How To Use the Shared Model Map

Technical Guide

Date: Jun 2014

Version: v1

Revision History

Rev Number	Date	Summary of changes
v1.0	Jun 2014	Initial document creation by CA/ITKO Professional Services

Table of Contents

Shared Model Map

The Shared Model Map provides a **small** memory space where information can be stored and retrieved from a thread safe Hash table kept in memory by the VSE. SharedModelMap is thread-safe - two VSMs can access and update it at the same time (internally, access to each namespace map is synchronized on the map itself).

The maps are not persisted across Virtual Environment Service restarts. Refer to *PersistentModelMap* for a persistent map implementation of this feature.

NOTES:

1) Shared Model Maps are NOT sharable across VSEs. If multiple VSEs are running, the Shared Model Map managed by one VSE is not visible to another VSE.
2) Shared Model Maps are "in memory". If the VSE is stopped, the values contained in the Shared Model Map are lost. (e.g., unrecoverable)
3) When executing ITR from the LISA Workstation, the SharedModelMap (SMM) is local to the workstation; hence, the VSE has no visibility to the SMM.
4) The application must manage memory (e.g., remove entries when no longer required)
5) If the map reaches its maximum capacity, the oldest entry is removed to make room for the entry being added.
6) If a value having a LISA *{{property}}* reference is to be stored on an SMM, it is a good practice to force LISA to replace the *{{property}}* with the actual value prior to placing the data on the SMM. For example:
   ```
   String aField = testExec.parseInState( testExec.getStateValue("<propertyNameHere>"));
   ```
   Then, add "aField" to the SMM.

Shared Model Maps assist LISA in understanding the relationship between variables that are to be shared by multiple services.

The following example provides a high level view of how one service updates the SMM and another service reads the SMM to provide a complete set of business functionalities.

Establishing the Size of a Shared Model Map

The Shared Model Map (SMM) is a thread safe hash table that is managed in the LISA Virtual Service Environment. Only one SMM exists for each instance of the Virtual Service Environment (VSE).

LISA enables the use of the SMM so Services can store information.

Following is a code fragment of the Establish SMM Size step:

```
import com.itko.lisa.vse.SharedModelMap;
import com.itko.lisa.VSE;

//  Create a specific Shared Model Map for a Namespace
//  Increase the size from the default 256 to 1000 entries
//  Replace <myNamespaceHere> with a meaningful name
//  NOTE: the namespace value can be parameterized and taken from the Config
//  example:   SharedModelMap.setCapacity(testExec.getStateValue("someConfigValue", 1000);

SharedModelMap.setCapacity("<myNamespaceHere>", 1000);

return true;
```

Iterating Over a Shared Model Map

There are multiple techniques for iterating over a SharedModelMap. This example simply demonstrates one technique.

```
// Iterate the MemoryMap and remove all entries for the given Namespace

// The only way to totally destroy an SMM is to stop the VirtualServiceEnvironmentService

Collection colAsync = SharedModelMap.keySet("<myNamespaceHere>");
Iterator itAsync = colAsync.iterator();

while (itAsync.hasNext()) {
    asyncToken = itAsync.next();
}

return true;
```

Removing Entries from a Shared Model Map

There are multiple techniques for removing entries from a SharedModelMap.

```
// Iterate the MemoryMap and remove all entries for the given Namespace
Collection colAsync = SharedModelMap.keySet("<myNamespaceHere>");
Iterator itAsync = colAsync.iterator();

while (itAsync.hasNext()) {
    asyncToken = itAsync.next();
    SharedModelMap.remove("<myNamespaceHere>", asyncToken);
}

return true;
```

OR

```
SharedModelMap.remove("<myNamespaceHere>", "<enterKeyValueHere>");
```

Adding a Key/Value Pair to an SMM

The SMM functions on the notion of using a key/value pair. The value inserted into the SMM can be a single value or an object. The following example inserts a Properties object holding on to several elements that belong to a single “key”.

The logic that builds the meters into the Shared Model Map follows this pattern:

```
import com.itko.lisa.vse.SharedModelMap;
import com.itko.lisa.VSE;

// Create a properties object to hold the desired value information
Properties  myProps = new Properties();

myProps.setProperty("smmMyMessage", testExec.getStateValue("prop_SomeMessage") );
myProps.setProperty("smmMyCntr", "0");
myProps.setProperty("smmMyOtherData", "No");

SharedModelMap.putObject(“<myNamespaceHere>”, “<enterKeyValueHere>”, myProps);

return true;
```

Accessing a Key/Value Pair on an SMM

The SMM provides accessor methods to enable the retrieval of data. In the example below, a value represented by the namespace (*<myNamespaceHere>*) and key (*<enterKeyValueHere>*) are returned. Since the above example stored the value as a Properties object, the Properties object is cast and each value is retrieved.

```
import com.itko.lisa.vse.SharedModelMap;
import com.itko.util.ParameterList;

// Retrieve an entry from the SMM
Properties myProps = (Properties) SharedModelMap.getObject("<myNamespaceHere>",
"<enterKeyValueHere>");

if (myProps == null)
    return false;             // No Entry on the map

if (myProps.isEmpty())
    return false;             // Entry found but object contains no values

// For demonstration purposes, this code accesses each of the properties
// on the Properties object
String myMessage      = myProps.getProperty("smmMyMessage");

int myCntr            = Integer.parseInt(myProps.getProperty("smmMyCntr"));

String myOtherData    = myProps.getProperty("smmMyOtherData");

return true;
```

Appendix

SharedModelMap Javadoc

```
package com.itko.lisa.vse;
/**
 *
 * Maintains a statically shared set of maps, meant for sharing data across virtual service models at runtime in a
 * single JVM. Think of it as you would shared memory between processes in a traditional OS.  Maps are accessed
 * by namespace and key in most cases. If you are feeling lazy you can use the default namespace but this is not
 * encouraged, it's better to agree on a namespace that two or more VSMs can share that is meaningful to those VSMs.
 * <p>
 * The get and put operations have String and Object variations with the assumption that most of the time you are
 * storing and retrieving String values.  Keys must be strings.
 *
 * Each namespace has an LRU map created on demand the first time the namespace is used. It's backed by an
 * org.apache.commons.collections.map.LRUMap, see the javadoc there for details.
 *
 * All methods are thread-safe.
 */
public class SharedModelMap {
    public static final int DEFAULT_CAPACITY = 256;


    // ====================================================================
    // These methods work with the default namespace.
    // ====================================================================


    /**
     * This method returns the number of entries in the default namespace.
     *
     * @return the number of entries in the default namespace.
     */
    public static int size()


    /**
     * This method returns whether the default namespace is empty or not.
     *
     * @return {@code true} if the default namespace is empty or {@code false} if not.
     */
    public static boolean isEmpty()


    /**
     * This method returns whether the default namespace contains an entry with the given key
     * or not.
     *
     * @return {@code true} if the default namespace contains the specified key or {@code
     *         false} if not.
     */
    public static boolean containsKey(final String key)


    /**
     * This method returns whether the default namespace contains an entry with the given value
     * or not.
     *
     * @return {@code true} if the default namespace contains the specified value or {@code
     *         false} if not.
     */
    public static boolean containsValue(final Object value)


    /**
     * This method gets a value from the default namespace.
     *
     * @param  key the key to the desired value.
     * @return the value known by the key or {@code null}.
     */
    public static Object getObject(final String key)


    /**
     * This method gets a value from the default namespace cast as a string.
     *
     * @param  key the key to the desired value.
     * @return the value known by the key or {@code null}.
     */
    public static String get(final String key)


    /**
     * This method puts a value into the default namespace.
     *
     * @param  key the key to make the value known by.
     * @param  value the value to associate with the key.
     * @return the previous value for the key or {@code null}.
     */
    public static Object putObject(final String key, final Object value)


    /**
     * This method puts a value into the default namespace cast as a string.
     *
     * @param  key the key to make the value known by.
     * @param  value the value to associate with the key.
     * @return the value known by the key or {@code null}.
     */
    public static String put(final String key, final String value)


    /**
     * This method removes a value from the default namespace.
     *
     * @param  key the key to the value to remove.
     * @return the value known by the key or {@code null}.
     */
    public static Object remove(final String key)


    /**
     * This method gets a value from the default namespace cast as a string.
     */
    public static void clear()


    /**
     * This method returns the set of keys currently known in the default namespace.  Note that
     * the set returned, unlike the standard map semantics, is detached from the source map;
     * this helps us ensure thread-safety.
     *
     * @return the set of known keys in the default namespace.
     */
    public static Set<String> keySet()


    // ====================================================================
    // These methods work with a specified namespace.
    // ====================================================================
    /**
     * This method returns the number of entries in the specified namespace.
     *
     * @param  namespace the namespace to get the size of.
     * @return the number of entries in the specified namespace.
     */
    public static synchronized int size(final String namespace)


    /**
     * This method returns whether the specified namespace is empty or not.
     *
     * @param  namespace the namespace to get the empty state for.
     * @return {@code true} if the specified namespace is empty or {@code false} if not.
     */
    public static boolean isEmpty(final String namespace)
```

```java
/**
 * This method returns whether the specified namespace contains an entry with the given key
 * or not.
 *
 * @param  namespace the namespace to check for key containment.
 * @return {@code true} if the specified namespace contains the given key or {@code false}
 *         if not.
 */
public static synchronized boolean containsKey(final String namespace, final String key)


/**
 * This method returns whether the specified namespace contains an entry with the given
 * value or not.
 *
 * @param  namespace the namespace to check for value containment.
 * @return {@code true} if the default namespace contains the specified value or {@code
 *         false} if not.
 */
public static synchronized boolean containsValue(final String namespace, final Object value)


/**
 * This method gets a value from the specified namespace.
 *
 * @param  namespace the namespace to get the value from.
 * @param  key the key to the desired value.
 * @return the value known by the key or {@code null}.
 */
public static synchronized Object getObject(final String namespace, final String key)


/**
 * This method gets a value from the specified namespace cast as a string.
 *
 * @param  namespace the namespace to get the value from.
 * @param  key the key to the desired value.
 * @return the value known by the key or {@code null}.
 */
public static String get(final String namespace, final String key)


/**
 * This method puts a value into the specified namespace.
 *
 * @param  namespace the namespace to put the value in.
 * @param  key the key to make the value known by.
 * @param  value the value to associate with the key.
 * @return the previous value for the key or {@code null}.
 */
public static synchronized Object putObject(final String namespace, final String key, final Object value)
```

```
    /**
     * This method puts a value into the specified namespace.
     *
     * @param  namespace the namespace to put the value in.
     * @param  key the key to make the value known by.
     * @param  value the value to associate with the key.
     * @return the previous value for the key or {@code null}.
     */
    public static String put(final String namespace, final String key, final String value)

    /**
     * This method removes a value from the specified namespace.
     *
     * @param  namespace the namespace to remove the key from.
     * @param  key the key to the value to remove.
     * @return the value known by the key or {@code null}.
     */
    public static synchronized Object remove(final String namespace, final String key)

/**
     * This method gets a value from the specified namespace cast as a string.
     *
     * @param namespace the namespace to clear.
     */
    public static synchronized void clear(final String namespace)

    /**
     * This method returns the set of keys currently known in the specified namespace.  Note
     * that the set returned, unlike the standard map semantics, is detached from the source
     * map; this helps us ensure thread-safety.  The value returned will never be {@code null}.
     *
     * @param  namespace the namespace to get the set of keys for.
     * @return the set of known keys in the default namespace.
     */
    public static synchronized Set<String> keySet(final String namespace)

    /**
     * This method may be used to resize the capacity of the specified namespace.
     *
     * @param namespace the namespace to change the capacity for.
     * @param newCapcity the new capacity for the namespace.
     */
     public synchronized static void setCapacity(final String namespace, final int newCapcity)



}
```