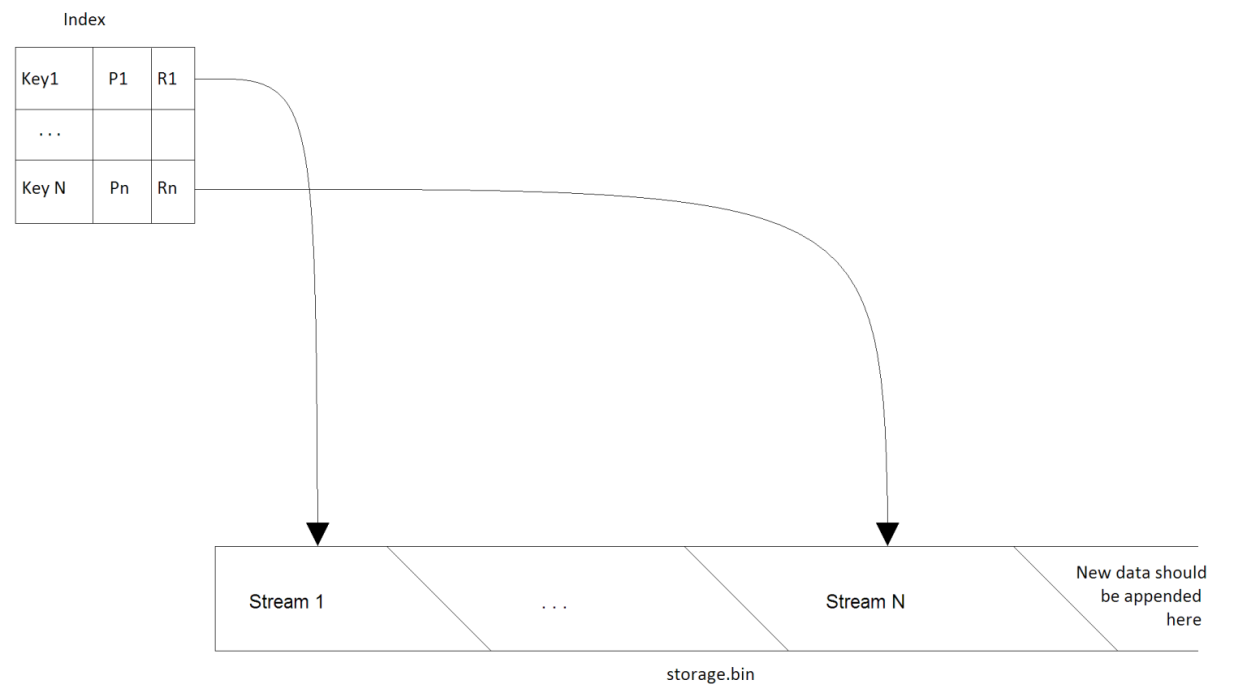
Binary Storage

# Introduction

**Binary Storage** is a write once read many data structure stored on the hard drive. It should provide persistent storage for arbitrary binary content (stream of bytes). When adding new data to the storage client provides a string key that will be associated with this data. Once added to the storage the data cannot be modified through storage API. After data has been successfully added to the storage client could request it for reading by key and it should be returned as a stream of bytes. The storage should be thread-safe and should support multi-threaded addition of new data and reading of existing data. Logical organization of the **Binary Storage** is presented on the picture below:



It consists of two logical parts **Index** and **Storage File.**

* **Storage File** is a plain file on the disk (*storage.bin*) where all binary streams are stored. New data should be appended to the end of the storage file.
* **Index** is a persistent data structure that is used to associate keys (Ki) with streams by means of references (Ri). The reference is simply a pair of byte offset within the Storage File and size of data also in bytes. All information (Pi) required for a normal operation of the Binary Storage should also be stored in the Index. Pi can be hash value of the content, CRC code or a combination of any other properties.

The task is to implement the Binary Storage that satisfies requirements listed below and passes acceptance criteria. The solution should be accompanied with a short description and performance measurements. If any optional requirement is implemented, it should be mentioned in the description. Limitations, important design decisions and usage of 3rd party components should be described.

# Requirements

| Id | Description |
| --- | --- |
| #1 | The solution should be implemented in C# language using .NET 4 or higher in Visual Studio 2010-2015. Use of 3rd party libraries should be minimal and limited to open source public libraries. All sources should be provided. Solution should be standalone and should not require any additional software installed on the host machine like MS SQL Server, MS Message Queue, etc. |
| #2 | *IBinaryStorage* interface should be implemented in a class called *BinaryStorage.* See attached Visual Studio solution. |
| #3 | In the attached solution, *TestApp* project should not be modified. It will be used to test the code against the large set of files. |
| #4 | The solution can use as much memory as the host system provides. But it should work on the systems with 1Gb of RAM and should not fail with OutOfMemory exceptions. |
| #5 | Index structure should have compact representation on the disk and in the memory. It should support arbitrary amount of records and might keep frequently used records in the memory to achieve the best performance during lookups for information. Index structure should store MD5 hash of the input data calculated by Binary Storage implementation during add operation. |
| #6 | It should be possible to add a new data to the Binary Storage by providing a string key and instance that implements Stream (see *IBinaryStorage*).  When disk is full a proper exception should be thrown during attempt to add new data. If data with the same key is already present in the storage a proper exception should be thrown. |
| #7 | After *IBinaryStorage*.Add method returns it should be possible to retrieve binary data as stream from the Binary Storage by providing a key (*IBinaryStorage.Get)*. The data retrieved from the storage should be exactly the same as the data previously added to the storage for persistence.  It there is no data for a given key a proper exception should be thrown. |
| #8 | Adding and reading data from the Binary Storage should be thread-safe and multi-threaded. Thread-safe means that many threads might try to add data to the storage at the same time. Multi-threaded means that adding data distributed among 2 or more threads should be generally faster than adding the same data using only one thread. |
| #9 | It should be possible to check if a given key is present in the Index (*IBinaryStorage.Contains)* |

# Acceptance Criteria

We will use the test application that will run a number of tests against provided solution. Therefore it is very important to implement API exactly as described in this document.

Your solution will be judged based on the outcome from the test as well as the source code review and supporting materials (description of the solution).

The decision will be made based on the following criteria:

* Conformance to API, requirements
* Code and design quality
* Performance
* Support for concurrency
* RAM consumed (less for the same performance is better)
* HDD space consumed for persisting data (less is better)