MultiUberspect

```
/** org.apache.velocity.tools.generic.introspection.MultiUberspect
    Allows Velocity to Introspect other types of Objects
    November 2004
    Eric Fixler <fix@smete.org>
    $Id: MultiUberspect.java,v 1.1.2.7 2004/12/20 00:20:34 fix Exp $
*/

/*
 * Copyright 2003-2004 The Apache Software Foundation.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.apache.velocity.tools.generic.introspection;

import java.util.*;
import java.lang.reflect.Constructor;

import org.apache.log4j.Logger;

import org.apache.velocity.util.introspection.Info;
import org.apache.velocity.util.introspection.Uberspect;
import org.apache.velocity.util.introspection.UberspectImpl;
import org.apache.velocity.util.introspection.VelPropertyGet;
import org.apache.velocity.util.introspection.VelPropertySet;

/**
 * Expanded Uberspect implementation that allows you to designate custom Uberspects on
 * a per class basis, so that classes that don't follow the Java Beans specification can
 * be handled like beans in Velocity Templates.
 *
 * <p>To use, tell Velocity to use this class for introspection
 * by adding the following to your velocity.properties:<br />
 *
 * <code>
 * runtime.introspector.uberspect = org.apache.velocity.tools.generic.introspection.MultiUberspect
 * </code>
 * </p>
 *
 * @author <a href="mailto:fix@smete.org">Eric Fixler</a>
 * @version $Id: MultiUberspect.java,v 1.1.2.7 2004/12/20 00:20:34 fix Exp $
 */

 public class MultiUberspect extends UberspectImpl {
     private final Map    customIntrospectors = new HashMap();
     private final Map    derivedIntrospectors = new HashMap();

     public MultiUberspect() {}

     protected boolean useSuperclassIntrospector = false;

     /** <p>If true, and a class is encountered for which a custom Uberspect has not been registered,
         will check to see if any superclasses of class have registered Uberspects.  If so, will use
         the superclass' Uberspect.</p>
         <p>When false (the default), will only return a custom Uberspect if the requested Objects  class
         has had a Uberspect explicitly registered.</p>
```

```
         @param b the flag
    */

    public void             setUseSuperclassIntrospector(boolean b) { this.useSuperclassIntrospector = b; }

    /** Whether of not we will look for a superclass Uberspect if an exact match for a custom Uberspect is not
found. */

    public final boolean     getUseSuperclassIntrospector() { return this.useSuperclassIntrospector; }

    /** Register a custom uberspect

        @param clz     The class handled by this Uberspect
        @param u    The Uberspect
    */

    public void addIntrospector(Class clz, Uberspect u) {
        try {
            u.init();
        } catch (Exception e) {
            return;
        }
        this.customIntrospectors.put(clz, u);
        this.derivedIntrospectors.clear();
        this.derivedIntrospectors.putAll(this.customIntrospectors);
        Logger.getRootLogger().info("Added uberpect " + u.getClass().getName() + " for " + clz.getName());
    }

    /** Unregister a custom uberspect.

        @param clz     The class handled by the Uberspect
    */

    public void removeIntrospector(Class clz) {
        this.customIntrospectors.remove(clz);
        this.derivedIntrospectors.clear();
        this.derivedIntrospectors.putAll(this.customIntrospectors);
    }

    /** Get the Uberspect, if one exists

        @param clz The class for which to get an Uberspect
        @return an Uberspect, or null if none exists
    */

    protected Uberspect getCustomIntrospector(Class clz) {
        Uberspect uber = (Uberspect) this.derivedIntrospectors.get(clz);
        if ((! this.useSuperclassIntrospector) || (uber != null)) return uber;
        synchronized(this.derivedIntrospectors) {
            Iterator it = this.customIntrospectors.entrySet().iterator();
            while (it.hasNext()) {
                Map.Entry entry = (Map.Entry) it.next();
               Class checkClz     = (Class) entry.getKey();
               if (checkClz.isAssignableFrom(clz))  {
                   uber = (Uberspect) entry.getValue();
                   this.derivedIntrospectors.put(clz, uber);
                   return uber;
               }
            }
        }
        return null;
    }

    /**
    * Property iterarot - returns Iterator apropos for #foreach($foo in $bar.).
    * <br />
    * Looks to see if a custom Uberspect has been registered for obj.getClass().
    * If an Iterator was not found, tries the regular routine.
    *
    * @param obj the object
    * @param info a bunch of information.
```

```
 * @return a valid <code>Iterator</code>, if it was found.
 * @throws Exception failed to create a valid <code>Iterator</code>.
 */
public Iterator getIterator(Object obj, Info info)
        throws Exception {
        Uberspect uberspect = (Uberspect) this.getCustomIntrospector(obj.getClass());
    return (uberspect != null)
            ? uberspect.getIterator(obj, info)
            : super.getIterator(obj, info);


}


 /**
 * Property getter - returns VelPropertyGet appropos for #set($foo = $bar.woogie).
 * <br />
 * Looks to see if a custom Uberspect has been registered for obj.getClass().
 * If a getter was not found, tries the regular routine.
 *
 * @param obj the object
 * @param identifier the name of the property
 * @param info a bunch of information.
 * @return a valid <code>VelPropertyGet</code>, if it was found.
 * @throws Exception failed to create a valid <code>VelPropertyGet</code>.
 */
public VelPropertyGet getPropertyGet(Object obj, String identifier, Info info)
        throws Exception
{

    Uberspect uberspect = (Uberspect) this.getCustomIntrospector(obj.getClass());
    return (uberspect != null)
            ? uberspect.getPropertyGet(obj, identifier, info)
            : super.getPropertyGet(obj, identifier, info);
}

 /**
 * Property setter - returns VelPropertySet appropos for #set($foo.bar = "geir").
 * <br />
 * Looks to see if a custom Uberspect has been registered for obj.getClass().
 * If a setter was not found, tries the regular routine.
 *
 * @param obj the object
 * @param identifier the name of the property
 * @param arg the value to set to the property
 * @param info a bunch of information.
 * @return a valid <code>VelPropertySet</code>, if it was found.
 * @throws Exception failed to create a valid <code>VelPropertySet</code>.
 */
public VelPropertySet getPropertySet(Object obj, String identifier,
        Object arg, Info info) throws Exception
{
    Uberspect uberspect = (Uberspect) this.getCustomIntrospector(obj.getClass());
    return (uberspect != null)
        ? uberspect.getPropertySet(obj, identifier, arg, info)
        : super.getPropertySet(obj, identifier, arg, info);
}

 /**
     Change this source if you use this, since you won't have the classes referenced here.
     (Obviously, would be better to assign the mappings elsewhere, and it's set up to do its
     configuration work based on a map, in the hopes that, at some point, it'll be
     configurable velocity.properties)
 */
 public void init() throws Exception {
     super.init();
     Map m = new HashMap();
     m.put("org.apache.velocity.tools.generic.introspection.CumulusRecordUberspect",
                             new String[] { "com.canto.cumulus.Record" });
     m.put("org.apache.velocity.tools.generic.introspection.CumulusCategoriesUberspect",
                             new String[] { "com.canto.cumulus.Categories" });
```

```
        m.put("org.apache.velocity.tools.generic.introspection.LuceneDocumentUberspect",
                                new String[] { "org.apache.lucene.document.Document" });
        m.put("org.apache.velocity.tools.generic.introspection.SearchResultItemUberspect",
                                new String[] { "org.smete.search.SearchResultItem" });

        Iterator it = m.entrySet().iterator();

        while (it.hasNext()) {
                Map.Entry entry = (Map.Entry) it.next();
                String key         = (String) entry.getKey();
                Object val        = entry.getValue();
                String[] clzNames = (val instanceof Object[])
                                                                ? (val instanceof String[])
                                                                        ? (String[]) val
                                                                        : new String[0] //need better handler
here
                                                                : new String[] { val.toString() };


                Class                 introspectorClass         = null;

                try {
                        introspectorClass = Class.forName(key);
                } catch (Exception e) {
                        Logger.getRootLogger().warn("Uberspect class " + key + " not found, ignoring", e);
                        continue;
                }

                Constructor ubc = null; Object[] args = null;

                try {
                        ubc = introspectorClass.getConstructor(new Class[] { MultiUberspect.class} );
                        args = new Object[] { this };
                } catch (Exception e) {
                        try {
                                ubc = introspectorClass.getConstructor(new Class[0]);
                                args = new Object[0];
                        } catch (Exception ee) {
                                Logger.getRootLogger().warn("Can't find constructor for Uberspect " + key,e);
                                continue;
                        }
                }

                        Uberspect         introspector = null;
                        try {
                                introspector = (Uberspect) ubc.newInstance(args);
                        } catch (Exception e) {
                                Logger.getRootLogger().warn("Can't instantiate Uberspect " + key,e);
                                continue;
                        }

                for (int i = 0; i < clzNames.length; i++) {
                        try {
                                Class clz = Class.forName(clzNames[i]);
                                this.addIntrospector(clz,introspector);
                        } catch (Exception e) {
                                Logger.getRootLogger().warn("Can't find class " + clzNames[i] + " for custom
Uberspect, ignoring", e);
                                continue;
                        }
                }

        }

    }

 }

 // $Log: MultiUberspect.java,v $
 // Revision 1.1.2.7  2004/12/20 00:20:34  fix
 // docs
```

```
//
// Revision 1.1.2.6  2004/12/16 03:45:01  fix
// -- Uberspect for LuceneDocuments
// -- Modified init() in MultiUberspect to make configuration make more sense
//
// Revision 1.1.2.5  2004/11/24 00:46:36  fix
// -- Fixed MultiUberspect so delegation works for Iterators
//
// Revision 1.1.2.3  2004/11/05 06:35:40  fix
// bug fix
//
// Revision 1.1.2.2  2004/11/05 05:51:00  fix
// Added useSuperclassIntrospector to turn superclass scaling on and off
//
```