Project #2 - Key/Value Database

Version 1.5, Revised: 09/02/2015 13:31:09
Due Date: Wednesday October 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 .Net Framework Class Library and Visual Studio 2015, as provided in the ECS clusters.
2. (3) **Shall** implement a generic key/value in-memory database where each value consists of:
   - Metadata:
     - A 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 generic 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. (2) **Shall** support addition and deletion of key/value pairs.
4. (2) **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[2,3]. It is intended that this process be reversible, e.g., the database can be restored or augmented[4] 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.
7. (3) **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 metadata section.
   - 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.
8. (1) **Shall** support the creation of a new immutable database constructed from the result of any query that returns a collection of keys[5].
9. (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.
10. (2) **Shall** be accompanied by a test executive that clearly demonstrates you've met all the functional requirements #2-#9, above. If you do not demonstrate a requirement you will not get credit for it even if you have it correctly implemented.
11. (1) **Shall** contain a brief text document that compares this implementation with the concept developed in Project #1. Does this project fully implement its concept? Was the original concept practical? Were there things you learned during the implementation that made the origninal concept less relevant?

12. (5) **Bonus (only counted if you have implemented all other requirements):**
    Implement categories by using a Dictionary<key,value> where each key is the name of a category and each value is a list of keys for db items in that category[6]. This implies that each item's metadata should add a list for categories to which it belongs. Note that the list might be empty.

---

1. This instance of the generic type is the "data" that we store in the database. The metadata simply provides information about this "data".
2. XML is easy to parse with existing tools 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. For value instances that are not strings, you will need to override the generic type's ToString() method to persist the instance into an XML element.
4. 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.
5. The intent of this requirement is that the new database contains keys pointing to the values stored in the original database. This allows refining a query with a new query but doesn't allow the new databases to modify the original database's values.
6. So, you can find all the db items in a given category by enumerating the category's list of keys. This facility is very like conventional database indexes.

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 .Net Framework.
2. Use the .Net Containers.
3. Read and Write XML files.

Back Back