# STLdb Design

STLdb is an attempt to create a database based on the STL APIs, with the intention being that an application can use the familiar STL APIs, with the primary learning curve being to incorporate in transaction concepts to the manner in which the containers are updated. As much as possible, the goal is to minimize the learning curve.

There are a number of existing embedded databases that present an STL-like API as a façade layer above the implementation of the database storage layer. With STLdb, I’m trying to actually make the containers themselves the basis for the storage, based on the paradigm of boost::interprocess – in which the containers themselves can be within a mapped memory region corresponding to the database’s on-disk contents. This approach is not necessarily better than that which has been used traditionally, but it has some advantages – it may make it easier to add new physical storage approaches, like lists and queues to the database since each container actually encapsulates its own particular approach to physical storage.

Given this approach to physical storage, I’ve also sought to keep the impact on container design as small as possible. Checkpointing for example, has to intrusive impact on how containers are written, but containers do have to implement the means to permit the recovery of operations, and are ultimately responsible for a transactional concurrency model. The hope is that adding new containers is reasonably easy.

In this document, I’m going to discuss how STLdb ultimately works, with particular emphasis on its approach to optimizing disk I/O. My goal is explain ***why*** the design is as it is, not just explain how it works.

## The Basic Storage Approach

With STLdb, the database is a mapped file region into which containers are created. As a rule, any database can be thought of as consisting of a finite amount of storage on disk which can exceed the amount of memory available on machine. The approach STLdb uses for addressing this problem is to use a memory mapped file for the database, and allow the OS to perform paging as appropriate when accessing the data in the database. i.e. The database is manipulated as part of the virtual memory space of the machine.

There are some disadvantages of this approach – it could place a burden on the container to optimize the use of space within the memory map in order to minimize disk I/O. But with this burden comes the advantage of allowing containers the freedom to address these considerations as part of their design.