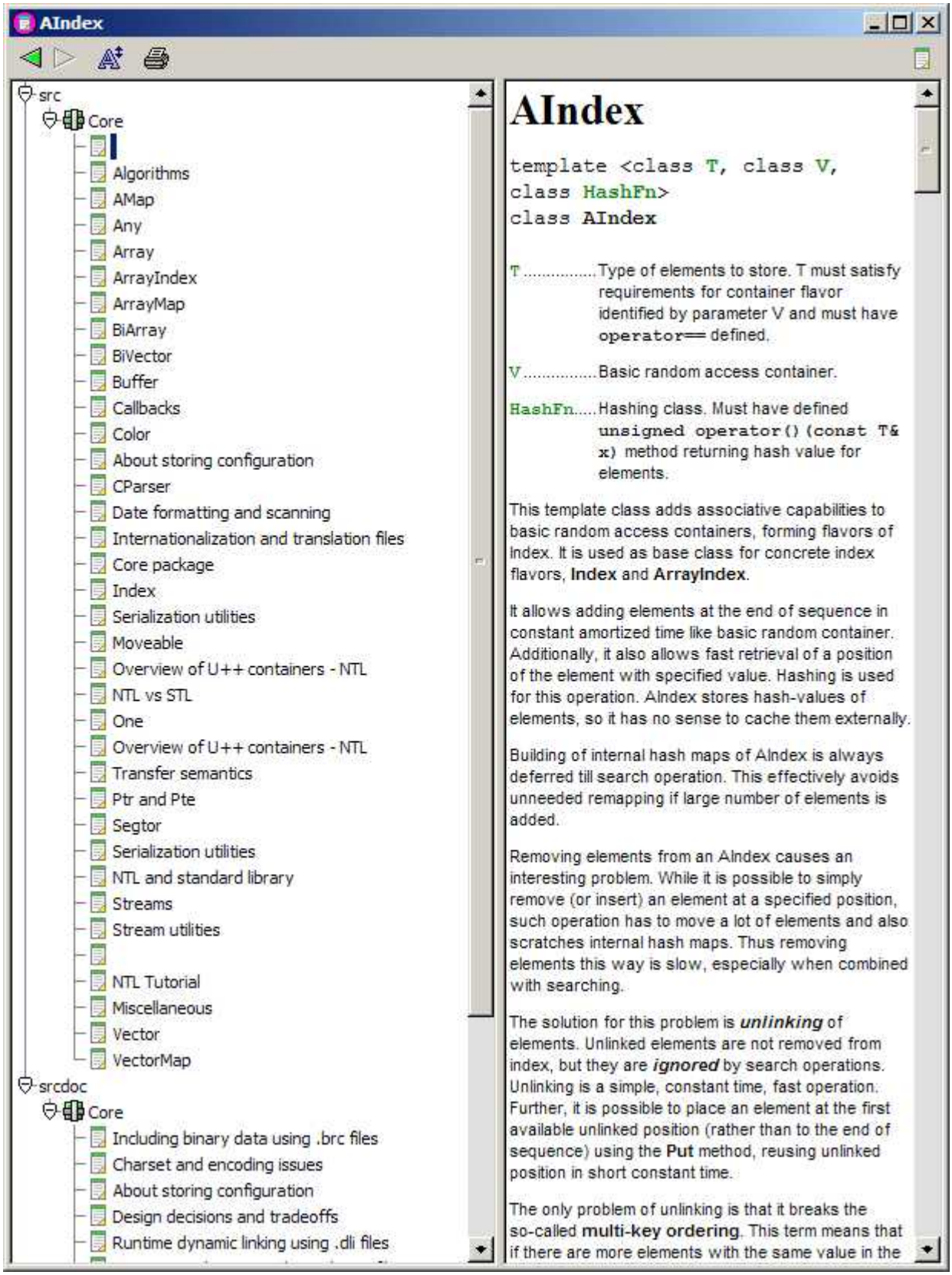
Subject: documentation and uvs
Posted by hojtsy on Mon, 30 Jan 2006 19:11:24 GMT
View Forum Message <> Reply to Message

I see all these files in the src.tpp direcory in uvs. Some of them are also visible in srcdoc.tpp, but I edited the version in the src.tpp directory. There are other strange things you can see in the screenshot: There are two "Overview of U++ containers" pages, two "Serialization" pages, some pages don't have a name, etc. Some pages are also visible in src.tpp and srcdoc.tpp. After looking more closely at this confusion, it is possible that I was editing some obsolete versions of those files which should have not been visible in src.tpp.

## File Attachments

1) docs_in_uvs.jpg, downloaded 2252 times


AIndex
src
Core
Algorithms
AMap
Any
Array
ArrayIndex
ArrayMap
BiArray
BiVector
Buffer
Callbacks
Color
About storing configuration
CParser
Date formatting and scanning
Internationalization and translation files
Core package
Index
Serialization utilities
Moveable
Overview of U++ containers - NTL
NTL vs STL
One
Overview of U++ containers - NTL
Transfer semantics
Ptr and Pte
Segtor
Serialization utilities
NTL and standard library
Streams
Stream utilities
NTL Tutorial
Miscellaneous
Vector
VectorMap
srcdoc
Core
Including binary data using .brc files
Charset and encoding issues
About storing configuration
Design decisions and tradeoffs
Runtime dynamic linking using .dli files
AIndex
template <class T, class V,
class HashFn>
class AIndex
T ..... Type of elements to store. T must satisfy requirements for container flavor identified by parameter V and must have operator== defined.
V ..... Basic random access container.
HashFn..... Hashing class. Must have defined unsigned operator()(const T& x) method returning hash value for elements.
This template class adds associative capabilities to basic random access containers, forming flavors of Index. It is used as base class for concrete index flavors, Index and ArrayIndex.
It allows adding elements at the end of sequence in constant amortized time like basic random container. Additionally, it also allows fast retrieval of a position of the element with specified value. Hashing is used for this operation. AIndex stores hash-values of elements, so it has no sense to cache them externally.
Building of internal hash maps of AIndex is always deferred till search operation. This effectively avoids unneeded remapping if large number of elements is added.
Removing elements from an AIndex causes an interesting problem. While it is possible to simply remove (or insert) an element at a specified position, such operation has to move a lot of elements and also scratches internal hash maps. Thus removing elements this way is slow, especially when combined with searching.
The solution for this problem is unlinking of elements. Unlinked elements are not removed from index, but they are ignored by search operations. Unlinking is a simple, constant time, fast operation. Further, it is possible to place an element at the first available unlinked position (rather than to the end of sequence) using the Put method, reusing unlinked position in short constant time.
The only problem of unlinking is that it breaks the so-called multi-key ordering. This term means that if there are more elements with the same value in the