

Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

TO9 Developer Briefing - Agenda

September 9, 2008



Purpose of Course

- Early developer-level introduction to facilitate cooperative development
 - Trying to evolve project toward an Open Source core
 - Everything a part of the baseline and open to improvement
- Early focus on architecture and design patterns
 - Get the big picture right, before moving into specific capabilities
 - Widen exposure to get more creative input
- Workstations with full installation of ADE 1.0
 - Source with Eclipse IDE
 - Server Side Run environment
 - CAVE visualization
 - Javadocs and other documentation



Training Prerequisites

- Reading materials
 - *Software Product Improvement Plan*
- Software
 - Pure Java
 - <http://java.sun.com/docs/books/tutorial>
 - CAVE: ECLIPSE IDE Framework & Plug-Ins
 - <http://www.eclipse.org>
 - Eclipse RCP
 - EDEX: Introductory level of Spring and Mule ESB
 - <http://mule.codehaus.org>
 - All: Introductory level of ANT
 - All: Introductory level of XML



Course Content

- Today's course consists of 5 segments
 - A look back to Task Order 8 training
 - Three new modules detailing important modifications made in Task Order 9
 - A look ahead to modifications planned for Task Order 10



Course Objectives

Module 13: TO9 EDEX Updates

- Describe EDEX platform updates
- Describe critical DR fixes
- Describe new (Python based) scripting engine
- Describe AWIPS II localization changes
- Describe database updates
- Describer Mule service endpoint enhancements
- Describe data decoder enhancements

Module 14: TO9 CAVE Updates

- Describe CAVE platform updates
- Addition of Python interface
- Describe updates to WarnGen templates
- Describe localization updates
- Describe scripting engine updates
- Describe derived parameters updates
- Startup using cave.sh



Course Objectives (Continued)

Module 15: TO9 AWIPS II ADE Updates

- Describe updates to platform
- Describe updates to installers
- Creation of flow-tags for system installation



TO 8 Look Back



TO 8 Training Material Additions/Corrections

- Updated training materials
 - *Launching CAVE from the Baseline.ppt* – incorrect Eclipse plug-in identified in the TO 8 training materials.
- Supplementary materials
 - *AWIPS II EDEX DB Purge Scripts.doc* – contains sample scripts for manually purging EDEX database
 - *Briefing questions.doc* – written answers to questions submitted prior to TO 8 training.
 - *Build Shell Scripts.doc* – shell scripts that can help simplify building and deploying EDEX
 - *Useful Developer Tools.doc* – descriptions of tools used by EDEX developers



Questions?



Module 13 – TO 9 EDEX Updates



Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

Module 13: EDEX Updates for TO 9

September 9, 2008



Objectives

- Upon completion of the module, the student will understand the modifications to the EDEX architecture that were implemented in AWIPS II TO 9



Topics

- Platform Updates for TO 9
- Memory leak fixed, other DR's
- Addition of Python scripting engine
- EDEX localization changes
- Database Updates
- Endpoint enhancements
- Decoder enhancements



Platform Updates



Platform Updates - Java

- Update:
 - Java has been updated to Java 1.6.0_05 (from 1.6.0_01)
- Rational:
 - Latest Java Version available at the appropriate time in the TO, contains latest Java bug fixes and enhancements
- Impacts:
 - Required update of several support packages
- Install:
 - packaged with AWIPS II Installers



Platform Updates - PostgreSQL

- Update:
 - Postgres has been updated to PostgreSQL 8.3.0
- Rational:
 - Latest Version available at the appropriate time in the TO, contains latest bug fixes and enhancements
- Impacts:
 - Minimal changes required – will need a need version of pgAdmin III
- Install:
 - packaged with AWIPS II Installers



Questions?



Critical Fixes



AWIPS II Problem Fixes

- The TO 8 version of EDEX shipped with a major memory leak. This leak has been corrected – TO 9 EDEX has much more stable memory utilization
- The TO 9 release includes a number of DR fixes. This include fixes to TTR 1, TTR 34, TTR 39, TTR 47, TTR 48 and TTR 113
 - This list includes both CAVE and EDEX fixes



Questions?



Scripting Enhancements



Scripting Enhancements

- In TO 6, the EDEX μ Engine was converted from an XML based language to JavaScript
 - Information on why the change was made is included in the updated training package (Module 3) delivered with TO8.
 - Module 3 also includes
 - information on writing μ Engine tasks in Java
 - writing μ Engine scripts in JavaScript, including the preferred tri-tiered approach to μ Engine scripts
- In TO 9, AWIPS II scripting has been expanded to add scripting capability in Python



Why Python?

- Python is a familiar scripting language within NWS
- We first looked at incorporating Python like scripting in TO 6.
 - The pre-TO 6 μ Engine had some capability to run scripts written in Jython, which is an all Java implementation of Python
 - There were performance problems with Jython
- When we decided to rewrite (and simplify) the μ Engine in TO 6, we looked at using Jython and JavaScript as replacement languages
 - Jython was a promising replacement, but suffered from lack of community support
 - We were unable to find a bridge between Java and Python
 - JavaScript (Rhino) is a standard embedded scripting language for Java



Why Python?

- We decided to convert the μ Engine to JavaScript
 - This enabled an extreme simplification of μ Engine tasks
- We also created a tri-tiered model for μ Engine scripts
 - This enables client applications to efficiently auto generate μ Engine scripts
 - Both CAVE and the μ Engine test driver use this capability
- As we looked at implementing Derived Parameters for CAVE in TO 9
 - it became evident that the current μ Engine (using JavaScript) would need require porting Python code to Java (as μ Engine tasks)
 - Among the implications: accuracy of algorithms ported from Python to Java



Why Python?

- The decision was made to again look for a bridge between Java and Python
 - A product called JEP (Java Embedded Python) was identified as such a bridge
- JEP allows:
 - Java to run Python scripts, and
 - Python scripts to utilize Java objects
- As a result, we were able to create a Python based μ Engine that
 - supports the tri-tiered architecture to μ Engine introduced in TO 6, and
 - is able to utilize μ Engine tasks written for that architecture
- EDEX now supports scripting in both JavaScript and Python
 - μ Engine written for TO 8 EDEX will still work



More on JEP

- From the Java Embedded Python web site:
 - “Jepp embeds CPython in Java. It is safe to use in a heavily threaded environment, it is quite fast and its stability is a main feature and goal.”
- AWIPS II includes a custom version of JEP that includes better exception handling between Java and Python.
 - always use the JEP jar from the AWIPS II baseline
 - source code is available in the AWIPS II baseline
 - Once inside JEP, any exception are re-cast to a JepException.
 - Stack traces are included
- both CAVE and EDEX require an environment variable
 - LD_PRELOAD – normally set by the installer and startup scripts



Python Benefits

- Speed: Python scripts run faster than JavaScript, in some cases faster than Java
- Familiarity: Python is widely used within the NWS
- Reuse: the same scripts can be used by μ Engine, GFE derived parameters, and GFE tools
- Reuse: can import Python modules
- Reuse: (with JEP) can import Java classes



Java/Python Bridge Issues

- Java classes are usable from inside Python
 - existing μ Engine tasks can be utilized, generally without modification
- A Java class constructor can't take a Java array as an argument
 - you can pass the array as an object and cast it to an array
 - you can use a setter to set the array into the object after it has been constructed
- Java can't generally handle Python objects
 - Java can handle Python primitives and strings
 - Java objects can be passed in and out of the Java/Python interface



Java/Python Bridge Issues

- Java can't handle Python objects; specifically
 - Problem: a Python list can't be passed to Java
 - in Python, a list is a resizable array of object references
 - the comparable structure in Java is the ArrayList
 - Solution: create and populate a Java ArrayList in Python and pass it to the Java object.
- for a Java object *obj*, calling *str(obj)* in Python is the same as calling *obj.toString()* in Java



Python μ Engine Scripts in AWIPS II

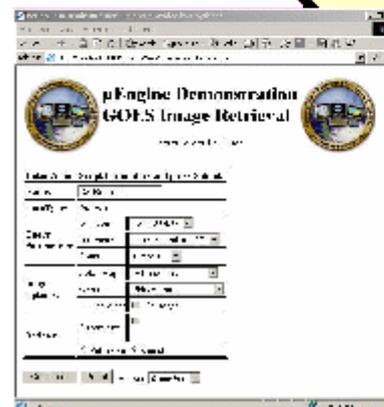
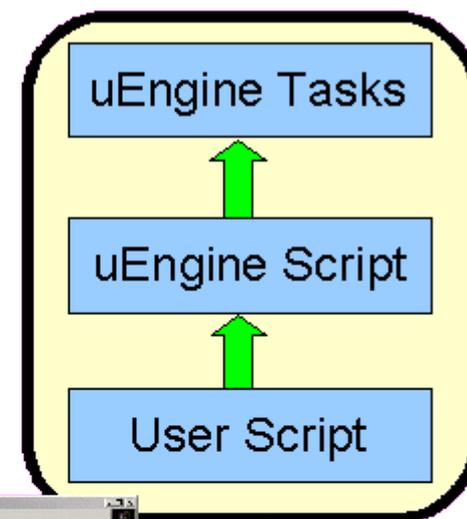
- In the AWIPS II ADE baseline, Python μ Engine scripts are located in AWIPSEdex/opt/utility/edex_static/base/python
- In the running environment, Python μ Engine scripts are located in awips/edex/opt/data/utility/edex_static/base/python
 - requires an EDEX/Mule restart before a new script is available



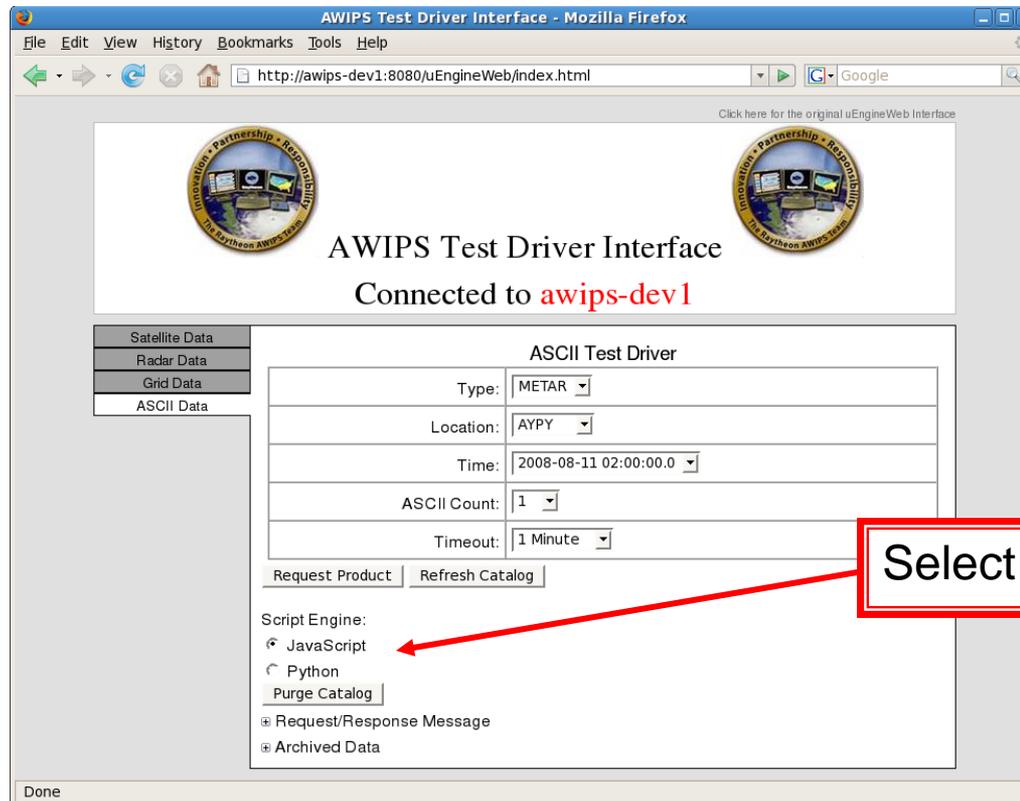
μEngine Scripting: Three-Tiered Approach

- μEngine supports a three-tiered approach to script writing
 - μEngine tasks are created to perform a specific function such as querying the data store
 - a μEngine class is created to perform a general task such as retrieving satellite imagery.
 - A user writes a short script to utilize the μEngine class.
 - the μEngine class and user script are written in the same language, either JavaScript or Python

Note: The actual code for creating the script may be hidden behind a GUI interface.



EDEX Test Driver Support



- In TO 9, the EDEX test driver supports both the new Python μ Engine and existing JScript μ Engine.
 - Both μ Engine scripts call the same underlying Java classes



Exercise:

The next few slides provide a comparison of the METAR retrieval scripts used by the EDEX test driver



Test Driver Scripts – Top Tier

JavaScript version:

```
include("BaseRequest.js");
var dataRequest = new
    BaseRequest("obs");
dataRequest.setCount(1);
dataRequest.addParameter("reportType",
    "METAR");
dataRequest.addParameter("stationID",
    "AYPY");
dataRequest.addParameter("dataTime",
    "2008-08-11 02:00:00.0");
dataRequest.execute();
```

Python version:

```
import BaseRequest
dataRequest =
    BaseRequest.BaseRequest("obs")
dataRequest.setCount(1)
dataRequest.addParameter("reportType",
    "METAR")
dataRequest.addParameter("stationID",
    "AYPY")
dataRequest.addParameter("dataTime",
    "2008-08-11 02:00:00.0")
return dataRequest.execute()
```



Data Retrieval Scripts – Middle Tier

- Using Eclipse, look at BaseRequest.js and BaseRequest.py in the code baseline
- Note:
 - Python provides a somewhat cleaner coding style
 - Python provides better support for class inheritance



Exercise: Hello World Revisited

Problem:

Implement a “Hello World” script for the Python μ Engine similar to the example from TO8. The script will log a “Hello World” message to the EDEX log and echo the response to the client.

Solution:

The implementation requires three components

- the μ Engine task to log the message
- the tier two (library) script to handle the logging echo the message
- the tier one script sent by the client



HelloWorld: Tier Three

- Tier three is the Java code available via the μ Engine
 - For this example, we add a μ Engine task called *SystemLog* that extends *ScriptTask*
 - This task is described in AWIPS II Training materials
 - See the μ Engine documentation for more on task creation

```
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();
    }
}
```



HelloWorld: Tier Two

```
from com.raytheon.edex.uengine.tasks.process import SystemLog
from com.raytheon.edex.uengine.tasks.query import TermQuery
from com.raytheon.edex.uengine.tasks.response import MakeResponseNull

class HelloWorld():

    def __init__(self):
        self.message = ""

    def setMessage(self,message):
        self.message = message

    def execute(self):
        logger = SystemLog()
        logger.log("info",self.message)
        return MakeResponseNull(self.message,TermQuery("obs")).execute()
```

- This script represents a Python class having
 - single attribute, message, with a setter
 - a single instance method, execute()
- Although a simple class, this general pattern is followed for most tier two scripts



HelloWorld: Tier One

This is a simple client script

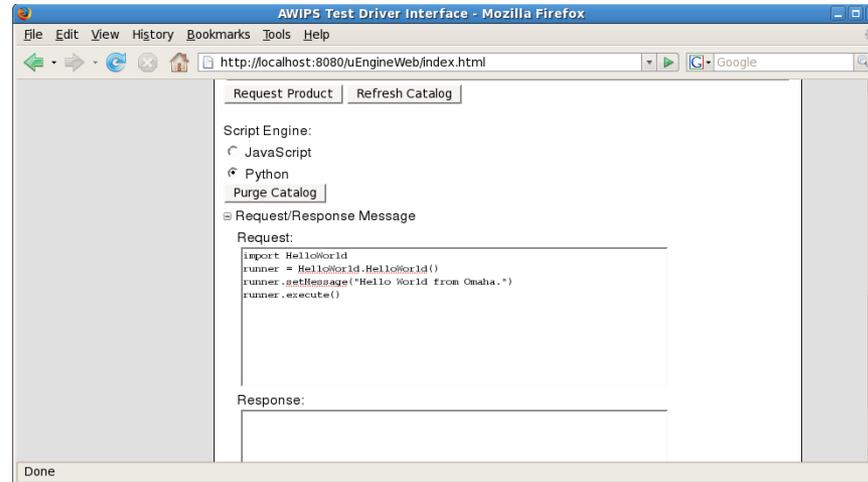
- Uses the HelloWorld script to do most of the work
- Client scripts follow this general pattern
 - import the tier two script
 - create a script instance
 - set script attributes
 - call the *execute* method.

```
import HelloWorld  
  
runner = HelloWorld.HelloWorld()  
  
runner.setMessage("Hello World from Omaha.")  
  
runner.execute()
```

Note: this script can auto generated by the client using a Velocity template



HelloWorld: Testing the Script



- μ Engine scripts can be tested using the AWIPS Test Driver
 - Note that this script will return an error message (“no response”) to the browser
- The EDEX log message is shown below

INFO 2008-08-12 15:26:14,723 [Awips.Edex.Service.PyProductSrv.2]
SystemLog: Hellow World from Omaha.



Questions?



Apache Velocity



Apache Velocity

- Velocity is a template language that is used by CAVE to facilitate auto generation of client μ Engine scripts based on user inputs
- It is also used by the μ Engine test driver generate product retrieval scripts based on how the user fills out a web form
- Velocity templates are discussed in detail in module 14.



Questions?

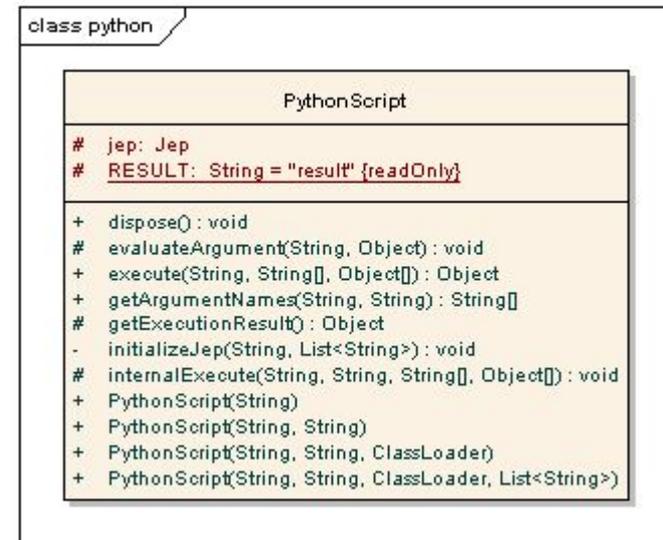


Python outside of the μ Engine



Python from Java: PythonScript Class

- Both CAVE and EDEX may use Python without using the μ Engine
 - For example, CAVE may bridge to existing Python applications
- the PythonScript class provides the basic functionality
 - may be extended to provide additional functionality
- Examples of extentions:
 - DerivParamScript
 - SmartInitScript



Python from Java: PythonScript Class

- The PythonScript constructor takes up to 4 arguments
 - (required) A the path to a python script to run.
 - (optional) A Python include path of directories of modules you'd like to import
 - multiple directories separated by colon (:)
 - may be specified as an empty string ("")
 - You don't need to include directories already on the python path.
 - (optional) A Java ClassLoader to ensure Java classes are accessible from Python.
 - (optional) A list of Strings of Python code to evaluate before the python script is run.
- Arguments must be included in the order listed
 - i.e., to include a class loader, you must specify the Python include path, which can be empty



Python from Java: PythonScript Class

■ Other key methods

- `getArgumentNames(String, String)`: returns a `String[]` containing the argument names for a method. Throws `JepException`.
- `execute(String, String[], Object[])`: returns an `Object` containing the results of executing the scripts. Throws `JepException`.
- `dispose()`: disposes the underlying JEP instance. Should be called to release system resources.

See the JavaDoc for additional information.



Python from Java: PythonScript Class

- Utilizing PythonScript
 - Instantiate PythonScript with the appropriate arguments
 - Call getArgumentsNames() to get a list of argument names
 - Create an Object[] containing the argument values
 - Call dispose() to release the system resources



Exercise:

Problem:

Create a Python “Hello World” script and provide code for executing the script from Java

Solution:

Implementation involves two components

- The Python script that will print the “Hello World” message
- The Java method to execute the script



Hello World: Python From Java

- This is a simple Python script with a single static method
 - you would save the script in a file named *PrintHello.py*
- The script simply prints and returns a “Hello World” message

```
def myMethod(argument):  
    hello = "Hello world and " + str(argument)  
    print hello  
    return hello
```



Hello World: Python From Java

- The Java driver:
 - creates a PythonScript object
 - gets the argument list for *myMethod*
 - sets up the arguments for *myMethod*
 - calls *myMethod* and captures the result
 - prints the result
- Also note: the call to the *dispose()* method is in the *finally* block to ensure system resources are released

```
import com.raytheon.edex.python.PythonScript;
import jep.JepException;

public class PythonDriver {

    public static void main(String[] args) {
        PythonScript py = null;
        try {
            py = new PythonScript("PrintHello.py");
            String argName = py.getArgumentNames("myMethod",
                null)[0];
            String result = (String) py.execute("myMethod",
                new String[] { argName },
                new Object[] { "developer" });
            System.out.println(result);
        } catch (JepException e) {
            e.printStackTrace();
        } finally {
            if (py != null) {
                py.dispose();
            }
        }
    }
}
```



Questions?



EDEX Localization



EDEX Localization

- EDEX provides a centralized localization repository for AWIPS II
 - In the baseline, localization files are in trunk/edex/opt/utility/...
- AWIPS II uses the concept of contexts to separate files depending on where they are used
 - Current contexts are
 - cave_config: contains CAVE configuration files
 - cave_plugin: contains CAVE plug-in based files such as menu definitions
 - cave_static: contains CAVE specific static files such as color maps
 - edex_static: contains EDEX specific static files such as Python μ Engine scripts
 - common_static: contains static files used by both CAVE and EDEX
- Other localization changes are covered in Module 14.



Questions?



Database Updates



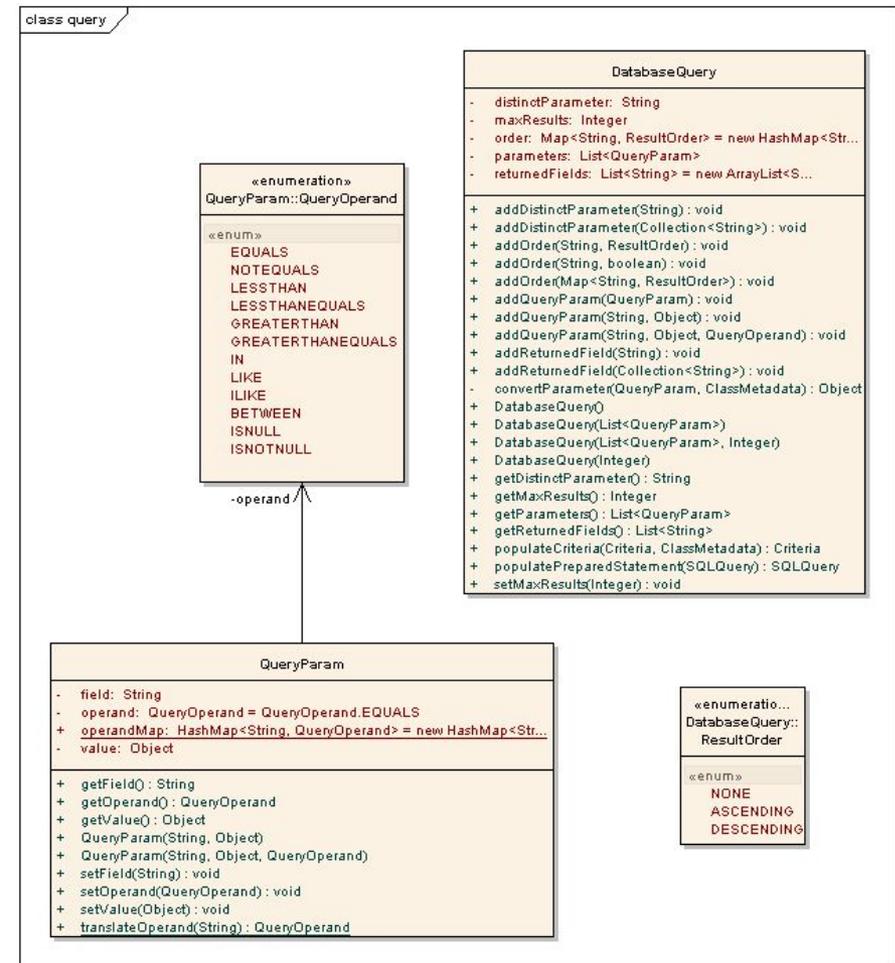
Database Updates

- added new package, *com.raytheon.edex.db.query*, for performing database queries
 - provides a standard API for specifying EDEX database queries
- the use of Data Access Object (DAO) pooling has been deprecated
 - adds a redundant layer on top of the database connection pooling provided by Hibernate
 - code utilizing the DAO pool will continue to work in TO9
 - Clients using μ Engine are not effected by this change



com.raytheon.edex.db.query Package

- There are two classes
 - DatabaseQuery:
 - encapsulates a database query.
 - Provides several constructors and other methods for building queries
 - QueryParam:
 - encapsulates a single *where* clause a query
 - multiple QueryParam objects are *anded*
- There are two enums
 - QueryOperand:
 - defines available operations
 - defined in QueryParams
 - ResultOrder:
 - defines return order of query
 - defined in DatabaseQuery



Exercise:

Problem:

Write a query to retrieve the latest 5 METAR records for KOMA. The data decoder to use is *plugin-obs*.

Note: this is essentially the same as ***select * from awips.obs where reporttype='METAR' and stationid='KOMA' order by insert_time desc limit 5***

Solution:

Use the TO9 DB API – details on the following slides



Solution:

- This code snippet illustrates the setup of the DatabaseQuery object
 - Note that the table is specified when selecting the DAO to use for the query and is not shown here

- This illustrates the setup for a ***select * from ...*** style query.
 - DatabaseQuery can also specify the return of specific fields.

```
import com.raytheon.edex.db.query.DatabaseQuery;

public Object myQuery() {
    Object retVal = null;
    DatabaseQuery query = new DatabaseQuery();
    query.addQueryParam("reporttype", "METAR");
    query.addQueryParam("stationid", "KOMA");
    query.setMaxResults(5);
    query.addOrder("insert_time", false);
    /* additional code to perform and use the query */

    return retVal;
}
```



Questions?



Endpoint enhancements



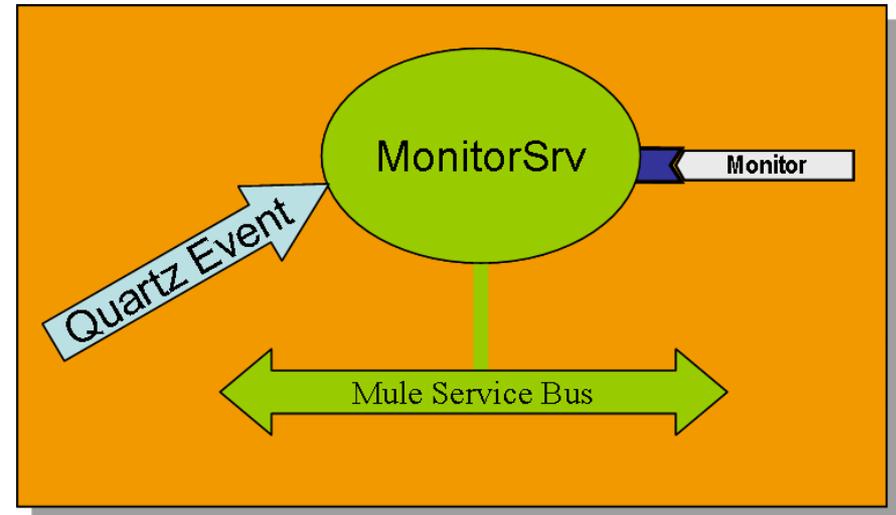
EDEX Endpoint Enhancements

- Monitoring endpoint
- ability to turn off memory logging for an endpoint
- JMX control of logging by logger



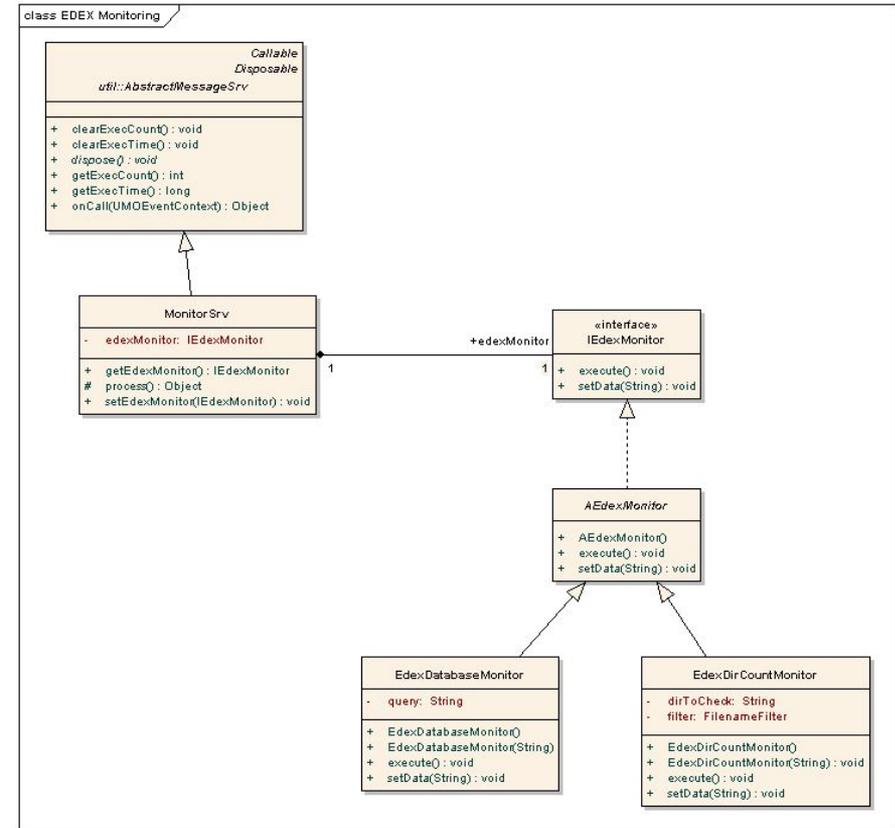
Monitoring Endpoint

- TO 9 provides a monitor endpoint that allows EDEX to periodically monitor itself and report status to the system log
- Monitor endpoint is usually triggered by a timer event
- Monitor endpoint uses pluggable monitors to perform the actual monitoring



Monitoring Endpoint – Class Diagram

- The MonitorSrv extends the EDEX AbstractMessageSrv class
 - Mule uses injection to provide the endpoint with an IEdexMonitor instance
- IEdexMonitor is an interface that specifies basic functionality of the monitor
 - implementations include EdexDatabaseMonitor and EdexDirCountMonitor
- Monitors are configured in Mule as Quartz (timer) driven endpoints



Monitoring Endpoint – Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
    "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <!-- the database monitor -->
  <bean id="DirCountMonitor"
    class="com.raytheon.edex.monitors.EdexDirCountMonitor">
    <constructor-arg index="0">
      <value>../../processing</value>
    </constructor-arg>
  </bean>
</beans>
```

- This is part of a bean specification document for Mule
 - This specifies a bean called *DirCountMonitor*
 - This bean will log a count of files in a single directory
 - The implementation is *EdexDirCountMonitor*
 - The constructor takes a single argument
 - the value, *../../processing*, is the directory to monitor



Monitoring Endpoint – Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mule-configuration PUBLIC
"-//MuleSource //DTD mule-configuration XML V1.0//EN"
"http://mule.mulesource.org/dtds/mule/mule-1.4.0-spring-configuration.dtd">

<mule-configuration version="1.0">
  <model name="edex" type="seda">
    <mule-descriptor name="Awips.Edex.Service.MonitorSrv-processing"
      singleton="true"
      implementation="com.raytheon.edex.services.MonitorSrv">
      <inbound-router>
        <endpoint name="quartz.monitor.Endpoint.processing"
          address="quartz://processing.monitor">
          <properties>
            <property name="cronExpression" value="0/10 * * * * ?"/>
            <property name="payloadClassName"
              value="org.mule.providers.NullPayload" />
          </properties>
        </endpoint>
      </inbound-router>
      <threading-profile maxThreadsActive="1" maxThreadsIdle="1" />
      <properties>
        <container-property name="edexMonitor"
          reference="DirCountMonitor"
          required="true" />
        <property name="report" value="false"/>
      </properties>
    </mule-descriptor>
  </mule-configuration>
```

This Mule Configuration document specifies a monitor endpoint that uses the *DirCountMonitor* bean to check and report the size of a directory every 10 seconds.



Memory Status Logging

- Memory status logging is now configurable on EDEX service endpoints
 - the default setting is to log memory usage
- To turn off memory logging for an endpoint, set the “report” property for the endpoint to “false”
 - This is configured for the new Monitor Service endpoints



Memory Status Logging

```
<mule-descriptor name="Awips.Edex.Service.MonitorSrv-processing"
  singleton="true"
  implementation="com.raytheon.edex.services.MonitorSrv">
  <inbound-router>
    <endpoint name="quartz.monitor.Endpoint.processing"
      address="quartz://processing.monitor">
      <properties>
        <property name="cronExpression" value="0/10 * * * * ?"/>
        <property name="payloadClassName"
          value="org.mule.providers.NullPayload" />
      </properties>
    </endpoint>
  </inbound-router>
  <threading-profile maxThreadsActive="1" maxThreadsIdle="1" />
  <properties>
    <container-property name="edexMonitor"
      reference="DirCountMonitor"
      required="true" />
    <property name="report" value="false"/>
  </properties>
</mule-descriptor>
```

Turn off memory status logging on this endpoint.



JMX Control of Logging

- TO9 EDEX enables run-time setting of logging levels via JMX
- Logging levels are determined by “logger”
 - generally, for a logger you can use the full class name of the endpoint
 - for example, the logger for IngestSrv is *com.raytheon.edex.services.IndexSrv*
- Logging levels are manipulated using a JMX console
 - JConsole, which ships with Sun’s Java, may be used
- Available logging levels are, in order, DEBUG, INFO, WARN, ERROR, and FATAL
- Log level is set using the *log4j.settings* bean. The operation to invoke is *setLevel(logger,level)*



JMX Control of Logging

There are a couple of considerations

- It is possible to set a level for a logger that is never used
 - This can occur when an invalid class name is entered, for example not entering the correct class name
- Loggers are inherited via the Java package hierarchy
 - within a single Mule endpoint, multiple loggers may be used
 - modifying the logging level of an endpoint may not change the logging for some of the classes used by that endpoint.
 - Example: the Monitoring Endpoint, MonitorSrv uses the *com.raytheon.edex.services* based logger. The actual monitors used by this endpoint utilize *com.raytheon.edex.monitors* based loggers. Since the packages are different, changing the logging level for MonitorSrv will not change the level for the actual loggers.



Exercise:

Problem:

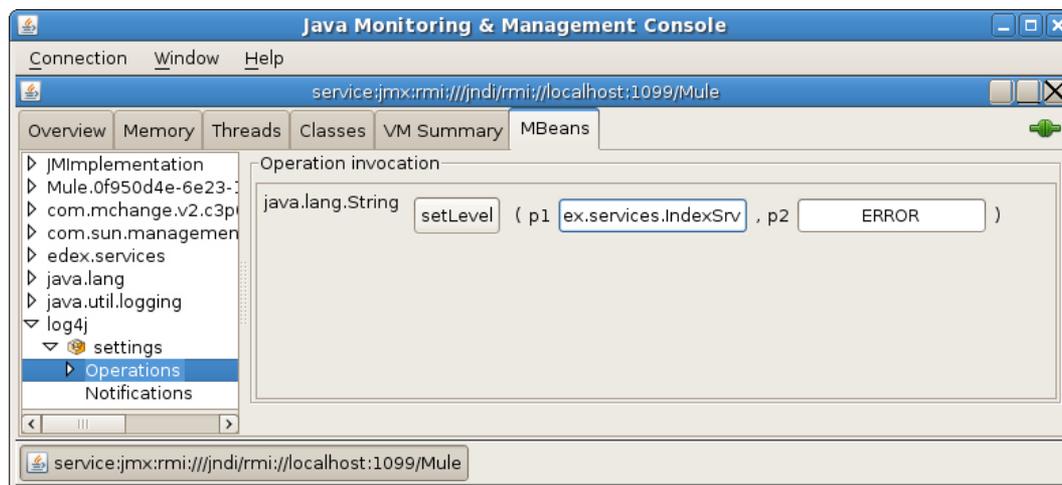
Modify the logging level of IndexSrv to ERROR.

Solution:

Use JConsole to connect to EDEX and change the logging level. The logger is *com.raytheon.edex.services.IndexSrv*. The level to set is ERROR.



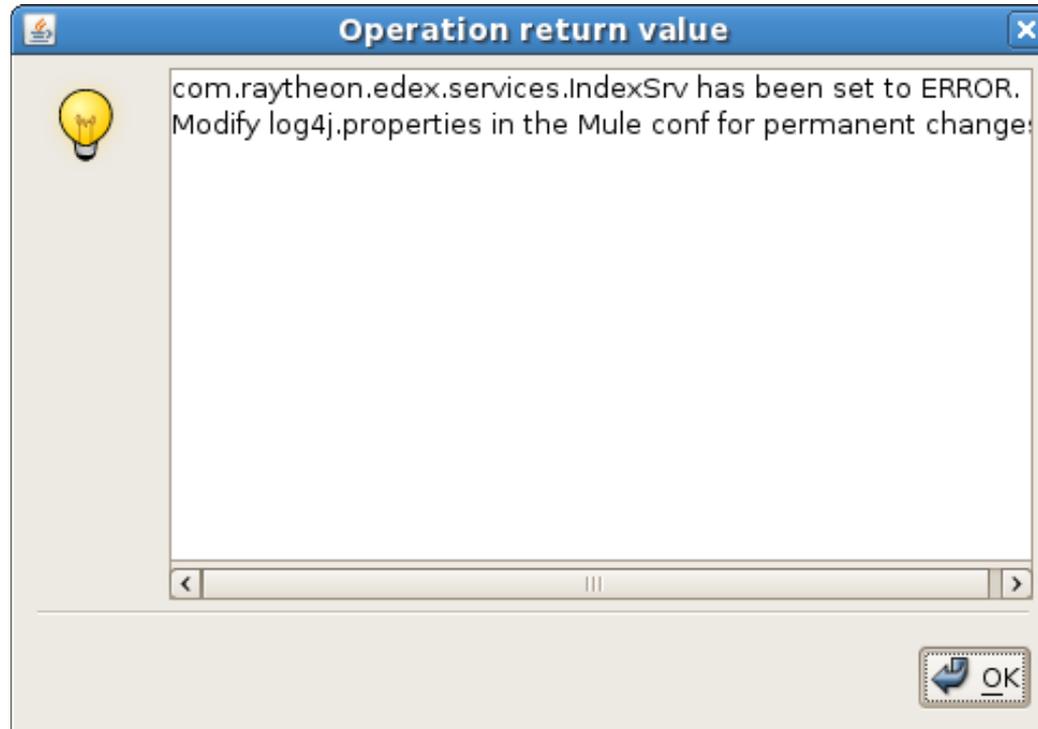
Exercise:



- Once JConsole is running and connected to Mule:
 - Select the *MBeans* tab and scroll down to find the *log4j* entry
 - Expand the *log4j* entry and the *settings* sub-entry
 - Select the *operations* entry
- Enter the values into the *setLevel* operation
 - enter the logger as p1, the level as p2
- Click the *setLevel* button to execute the operation



Exercise:



- After clicking *setLevel*, a confirmation message is displayed
 - click *OK* to dismiss the message



Questions?



Decoder Enhancements



Decoder Enhancements

- The main change made to the decoder plug-ins involves the ability to track products through the system
 - When a file is first processed by StagingSrv, it is assigned a unique identifier
 - The file name and unique id are logged
 - The unique ID is passed to all subsequent ingest end-points
 - In IngestSrv, the unique ID is passed to the actual data decoder
 - All logging in IngestSrv includes the unique ID
 - Most decoders also use the unique ID when logging messages
 - This provides time stamped tracking of files through the entire ingest chain
 - At this time, there is no tracking of files to database insertion available



Questions?



Wrap-Up



Summary

- Covered platform updates for TO 9
- Covered memory leak fixed, other DR's
- Covered addition of Python scripting engine
- Covered EDEX localization changes
- Covered database Updates
- Covered endpoint enhancements
- Covered decoder enhancements



Resources

- On the ADE TO9 DVD
 - Current code available for examination in the ADE baseline
 - JavaDoc documentation available
- Also available
 - TO 9 Training Updates
 - TO T1 Training Materials



Module 14 - TO 9 CAVE Updates



Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

Module 14: Cave Updates for TO9

September 9, 2008



Objectives

- Upon completion of the module, the student will understand the modifications to the CAVE architecture that were implemented in AWIPS II TO 9



Topics

- Platform updates
- Addition of Python interface
- Use of Apache Velocity
- WarnGen Templates
- Localization
- Scripting engine updated
- Derived parameters
- Modified CAVE Startup



Platform Updates



Platform Updates - Java

- Update:
 - Java has been updated to Java 1.6.0_05 (from 1.6.0_01)
- Rational:
 - Latest Java Version available at the appropriate time in the TO, contains latest Java bug fixes and enhancements
- Impacts:
 - Required update of several support packages
- Install:
 - packaged with AWIPS II Installers



Platform Updates - Eclipse

- Update:
 - Eclipse has been updated to Version 3.3.2
- Rational:
 - Latest Version available at the appropriate time in the TO, contains latest bug fixes and enhancements
- Impacts:
 - Minimal changes required – will be installed by the ADE installer
- Install:
 - packaged with AWIPS II ADE Installer
- Important: If you are doing CAVE development, you need to use the Eclipse that is packaged in the ADE Installer



Platform Updates – Other Packages

- Other software has been updated as needed to be compatible with the platform
 - Specific version information is available in the AWIPS II SVD document (included on the install media)



Questions?



Addition of Python Editing Perspective



Python Editing Perspective

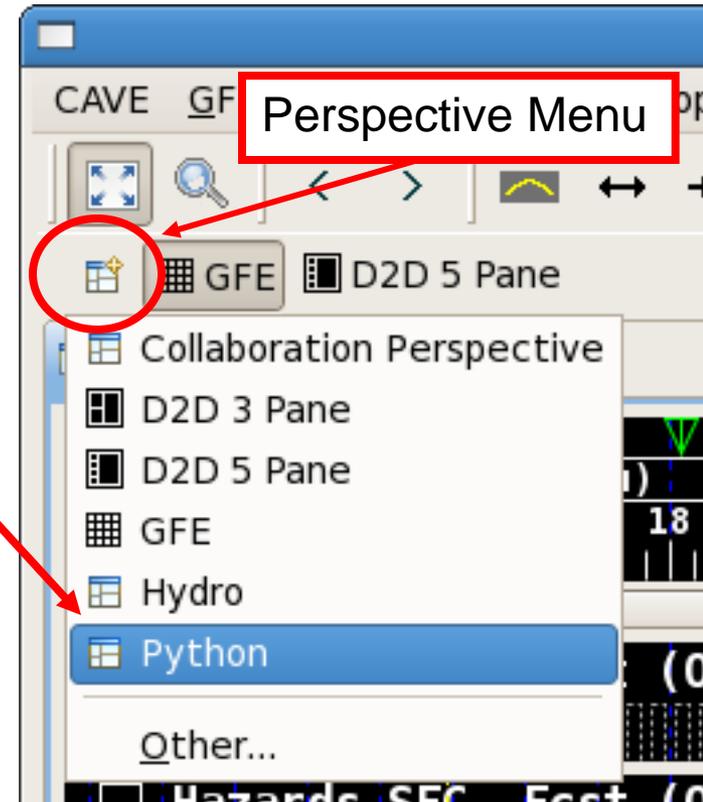
- TO 9 CAVE introduces the Python editing perspective for editing Smart Tools.
 - Smart tools are written in Python
- Not all functionality has been implemented
 - You are able to create a smart tool by editing an existing smart tool
 - Your smart tool is automatically saved



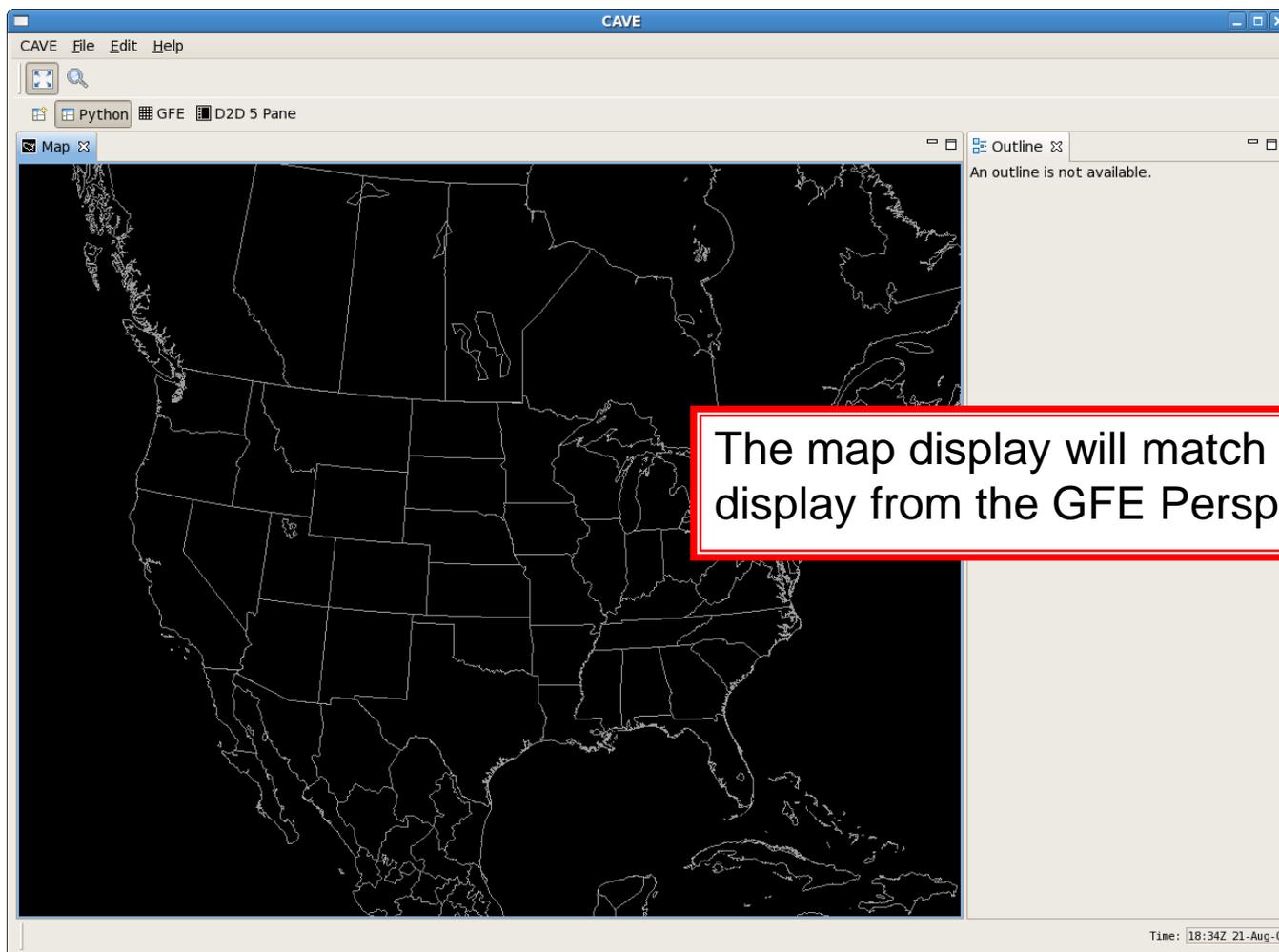
Python Editing Perspective

- The Python Editing Perspective is accessed via CAVE's perspective menu
 - click CAVE's perspective menu for a list of available perspectives
 - Select the Python perspective

Note: The Python perspective is usually used in conjunction with the GFE Perspective; although it can be loaded from other perspectives, not all functionality is available unless the GFE perspective is active.



Python Editing Perspective

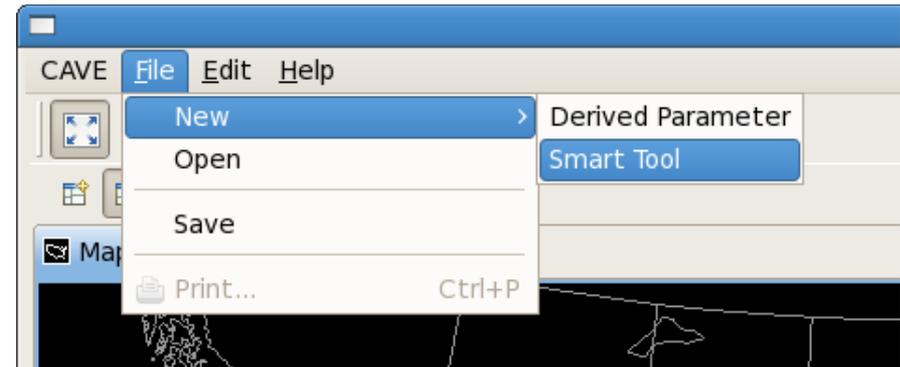


Python Editing Perspective

- The Python editor can be used to create a new Smart Tool from an existing Smart Tool

- To create a new Smart Tool
 - select *New->Smart Tool* from the File menu.

- This will open the *MyTool* dialog.

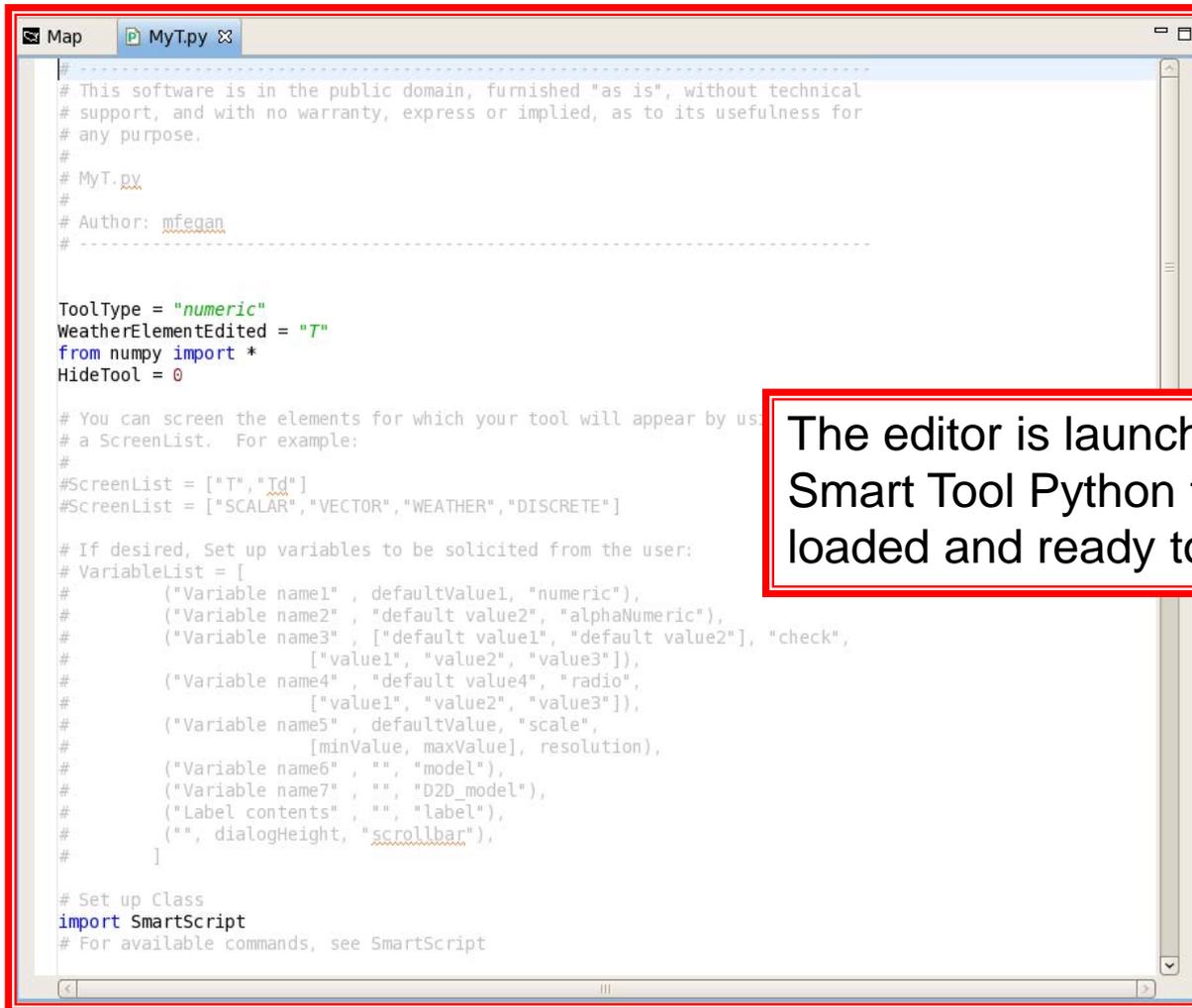


Python Editing Perspective

- In the *MyTool* dialog,
 - Edit the name for the new tool
 - Select the Weather Element for the new tool
 - Click *OK* to open the Python editor
 - clicking *Cancel* closes *MyTool* w/o opening the editor



Python Editing Perspective



The screenshot shows a Python editor window titled "MyT.py" with a red border. The code is a template for a Smart Tool, including a license notice, author information, and a list of variables to be solicited from the user. The code is as follows:

```
# This software is in the public domain, furnished "as is", without technical
# support, and with no warranty, express or implied, as to its usefulness for
# any purpose.
#
# MyT.py
#
# Author: mfeagan
# -----

ToolType = "numeric"
WeatherElementEdited = "T"
from numpy import *
HideTool = 0

# You can screen the elements for which your tool will appear by using
# a ScreenList. For example:
#
#ScreenList = ["T", "Id"]
#ScreenList = ["SCALAR", "VECTOR", "WEATHER", "DISCRETE"]

# If desired, Set up variables to be solicited from the user:
# VariableList = [
#     ("Variable name1" , defaultValue1, "numeric"),
#     ("Variable name2" , "default value2", "alphaNumeric"),
#     ("Variable name3" , ["default value1", "default value2"], "check",
#         ["value1", "value2", "value3"]),
#     ("Variable name4" , "default value4", "radio",
#         ["value1", "value2", "value3"]),
#     ("Variable name5" , defaultValue, "scale",
#         [minValue, maxValue], resolution),
#     ("Variable name6" , "", "model"),
#     ("Variable name7" , "", "D2D_model"),
#     ("Label contents" , "", "label"),
#     ("", dialogHeight, "scrollbar"),
# ]

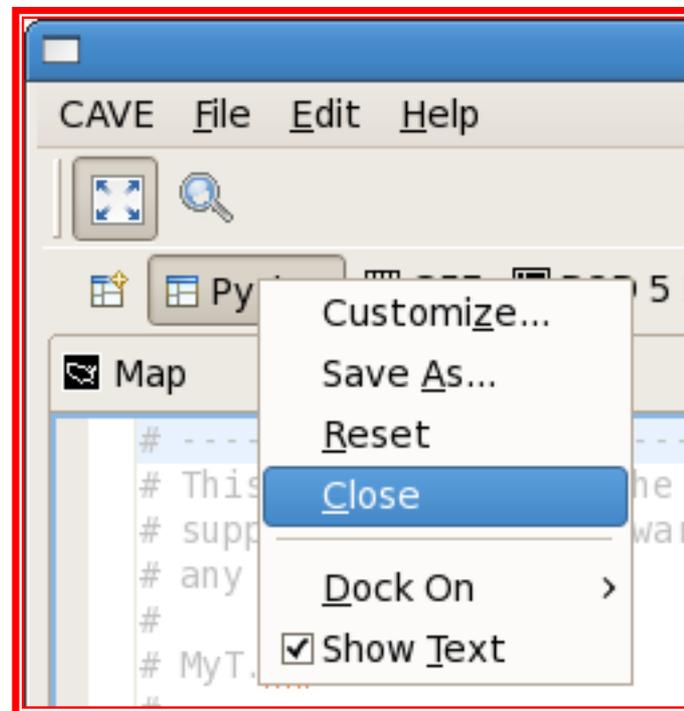
# Set up Class
import SmartScript
# For available commands, see SmartScript
```

The editor is launched with the Smart Tool Python template loaded and ready to customize.



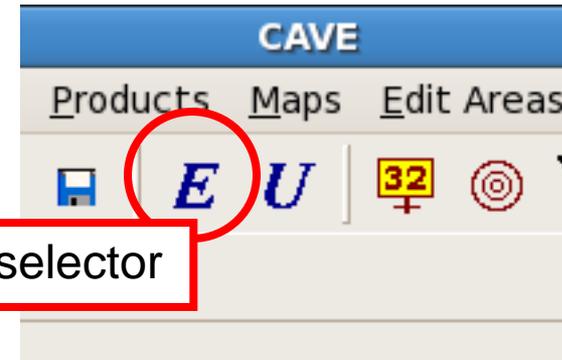
Python Editing Perspective

- To close the Python Editing Perspective,
 - first close the Python editor
 - Right click on the Python perspective tab and select close

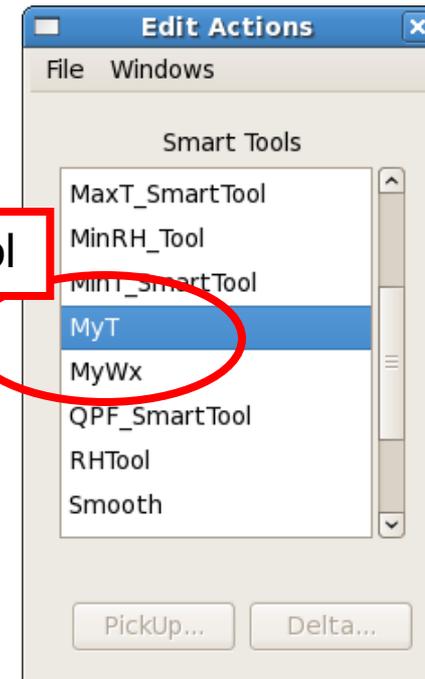


Python Editing Perspective

- Your new Smart Tool is now available via the Edit Actions Dialog in the GFE Perspective.



Newly Created Smart Tool



Questions?



Use of Apache Velocity



Apache Velocity

- From the Apache Velocity web site (velocity.apache.org):
The Apache Velocity Engine is a free open-source templating engine.

Velocity permits you to use a simple yet powerful template language to reference objects defined in Java code. It is written in 100% pure Java and can be easily embedded into your own applications.

- Velocity provides a standard paradigm for writing product templates
- AWIPS II currently uses Velocity in two areas:
 - generation of client level μ Engine scripts
 - generation of warnings from templates



Velocity Templates

- A velocity template is a boilerplate document written using the Velocity Template Language (VTL)
- A Velocity template includes
 - plain text which is not modified when the template is processed
 - comment: a statement for documenting the template
 - comments start with `##` and finish at the end of the line
 - directive: a statement providing direction as the template is processed
 - directives can provide specific (runtime) values to the template
 - directives can allow flow of control within the template
 - reference: a statement providing dynamic content for the template
 - references can link to variables defined in Java code
- From the Velocity Web Site



Example (from Velocity Web Site)

- This Velocity template generates a simple “Hello World” web page

```
<html>  
<body>  
#set( $name = "Velocity" )  
Hello $name World!  
</body>  
</html>
```



Example: μ Engine Script Generation

CAVE μ Engine Script Generation:

CAVE uses the following Velocity template to generate data retrieval scripts:

```
#macro(standardSelect $scriptLibrary $maxRecords $scriptMetadata $ignoreDateTime)
import ${scriptLibrary}
#set($pluginName = $scriptMetadata.get("pluginName").constraintValue)
dataRequest = ${scriptLibrary}.${scriptLibrary}("${pluginName}")
dataRequest.setCount(${maxRecords})
#foreach (${key} in ${scriptMetadata.keySet()})
#if(${key})
#if(${key} != "pluginName" && (${key} != "dateTime" || !${ignoreDateTime}))
#set($constraint = $scriptMetadata.get($key))
#if(${constraint.constraintType} == "IN")
dataRequest.addList("${key}", "${constraint.constraintValue}")
#elseif(${constraint.constraintType} == "LIKE")
dataRequest.addParameter("${key}", "${constraint.constraintValue}", "like")
#else
dataRequest.addParameter("${key}", "${constraint.constraintValue}")
#end
#end
#end
#end
return dataRequest.execute()
#end
```

Note: This actually defines a subroutine which is called standardSelect. It can be called from another template.



Example: μ Engine Script Generation

- Assume we have the following values passed in
 - scriptLibrary = "BaseRequest"
 - max records = 1
 - scriptMetadata = {"pluginName"=>"obs", "reportType"=>"METAR", "stationID"=>"AYMO", "dataTime"=>"2008-08-23 18:00:00.0"}
- Velocity will render this as

```
import BaseRequest
dataRequest = BaseRequest.BaseRequest("obs")
dataRequest.setCount(1)
dataRequest.addParameter("reportType", "METAR")
dataRequest.addParameter("stationID", "AYMO")
dataRequest.addParameter("dataTime", "2008-08-23 18:00:00.0")
return dataRequest.execute()
```



Questions?



WarnGen Templates



WarnGen Template Files

- A WarnGen template consists of two files:
 - the *.cfg* file and the *.vm* file
- The *.vm* file contains the warning template, which is written in Velocity's VTL language
- The *.cfg* file is an XML file containing values used by CAVE to populate the WarnGEN tool.
- The files are named *template.vm* and *template.cfg*
 - for a tornado warning, the files are *tornado.vmand tornado.cfg*
- More information on writing/modifying WarnGen Templates is included in *AWIPS II WarnGen Templates* (included in the ADE release materials as *AWIPS_II_WARNGEN_TEMPLATES.ppt*)



template.vm

- The `template.vm` files contain a mix of plain text and VTL statements. VTL has the capability of interacting with Java objects which are passed in.
- The plain text in these templates can be modified without error. Incorrectly modifying VTL statements may cause errors during warning generation, so exercise caution.
- The `.vm` files are in the AWIPS II ADE baseline in *build/static/common/cave/etc/warngen*



Exercise:

- We'll take a look at *tornado.vm* at this point.



template.cfg

- The template.cfg files contain user-configurable durations and bullets
 - Also contain other internal data which should not be modified.
- Example: To change the list of durations available in Warngen, add, delete, or modify a <duration/> item
 - <duration/> items are in the <durations> tag set.
- Example: To change the list of bullets, add, delete, or modify a <bullet/> item.
 - The <bullet/> items are in the <bullets> tag set
 - Additional changes must be made to template.vm as well.
- The current .cfg files are in the AWIPS II ADE baseline in *build/static/common/cave/etc/warngen*



Exercise:

- We'll take a look at *tornado.cfg* at this point.



Accessing Data in a Template

- Any Java element is accessed by VTL's variable syntax
 - simply enclose the element name in `${}`
- Any Java String or primitive type can be accessed directly within the Warngen template using:
 - *Syntax:* `${myJavaValue}`
- Any Java object's attributes can be accessed using a short cut notation
 - *Syntax:* `${myObject.attribute}`
 - *Example:* `${timeFormat.clock}`
- Any Method or attribute on a Java object can be accessed using standard Java-like syntax
 - *Syntax:* `${object.method(${arg1},${arg2}, ...)}`
 - *Example:* `${dateUtil.format(${pc.time}, ${timeFormat.clock}, ${localtimezone})}`
 - *note that the arguments are also specified using VTL syntax*



WarnGen Example: Variable Substitution

`${officeLong}` HAS ISSUED A

- This is an example of variable substitution.
 - the value of *officeLong* is set by the WarnGen code
 - It is obtained from the site localization
 - For KOAX (Omaha/Valley) it is *THE NATIONAL WEATHER SERVICE IN OMAHA*
- When the template is executed, this line will be expanded
 - *THE NATIONAL WEATHER SERVICE IN OMAHA HAS ISSUED A* is added to the output.



WarnGen Example: #if directive

```
#if( ${mode}=="test" || ${mode}=="practice" )  
TEST...SEVERE THUNDERSTORM WARNING...TEST  
#else  
SEVERE THUNDERSTORM WARNING  
#end
```

- This is an example of a an *#if* directive.
 - *\${mode}* is a value passed in from WarnGen
- If the mode is set to *test* or *practice* then *TEST...SEVERE THUNDERSTORM WARNING...TEST* is added to the output
 - otherwise “SEVERE THUNDERSTORM WARNING” is added to the output.



WarnGen Example: Nested Control

```
#if(${list.contains($bullets, "doppler")})  
#if(${stormType} == "line")  
#set ($report = "NATIONAL WEATHER SERVICE DOPPLER RADAR  
INDICATED A LINE OF SEVERE THUNDERSTORMS")  
#else  
#set ($report = "NATIONAL WEATHER SERVICE DOPPLER RADAR  
INDICATED A SEVERE THUNDERSTORM")  
#end  
#end
```

Note: `bullets` is a Java `String[]` provided by WarnGen

- `list.contains()` is a utility method for checking the contents of an array
- This example shows a compound control structure.
- The outer `#if` directive checks the `bullets` list; if it “doppler”, the inner if statement is Executed.
 - The inner `#if` directive checks the `stormType`.
 - The value of `report` is set depending on the value of `stormType`.
- Note that each `#if` requires a matching `#end`.



WarnGen Example: Looping

```
#foreach (${city} in ${pc.points})  
if(${city.roundedDistance} < 3)  
## close enough to not need azran, considered OVER the area  
${city.name}##  
else  
## needs azran information  
${city.roundedDistance} MILES #direction(${city.roundedAzimuth}) OF  
  ${city.name}##  
#end  
#end
```

Note: *pc* is an object provided by WarnGen; *pc.points* is an array of objects, each having a *roundedDistance* method.

- This example shows a *#foreach* loop.
 - Each iteration of the loop gets an object from the *pc.points* array.
 - Depending on the value returned by the object's *roundDistance* method, an appropriate message is added to the output.
 - The *#foreach* directive is matched by an *#end* directive.

Note: the lines beginning with two pound symbols (##) is are comments.

- Comments can be used anywhere within a template.
- Comments do not become part of the template output.



Questions?



Localization



CAVE Localization: File locations

- "Static" files (essentially the text or xml files in the 'etc' directory) are now loaded on-demand rather than as CAVE starts.
 - provides a quicker startup
 - better supports the GFE way of doing business
 - keeps things more in sync.



CAVE Localization: Coding Changes

- VizApp.getBaseDir() use is now strongly discouraged. The use of PathManager/LocalizationFile instead is recommended.
 - This interface has far more capability including listing files available at multiple tiers of localization on the server.
 - IPathManager is currently only implemented on the CAVE side, but the EDEX interface will eventually be developed.
- An InputStream/OutputStream interface for LocalizationFile is planned in the future to address cache-coherency issues with storing the file temporarily on the filesystem, for frequently changed files. This will allow reading files directly being streamed from the server.



Questions?



Scripting engine updated



CAVE μ Engine Script Generator

- CAVE generates *client side* μ Engine scripts to perform database queries and product retrievals
 - The CAVE script generator uses Velocity's VTL language to specify script templates
 - CAVE uses Python of all generated μ Engine scripts
- CAVE uses two VLT template files:
 - Both files are located in `com.raytheon.viz.core/scriptTemplates`
 - `standardTemplate.vm` contains single top level template
 - `VM_global_library.vm` contains macro definitions used by the top level template



Template File: StandardTemplate.vm

```
## The standard template gets all of its behavior from
   VM_global_library macros
#if($mode == "select")
#standardSelect($scriptLibrary $maxRecords $scriptMetadata false)
#elseif($mode == "catalog")
#standardCatalog($scriptLibrary $scriptMetadata)
#elseif($mode == "latestTime")
#standardLatestTime($scriptMetadata)
#elseif($mode == "plot")
#standardSelect($scriptLibrary $maxRecords $scriptMetadata true)
#elseif($mode == "dbquery")
#standardDbQuery($scriptMetadata)
#end
```



Calling Velocity from CAVE

- The client code calls *ScriptCreator.createScript()*, then passes the script to the μ Engine
- When Velocity is called, the following values are available to the template
 - *metaData*: Java Map containing the request constraints
 - *maxRecords*: the requested number of records
 - *scriptLibrary*: normally contains *VM_global_library.vm*
 - *mode*: type of query to perform; determines which macro to use to create the script
 - must match one of the conditions in *standardTemplate.vm*
 - ▶ current values are *select*, *catalog*, *latestTime*, *plot* and *dbquery*
 - this maps to the name of the macro from *VM_global_library.vm* that creates the script



Example: “select” macro

```
from VM_global_library.vm:
#macro(standardSelect $scriptLibrary $maxRecords $scriptMetadata
  $ignoreDateTime)
import ${scriptLibrary}
#set($pluginName = $scriptMetadata.get("pluginName").constraintValue)
dataRequest = ${scriptLibrary}.${scriptLibrary}("${pluginName}")
dataRequest.setCount(${maxRecords})
#foreach ($key in ${scriptMetadata.keySet()})
#if(${key})
#if(${key} != "pluginName" && (${key} != "dateTime" || !${ignoreDateTime}))
#set($constraint = $scriptMetadata.get($key))
#if(${constraint.constraintType} == "IN")
dataRequest.addList("${key}", "${constraint.constraintValue}")
#elseif(${constraint.constraintType} == "LIKE")
dataRequest.addParameter("${key}", "${constraint.constraintValue}", "like")
#else
dataRequest.addParameter("${key}", "${constraint.constraintValue}")
#end
#end
#end
#end
return dataRequest.execute()
#end
```



Questions?



Derived parameters



Derived Parameters

- TO 9 CAVE includes a partial implementation of derived parameters for the Volume Browser
- Derived Parameters are implemented as Python scripts
 - Mapping of Volume Browser text to Python script for derived parameters is defined in BrowserXXXFields.xml, where XXX represents the desired Volume Browser mode
 - Example: BrowserTimeFields.xml
 - Each mapping is represented by a <menuButton/> tag set
 - Example: the Dewpoint parameter is defined by

```
<menuButton>  
    <buttonText> Dewpoint</buttonText>  
    <buttonName> DpT </buttonName>  
</menuButton>
```



Derived Parameters

- Each derived parameter is defined in a Python script
 - the Python scripts are in `build/static/cave/etc/derivParamScripts` in the AWIPS II ADE baseline
- The derived parameter script is named *parameter.py*
 - Example: for the *Dewpoint* (DpT) derived parameter, the scrip is contained in *DpT.py*
- Detailed information on creating derived parameter scripts is documented in the AWIPS II ADE. The document to read is *DerivedParmScripting.pdf*



Derived Parameter Scripts

- Derived Parameter Scripts are written in Python
 - Scripts have full access to Python
 - CAVE requires code conventions to treat as derived parameter
 - Interface to/from Java limits types of objects received/returned

- Quick example:

```
variableId = "Qual"  
variableName = 'Air quality'  
variableUnit = "ppm"  
parameters1 = "|T|RH"  
class DerivedParameter():  
    def execute1(self,T,RH):  
        return T + RH
```



Interface Requirements

- Derived parameter metadata - all required
 - variableId - unique identifier for derived parameter
 - variableName - short description
 - variableUnit - units of measure

- Example:

```
variableId = "Qual"  
variableName = "Air quality"  
variableUnit = "ppm"
```



Parameter Specification

- Parameter string(s) - at least one (and currently limited to one)
 - Describes parameters and level restrictions
 - variable name is "parameters<n>"
 - separators are commas and pipes
 - commas separate level restrictions
 - pipes separate parameters
 - level restrictions on derived parameter appear before first pipe
 - currently only support composite levels in level restrictions
 - currently doesn't support 3D parameters

- Examples:

```
parameters1=' |T'
```

```
parameters2="Surface |T, 3D |RH"
```



Class Definition

- DerivedParameter class declaration
 - Must be named DerivedParameter
 - constructor can't take parameters
- Execute<n> method(s)
 - named execute<n>
 - Instance method of class
 - One for every parameters<n> above (currently limited to one)
 - Takes "self" + parms from matching parameters<n>
 - Can't use *args or **kwargs
 - Return value is passed to Java - RESTRICTIONS!

- Example:

```
def execute1(self,T,RH):  
    return T/273.15 + RH
```



Java/Python Interface

- Parameter types from Java:
 - scalars
 - 1D arrays of numpy float32s
 - 2D arrays of numpy float32s
 - tuples of 2D arrays of numpy float32s
 - 3D arrays will be passed as tuples of 2D arrays (not implemented)

- Things it's OK to return to Java:
 - scalars
 - arrays of int8s
 - arrays of float32s
 - simple tuples or lists of any of the above



Java/Python Interface

- Things it's definitely NOT OK to return to Java:
 - arrays of float64s - numpy default, beware of conversions
 - masked arrays
 - NaN (not a number) - Java is OK with it, database may not be
 - Arrays of complex, not tried but very unlikely to work
 - tuples or lists of the above
- Things you might not want to return to Java:
 - Arrays of other data sizes, might be converted
 - NaN (not a number) - Java is OK with it, database may not be
 - strings - works, but how would CAVE display it on a map?
 - dictionaries - works, but how would CAVE...



Complete Example

```
## @file AV.py

import functions.Vorticity as Vorticity

variableId = "AV"
variableName = "Absolute Vorticity"
variableUnit = "s"

parameters1 = "|uW|vW|coriolis|dx|dy"

##
# AV derived parameter class.
class DerivedParameter():
    "AV derived parameter class."

    ##
    # Calculate wind vorticity.
    #
    # @param Wind: tuple(U,V) of wind velocity.
    # @param coriolis: coriolis value to add to relative vorticity
    # @param dx: spacing between X data points (scalar or array)
    # @param dy: spacing between Y data points (scalar or array)
    # @return: wind vorticity array
    def execute1(self, uW, vW, coriolis, dx, dy):
        "Calculate wind vorticity."
        result = Vorticity.execute(uW, vW, coriolis, dx, dy)
        return result
```



Saving the Script

- Saving the script
 - Script name must match variableId, with ".py" extension
 - All to same directory (build/eclipse/etc/derivParamScripts)
 - Available to CAVE immediately
- Path to derived parameter script directory is `${HOME}/workspace/build/eclipse/etc/derivParamScripts` on my machine; not sure where they'll find it, but `find . -maxdepth 10 -name derivParamScripts` should locate it.



Questions?

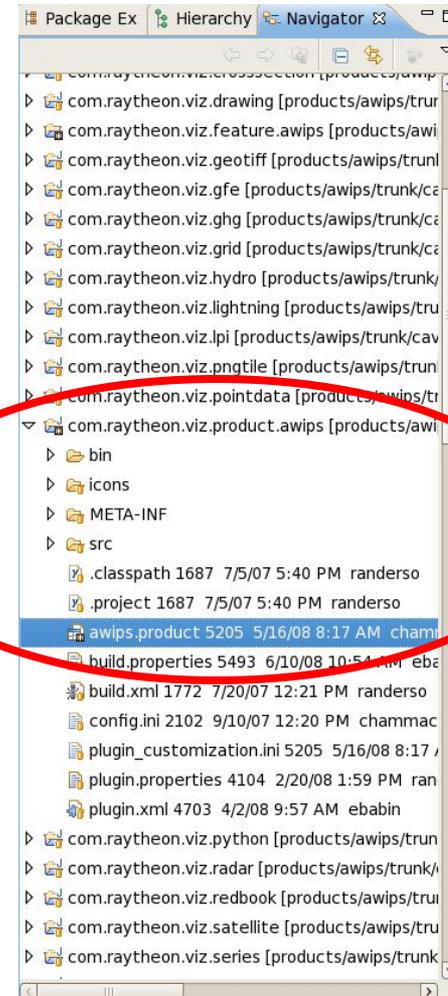


Modified CAVE Startup



Modified CAVE Startup - Developer

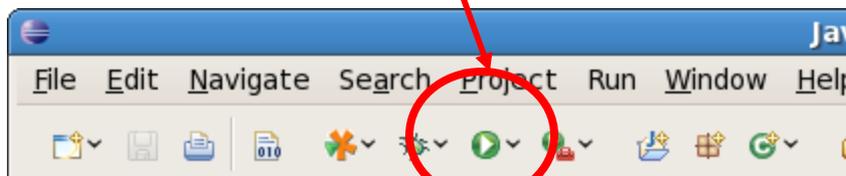
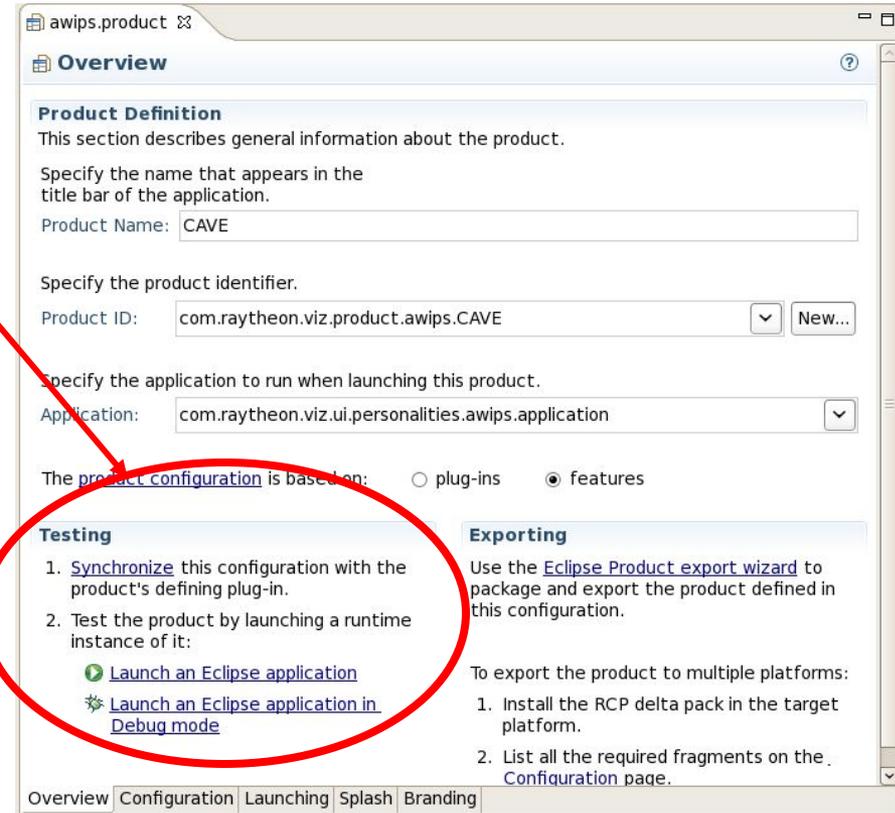
1. From inside Eclipse, locate the **com.raytheon.viz.product.awips** project
2. Expand the project (click the triangle)
3. Double click on **awips.product**



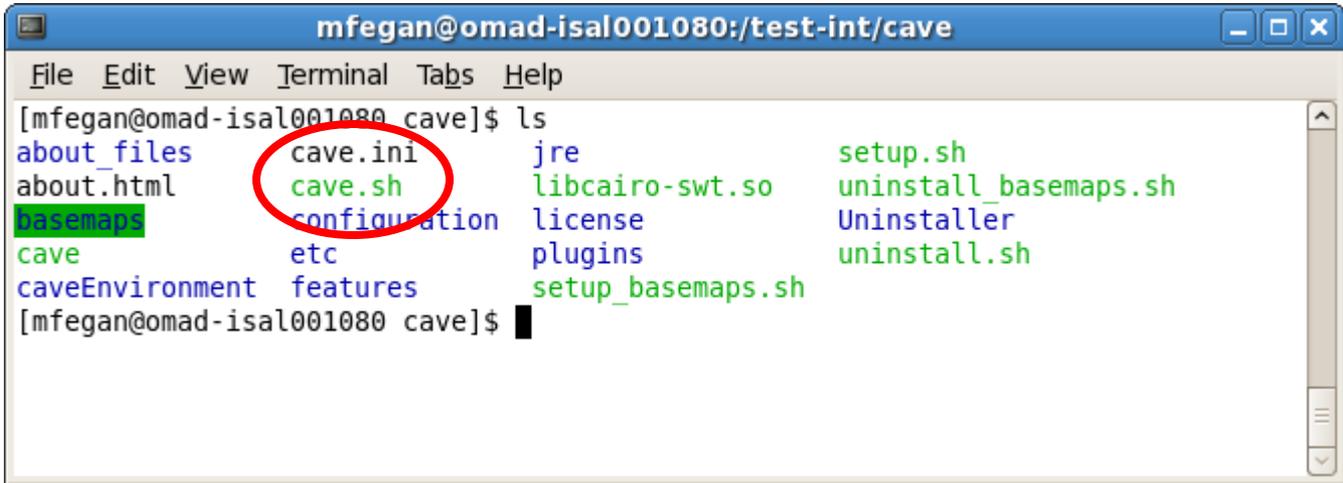
Modified CAVE Startup - Developer

After the description page loads:

4. click the *features* radio button,
5. click on the [blue Synchronize](#) hyperlink, then
6. click on the [blue Launch the Product](#) hyperlink
7. Once this has been done, you can usually launch CAVE by clicking the green *Run As* button on the Eclipse toolbar.



Modified CAVE Startup – Installed User



```
mfegan@omad-isal001080:/test-int/cave
File Edit View Terminal Tabs Help
[mfegan@omad-isal001080 cave]$ ls
about_files  cave.ini      jre          setup.sh
about.html   cave.sh      libcairo-swt.so  uninstall_basemaps.sh
basemaps    configuration license      Uninstaller
cave        etc          plugins      uninstall.sh
caveEnvironment  features    setup_basemaps.sh
[mfegan@omad-isal001080 cave]$
```

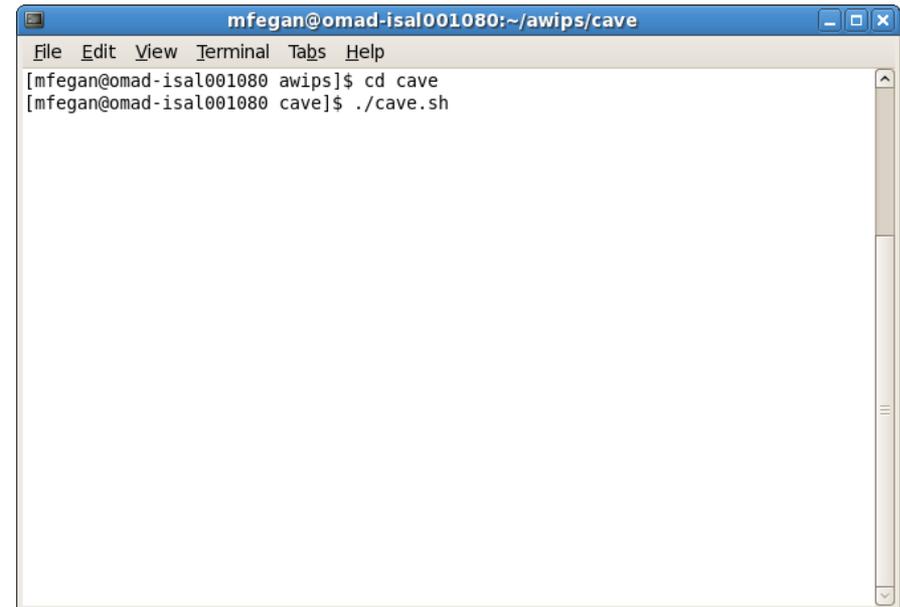
- Starting in TO 9, CAVE requires specific environmental settings to run successfully
 - The values are provided when CAVE (or the ADE) is installed
- For an installed version of CAVE, these values are provided by the CAVE startup script, cave.sh
 - cave.sh is in the CAVE install directory
- CAVE should be started using this script



Modified CAVE Startup – Installed User

To run CAVE following an IzPack installation:

- open a terminal window
- change directory to the CAVE install directory
- execute cave.sh

A terminal window titled "mfegan@omad-isal001080:~/awips/cave" with a menu bar containing "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal shows the following commands and output:

```
[mfegan@omad-isal001080 awips]$ cd cave  
[mfegan@omad-isal001080 cave]$ ./cave.sh
```



Questions?



Wrap-Up



Summary

- Covered Platform updates
- Covered Addition of Python interface
- Covered Use of Apache Velocity
- Covered WarnGen Templates
- Covered CAVE Localization
- Covered Scripting engine updated
- Covered Derived parameters
- Covered Modified CAVE Startup



Resources

- On the ADE TO9 DVD
 - Current code available for examination in the ADE baseline
 - JavaDoc documentation available
- Also available
 - TO 9 Training Updates
 - TO T1 Training Materials
- Velocity Template Language (VTL).
 - Velocity and the VTL is released under the Apache Jakarta Project.
 - <http://click.sourceforge.net/docs/velocity/vtl-reference-guide.html>



Module 15 – TO 9 ADE Updates



Advanced Weather Interactive Processing System II (AWIPS II)

AWIPS Development Environment (ADE) and the Common AWIPS Visualization Environment (CAVE)

Module 15: AWIPS II ADE Updates for TO9

September 9, 2008



Objectives

- Upon completion of this module, the student will understand modification made to the AWIPS II ADE for TO 9



Topics

- **ADE platform updates**
- **AWIPS II installer updates**
- **System Installation now supported by Flow Tags**



Platform Updates



Platform Updates - Java

- Update:
 - Java has been updated to Java 1.6.0_05 (from 1.6.0_01)
- Rational:
 - Latest Java Version available at the appropriate time in the TO, contains latest Java bug fixes and enhancements
- Impacts:
 - Required update of several support packages
- Install:
 - packaged with AWIPS II Installers



Platform Updates - PostgreSQL

- Update:
 - Postgres has been updated to PostgreSQL 8.3.0
- Rational:
 - Latest Version available at the appropriate time in the TO, contains latest bug fixes and enhancements
- Impacts:
 - Minimal changes required – will need a need version of pgAdmin III
- Install:
 - packaged with AWIPS II Installers



Platform Updates - Eclipse

- Update:
 - Eclipse has been updated to Version 3.3.2
- Rational:
 - Latest Version available at the appropriate time in the TO, contains latest bug fixes and enhancements
- Impacts:
 - Minimal changes required – will be installed by the ADE installer
- Install:
 - packaged with AWIPS II ADE Installer



Platform Updates – Other Packages

- Other software has been updated as needed to be compatible with the platform
 - Specific version information is available in the AWIPS II SVD document (included on the install media)



Questions?



AWIPS II Installer Updates



Installer Updates – ADE Installer

- AWIPS II ADE installer now includes
 - ADE Baseline (EDEX/CAVE source code and JavaDoc)
 - ANT 1.70
 - Eclipse 3.3.0
 - Java JDK 1.6.0u5
- All components except the ADE Baseline are optional
- ADE Installer is designed to coordinate with an existing EDEX runtime installation
 - EDEX development requires an EDEX runtime installation
 - ADE installer defaults to standard EDEX locations – may be changed by the user



Installer Updates – EDEX Installer

- EDEX Installer includes updated versions of platform software
- EDEX installer adds required environment settings to startup scripts used to run EDEX as a service on Linux servers



Installer Updates – CAVE Installer

- CAVE Installer includes updated versions of platform software
- CAVE is now designed to run using a startup script, `cave.sh`
 - `cave.sh` sets environment appropriately for running CAVE
 - starting CAVE directly will result in runtime errors



Questions?



AWIPS II Installation Flow Tags



AWIPS II Installation Flow Tags

- Starting with TO 10, AWIPS II Installers are supported with Installation Flow Tags
- Flow tags consist of detailed instructions for performing a specific installation
- Flow Tags are available for
 - Cluster based EDEX/CAVE install
 - Standalone EDEX/CAVE install
 - Developer (ADE) install
- Flow Tags have been tested/refined in the Omaha lab
 - feed back will be appreciated

AWIPS

Flow Tag Record
Cluster Deployment

Integration Server Information

| | |
|--------------------------------|--|
| NAS Server | |
| EDEX DB Server | |
| EDEX Server | |
| EDEX Server IP Address | |
| EDEX Client(s) | |
| EDEX Client(s) IP Address | |
| CAVE Machine(s)/user(s) | |

| | |
|---------------------|--|
| Task Order | |
| Build Number | |

Final Readiness Approval Record

| Department | Assignee | Signature | Date |
|----------------------|----------|-----------|------|
| Systems Engineering: | | | |
| Quality Assurance: | | | |

Hand-copies are uncontrolled. Verify version prior to use.
UNLIMITED RIGHTS NOTICE
This documentation/technical data was developed pursuant to Contract Number D0133R-05-C-01057 with the US Government. The US Government's rights in and to this copyrighted data are as specified in DPA RO 252.227-7013, which was made part of the above contract.



Questions?



Wrap-Up



Summary

- Discussed ADE platform updates
- Discussed AWIPS II installer updates
- Discussed Flow Tags for system installation



Resources

- On the ADE TO9 DVD
 - Current code available for examination in the ADE baseline
 - JavaDoc documentation available
- Also available
 - TO 9 Training Updates
 - TO T1 Training Materials



TO 10 Look Ahead



TO 10 Platform Updates

- Move to Eclipse 3.4 in ADE
- Move to either Mule 1.4.4 or 2.0 for EDEX
- Move to ActiveMQ 5.0
- Update other FOSS packages as needed



AWIPS II Baseline Reorganization

- The plan is to reorganize both CAVE and EDEX into a “core” vs “non-core” as opposed to CAVE vs EDEX organization.
 - the intent is to support distributed development
 - EDEX may be organized more like CAVE
- EDEX build will include more packaging options



More use of Proxy Pattern for Services

- Currently, IndexSrv is configurable as to the data decoder each service endpoint uses
- The plan is to extend this to other service endpoints
- Potential candidates for this treatment
 - ProductSvr
 - AutoBldSrv
 - PurgeSrv



Enhanced μ Engine Scripting

- μ Engine will become a script runner factory
 - Script runners (currently JScript and Python) will conform to a well defined interface
- Endpoints will use the μ Engine factory to obtain an appropriate script runner
 - May be configured at the endpoint or based on data
- Additional script runners can be created



Text Database Implementation

- Enhanced implementation of the Text Product decoder
 - decoded products are stored
 - product notifications are sent to CAVE
- Text Workstation component of CAVE will have more features added
- Command line interfaces will be provided
 - textDB – provides a command line interface to the text database
 - existing command line interface will be preserved on the client
 - local apps using the existing command line interface will work w/o modification
 - uEngine – provides a command line interface for executing μ Engine scripts
 - (new) provides access to uEngine script running



CAVE Enhancements

- Implementation of additional legacy features
- Will be looking at improved serialization mechanism (also for EDEX)
- Will be streamlining the IVisResource interface
 - will simplify visualization implementations



Database Enhancements

- General reorganization to facilitate operation
 - Structure of some tables will change
 - more use of primary keys and indices
- Improved Purge
 - currently time based, will provide other strategies
 - extensible using Proxy Pattern
- DAO Pool will be removed
- Possibly update Hybernate
 - leverage Java SE6 style annotations



Questions?



Wrap-Up



Resources

- On the ADE TO9 DVD
 - Current code available for examination in the EDEX baseline
 - JavaDoc documentation available
- Also available
 - TO 9 Training Updates
 - TO T1 Training Materials

