Project #1 - Key/Value Database

Version 1.0, Revised: 01/16/2017 19:53:52
Due Date: Tuesday February 7th

Purpose:

There is currently a lot of technical interest in "Big Data". Extreme examples are: data collection and analyses from the Large Hadron Collider, the Sloan Sky Survey, analyses of Biological Genomes, measuring weather patterns, collecting data for global climate models, and analyzing client interactions in social networks.

Conventional SQL databases are not well suited for these kinds of applications. While they have worked very well for many business applications and record keeping, they get overwhelmed by massive streams of data. Developers are turning to "noSQL" databases like MongoDB, couchDB, Redis, and Big Table to handle massive data collection and analyses.

In this project we will explore how a non SQL database can be constructed and used.

Requirements:

Your Key/Value Database project:

1. (1) **Shall** be implemented in C++ using the facilities of the standard C++ Libraries and Visual Studio 2015, as provided in the ECS clusters.
2. (4) **Shall** implement a template class providing key/value in-memory databases where each value consists of:
   - Metadata:
     - An item name string
     - A category name string
     - A text description of the item
     - A time-date string recording the date and time the value was written to the database.
     - a finite number (possibly zero) of child relationships with other values. Each element of the child relationships collection is the key of another item in the database. Any item holding a key in this collection is the parent of the element accessed by the key.
   - An instance of the template type[1]. This might be a string, a container of a set of values of the same type, or some other collection of data captured in some, perhaps custom, data structure.
3. (3) **Shall** support addition and deletion of key/value pairs.
4. (3) **Shall** support editing of values including the addition and/or deletion of relationships, editing text metadata, and replacing an existing value's instance with a new instance. Editing of keys is forbidden.
5. (2) **Shall**, on command, persist database contents to an XML file[2]. It is intended that this process be reversible, e.g., the database can be restored or augmented[3] from an existing XML file as well as write its contents out to an XML file.
6. (1) **Shall** accept a positive time interval or number of writes after which the database contents are persisted. This scheduled "save" process shall continue until cancelled. By default, timed saving is turned off.
7. (4) **Shall** support queries for:
   - The value of a specified key.
   - The children of a specified key.
   - The set of all keys matching a specified pattern which defaults to all keys.
   - All keys that contain a specified string in their item name
   - All keys that contain a specified string in their category name
   - All keys that contain a specified string in their template data section when that makes sense.
   - All keys that contain values written within a specified time-date interval. If only one end of the interval is provided shall take the present as the other end of the interval.

   Each query accepts a list of keys[4] and returns a collection of keys from the list that match the query.
8. (2) **Shall** support making the same kinds of queries, cited in the requirement above, on the set of keys returned by an earlier query.
9. (1) **Shall** support forming a union of the keys from two or more earlier queries.
10. (2) **Shall** be submitted with contents, in the form of an XML file, that describe your project's package structure and dependency relationships that can be loaded when your project is graded.
11. (2) **Shall** be accompanied by a test executive that clearly demonstrates you've met all the functional requirements #2-#9, above. The XML file you supply in the previous requirement will be used as part of that demonstration. If you do not demonstrate a requirement you will not get credit for it even if you have implemented it correctly.

T N P B

12. (5) **Bonus (only counted if you have implemented all other requirements):** Support regular expression matching for the query types described above.

---

1. This instance of the template type is the "data" that we store in the database. Metadata simply provides information about this "data".

2. XML is easy to parse with existing tools, like the XmlDocument class provided in the Repository. So it makes a good storage mechanism for data that will be examined by other tools. It is not very space-efficient so a "production" version of this database might use some other more efficient, but more complex storage implementation.

3. By augmented we mean that the database already contains data and the contents of the XML file are read and inserted into the database without destroying the original data.

4. So, the first query starts with a list of all the keys in the database. Each subsequent query, if any, use the list returned by the previous query or a union of keys from previous queries.

What you need to know:

In order to successfully meet these requirements you will need to:

1. Write C++ code and use basic facilities of the standard C++ libraries.

2. Use the STL Containers.

3. Read, Edit, and Write XML files using the XmlDocument class.

Back Back

T N P B