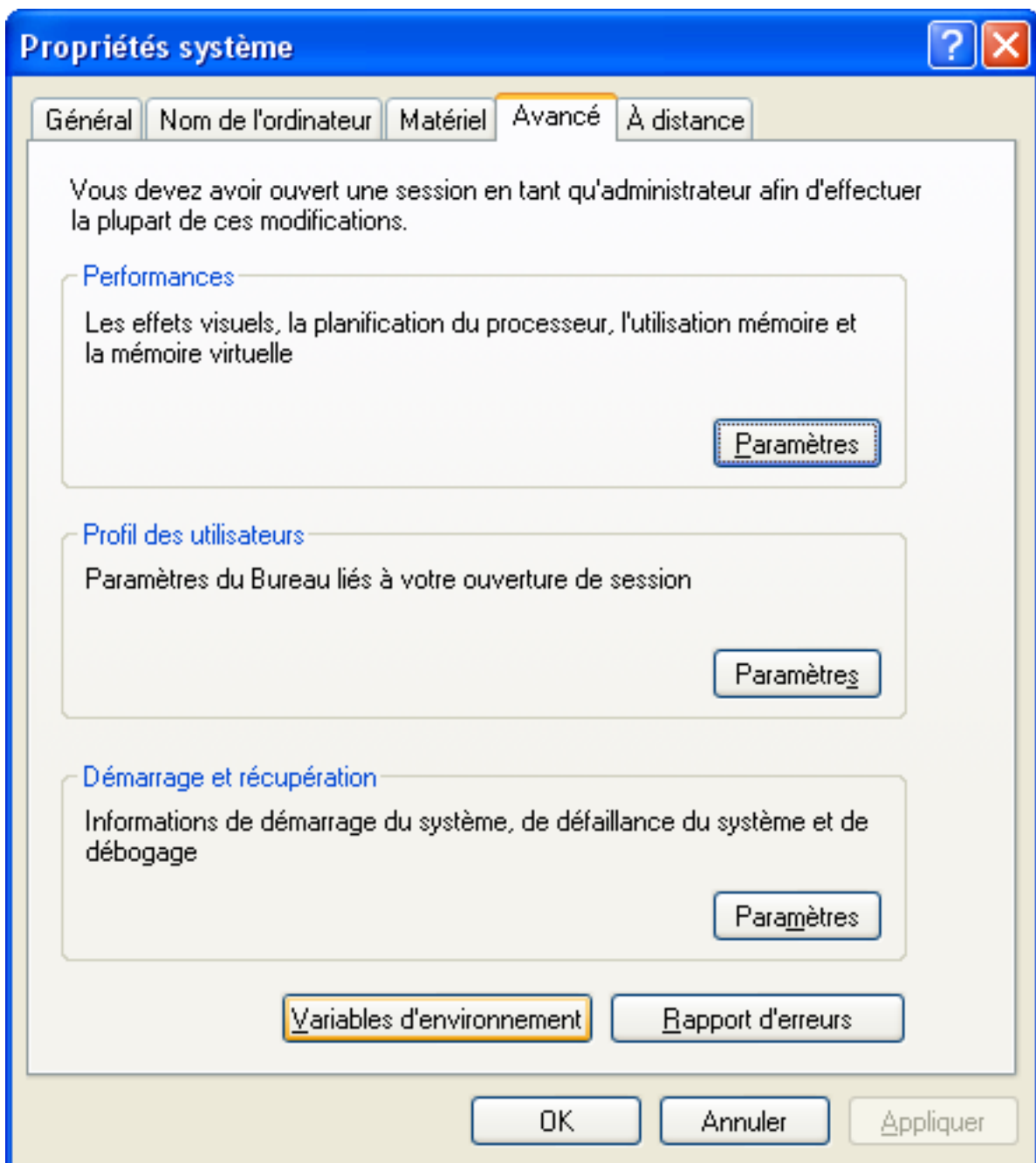
Configuration du LIB_PATH

Visual C++ est maintenant configuré pour pouvoir utiliser Boost et Boost.TR1 (si les répertoires renseignés sont corrects bien sûr).

Configuration du PATH de Windows

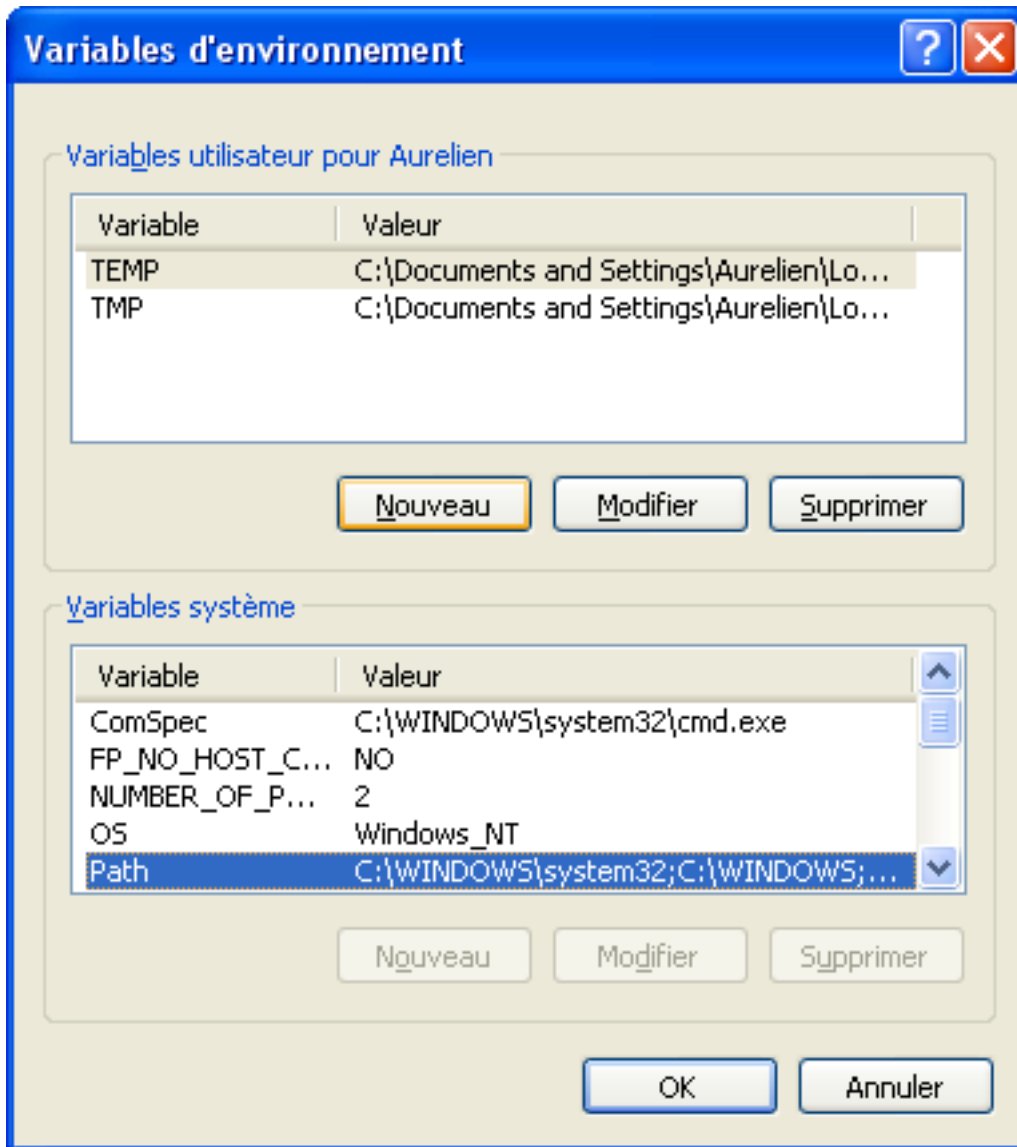
Visual C++ sait maintenant où trouver les fichiers d'en-tête et de liaison de Boost, et peut donc compiler des programmes qui s'en servent. Cependant, en fonction des options de compilation, ces programmes risquent de faire appel aux dlls contenues dans le répertoire *lib*. Et cette fois-ci, c'est Windows qui émettra une erreur au moment de l'exécution car il ne sait pas que ce répertoire contient les dlls en question. Il faut donc le lui dire à lui aussi, en renseignant ce répertoire dans le PATH.

Pour cela, allez dans les propriétés du Poste de travail (Touche Windows + Pause), cliquez sur l'onglet *Avancé* puis *Variables d'environnement* :



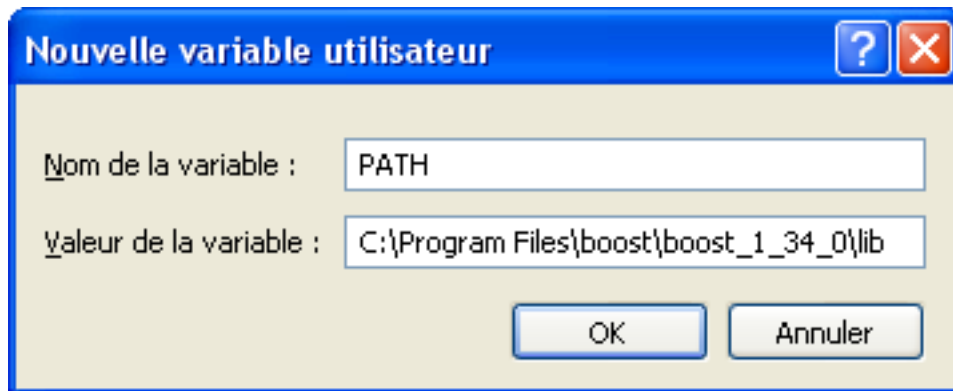
Propriétés système

Une fenêtre s'ouvre, vous donnant accès aux variables d'environnement locales à votre session et globales au système :



Variables d'environnement

La variable PATH est à la base une variable système. Sur l'image ci-dessus, il est interdit de la modifier car je ne travaille pas en tant qu'administrateur. Je crée donc une variable locale PATH, qui sera automatiquement ajoutée au contenu de la variable système du même nom :



Ajout des dlls Boost au PATH

Il suffit d'ajouter l'emplacement des dlls Boost à cette variable (*C:\Program Files\boost\boost_1_34_0\lib* dans mon cas) pour que Windows prenne désormais en compte ce répertoire lors de la recherche de dlls (entre autre), et puisse donc trouver les dlls Boost.

Si vous êtes administrateur et que vous souhaitez modifier de manière globale (pour tous les utilisateurs) la variable PATH, vous pouvez modifier la variable système existante en ajoutant le répertoire *<Boost>/lib* à la suite de ceux déjà présents en le séparant d'un point virgule ';'.

Programme d'exemple

Nous allons maintenant vérifier que tout est bien configuré en compilant un exemple utilisant boost::regex pour tester la validité d'une adresse mail. En réalité, nous allons utiliser std::tr1::regex grâce à Boost.TR1. Voici le code source, disponible en téléchargement en fin d'article :

test_regex.cpp

```
#include <iostream>
#include <string>

// #define BOOST_DYN_LINK // pour utiliser Boost sous forme de dll
#define BOOST_LIB_DIAGNOSTIC // pour afficher le nom du fichier lib utilisé
#include <tr1/regex>

bool is_valid_email( const std::string & Str )
{
    static std::tr1::regex email_pattern( "(\\w[-._\\w]*\\w@\\w[-._\\w]*\\w\\.\\.\\w{2,3})" );

    return std::tr1::regex_match( Str, email_pattern );
}

void test( const std::string & Str )
{
    std::cout << "Test de '" << Str << "' : ";
    std::cout << ( is_valid_email( Str ) ?
        "ok\n" :
        "erreur\n" );
}

int main()
{
    using namespace std;

    test( "contact@site-web.com" );
    test( "test-mail100@site.web.fr" );
    test( "mail@site.fr.nospam" );

    cout << "Entrez une adresse mail : ";
    string str;
    if ( cin >> str )
    {
        test( str );
    }
}
```

Ce code devrait compiler et linker sans problème si tout est bien configuré. Si vous obtenez une erreur du type :

```
fatal error C1083: Impossible d'ouvrir le fichier include : 'tr1/regex' : No such file or directory
```

C'est que vous avez mal configuré l'INCLUDE_PATH, et si vous obtenez l'erreur :

```
LINK : fatal error LNK1104: impossible d'ouvrir le fichier 'boost_regex-vc80-mt-gd-1_34.lib'
```

C'est que vous avez mal configuré le LIB_PATH, ou alors que vous n'avez pas téléchargé la version DLL de la bibliothèque que vous essayez d'utiliser.

Vous noterez à ce propos qu'un fichier .lib a été automatiquement ajouté à la liste des bibliothèques de liaisons par Boost (boost_regex-vc80-mt-gd-1_34.lib) via un #pragma spécifique à VC++ (voir le fichier *boost/config/auto_link.hpp*). La définition de BOOST_LIB_DIAGNOSTIC permet d'être informé de la bibliothèque utilisée pour l'édition de liens.

Pour plus d'information sur cette macro ainsi que sur BOOST_DYN_LINK, consultez la documentation de Boost à leur sujet : **Boost Configuration Reference : Macros for libraries with separate source code**.

Par défaut, en Debug comme en Release, le fichier .lib utilisé correspond à la version statique de la bibliothèque. Pour utiliser la version dynamique (c.a.d utiliser Boost.Regex sous forme de DLL), il convient de définir BOOST_DYN_LINK (ou BOOST_REGEX_DYN_LINK qui est spécifique à Boost.Regex) avant d'inclure boost/regex.hpp (via tr1/regex). En faisant cela, votre programme va être lié à une dll de Boost et vous pouvez donc vérifier que le PATH a été correctement configuré en exécutant le programme avec succès. Si Windows vous gratifie d'une erreur de ce style :

e - Composant introuvable

l'application n'a pas pu démarrer car boost_regex-vc80-mt-gd-1_34.dll est introuvable. La réinstallation de cette application est recommandée.



DLL Boost introuvable

C'est que la variable PATH est mal configurée.

CRT statique / dynamique (/MT[d], /MD[d])

Le but de ce tutoriel n'est pas de présenter en détail le principe de la CRT. Sachez juste que tout programme C/C++ (exe, dll, ...) compilé avec VC++ est lié à une bibliothèque que l'on appelle généralement CRT (*C/C++ RunTime*). Cette bibliothèque contient le code de la bibliothèque standard du C et du C++ (gestion de la mémoire, manipulation des fichiers, etc...). Elle existe en plusieurs versions : Debug et Release bien sûr, mais aussi sous forme de bibliothèque statique (.lib) ou dynamique (.dll). Nous avons donc 4 combinaisons possibles, identifiées par 4 options de compilations :

- /MD : CRT Release sous forme de dll (msvcp80.dll, msvcr80.dll)
- /MDd : CRT Debug sous forme de dll (msvcp80d.dll, msvcr80d.dll)
- /MT : CRT Release sous forme de lib static
- /MTd : CRT Debug sous forme de lib static

Pour plus d'informations, reportez-vous à la documentation de Visual C++ ([/MD, /MT, /LD \(Use Run-Time Library\)](#)).


Cette notion de CRT a son importance, car dans un programme, à moins de chercher les ennuis, il convient de faire en sorte que l'ensemble des modules compilés utilisent la même et unique CRT. Par module, comprenez un exe ou une dll. Dans notre cas, cela veut dire que si vous utilisez Boost sous forme de dll, vous devez veiller à compiler votre exécutable (ou votre module) avec la même option que la dll Boost utilisée. Et pour des raisons que je ne détaillerai pas ici, les dlls Boost sont compilées pour utiliser dynamiquement la CRT, c.a.d avec l'option /MDd en Debug et /MD en Release.

Si vous n'avez pas tout suivis, retenez juste que si vous utilisez Boost sous forme de dll, il convient de compiler votre programme avec l'option /MDd en Debug, et /MD en Release. D'ailleurs, si vous ne le faites pas, Boost vous rappellera à l'ordre en faisant échouer la compilation :


```
fatal error C1189: #error : "Mixing a dll boost library with a static runtime is a really bad idea..."
```

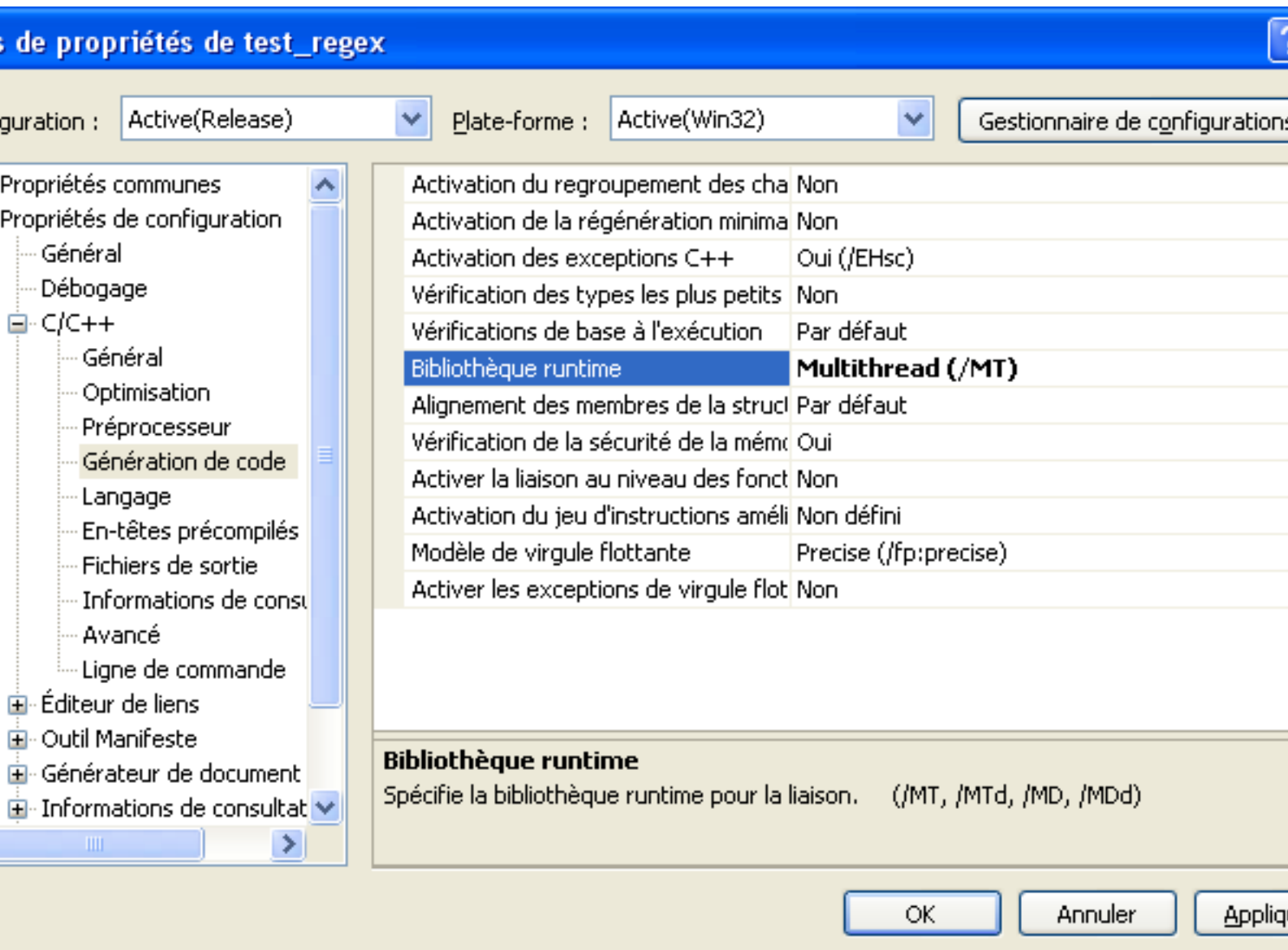
Si vous rencontrez cette erreur, ajustez les options de votre projet pour compiler en /MD[d] (la procédure est détaillée en fin de ce chapitre).

Au niveau du résultat final, la principale différence entre /MD[d] et /MT[d] est que dans le premier cas l'exécutable obtenu est plus petit **mais dépendant de la CRT sous forme de 2 dlls**, alors que dans le second le programme est plus gros, mais autonome. Le problème de la dépendance aux 2 dlls de la CRT est que depuis VC++ 8, ces dernières ne peuvent plus être simplement redistribuées aux côtés de votre exécutable et nécessitent d'être installées de manière spécifiques sur le système.

 **En utilisant Boost sous forme de dll, vous devrez compiler en /MD, et le programme obtenu ne pourra pas être facilement redistribué dans la mesure où il sera lié aux dlls Boost bien sûr, mais aussi à 2 dlls de VC++ 8 qui doivent être redistribuées au moyen d'un installateur (la simple copie dans le même répertoire que votre fichier exécutable ne fonctionne pas).**

Sachez aussi qu'indépendamment du fait que vous utilisiez Boost ou non, la configuration par défaut des projets Release de VC++ est /MD, c'est à dire d'utiliser la version dynamique de la CRT (dll). Si vous avez suivi, cela veut dire que votre exécutable compilé en Release ne peut pas être déployé simplement (nécessité d'installer la CRT sur les postes cibles).

 Je vous conseille donc dans la mesure du possible de toujours compiler vos versions de production (Release) en /MT, afin d'obtenir des exécutables "autonomes" faciles à déployer. Cette option de compilation se règle au niveau des propriétés C++ du projet. Menu Projet->Propriétés... :



Configuration : Active(Release) Plate-forme : Active(Win32) Gestionnaire de configurations

Propriétés communes
Propriétés de configuration

- Général
- Débogage
- C/C++
 - Général
 - Optimisation
 - Préprocesseur
 - Génération de code
 - Langage
 - En-têtes précompilés
 - Fichiers de sortie
 - Informations de cons
 - Avancé
 - Ligne de commande
- Éditeur de liens
- Outil Manifeste
- Générateur de document
- Informations de consultat

Activation du regroupement des cha	Non
Activation de la régénération minima	Non
Activation des exceptions C++	Oui (/EHsc)
Vérification des types les plus petits	Non
Vérifications de base à l'exécution	Par défaut
Bibliothèque runtime	Multithread (/MT)
Alignement des membres de la struct	Par défaut
Vérification de la sécurité de la mém	Oui
Activer la liaison au niveau des fonct	Non
Activation du jeu d'instructions améli	Non défini
Modèle de virgule flottante	Precise (/fp:precise)
Activer les exceptions de virgule flot	Non

Bibliothèque runtime
Spécifie la bibliothèque runtime pour la liaison. (/MT, /MTd, /MD, /MDd)

OK Annuler Appliq

Modification des propriétés pour compiler en /MT

Dans les *Propriétés de configuration->C/C++->Génération de code->Bibliothèque runtime*, modifier la valeur par défaut *DLL multithread (/MD)* en *Multithread (/MT)* comme sur l'image ci-dessus.

Recompilez le projet, vous devriez obtenir un exécutable qui ne dépend d'aucune dll liée à Boost ou VC++, ce qui le rend facilement redistribuable (vous pouvez vous en assurer au moyen de l'utilitaire **Dependency Walker**).

Conclusion

Vous pouvez télécharger le projet d'exemple avec un exécutable compilé en Release /MT ici : [test_regex.zip](#) ([miroir](#)).

J'espère qu'à l'issue de ce tutoriel vous serez parvenus à installer et utiliser avec succès la bibliothèque Boost avec Visual C++. N'hésitez pas à me contacter ou à **laisser un commentaire** au sujet de cet article. Merci pour votre lecture.

