



Raytheon

AWIPS II EDEX Training Resource Material Creating a MicroEngine Task

Prepared in Support of the AWIPS Software
Continuous Technology Refresh Re-Architecture,
Task Order T1

Document No. AWP.TRG.SWCTR/TOT1-02.00
28 February 2008

Prepared Under

Contract DG133W-05-CQ-1067
Advanced Weather Interactive Processing System (AWIPS)
Operations and Maintenance

Prepared by:

Raytheon

Raytheon Technical Services Company LLC
8401 Colesville Road, Suite 800
Silver Spring, MD 20910

This document includes data that shall not be duplicated, used, or disclosed – in whole or in part – outside the Government for any purpose other than to the extent provided in contract DG133W-05-CQ-1067. However, the Government shall have the right to duplicate, use, or disclose the data to the extent provided in the contract. This restriction does not limit the Government's right to use information contained in this data if it is obtained from another source without restriction. The data subject to this restriction are contained in all sheets.

Table of Contents

	<i>Page</i>
1 Objective.....	1
2 Description.....	1
3 Basic μ Engine Task Structure.....	2
4 Java/JavaScript Interface	3
5 Example: A Logging Task.....	3
5.1 The μ Engine Script.....	3
5.2 Design.....	4
5.3 The SystemLog Class	4
5.4 The Log Entries.....	5
6 Additional Considerations	5
7 Setup Prior to Task Creation.....	6
8 Procedure for Creating a μ Engine Task.....	6
9 Known Issues with μ Engine Task Creation.....	7
Appendix A. References.....	A-1
Appendix B. Acronym List.....	B-1

List of Figures

	<i>Page</i>
Figure 1. μ Engine Data Transformation.....	1
Figure 2. μ Engine Script Building Blocks.....	1
Figure 3. ScriptTask Class Diagram.....	2
Figure 4. SystemLog Task Code Example	3
Figure 5. SystemLog Class Diagram	4
Figure 6. SystemLog Source Code	5
Figure 7. SystemLog Sample Output.....	5
Figure 8. μ Engine Query Package Class Diagram	6

1. Objective

This document outlines the basic procedure for creating an AWIPS II Environmental Data Exchange (EDEX) MicroEngine (μ Engine) task.

2. Description

The Environmental Data Exchange (EDEX) provides the server functionality of the AWIPS Development Environment (ADE). It consists of two parts, a static runtime environment and modifiable software – the AWIPS II EDEX – which is based on the design patterns of the Service Oriented Architecture (SOA). Also included is the μ Engine Web, which is a set of Apache Tomcat-based Web pages that are available for EDEX development testing and demonstration. For development purposes, the ADE supports building and installation of the EDEX software, as well as the μ Engine demo, on a development computer.

Within the AWIPS II EDEX, the μ Engine provides a scripting capability for transforming raw input data into various displayable products. The actual transformation is performed by the μ Engine running inside one of the EDEX services, e.g., the Auto Build Service. Figure 1 provides a basic view of this data transformation process.

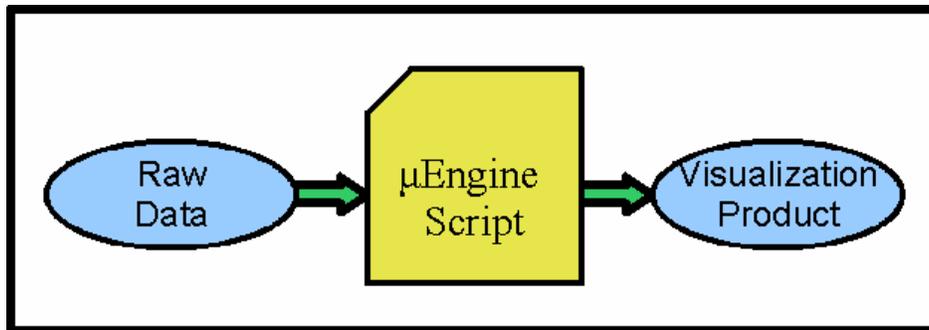


Figure 1. μ Engine Data Transformation

The AWIPS II EDEX ADE supports a multi-tiered approach to script writing, as shown in Figure 2.

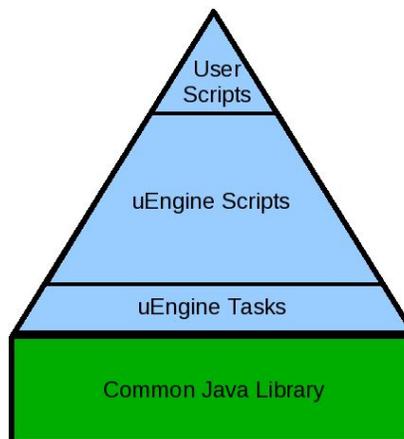


Figure 2. μ Engine Script Building Blocks

User scripts are short scripts that are normally generated by a client application, such as the Common AWIPS Visualization Environment (CAVE), based on user inputs. μ Engine scripts are library scripts, normally defining JavaScript classes, and are referenced from within user scripts. User scripts and μ Engine scripts are written in JavaScript and are covered elsewhere. The AWIPS II ADE includes the source code for a number of user scripts and μ Engine scripts.

A μ Engine Task is a Java class that is designed to perform a specific computation. The Task also forms a bridge between the JavaScript used to write μ Engine Scripts and the Common Java Library, which implements most EDEX functionality.

The remainder of this document discusses the creation of μ Engine Tasks.

3. Basic μ Engine Task Structure

In the μ Engine, each μ Engine task serves as a bridge between the μ Engine script, which is written in JavaScript, and the EDEX common library, which is written in Java. The μ Engine Task may be self-contained, but more commonly it utilizes additional Java classes to perform its functions.

A μ Engine task is implemented as a plain Java class that extends the abstract ScriptTask class, `com.raytheon.edex.uengine.tasks.ScriptTask`. ScriptTask defines the functionality required for a μ Engine task. Most μ Engine Task implementations follow Java's "Bean" pattern¹. The basic design of ScriptTask is shown in Figure 3.



Figure 3. ScriptTask Class Diagram

ScriptTask provides a LOG4J logger and defines a single public method that must be implemented by each μ Engine task. This method is:

1. `execute()`: performs the actions needed to carry out the task's processing.

`execute()` returns a Java Object containing the results of its primary operation. In a chain of Tasks that makes up a μ Engine script, this return value is normally passed to the next task.

In addition, each ScriptTask implementation must provide at least one public constructor. The μ Engine Task may provide additional methods, both private and public, as needed. The μ Engine Task may also utilize other Java classes to perform its function.

In general, utilization of a Task within a μ Engine script follows a well-defined pattern. This basic pattern is:

¹ All class attributes are private and accessed via "getters" and "setters."

1. Instantiate the Task,
2. Set any additional properties on the Task,
3. Call the Task's execute() method, and
4. Retrieve any results from the Task.

Steps 2 and 4 are optional and may be omitted when using the simplest Tasks.

In general, a μ Engine Task is designed to take a single input data set and produce a single output data set.

4. Java/JavaScript Interface

There are a couple of additional considerations for writing μ Engine Tasks.

1. Because the μ Engine Task provides the bridge between Java and JavaScript, all public methods in the task must be able to accept objects. In addition, JavaScript is very weakly typed. As a result, you cannot overload Java functions (that are called from JavaScript) where the only difference in the footprint is the argument types.
2. Java and JavaScript handle arrays differently. In JavaScript, there are no "arrays of primitives" such as an `int[]` or `float[]`. Rather, JavaScript arrays are more like Java `ArrayList` or `HashMap` objects. Because of this, if the JavaScript μ Engine script must iterate the result of executing a task, the `execute()` method should return an `ArrayList`.

As a general rule, μ Engine Tasks should be written to expect and work on objects. Note that, starting with Java 5, the Java Virtual Machine (JVM) provides "auto boxing" between Java primitives and matching class objects. This allows easier passing of primitives between Java and JavaScript.

For additional information on existing μ Engine tasks, see the JavaDoc or examine the source code for the various tasks in the AWIPS EDEX μ Engine code baseline.

5. Example: A Logging Task

(This is the "Hello World" example, written as a μ Engine script.) This example demonstrates a simple μ Engine task that prints a message to the EDEX/Mule system log.

5.1 The μ Engine Script

A simple μ Engine action script that uses a logging task is shown in Figure 4.

```
// create the logger
var logger = new SystemLog();
// log a message
logger.log("info", "This is a log message.");
```

Figure 4. SystemLog Task Code Example

5.2 Design

The Logging Task will use the LOG4J logger provided by ScriptTask to write to the EDEX system log. For this reason, it will need to support the logging levels provided by LOG4J. Because the logger may be used multiple times in the same μ Engine script, a separate method, `log(String, String)`, is provided to perform the logging. The class diagram is shown in Figure 5.

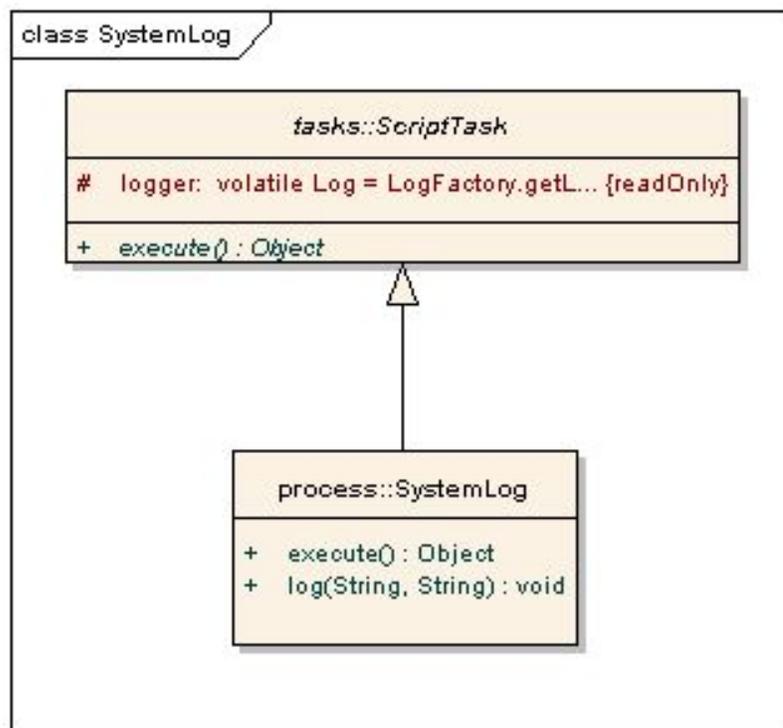


Figure 5. SystemLog Class Diagram

5.3 The SystemLog Class

The complete code for the SystemLog class is shown in Figure 6.

```
package com.raytheon.edex.uengine.tasks.process;

import com.raytheon.edex.uengine.tasks.ScriptTask;

public class SystemLog extends ScriptTask {

    private String level = "info";
    private String message = "";

    @Override
    public Object execute() {
        if (level.equalsIgnoreCase("fatal")) {
            logger.fatal(message);
        } else if (level.equalsIgnoreCase("warn")) {
            logger.warn(message);
        } else if (level.equalsIgnoreCase("error")) {
            logger.error(message);
        } else if (level.equalsIgnoreCase("info")) {
            logger.info(message);
        } else {
            logger.debug(message);
        }
        return null;
    }
    public void log(String level,String message) {
        this.level = level;
        this.message = message;
        execute();
    }
}
```

Figure 6. SystemLog Source Code

Note: To save space in this document, this listing does not include any comments.

5.4 The Log Entries

A partial log listing from running a μ Engine script using SystemLog is shown in Figure 7.

```
INFO 2008-02-12 17:11:18,284 [Awips.Edex.Service.ProductSrv.2]
SystemLog: This is a log message.
```

Figure 7. SystemLog Sample Output

6. Additional Considerations

Each μ Engine task must extend the abstract ScriptTask base class. This allows EDEX to use an introspective approach to generating a list of available μ Engine Tasks. It is not necessary, however, for the Task to directly extend ScriptTask. In some cases, it is desirable to group Tasks that share common functionality and factor that common functionality into an intermediate class. An example of this exists in the μ Engine query tasks, which are located in the com.raytheon.edex.uengine.tasks.query package. The class diagram for part of this package is shown in Figure 8.

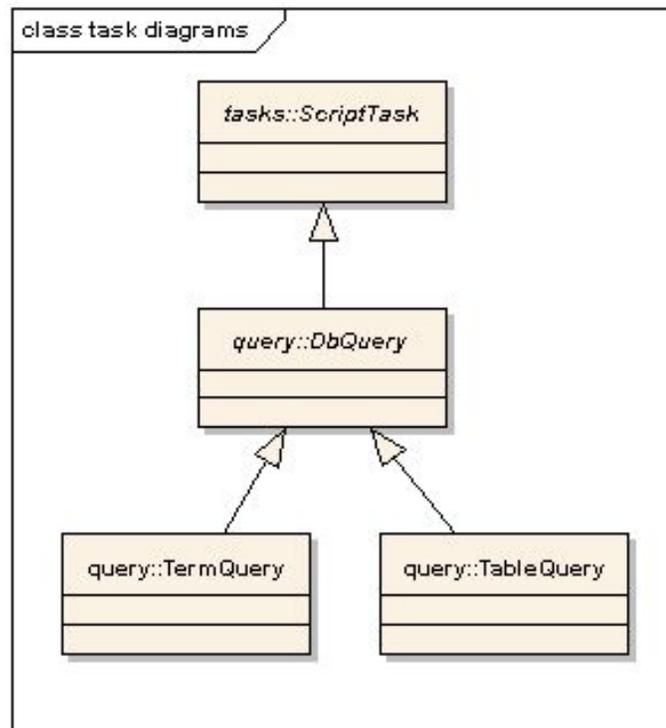


Figure 8. μ Engine Query Package Class Diagram

As Figure 8 shows, both TermQuery and TableQuery extend DBQuery, which in turn extends ScriptTask. DBQuery is an abstract class that provides many of the attributes required for the database queries implemented in TermQuery and TableQuery. It provides getters and setters for these attributes as well as methods for setting the search parameters used in the queries. Both TermQuery and TableQuery provide additional attributes as well as independent implementations of execute().

7. Setup Prior to Task Creation

Prior to creating a μ Engine task, the AWIPS ADE, including both AWIPS EDEX and the ADE, must be installed. See the AWIPS ADE documentation for information on installing the ADE.

8. Procedure for Creating a μ Engine Task

Note: The following procedure is a very general list of the steps required to create a μ Engine task. It assumes the reader is familiar with both the Java programming language and the Eclipse IDE. In most cases, studying the code of the existing μ Engine tasks will provide insight into the implementation of a new task.

1. Prior to creating the task, determine the general design of the task. In particular:
 - a) Determine the name to use for the task. Task names must be unique. It is also helpful to determine the general location within the EDEX code and package structure where the task should reside.
 - b) Determine the inputs required for the task

- Identify the inputs that will be provided by prior tasks in the μ Engine script. These inputs must match the types that are output by those tasks.
 - Decide which inputs should be provided via the Task's constructor and which inputs can be provided via other methods.
- c) Identify the desired output from the task.
 - d) Identify the processing required to transform the task's inputs into the desired output.
2. Using the transformation identified in (1d) and the output identified in (1c), implement the execute() method. Once the execute() method transforms the data, it can either return the data directly or set class attributes so that the information may be retrieved later using getter methods. See the source code for the existing μ Engine tasks for examples of implementing the execute() method.
 3. Use ANT to compile and deploy the modified μ Engine. See the ADE documentation for information on compiling and deploying the EDEX μ Engine.

9. Known Issues with μ Engine Task Creation

There are two main issues to consider when creating a new μ Engine Task.

1. As of AWIPS II TO 8, μ Engine Tasks can only exist in two places in the code baseline. A Task can be created as part of the μ Engine component, which is located in awips/trunk/edex/edex/uEngine in the code baseline. A Task can also be created as part of a data-type plug-in, which is located in awips/trunk/edex/extensions in the code baseline.
2. For the Task to compile correctly, the component containing the task must include the μ Engine in its dependency list.

Appendix A. References

The AWIPS EDEX:

1. An up-to-date (as of TO 8) listing of μ Engine Tasks is contained in AWIP II EDEX Micro Engine Tasks, which is in the docs directory on the ADE CD.
2. AWIPS EDEX/CAVE training materials (AWP.TRG.SWCTR/TO6.ADE/CAVE-00.01 [Rev. 1]), located on the ADE Resources CD delivered 7 February 2008 under TO8. Refer especially to Module 3, MicroEngine Scripting.
3. The AWIPS EDEX code baseline is located on the Installer's media. The code is normally installed on a development computer. See the Release Notes (ReleaseNotes.txt) for more information.
4. AWIPS EDEX documentation is located in the docs directory on the ADE Resources CD delivered 7 February 2008 under TO8.
5. JavaDoc is embedded in the baseline code. The Web version is generated using the ANT build script. See the "AWIPS ADE EDEX Build Procedure" (AWIPS_ADE_EDEX_Build_Procedure.pdf), located on the ADE Resources CD delivered 7 February 2008 under TO8 for information on generating the browser viewable JavaDoc.

Appendix B. Acronym/Abbreviation List

ADE	AWIPS Development Environment
ANT	(not an acronym) Another Nifty Tool, a Java oriented “build” tool
AWIPS	Advanced Weather Information Processing System
EDEX	Environmental Data Exchange
IDE	Integrated Development Environment
JAR	Java ARchive
JDK	Java Development Kit
JRE	Java Runtime Environment
JVM	Java Virtual Machine
SOA	Service Oriented Architecture
μEngine	MicroEngine
XML	eXtensible Markup Language