

Boost.PropertyMap ou comment associer des objets clé à des objets valeur

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

Date de publication : 04/12/2006

Dernière mise à jour :

Les Property Maps de Boost est un moyen d'utiliser dans des algorithmes des propriétés différentes. Par exemple, dans les algorithmes de graphes, les Property Maps permettent de donner à ceux-ci les capacités des arcs.

Introduction

I - Les concepts

I-A - Readable Property Map

I-B - Writable Property Map

I-C - Read/Write Property Map

I-D - Lvalue Property Map

II - Les classes template de Property Map

II-A - identity_property_map

II-B - iterator_property_map<RandomAccessIterator, OffsetMap, T, R>

II-C - associative_property_map<UniquePairAssociativeContainer>

II-D - const_associative_property_map<UniquePairAssociativeContainer>

II-E - vector_property_map<T, IndexMap = identity_property_map>

II-F - Transformer un pointeur en Property Map

III - Conclusion

Introduction

Boost.PropertyMap implémente quatre concepts à l'aide de plusieurs classes template. Chacune de ces classes implémente les concepts nécessaires à leur utilisation dans des algorithmes complexes, et ceci de manière interchangeable, qu'elles soient stockées au départ dans des `std::map` ou de simple tableaux.

I - Les concepts

Au nombre de 4, ces concepts proposent une interface aux classes qui les implémentent, selon leurs capacités.

I-A - Readable Property Map

Cette Property Map peut être lue à l'aide de la fonction `get()`. Outre cette fonction, plusieurs types sont indispensables, dépendant de **PMap**, le type de la Property Map.

Code	correspond	Explication
<code>boost::property_traits<PMap>::value_type</code>	de	de la propriété valeur
<code>boost::property_traits<PMap>::reference</code>	de	qui référence peut se convertir en un type de valeur
<code>boost::property_traits<PMap>::key_type</code>	de	d'effectuer une recherche d'une valeur. Peut valoir void si le type de clé est template
<code>boost::property_traits<PMap>::category</code>	de	de la catégorie, doit pouvoir se convertir en <code>readable_property_map_tag</code>

Une seule fonction doit être définie.

No. de ret.	Typ.	Description
<code>get(const PropertyMap& pm, const Key& key)</code>	reference	Retourne la valeur dans la Property Map pm associée à la clé key

I-B - Writable Property Map

Cette Property Map est capable d'associer à une clé une valeur grâce à la fonction *put()*. Plusieurs types sont aussi indispensables.

Code	correspond	Explication
<code>boost::property_traits<PM>::value_type</code>	de	de la propriété valeur
<code>boost::property_traits<PM>::key_type</code>	de	d'effectuer une recherche d'une valeur. Peut valoir void si le type de clé est <code>template</code>
<code>boost::property_traits<PM>::category</code>	de	de la catégorie, doit pouvoir se convertir en <code>writable_property_map_tag</code>

Une seule fonction doit être définie.

No	pression	type	Description
	<code>put(pmap, key, Valueval)</code>	void	Affecte dans la Property Map pmap la valeur val à la clé key

I-C - Read/Write Property Map

Ce concept hérite des 2 précédents, avec une petite différence pour le type de catégorie.

Code	correspond	Explication
<code>boost::property_traits<PM>::category</code>	de	de la catégorie, doit pouvoir se convertir en <code>lvalue_property_map_tag</code>

I-D - Lvalue Property Map

Une telle Property Map peut hériter de **Readable Property Map** ou de **Read/Write Property Map** selon qu'elle soit constante ou non - mutable -. On peut accéder à une référence indexée par une clé grâce à une interface particulière.

Code	correspondance	Explication
<code>boost::property_map<PM, traits<PM>></code>	de référence	qui peut se convertir en un type de valeur
<code>boost::property_map<PM, traits<PM>, category></code>	de la catégorie,	doit pouvoir se convertir
<code>en writable_property_map_tag</code>		

Une fonction d'accessor particulier doit être défini.

No	Accessor	Type de retour	Description
1	<code>accessor_key</code>	<code>key_type</code>	Retourne une référence à l'objet
2	<code>accessor_value</code>	<code>value_type</code>	Retourne une référence à l'objet
3	<code>accessor_key_const</code>	<code>key_type</code>	Retourne une référence à l'objet
4	<code>accessor_value_const</code>	<code>value_type</code>	Retourne une référence à l'objet
5	<code>accessor_key_mutable</code>	<code>key_type</code>	Retourne une référence à l'objet
6	<code>accessor_value_mutable</code>	<code>value_type</code>	Retourne une référence à l'objet

II - Les classes template de Property Map

II-A - identity_property_map

Cette Property Map est une **Readable Property Map** qui retourne la clé donnée en entrée.

Code	Description
<code>st::property_traits<identity_property_map>::value_type</code>	est void , les fonctions de la Property Map sont templatées
<code>st::property_traits<identity_property_map>::key_type</code>	est void , les fonctions de la Property Map sont templatées
<code>st::property_traits<identity_property_map>::category</code>	est void , les fonctions de la Property Map sont templatées

Quelques fonctions membres indispensables sont utilisables.

Membre	Description
<code>identity_property_map()</code>	Constructeur par défaut
<code>identity_property_map(const identity_property_map& x)</code>	Constructeur par copie
<code>template <class T> T operator[](T x) const</code>	Retourne une copie de l'élément x
<code>template <class T> T get(const identity_property_map& pmap, T x)</code>	Fonction non-membre qui retourne une copie de x

II-B - iterator_property_map<RandomAccessIterator, OffsetMap, T, R>

Cette classe implémente le concept de **Lvalue Property Map**. À partir d'un itérateur à accès aléatoire, on peut accéder à n'importe quel élément avec un entier - comme pour un itérateur sur des vecteurs standards, on peut l'avancer ou le reculer à l'aide des additions, ici, c'est la même chose -. Le deuxième élément indispensable est un outil pour convertir la clé en cet entier qu'on va utiliser. Les autres paramètres ont des valeurs par défaut décrites dans le tableau suivant.

Paramètre	Description	Valeur par défaut
<code>Iterator</code>	Modèle d'itérateur à accès aléatoire	
<code>OffsetMap</code>	Modèle de Readable Property Map et la valeur associée doit pouvoir être convertie en type de différence pour l'itérateur précédent	
<code>std::iterator_traits<RandomAccessIterator>::value_type</code>	de valeur de l'itérateur	
<code>std::iterator_traits<RandomAccessIterator>::reference</code>	de référence de l'itérateur	

Et voici les membres associés à cette Property Map :

Membre	Description
<code>property_traits<iterator_property_map<Iterator, OffsetMap>::value_type</code>	
<code>iterator_property_map(i)</code>	OffsetMap est construit par défaut
<code>iterator_property_map(i, OffsetMap m)</code>	Iterator
<code>reference operator[](difference_type v)</code>	Return type une référence selon la différence donnée

II-C - `associative_property_map<UniquePairAssociativeContainer>`

Cet adaptateur permet de transformer des objets répondants aux critères des conteneurs associatifs à paires - une paire clé/valeur - et des conteneurs associatifs à clé unique, comme `std::map` pour les transformer en **Lvalue Property Map**.

Paramètre	Description	Valeur par défaut
<code>UniquePairAssociativeContainer</code>	Conteneur aux	

	critères des conteneurs associatifs à paires et à clé unique
--	---

Et voici les membres associés à cette Property Map :

Membre	Description
<code>const</code>	Constructeur par défaut
<code>operator()</code>	Retourne une référence selon la différence donnée. Les types sont dérivés de <code>UniquePairAssociativeContainer</code>

II-D - `const_associative_property_map<UniquePairAssociativeContainer>`

Cet adaptateur diffère du précédent en ne proposant qu'un accesseur retournant un élément constant.

Membre	Description
<code>const</code>	Retourne une référence selon la différence donnée. Les types sont dérivés de <code>UniquePairAssociativeContainer</code>

II-E - `vector_property_map<T, IndexMap = identity_property_map>`

Cette classe particulière se trouve entre une Property Map associative et une Property Map itérative. À la différence de cette dernière, le nombre d'éléments stockés peut varier. En fait, les données stockées sous la forme d'un vecteur qui croît à la demande. C'est à nouveau une implémentation du modèle **Lvalue Property Map**.

Paramètre	Description
<code>T</code>	Le type de valeur par défaut

Implémente le modèle de **Readable Property Map** et doit être convertible en `std::vector<T>::size_type`

Et voici les membres associés à cette Property Map :

Membre	Description
<code>vector<property_map></code>	Constructeur <code>IndexMap& index = IndexMap()</code> en paramètre
<code>vector<property_map></code>	Constructeur <code>initial_size, const IndexMap& index = IndexMap()</code> de la Property Map
<code>vector<property_map></code>	Constructeur <code>property_map&</code>
<code>vector<property_map></code>	Constructeur <code>operator=(const property_map&)</code>
<code>vector<property_map></code>	Constructeur <code>operator()</code>
<code>reference</code>	Constructeur <code>operator() const</code> d'accès
<code>std::vector<T>::reverse_iterator</code>	Constructeur <code>storage_begin()</code> itérateurs sur la Property Map
<code>std::vector<T>::iterator</code>	Constructeur <code>storage_end()</code>
<code>std::vector<T>::const_iterator</code>	Constructeur <code>storage_begin()</code>
<code>std::vector<T>::const_iterator</code>	Constructeur <code>storage_end()</code>
<code>void</code>	<code>reserve(unsigndet size)</code> de réserver une taille pour éviter les changements

	de taille coûteux
--	----------------------

II-F - Transformer un pointeur en Property Map

Afin d'utiliser un pointeur **T***, des fonctions put et get existent pour palier l'absence naturelle de ces fonctions pour les pointeurs, ainsi qu'une spécialisation de **property_traits** afin de répondre aux nécessités d'une Read/Write Property Map.

III - Conclusion

Chaque Property Map, sauf **identity_property_map**, peut être construite à l'aide d'un **make_xxx** où xxx est le type à construire. Ces classes sont utilisées couramment dans Boost.Graph que nous étudierons par la suite, mais vous pouvez les utiliser dès que vous voulez utiliser des propriétés associées à un type de données.

