# Technical Approach for Issue #222: Multithreaded Use Cases

## Overview

1. Update stateless utility classes (ISMVocabulary and Util) to prevent multithreaded side effects.
2. Convert static global classes to be localized within a single Thread (DDMSVersion, PropertyReader).

By limiting the changes to the above 4 classes using the technical approach described below, no broader changes to the rest of the DDMSence API should be necessary. For example, early prototyping involved adding a DDMSVersion input parameter to every DDMSComponent constructor and Builder commit() call – this level of change is not needed.

## DDMSVersion

This class was used to set a global DDMSVersion that would apply to all component building and creation occurring after that point in time. Previously, the version was stored in a static variable. Now, the version will be stored in a ThreadLocal wrapper. This greatly minimizes other API changes.

### Technical Approach

1. Update \_currentVersion to exist within a ThreadLocal wrapper.

### Backwards Compatibility

This change is backwards compatible.

## ISMVocabulary

This utility class is used to validate the values of ISM security attributes. Previously, the current version of DDMS was set here whenever it was changed. That value was then used to load and cache a specific set of ISM enumeration files so they could be used in validation. Now, the current DDMS version will not be stored. Instead, the version will be passed into the validation methods at validation time, effectively converting this class into a stateless utility class.

For the sake of performance, the most recent set of enumeration files can still be cached locally within a single Thread. However, this feature should be invisible to the caller.

### Technical Approach

1. Eliminate global caching of enumeration files. Refactor code to load enumerations whenever needed by the two validation methods:
   1. getEnumerationTokens(String)
   2. validateEnumeration(String, String)
2. Eliminate setDDMSVersion(String) method entirely.
3. Update validation method signatures to include DDMSVersion:
   1. getEnumerationTokens(DDMSVersion, String)
   2. validateEnumeration(DDMSVersion, String, String)
4. After regression testing, restore caching of enumeration files based on passed in DDMSVersion. Caches should be localized within a single Thread.

### Backwards Compatibility

This change will result in a **minor break** in backwards compatibility. The two validation methods are not necessarily high-traffic exposed API calls, but if anyone has used them for purposes outside of DDMSence (for example, developing a web application that manipulates security attributes outside of a metacard), they will need to add a parameter to their method calls. In my opinion, this change is minor enough to not trigger a major version number change.

## PropertyReader

This utility class maintains DDMSence properties, some of which can be configured at run-time. Previously, properties were stored in a static singleton. Now, this singleton will be updated to be thread-local, which would allow multiple Threads to have different sets of run-time properties. For example, Thread 1 could output raw JSON while Thread 2 outputs pretty-printed JSON.

### Technical Approach

1. Update INSTANCE singleton to exist within a ThreadLocal wrapper.

### Backwards Compatibility

This change is backwards compatible.

## Util

This utility class is used for a variety of stateless functional calls. Previously, the method, buildSchematronTransform(File), maintained a global cache of XSLTransform objects used for resource Schematron validation. Now, these caches should be converted into local caches within a single Thread.

### Technical Approach

1. Replace the global static caches with thread-safe, thread-local versions.
   1. \_schematronIncludeTransform
   2. \_schematronAbstractTransform
   3. \_schematronSvrlTransforms

### Backwards Compatibility

This change is backwards compatible.