Un aperçu des possibilités des bibliothèques de Boost

par Matthieu Brucher (http://matthieu-brucher.developpez.com/) (Blog)

Date de publication :

Dernière mise à jour : 15/04/2008

Une page d'introduction à l'étude des bibliothèques de Boost.

Boost est un ensemble de bibliothèques bâties sur le standard du C++. Chaque bibliothèque répond à un besoin précis d'une manière homogène et réutilisable. Plusieurs articles seront écrits pour vous permettre de vous familiariser avec certaines parties sélectionnées pour leurs capacités et leur utilité.

Sous Linux, il n'est pas nécessaire de télécharger puis compiler et installer Boost, il se trouve présent sur toutes les distributions actuelles. En revanche, dans le cas où une version plus récente serait nécessaire, il vous faut télécharger et installer la bibliothèque.

I - Téléchargement

Le téléchargement de la dernière version de Boost s'effectue directement sur Sourceforge à **cette adresse**. Le paquet **boost** est celui qu'il faut télécharger.

Il n'est pas nécessaire de compiler Boost dans tous les cas. Cela est nécessaire uniquement pour certaines bibliothèques telles que Python ou Regex.

II - Compilation

La compilation de Boost nécessite l'utilisation de **bjam**, téléchargeable sur Sourceforge. L'exécutable doit être accessible en ligne de commande.

Une fois téléchargé, l'archive de Boost doit être décompressée, par exemple dans /home/user/local/src ou c:\. A l'aide de la ligne de commande, déplacez-vous dans ce dossier.

Boost peut être compilé avec plusieurs outils, dont gcc et Microsoft Visual Studio. Pour cela, il faut ajouter l'option --toolset=msvc sous Windows, sous Linux, gcc est le compilateur par défaut.

*Il est possible d'installer Boost dans un dossier spécifique à l'aide de l'option **install**. Dans ce cas, l'otpion --prefix=dossier indiquera quel est ce dossier. Si l'option **stage** est utilisée, seules les bibliothèques seront compilées dans le dossier ./stage/lib.*

Un exemple de ligne de commande pour compiler uniquement la bibliothèque Python sous Windows :

```
bjam --toolset=msvc --with-python stage
```

Un autre exemple d'installation sous Linux dans le dossier /home/user/local :

```
bjam --prefix=/home/user/local install
```

*Pour utiliser MinGW, on utilisera le toolset **mingw**, pour le compilateur Borland, il s'agira de **bcc**. Pour le compilateur d'Intel, il s'agira de **intel**.*

III - Utilisation

L'utilisation principale de Boost est sous la forme des fichiers d'entêtes qu'il propose. Ces fichiers sont dans le sous-dossier **boost**. Si vous avez installé Boost dans un autre sous-dossier (sous Windows ou sous Linux), vous devez ajouter le dossier **include** de ce sous-dossier à la variable d'environnement INCLUDE. Si vous utilisez Visual Studio, ajoutez-le dans la liste des dossiers d'entêtes dans les options de l'IDE.

Si vous utilisez les bibliothèques compilées de Boost, faites de même avec le dossier **bin** à ajouter à la variable PATH sous Windows et le dossier **lib** à ajouter à la variable LD_LIBRARY_PATH sous Linux.

*Dans le cas de l'utilisation de la cible **stage** lors de la compilation ou si vous n'utilisez pas de bibliothèque compilée de Boost, les en-têtes ne sont pas déplacés, c'est le dossier boost_1_xx qui servira de dossier pour la variable INCLUDE et c'est le dossier boost_1_xx/stage/lib qui servira pour la variable LIB ou LD_LIBRARY_PATH.*

Une fois ces dossiers ajoutés à la configuration de votre système par ces différents moyens, vous devriez être capables d'utiliser Boost depuis n'importe quel IDE ou avec tout outil de compilation.

IV - Contenu de Boost

- **any** est un pointeur vers tout type de données, mais sécurisé au niveau du type
- **array** décrit un conteneur de taille fixe **(tutoriel)**
- **assign** utilise le symbole **,** pour faciliter l'assignation de données à un conteneur **(tutoriel)**
- **bind** et **mem_fn** sont des pointeurs de fonctions et de méthodes
- **call_traits** définit des *traits* pour les types de données
- **compatibility** définit certains entêtes manquants sur certaines plateformes
- **compressed_pair** est une optimisation par rapport à std::pair
- **concept check** permet de vérifier qu'une classe vérifie un concept (utilisé en méta programmation)
- **config** permet à Boost de s'en sortir avec les différents compilateurs supportés
- **conversion** effectue des conversions lexicales et polymorphiques **(tutoriel)**
- **crc** calcule le code de redondance cyclique d'une variable
- **date_time** permet de travailler sur plusieurs systèmes de temps et date
- **dynamic_bitset** est une version dynamique de std::bitset
- **enable_if** active ou désactive certaines fonctions grâce au principe SFINAE
- **filesystem** permet de travailler sur le système de fichiers de manière portable **(tutoriel)**
- **foreach** définit la macro BOOST_FOREACH permettant d'itérer sur un conteneur
- **format** crée un chaîne de caractères à partir d'un format donné
- **function** encapsule une fonction ou une classe foncteur
- **functional** propose une amélioration des adaptateurs fonctions/objets
- **functional/hash** calcule une valeur de hash pour tout type de données
- **graph** expose une bibliothèque complète de traitement sur les graphes **(tutoriel)**
- **integer** décrit les nouveaux types d'entiers de la norme C99, entre autres
- **interval** décrit des fonctions sur des intervalles mathématiques
- **in_place_factory**, **typed_in_place_factory** permet d'éviter la construction par copie
- **io state savers** permet de sauvegarder l'état d'un flux et le restituer
- **iostreams** facilite la création de nouveaux flux
- **iterators** facilite la création d'itérateurs sur de nouveaux conteneurs
- **lambda** permet d'utiliser des variables sans les nommer
- **math** est un ensemble de plusieurs contributions
- **math/common_factor** calcule le PGCD et le PPCM
- **math/octonion** propose une classe d'octonions
- **math/quaternion** propose une classe de quaternions
- **math/special_functions** propose plusieurs fonctions mathématiques usuelles
- **minmax** calcule le min et le max d'un conteneur en même temps
- **mpl** est un ensemble d'outils de méta programmation
- **multi_array** est un conteneur multi-dimensionnel
- **multi_index** permet la création de conteneurs indexés par plusieurs types de données
- **numeric/conversion** expose un certain nombre d'outils de conversion numérique
- **operators** définit des opérateurs mathématiques à partir d'autres opérations existantes

- **optional** propose une solution lorsqu'une fonction ne retourne pas toujours une valeur
- **parameter** permet de définir des arguments formels
- **pointer container** définit des conteneurs de pointeurs qui seront supprimés
- **pool** gère un pool de mémoire
- **preprocessor** propose différents outils additionnels pour le préprocesseur
- **program_options** analyse les données passées en argument sur la ligne de commande
- **property map** définit des interfaces entre clés et valeurs **(tutoriel)**
- **python** permet la communication entre C++ et Python
- **random** génère plusieurs types de nombres aléatoires **(tutoriel)**
- **range** travaille sur un intervalle d'itérateurs
- **rational** définit une classe de nombres rationnels
- **ref** permet de passer des références sur des objets
- **regex** est une bibliothèque d'expressions régulières
- **serialization** permet de sauvegarder et de charger des objets quelconques
- **signals** gère des signaux et des sloats
- **smart_ptr** définit des pointeurs intelligents **(tutoriel)**
- **statechart** définit des machines à états finis
- **static_assert** permet d'exécuter des assertions à la compilation **(tutoriel)**
- **spirit** est un parseur LL
- **string_algo** définit des algorithmes pour les chaînes de caractères
- **test** gère des tests unitaires
- **thread** permet d'utiliser les threads de manière portable **(tutoriel)**
- **timer** gère des événements
- **tokenizer** découpe une chaîne en jetons/tokens
- **TR1** définit les outils du rapport TR1
- **tribool** définit un booléen à trois états
- **tuple** permet à une fonction de retourner plusieurs valeurs
- **type_traits** définit des types basés sur d'autres types
- **typeof** retourne le type d'un objet et permet d'en construire de nouveaux
- **uBLAS** est une bibliothèque matricielle
- **utility** définit plusieurs utilitaires
- **value_initialized** initialise de manière consistante des données
- **variant** permet à une variable de contenir plusieurs types de données
- **wave** est une implémentation standard du préprocesseur C99/C++
- **xpressive** est une autre bibliothèque d'expressions régulières, permettant de se référer l'une à l'autre