ServletLogChute

This is an implementation of the new LogChute interface in Velocity 1.5, that will pipe log messages to the servlet's log file. In accord with the upgrades to logging in Velocity 1.5, this provides the ability to check if specified levels are enabled. Since the Servlet API does not provide the ability to enable/disable log levels, log output from Velocity when using this LogChute may be controlled by setting the "runtime.log.logsystem.servlet.level" property (in your velocity.properties) to trace, debug, info, warn, or error.

This will be added to the trunk of the VelocityTools codebase after the release of VelocityTools 1.2.

```
/*
 * Copyright 2005 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.view.servlet;

import javax.servlet.ServletContext;
import org.apache.velocity.runtime.log.LogChute;
import org.apache.velocity.runtime.RuntimeConstants;
import org.apache.velocity.runtime.RuntimeServices;
import org.apache.velocity.util.StringUtils;

/**
 * Simple wrapper for the servlet log.  This sends Velocity log
 * messages to ServletContext.log(String).
 *
 * @author <a href="mailto:geirm@apache.org">Geir Magnusson Jr.</a>
 * @author <a href="mailto:nbubna@apache.org">Nathan Bubna</a>
 * @since VelocityTools 1.3
 * @version $Revision: 71982 $ $Date: 2004-02-18 12:11:07 -0800 (Wed, 18 Feb 2004) $
 */
public class ServletLogChute implements LogChute
{
    public static final String PREFIX = " Velocity ";
    public static final String TRACE_PREFIX = PREFIX + RuntimeConstants.TRACE_PREFIX;
    public static final String DEBUG_PREFIX = PREFIX + RuntimeConstants.DEBUG_PREFIX;
    public static final String INFO_PREFIX = PREFIX + RuntimeConstants.INFO_PREFIX;
    public static final String WARN_PREFIX = PREFIX + RuntimeConstants.WARN_PREFIX;
    public static final String ERROR_PREFIX = PREFIX + RuntimeConstants.ERROR_PREFIX;
    public static final String UNKNOWN_PREFIX = PREFIX + RuntimeConstants.UNKNOWN_PREFIX;

    protected ServletContext servletContext = null;

    private int enabled = TRACE_ID;

    /**
     * Initializes the logger.
     * <br>
     * NOTE: this class expects that the ServletContext has already
     *       been placed in the runtime's application attributes
     *       under its full class name (i.e. "javax.servlet.ServletContext").
     *
     * @throws IllegalStateException if the ServletContext is not available
     *         in the application attributes under the appropriate key.
     */
    public void init(RuntimeServices rs) throws Exception
    {
        Object obj = rs.getApplicationAttribute(ServletContext.class.getName());
        if (obj == null)
```

```
    {
        throw new IllegalStateException("Could not retrieve ServletContext from application attributes!");
    }
    servletContext = (ServletContext)obj;

    // look for a level config property
    String level = (String)rs.getProperty("runtime.log.logsystem.servlet.level");
    if (level != null)
    {
        // and set it accordingly
        if (level.equalsIgnoreCase("debug"))
        {
            setEnabledLevel(DEBUG_ID);
        }
        else if (level.equalsIgnoreCase("info"))
        {
            setEnabledLevel(INFO_ID);
        }
        else if (level.equalsIgnoreCase("warn"))
        {
            setEnabledLevel(WARN_ID);
        }
        else if (level.equalsIgnoreCase("error"))
        {
            setEnabledLevel(ERROR_ID);
        }
    }
}

protected String getPrefix(int level)
{
    switch (level)
    {
        case LogChute.WARN_ID:
            return WARN_PREFIX;
        case LogChute.INFO_ID:
            return INFO_PREFIX ;
        case LogChute.DEBUG_ID:
            return DEBUG_PREFIX;
        case LogChute.TRACE_ID:
            return TRACE_PREFIX;
        case LogChute.ERROR_ID:
            return ERROR_PREFIX;
        default:
            return UNKNOWN_PREFIX;
    }
}

/**
 * Logs messages to either std.out or std.err
 * depending on their severity.
 *
 * @param level severity level
 * @param message complete error message
 */
public void log(int level, String message)
{
    // pass it off
    log(level, message, null);
}

/**
 * Logs messages to the servlet log so long as the specified level
 * is equal to or greater than the level this LogChute is enabled for.
 * If a java.lang.Throwable accompanies the message, it's stack trace
 * will also be sent to the servlet log as the same level as the message.
 *
 * @param level severity level
 * @param message complete error message
 * @param t the java.lang.Throwable
 */
```

```
    public void log(int level, String message, Throwable t)
    {
        if (!isLevelEnabled(level))
        {
            return;
        }

        String prefix = getPrefix(level);
        servletContext.log(prefix + message);
        if (t != null)
        {
            servletContext.log(t.getMessage() + StringUtils.stackTrace(t));
        }
    }

    /**
     * Set the minimum level at which messages will be printed.
     */
    public void setEnabledLevel(int level)
    {
        this.enabled = level;
    }

    /**
     * Returns the current minimum level at which messages will be printed.
     */
    public int getEnabledLevel()
    {
        return this.enabled;
    }

    /**
     * This will return true if the specified level
     * is equal to or higher than the level this
     * LogChute is enabled for.
     */
    public boolean isLevelEnabled(int level)
    {
        return (level >= this.enabled);
    }

}
```