Skwish: A Blob Management Library

A lightweight library for storing blobs on the file system.

Table of contents

Skwish is a Java library for storing and retrieving blobs of arbitrary size--*entries*, in Skwish-speak. The entries (blobs) can contain arbitrary content: to Skwish, every entry is simply an uninterpreted byte sequence. Skwish maintains a simple, fast mapping from numeric entry IDs to entry contents. Think of Skwish as a random access list, not a map, though. Entry IDs are determined by Skwish on entry (blob) insertion: the IDs are doled out in ascending order. An application, thus, must maintain the entry IDs somewhere else--typically in an index or a database.

So the functionality provided is quite Spartan. This begs the question then, "*But what is it good for?*" The idea is for Skwish to do one thing, blob storage and retrieval, and do it very well. While similar functionality may be found in many existing indexing and database tools (and indeed the file system itself), Skwish is designed to address more niche scenarios. Here are some examples:

- The application manages a large number of records but does not use a full blown (e.g. relational) database implementation.
- The application uses a database implementation, but needs to store blobs separately. This requirement may arise for performance reasons, or where the contents of the database is generated from a set of *source* entries (records) that are maintained independently from the database (think, for example, lots and lots of XML files).
- The application uses an indexing library (such as Lucene) that does not directly support storing the contents of what is being indexed.
- The application needs all-or-nothing transactional semantics. For example, an application may need to commit a set of entries either *en bloc* or not at all.

More generally, if you find your application needs to maintain both offset and content files, where the offset file delineates the boundaries of entries within the content file, then this library hopefully provides a ready-made solution for that part of the problem. In fact, that's how this project came into being in the first place.

1. Characteristics & Features

The following table highlights features and characteristics of Skwish.

	Description
Operational modes	
Segment Store	The library provides an interface to a managed collection of segments. (A segment is the elementary storage unit of the system.) This interface is designed to support multiple concurrent readers together with multiple concurrent writers, and provides all-or-nothing

	commit semantics.
Standalone	The library also exposes a basic, unmanaged segment implementation. Unmanaged segments are even more lightweight than managed ones. While still thread-safe under concurrent read access, unmanaged segments require a little more care when also writing to them.
Durability	
Committed operations	Skwish is designed so that when a successfully completed write operation returns (i.e. one that doesn't raise an exception), the changes are guaranteed to be written out to persistent storage (to the extent the operating system and other things like the device controller allow). This is true whether or not the write occurs in the context of a transaction (though transactions do provide much stronger guarantees).
Abnormal shutdown	The system is relatively fail-safe in the face of a crash or other abnormal shutdown. That is, if a running instance is abruptly terminated, chances are very good that the system will still be in a consistent state on restart. We say, "chances are very good," because there is still a small window (the partial write of an 8 byte value denoting the entry count in a segment's index file) in which abrupt termination *can* result in data corruption. (This hole *will* be plugged in a future release.)
I/O	
Access methods	Skwish provides 2 basic ways for reading and writing entry contents. One is value-based and involves copying entry contents to and from memory (via a `ByteBuffer`, see below). This method is typically suitable for accessing smaller [size] entries. The second representation of entry contents is stream-based, and is especially suitable for accessing larger entries: obtaining a reference to an entry-stream (a `FileChannel`) costs at most one disk seek, and depending on how the stream is used, little to none of the actual entry contents need ever be loaded (copied) into memory (as when, for example, the contents is to be piped to another channel).

java.nio.*	Skwish leverages the lower level I/O abstractions Java exposes under the `java.nio` packages. These abstractions (e.g. `FileChannels` and `ByteBuffer`) are not just used under the hood; they are exposed all the way out to Skwish's public API. The goal is to allow layering efficient applications on top Skwish. The library's new experimental non-blocking HTTP interface is an example of such an application.

2. Latest News

2.1. Version 0.2.0 Released

5 March 2008

In addition to some maintenance-related refactorings, this release adds a new stream-based interface for entry insertion TxnSegment.getEntryInsertionChannel() to the API. Unlike the insertEntry(java.nio.channels.ReadableByteChannel) method of its parent class which *pulls* entry contents into the store before returning control to the caller, this new insertion method returns a `FileChannel` object into which the new entry contents can be written. Closing the channel completes the entry insertion. The purpose of this new *push* interface is to allow skwish to be used as an event sink. The immediate application here is for streaming output from a SAX processor into skwish. But it's easy to imagine using it in other scenarios. (Logging, for example.)

A new demo program (which also happens to use the new API additions) complements the "Getting Started" trail.

Download Skwish 0.2.0 from the project download page. While there, you might also want to download `skwish-eg-0.2.zip` (listed under **`Additional Files`**) which contains the demo project.

2.2. Version 0.1.5 Released

26 January 2008

This release now supports referencing an entry's id before it is committed. (Recall, in the general case, a newly inserted entry's id is fixed only *after* the transaction commits.) This is achieved by introducing a new *transaction id* which in combination with a new entry's *pre*-commit id uniquely determines the entry's *post*-commit id. More documentation about

the new feature and transactions, in general, is available here. A demo program is also included that uses the new feature.

To download Skwish 0.1.5 click here.

2.3. Version 0.1.4 Released Under Apache License, Version 2.0

28 December 2008

This is a redux of version 0.1.3, but released under the Apache License, Version 2.0. Other than license-specific documentation changes, there are no substantive changes from the previous release.

To download Skwish 0.1.4 click here.

The decision to change the license from GPL to Apache 2.0 came easy. After posting a message about Skwish on the Lucene developer mailing list and receiving some encouraging interest from that community, it appeared this little project might be finding an audience. So it made sense to align the license. The lib is a *small* offering of a well-established solution, besides; and therefore not so hard to rewrite, anyway. (And why would I want that?) No, the license should have been Apache style from the start. I just hadn't thought it through..

-Babak

2.4. Version 0.1.3 Released

21 December 2008

This release introduces a read-only HTTP interface for skwish segment stores. It uses a new, experimental embedded HTTP server that ships with the library. Besides providing a good public interface to the contents of a store, this also provides a convenient way to inspect ad hoc entries using just a browser (e.g., for debugging purposes). For more information about this new feature, see the package documentation.

To download Skwish 0.1.3 click here.

2.5. Version 0.1.2 Released

28 October 2008

In this release: API naming changes, introduced a bounded entry update feature in the API, and a possibly more efficient asynchronous scheduling and shared execution environment for "segment store"s.

2.6. Version 0.1.1 Released

29 September 2008

In this release: Feature completions and bug fixes.

2.7. Version 0.1.0 Released

14 September 2008

This the first (alpha) release of the software.
From the first forum post [First release, first post]:

..been banging out the code for Skwish for a few weeks now. The skeleton has a little meat now, but no so much that it can't easily change direction. So now seemed liked a good time to release: there's enough code there to show it rather than tell it.

To download Skwish 0.1.0 click here.