Derek Barnett – 22 March 2011

Porting to BamTools 1.x

This document describes the process of porting applications from BamTools 0.x (BT-0x) to BamTools 1.x (BT-1x).

Introduction

Our development philosophy with BamTools so far has been to allow periodic breaks in binary compatibility while avoiding source incompatibilities at all costs. (Changes in my codebase shouldn't force you to change your code.) However, this has led to a few operations growing overly complex and easy to get wrong. See the boolean parameter lists in BT-0x's BamReader::Open() or BamMultiReader::Open() as prime examples. In the interest of providing API methods that are easier to read, harder to get wrong, and more flexible going forward - the decision has been made to force (what we hope to be) a one-time break in source compatibility.

Therefore, BT-1x and beyond will be neither binary- nor source-compatible with the earlier BT-0x versions. Your programs compiled using BT-0x classes will not work with the new libraries. There are a few key operations that will need your code to be modified. It may seem like a nuisance to revisit working code, but the breaks in your codebase should only occur around the opening of BAM files. An explanation of the new methods is given below.

BT-1x adds some additional features and marks some methods as deprecated. These deprecated methods will remain available throughout the BT-1x series, but may be removed sometime down the road (BT-2x or later). While you are porting to BT-1x, we recommend making the switch with these methods as well. Deprecated methods are listed below along with the preferred new method.

BamReader

In BT-0x, opening a BAM file and optional index was performed in one step. While this seems convenient, the parameter list was a mess of default parameters and “naked” boolean values, leading to code like this:

```
BamReader reader1;
BamReader reader2;
reader1.Open(“data.bam”, “”, true, true);
reader2.Open(“data.bam”, “”, false, true);
```

Is it obvious what the two methods do differently? Parameter lists like this are prone to copy-and-paste bugs and quickly become annoying to someone maintaining your code (possibly you) .

To address this, and to allow for more flexible configurations of the BamReader as we go forward, we have separated this method into several methods.

Derek Barnett – 22 March 2011

BT-0x

```
bool Open(const string& filename,
          const string& indexFilename = "",
          const bool lookForIndex = false,
          const bool preferStandardIndex = false);
```

BT-1x

```
bool Open(const string& filename);
bool OpenIndex(const string& indexFilename);
bool LocateIndex(const BamIndex::IndexType& preferredType = BamIndex::STANDARD);
```

The ambiguity in the BT-0x method lies in index file handling. A few example cases will be illustrated here: (Note – most of the boolean return values will be ignored to simplify the illustations, but we recommend that they be used where applicable)

When you only need a BAM file, with no index:

```
reader.Open(bamFilename);
```

When you have an explicit index filename immediately at hand:

```
reader.Open(bamFilename);
reader.OpenIndex(indexFilename);
```

There is a new BamReader::LocateIndex() method that looks in the BAM file's directory for a matching index file and returns true or false, depending on whether one could be loaded. This can be used in cooperation with the existing BamReader::CreateIndex() method, when you definitely need an index, whether one exists already or not:

```
reader.Open(bamFilename);
if ( !reader.LocateIndex() )
    reader.CreateIndex();
```

When you have multiple index formats available for your BAM file, but prefer to use a specific one in this case:

```
reader.Open(bamFilename);
if ( !reader.LocateIndex(BamIndex::BAMTOOLS) )
    reader.CreateIndex(BamIndex::BAMTOOLS);

// Note that the type passed to LocateIndex() is a “preferred” type.  If this
// format is not found, the method will look for any other index files that
// are valid for this BAM file.
```

BamMultiReader

The changes to BamMultiReader mirror those made to BamReader.

BT-0x

```
bool Open(const vector<string>& filenames,
          bool openIndex = true,
          bool coreMode = false,
          bool preferStandardIndex = false);
```

BT-1x

```
bool Open(const vector<string>& filenames);
bool OpenIndexes(const vector<sstring>& indexFilenames);
bool LocateIndexes(const BamIndex::IndexType& preferredType = BamIndex::STANDARD);
```

The coreMode parameter in the Open() call is no longer necessary.

See BamReader section above for examples of different use cases for opening files. The concepts behind the methods are the same, only in this case, operating on multiple files. For example, BamMultiReader::LocateIndexes() will return false if ANY files lack an index, and BamMultiReader::CreateIndexes() will ONLY create indexes for readers that lack an index already. So the BamMultiReader code looks similar to the BamReader code to safely accomplish the same task (e.g. you need an index for your BAM file(s), regardless of whether it exists already or not) :

```
BamMultiReader reader;
reader.Open( filenames );
if ( !reader.LocateIndexes() )
    reader.CreateIndexes();
```

BamWriter

BamWriter received a small change to Open() as well. The compression switch has been moved to its own dedicated method.

BT-0x

```
bool Open(const string& filename,
          const SamHeader& samHeader,
          const RefVector& referenceSequences,
          bool writeUncompressed = false);
```

BT-1x

```
bool Open(const string& filename,
          const SamHeader& samHeader,
          const RefVector& referenceSequences);
void SetCompressionMode(const CompressionMode& compressionMode);
```

This method uses a newly introducted enum, BamWriter::CompressionMode, to specify the desired behavior. This makes it harder to change by accident but more importantly - it leaves open the flexibility in the future to implement various compression schemes, instead of just the simple true/false, on/off setup. The only real caveat with this new method is that it must be called **before** the output BAM file is opened.

```
BamWriter writer;
writer.SetCompressionMode(BamWriter::Uncompressed);
writer.Open(filename, header, references);
```

Deprecated methods

BamAlignment: Alignment Flag Modifiers

Some of the alignment flag modification methods have been deprecated in favor of methods that mirror the query methods:

Deprecated:	**Instead Use:**
`void SetIsMateUnmapped(bool ok)`	`void SetIsMateMapped(bool ok)`
`void SetIsSecondaryAlignment(bool ok)`	`void SetIsPrimaryAlignment(bool ok)`
`void SetIsUnmapped(bool ok)`	`void SetIsMapped(bool ok)`

In these cases, the deprecated method is simply a complement of the preferred method.

So instead of:

```
BamAlignment al;
al.SetIsUnmapped(true);
```

You would use:

```
BamAlignment al;
al.SetIsMapped(false);
```

And so on...

BamAlignment – Tag Data Access

Some of the BAM tags have dedicated methods in the API (e.g. BamAlignment::GetReadGroup()). These work perfectly fine, but in fact use the preferred method under the hood anyway. Retaining these tag-specific methods simply adds unecessary clutter to the interface since we are not going to implement a method for every possible tag.

Deprecated:

```
bool GetEditDistance(uint32_t& editDistance) const
bool GetReadGroup(string& readGroup) const
```

Instead Use:

```
BamAlignment al;
al.GetTag("NM", editDistance);
al.GetTag("RG", readGroup);
```

BamReader & BamMultiReader – Checking For Index

The original way to check for available index data was to use BamReader::IsIndexLoaded() or BamMultiReader::IsIndexLoaded(), respectively. However, the introduction of the BamIndex::IndexCacheMode left these methods with names that were possibly confusing or misleading about the current memory status. Index data may in fact be available even though it is technically not “loaded”.

Deprecated:	Instead Use:
`bool BamReader::IsIndexLoaded(void) const`	`bool BamReader::HasIndex(void) const`
`bool BamMultiReader::IsIndexLoaded(void) const`	`bool BamMultiReader::HasIndexes(void) const`

BamMultiReader – Printing Filenames

We plan to remove BamMultiReader::PrintFilenames(). This method is more at home in client code rather than the API. Client code can decide where the filenames are printed to, if there should be any formatting applied, etc.

Conclusion

I hope this guide proves helpful in porting over to BT-1x. If I've missed anything important, or you have any suggestions to improve this guide, please feel free to contact me with your feedback.

Thanks and happy hacking!

Derek Barnett
Marth Lab, Boston College
derekwbarnett@gmail.com