Java Object Cache Requirement Specification

1. Scope

1.1 Overview

An existing standalone Java batch job has a flaw in that it that it performs an expensive operation in order to build an object and then it discards that object, not knowing if it will need that same object on the next iteration.

This component design is for an object caching mechanism which will cache objects during the execution of the job.

1.1.1 Version

1.0

1.2 Logic Requirements

This component will cache serializable Java objects provided along with a key.

1.2.1 Memory Cache

The component needs to be configurable for how much memory the in-memory cache is allowed to consume. This amount should be specified in kilobytes and can be zero, in which case only persistent cache is used.

1.2.2 Persistent Cache

The component needs to be able to store the serialized Java objects in a relational database when needed. Part of this design is to design the database schema for the table(s) which are required for the storage in a relational database.

1.2.3 Cache Cleanup

The component needs to provide a simple cleanup mechanism that will be invoked at the beginning and end of the job. This mechanism will clear out any in-memory cache as well as the ENTIRE table in the relational database which stores the cached objects.

1.2.4 Non-serializable Objects

The component needs to throw a runtime exception in the case where an object is provided to cache but it is not serializable. This can be java.io.NotSerializableException, but it is not required to be this exception.

1.2.5 Named Cache Sets

The component needs to require a "name" (string value) to be provided as part of instantiation which will be used when storing serialized objects in the relational database. This will allow multiple instances of this module to be used by different parts of an application but with the same database.

For example, if a cache is instantiated as "Cache A" then when items are added to the cache and they are serialized and placed in the database, they will be associated with the Cache Set with name "Cache A."

Then, when it comes time to clear the cache, a simple SQL statement like the following can be used (pseudo-SQL): delete from CACHE where SET_NAME = 'Cache A'

This type of design will allow for multiple cache sets to be stored in the same table in the database without affecting other cache sets.

1.2.6 Statistics and Reporting

The component needs to provide a printout (to a provided PrintWriter) of statistics of the cache, including the following:

- Number of items in the cache
- Keys in the cache, including the number of accesses to each key
- Number of items in memory and number of items in persistent storage
- Number of misses, where an object is requested that is not in the cache

1.2.7 Cache Preferences

The object cache should use in-memory cache first and then use persistent cache if the memory cache is exhausted.

It should also be assumed that if an object is retrieved from persistent cache, that it is likely to be needed again sooner than any other object that may be in the in-memory

cache. Therefore, if an object is requested and it exists in persistent cache but not in memory cache, it should be placed into the in-memory cache, possibly displacing the "oldest" items in the in-memory cache. Of course, if the in-memory cache size is set to zero or if the object is too large to fit in the in-memory cache, then persistent cache is the only possible location for the object.

1.3 Required Algorithms

The component will provide methods which allow the batch job to ask for a cached object by the Key of that object. The component should return the object or null if not found or if any error occurred. The component should never throw an exception from an attempt to access the cache.

The component will provide methods which allow the batch job to provide a new or updated cached object, along with a unique key for that object. Null values are to be treated as clearing that key. If a key already exists in the cache, it is overwritten with the new object.

1.4 Example of the Software Usage

The Java batch job will start up, invoke the Clear Cache mechanism on this component, and then it will determine which items it needs to iterate over. For each iteration, the batch job will first ask the cache component if a cached object exists for that iteration. If it does, it will be used. If it does not have a cached object, it will perform the expensive operation to build the object and then provide it to the cache. At the end of all iterations, the cache will be cleared via the Clear Cache mechanism.

1.5 Future Component Direction

[none]

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

[none]

2.1.2 External Interfaces

The component should be compatible with DB2 v9 and above in terms of storing, updating and deleting BLOB data into a table. The database table schema should be included in the design.

2.1.3 Environment Requirements

- Development language: Java1.5
- Compile target: Java1.5
- DB2 v9+

2.1.4 Package Structure

com.ibm.support.electronic.cache

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

Need to be able to configure the amount of in-memory cache available, in kilobytes, including zero.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

Java persistence frameworks such as JPA (Apache OpenJPA, for example) MAY be useful, but are NOT required. It may be better to NOT use JPA in favor of a more simple approach since this component is not complex. If JPA can be used in a SIMPLE fashion that is easy to set up and configure, then this would be good. **OpenJPA v1.2.2** is acceptable, all other third-party packages need to be approved before being included in a design.

3.2.2 TopCoder Software Component Dependencies:

Do Not Include Any TopCoder Generic components in this design.

3.2.3 Third Party Component, Library, or Product Dependencies:

[none]

3.2.4 QA Environment:

- Aix 5.3
- Java 1.5

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

Access to this component should be threadsafe in all cases. It is possible/likely that multiple threads will be accessing this component simultaneously.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

Please generate from your TCUML the following (if applicable)

- PNG image files of all UML diagrams

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the 'Documentation' tab of the TCUML Tool.