

# Advanced Weather Interactive Processing System II (AWIPS II)

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

*Module 16: EDEX Updates for TO10*

February 18, 2009

AWP.TRG.SWCTR/TO10.ADE/CAVE-16.00

*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 the restriction are contained in all sheets.*



# Objective

---

- Understand the modifications to the EDEX architecture that were implemented in AWIPS II TO10



# Topics

- Describe EDEX platform updates
- Describe EDEX Code Reorganization
- Describe move to Camel as the Integration Framework
- Describe database improvements
- Describe data serialization improvements
- Describe new Command Line Interface tools package
- Describe improvements to data decoder plug-ins
- Describe improvements to EDEX service endpoints



# Platform Updates



# Platform Updates – Eclipse

## ■ Update:

- Eclipse has been updated to Version 3.4.1, built 9/11/2008

## ■ Rationale:

- Latest Version available at the appropriate time in the TO, contains latest bug fixes and enhancements
- Used for CAVE and EDEX development and builds

## ■ Impacts:

- Minimal changes required

## ■ Install:

- Packaged with AWIPS II Installers



# Platform Updates: PostgreSQL

- Update:
  - Postgres updated to PostgreSQL 8.3.4
- Rationale:
  - Latest Version available at the appropriate time in the TO, contains latest bug fixes and enhancements
- Impacts:
  - Minimal changes required
- Install:
  - Packaged with AWIPS II Installers



# Platform Updates: ActiveMQ

- Update:
  - ActiveMQ updated to version 5.2.0 (from 4.1.1)
- Rationale:
  - Latest ActiveMQ Version available at the appropriate time in the TO, contains latest bug fixes and enhancements
  - Supports embedding of other Apache products into an integrated running environment
- Impacts:
  - Minimal changes, updated configuration required
- Install:
  - Packaged with AWIPS II Installers



# Platform Updates: Mule

- Update:
  - Mule has been replaced by Apache Camel
- Rationale:
  - Changes in the MuleSource business model and licensing coupled with an uncertain release schedule made continuing with Mule problematical
- Impacts:
  - Some code rewrite required; much simplified deployment
- Install:
  - Packaged with AWIPS II Installers

Note: This is covered in more detail later.



# Platform Updates: Java Serialization

- Update:
  - JiBX has been replaced with Java Architecture for XML Binding (JAXB)
- Rationale:
  - JiBX doesn't work particularly well in our plug-in architecture.
  - Performance: Possible to do even better than JiBX
- Impacts:
  - Some code rewrite required; much builds
- Install:
  - Packaged with AWIPS II Installers

Note: This is covered in more detail later.



# Questions?



# EDEX Code Reorganization



# EDEX Code Reorganization

- With TO10, the EDEX code base in the ADE has been reorganized into Open Services Gateway Initiative (OSGi) compliant plug-ins
  - The EDEX runtime (Camel) does not leverage OSGi compliance
  - The ADE IDE (Eclipse) does leverage OSGi compliance
- Using OSGi provides a standard minimal framework for creating EDEX projects



# EDEX Code Reorganization (cont'd)

- EDEX components are Eclipse plug-ins
  - This leverages the use of Eclipse IDE to create the basic component structure
- Two components are used to enable this:
  - `com.raytheon.edex.feature.uframe`
    - Components that are part of EDEX must be registered in this plug-in
  - `com.raytheon.edex.ui.personalities.uframe`
    - Required by Eclipse, not used by EDEX



# EDEX Code Reorganization (cont'd)

- EDEX components extend the Eclipse plug-in model, adding additional directories and files that are used during deployment and execution of EDEX
  - Resources used by the component are included in a “res” directory
    - Camel-enabled components require descriptors under “res/spring”
    - Additional configuration may be provided in “res/conf”
  - Components contributing objects using JavaArchitecture for XML Binding (JAXB) serialization require an additional file under META-INF
  - All components require a deployment definition file, component-deploy.xml
- Other changes to the code base have eliminated
  - Most Hibernate definition files have been eliminated
  - Most JiBX definition files have been eliminated



# EDEX Code Reorganization (cont'd)

- EDEX Build has been simplified – integrated with Eclipse
  - Eclipse automatically builds, but does not deploy, EDEX as code is modified
  - Build is determined by linkages between the EDEX feature and each component.
- EDEX can be built from the command line
  - More on this later
- EDEX deploy is feature based – not directly supported by Eclipse
  - Details on deploying via Eclipse are in the examples
- EDEX can be deployed from the command line
  - More on this later



# EDEX Code Reorganization – Tools

- For TO9, the ADE shipped with an Eclipse plug-in – Plugin Creator – to assist in making a new data decoder plug-in for EDEX
- With the change to an OSGi/Eclipse plug-in structure for EDEX components, this tool no longer generates valid plug-in skeletons and should no longer be used
- All EDEX components can be created using the Eclipse Plug-in Development tools
  - These tools are shipped with Eclipse in the ADE
- The plan is to discontinue support for the EDEX-specific Plugin Creator



# EDEX Code Reorganization – Example

## Problem:

Create an EDEX component, YAHW, that logs messages to the EDEX system log.

## Solution:

Create a new OSGi plug-in to host the component. Provide a main class for the component that utilizes existing EDEX code to perform logging.

## To Do:

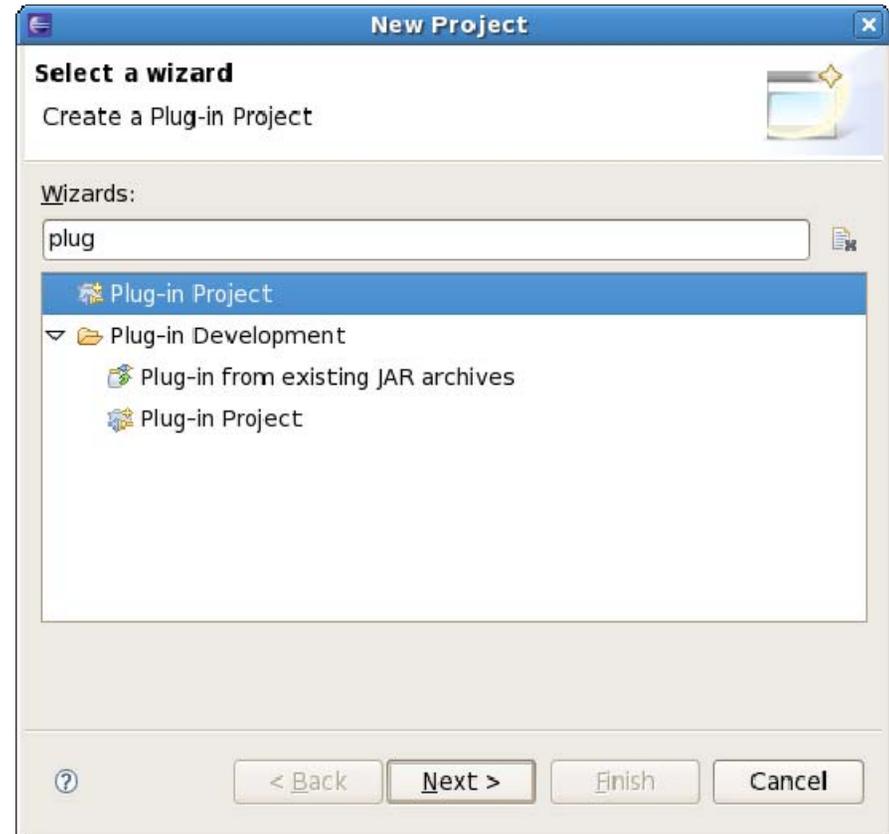
Camel wiring will be provided later.



# EDEX Code Reorganization – Example (cont'd)

First, create a new plug-in project:

- Open the *New Project* dialog.
  - Click the *New* icon in the Eclipse toolbar.
  - Select *Project...*
- On the *New Project* dialog, find *Plug-in Project* in the selection list.
  - Hint: The text box limits the search; type *plug* to quickly find *Plug-in Project*.
- Select *Plug-in Project* and click Next >.



# EDEX Code Reorganization – Example (cont'd)

Enter basic Plug-in details:

- Enter the Project Name.
  - Hint: This is the name Eclipse uses to create a project directory.
  - In this example, use *org.noaa.gov.to10.example*.
- Enter the project Location.
  - Hint: You can use the default location or browse to a different location (be sure to create the directory).
- Set the Eclipse version.
  - EDEX uses Eclipse version 3.4.
- Continue to the next page.
  - Click Next >.

**New Plug-in Project**

Create a new plug-in project

Project name:

Use default location

Location:

Project Settings

Create a Java project

Source folder:

Output folder:

Target Platform

This plug-in is targeted to run with:

Eclipse version:

an OSGi framework:

Working sets

Add project to working sets

Working sets:



# EDEX Code Reorganization – Example (cont'd)

Provide/verify Plug-in details:

- Verify that the *Plug-in Properties* are correct.
  - Enter a *Plug-in Provider*.
    - In this example, use *AWIPS II Training*.
  - Change *Execution Environment* to *<No Execution Environment>*
- Set the *Plug-in Options*.
  - De-select all entries under *Plug-in Options*.
- Make sure the *Rich Client Application* option has *No* selected.
- Create the Plug-in.
  - Click *Finish*.

**New Plug-in Project**

**Plug-in Content**  
Enter the data required to generate the plug-in.

**Plug-in Properties**

Plug-in ID:

Plug-in Version:

Plug-in Name:

Plug-in Provider:

Execution Environment:

**Plug-in Options**

Generate an activator, a Java class that controls the plug-in's life cycle  
Activator:

This plug-in will make contributions to the UI

Enable API Analysis

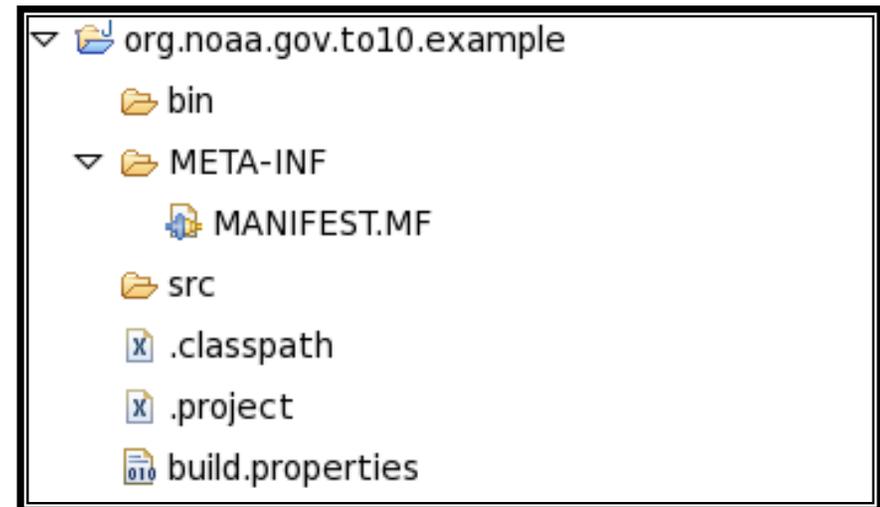
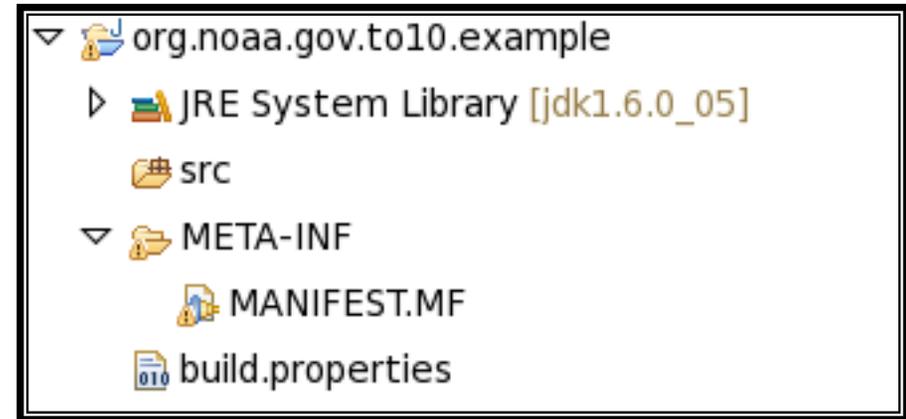
**Rich Client Application**

Would you like to create a rich client application?  Yes  No



# EDEX Code Reorganization – Example (cont'd)

- Eclipse will open the newly created plug-in in the *Plug-in Development perspective*.
  - Hint: You can change to the Java Perspective
- To see what Eclipse has generated for you.
  - Locate and expand the org.noaa.gov.to10.example node in the Package explorer.
  - Expand the META-INF node.
- You should see a directory structure similar to that displayed above right.
- The display lower right is in the Navigator view of the Java perspective.



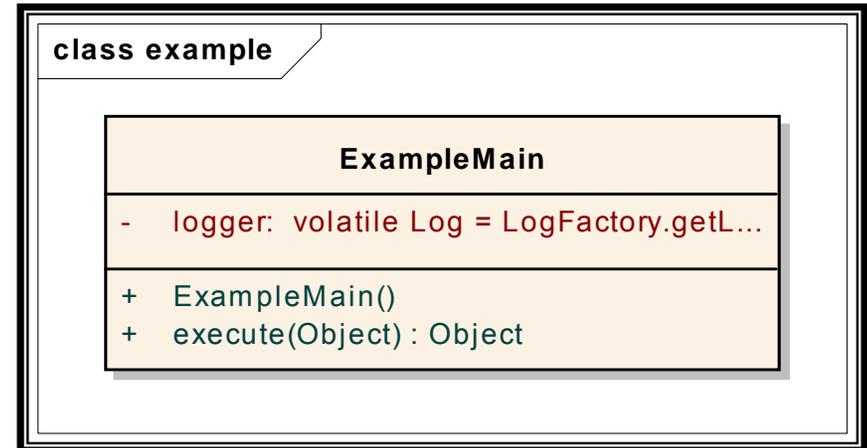
Note: The next few slides assume a switch to the Java Navigator.



# EDEX Code Reorganization – Example (cont'd)

Adding the Code:

- The design of this example is shown at right; the code is shown on a later slide.
- The code for this example is fairly simple.
- Note that this is a POJO.
  - There is no dependency in this design to any special classes.



# EDEX Code Reorganization – Example (cont'd)

Create the package structure:

- Open the *New Java Package* dialog.
  - In the Navigator, right click on the *src* directory.
  - *select New*→*Package*.
- Verify that the *Source folder* is correct.
  - In this example, the source folder is *org.noaa.gov.to10.example/src*.
- Enter the package name.
  - In this example, enter *org.noaa.gov.to10.example*.
- Create the package.
  - Click *Finish*.



Note: In Eclipse, the view updates to show the new package.



# EDEX Code Reorganization – Example (cont'd)

Add the main class for the plug-in:

- Open the *New Java Class* dialog.
  - Locate the newly created package in the *src* directory of *org.noaa.gov.to10.example*.
  - Right click on the package and select *New*→*Class*.
- Select the appropriate class options.
  - Enter the name of the new class.
    - In this example, use *ExampleMain*.
  - Optional: Select to generate method stubs and/or comments.
  - Click *Finish*.

A skeleton of the class is added to the project.



Note: In Eclipse, newly created class is opened in the Java editor.



# EDEX Code Reorganization – Example (cont'd)

Code the class:

- Add the constructor and a execute method as shown at right.
  - Hint: The code to add is in *italic* font.
- As you enter the code, Eclipse will flag errors in several statements.
  - Both imports will list errors.
  - The lines defining and using **logger** will show errors.
- The errors exist because each Eclipse plug-in has its own dependencies
  - At this point, this new plug-in doesn't know about the Apache classes in the import statements.

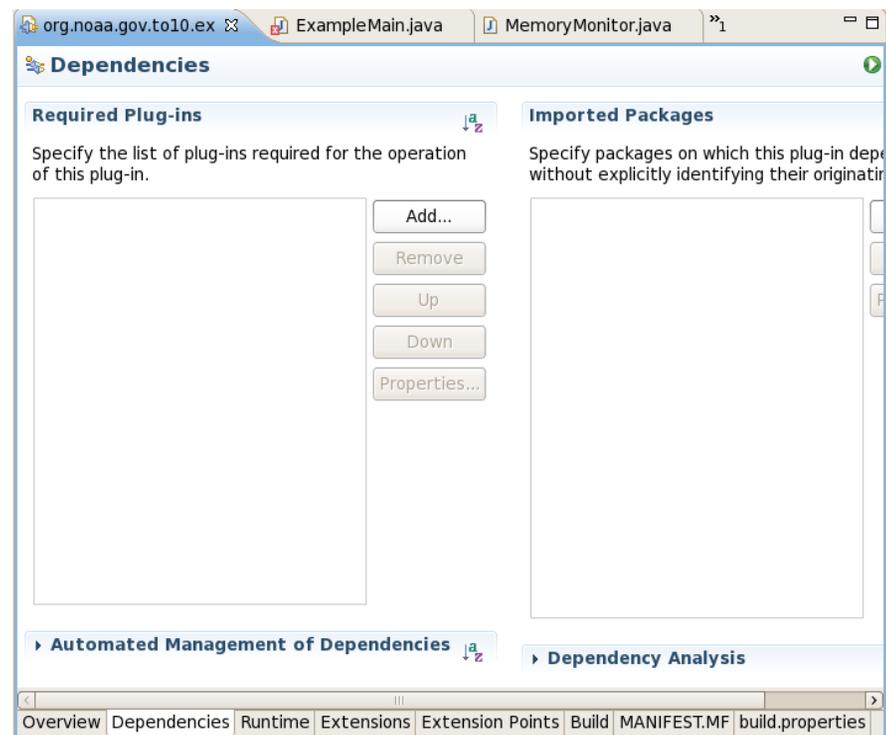
```
package org.noaa.gov.to10.example;  
  
import org.apache.commons.logging.Log;  
import org.apache.commons.logging.LogFactory;  
  
public class ExampleMain {  
    private transient Log logger  
        = LogFactory.getLog(getClass());  
  
    public ExampleMain() {  
        super();  
    }  
  
    public String execute(String msg) {  
        logger.info(msg);  
        return msg;  
    }  
  
}
```



# EDEX Code Reorganization – Example (cont'd)

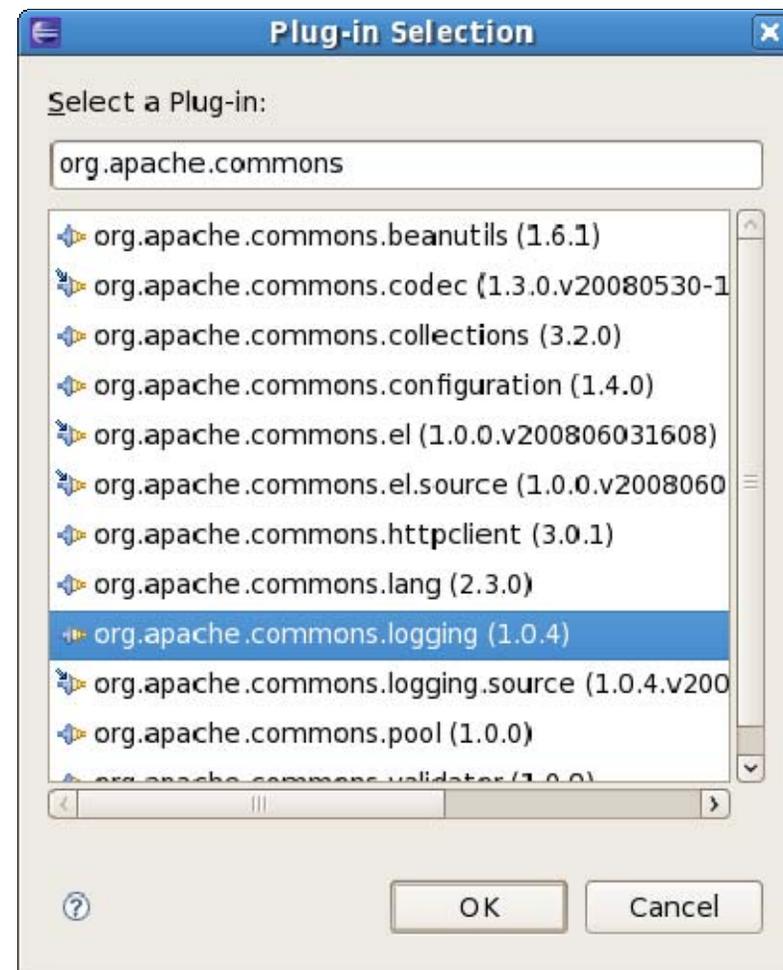
Add Apache Commons Logging:

- Open the plug-in Editor.
  - In the Navigator view, find and double click on *MANIFEST.MF*.
- Add the dependency.
  - Click on *Add...*



# EDEX Code Reorganization – Example (cont'd)

- Add Apache Commons Logging.
  - Locate and select *org.apache.commons.logging*.
  - Hint: Use the text box to limit the selections.
- Click OK.

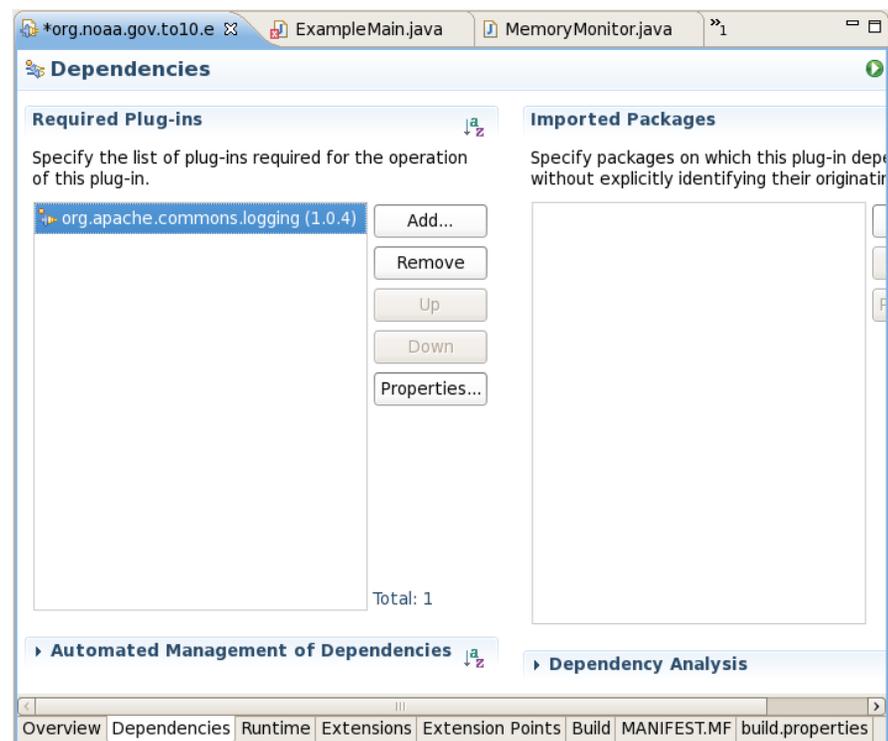


# EDEX Code Reorganization – Example (cont'd)

The editor will update to display the new dependency.

- Click the Save icon on the toolbar to save the modified project files.

Note: Adding this dependency should clear up the errors in ExampleMain.java. If it does not, you can force a rebuild by selecting *Clean...* under the *Project* menu.



# EDEX Code Reorganization – Example (cont'd)

Final steps:

- Once the plug-in has been created, we still need to perform two steps before we can deploy it as part of EDEX
  - First, we need to add the new plug-in to the EDEX feature;
  - Second, we need to add an ANT script, *component-deploy.xml* to the project.



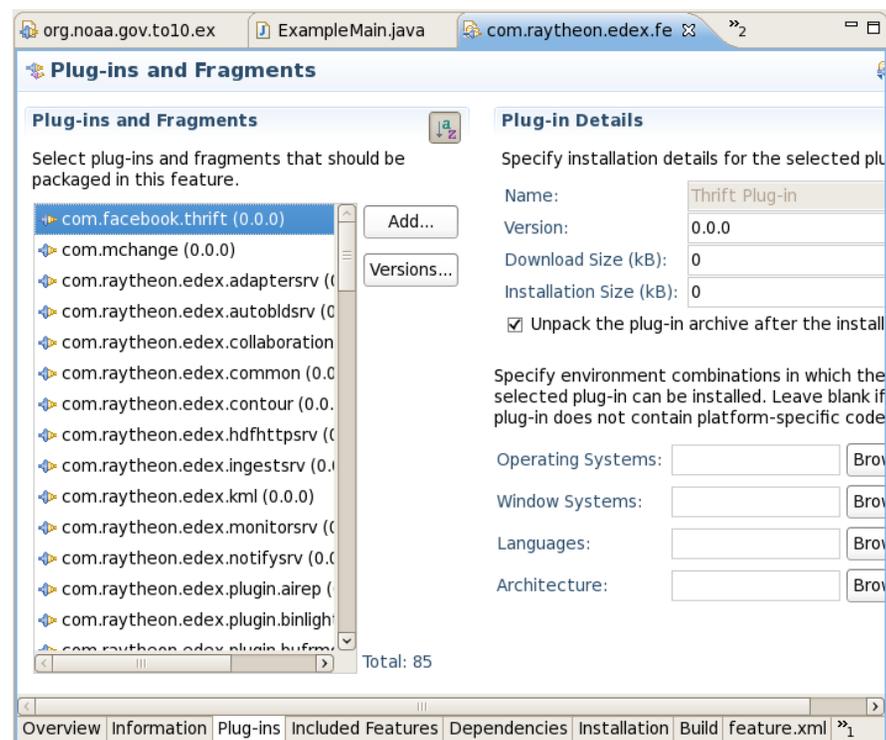
# EDEX Code Reorganization – Example (cont'd)

Open the EDEX feature for editing:

- Locate and expand the EDEX feature (com.raytheon.edex.feature.uframe)
- Double click on *feature.xml*.
- Click the *Plug-ins* tab.

Add the example project:

- Click *Add...*



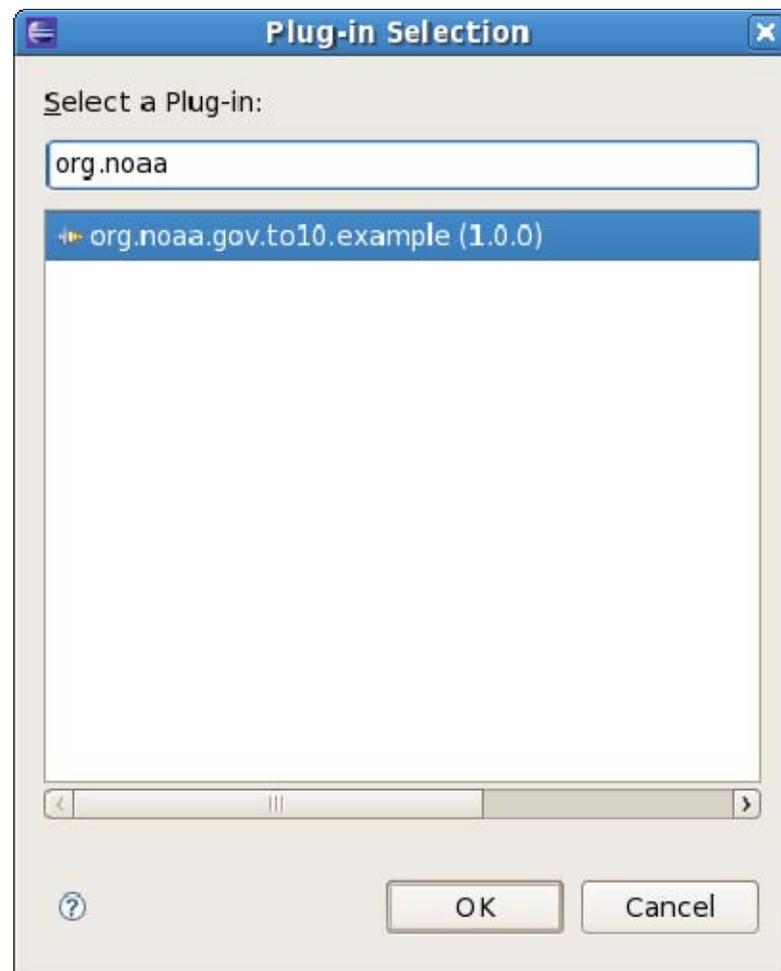
# EDEX Code Reorganization – Example (cont'd)

On the *Plug-in Selection* dialog:

- Locate the example plug-in.
  - Hint: Use the text box to limit selections.
- Select *org.noaa.gov.to10.example*.
- Click *OK*.

Back in the *Plug-in Editor*:

- Save the changes

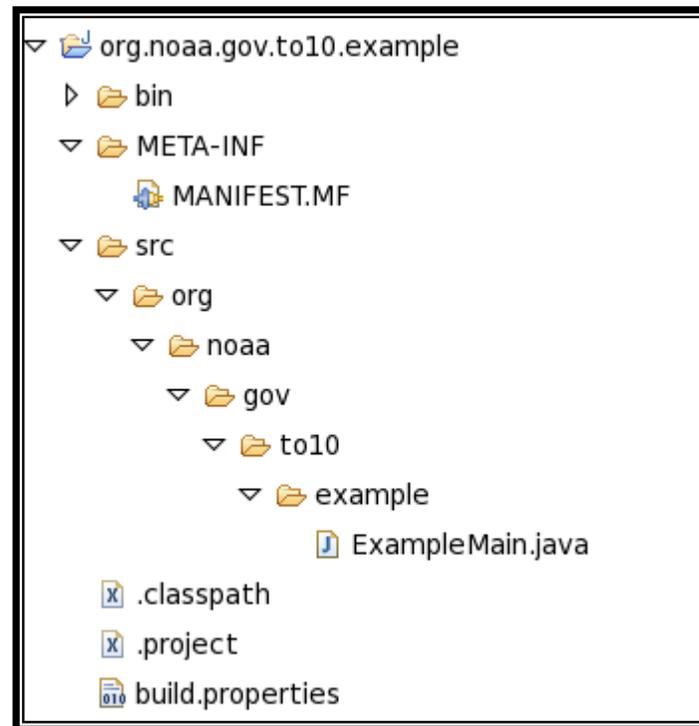


# EDEX Code Reorganization – Example (cont'd)

Add *component-deploy.xml*:

- Locate *org.noaa.gov.to10.example* in the Navigator.
- In the Navigator, right click on *org.noaa.gov.to10.example*.
- Select *New*→*File*.

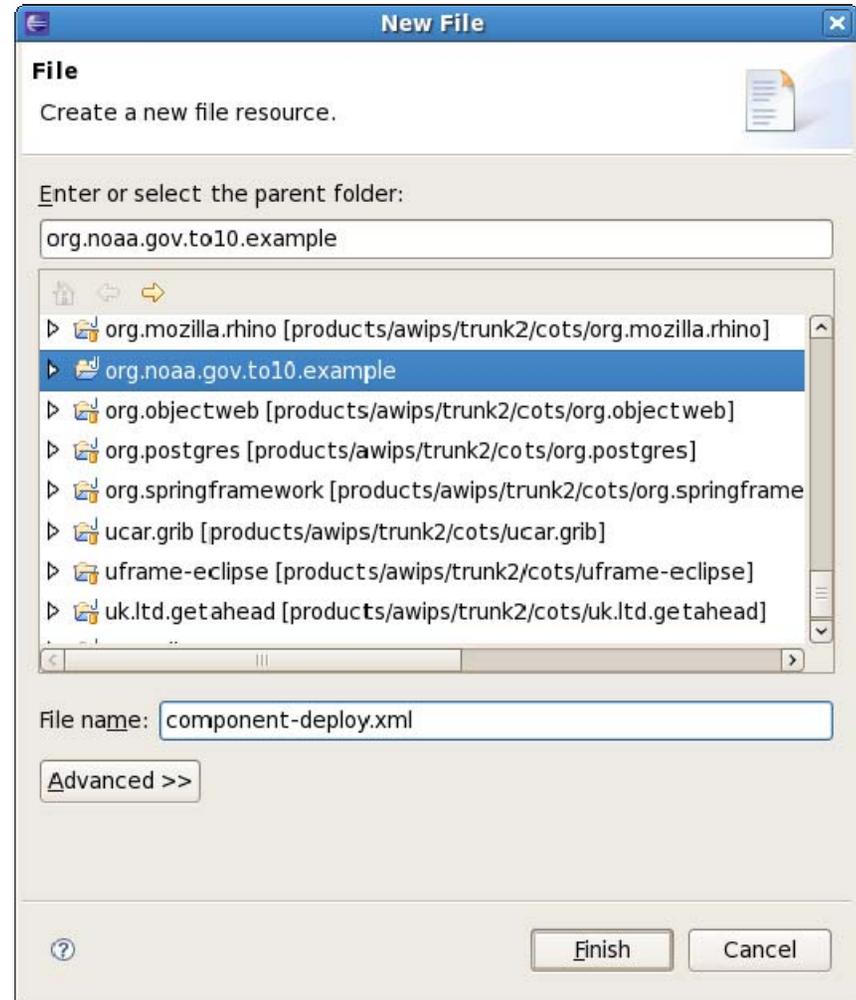
The *New File* dialog will open.



# EDEX Code Reorganization – Example (cont'd)

In the *New File* dialog:

- Enter *component-deploy.xml* in the *File name* text box.
- Click *Finish* to create the file.



# EDEX Code Reorganization – Example (cont'd)

Add the XML:

- The new file will be in the XML editor.
- Enter the XML shown below right
- Save the file.

```
<project basedir="." default="deploy" name="org.noaa.gov.to10.example">  
  <property name="component.name" value="TO10Example" />  
  
  <available file="../build.edex" property="build.dir.location"  
    value="../build.edex" />  
  
  <available file="../../../build.edex" property="build.dir.location"  
    value="../../../build.edex" />  
  
  <import file="${build.dir.location}/basebuilds/component_deploy_base.xml" />  
  
</project>
```

Note: Most EDEX plug-ins use essentially the same *plugin-deploy.xml*. You can save some time by copying the file from another plug-in project and modifying it. Be sure to modify the project in the first line and the property value in the second line **exactly** as shown here.



# EDEX Code Reorganization – Example (cont'd)

- EDEX deploy is controlled by an ANT build file (deploy-install.xml) in the build.edex project.
  - Deploying components is keyed by the directory name of the component.
  - By default, the directory must contain “raytheon” to package and deploy.
- The *includegen* ant task controls the package and deploy.
  - Add a providerfilter attribute to specify additional providers.
  - Open *deploy-install.xml* in Eclipse.
  - Locate the *includegen* tag.
  - Add the providerfilter as shown.
    - Because we have both Raytheon and NOAA code, we use “raytheon|noaa” as the filter value.

```
<includegen basedirectory="${projects.dir}"
  featurefile="${feature}"
  cotsout="${includes.dir}/cots.includes"
  plugsout="${includes.dir}/plugins.includes"
  coreout="${includes.dir}/core.includes" />
```

```
<includegen basedirectory="${projects.dir}"
  featurefile="${feature}"
  cotsout="${includes.dir}/cots.includes"
  plugsout="${includes.dir}/plugins.includes"
  coreout="${includes.dir}/core.includes"
  providerfilter="raytheon|noaa" />
```



# EDEX Code Reorganization – Example (cont'd)

- The example is nearly ready to go; we just need to build and deploy it.
  - The good news: Eclipse has already built it for us!
  - The new plug-in will deploy when EDEX is deployed.
    - Deploying the project is covered on the next few slides.

## Deploying the EDEX

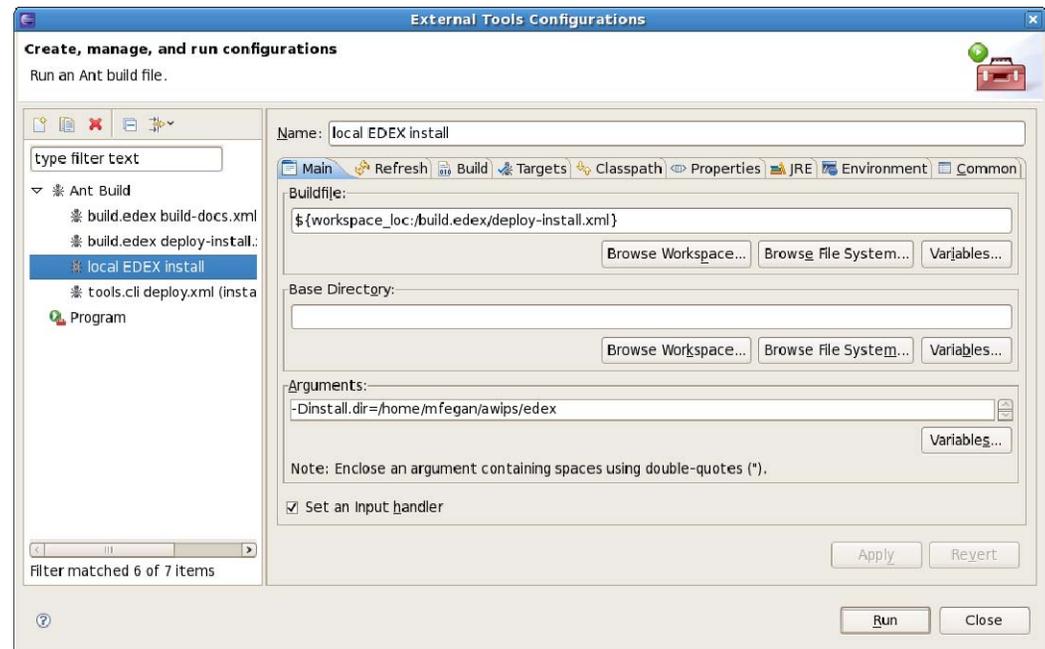
- The ADE ships with an ANT script deploying EDEX.
  - The script is located in the *edex.build* project.



# EDEX Code Reorganization – Example (cont'd)

Run the EDEX deploy script:

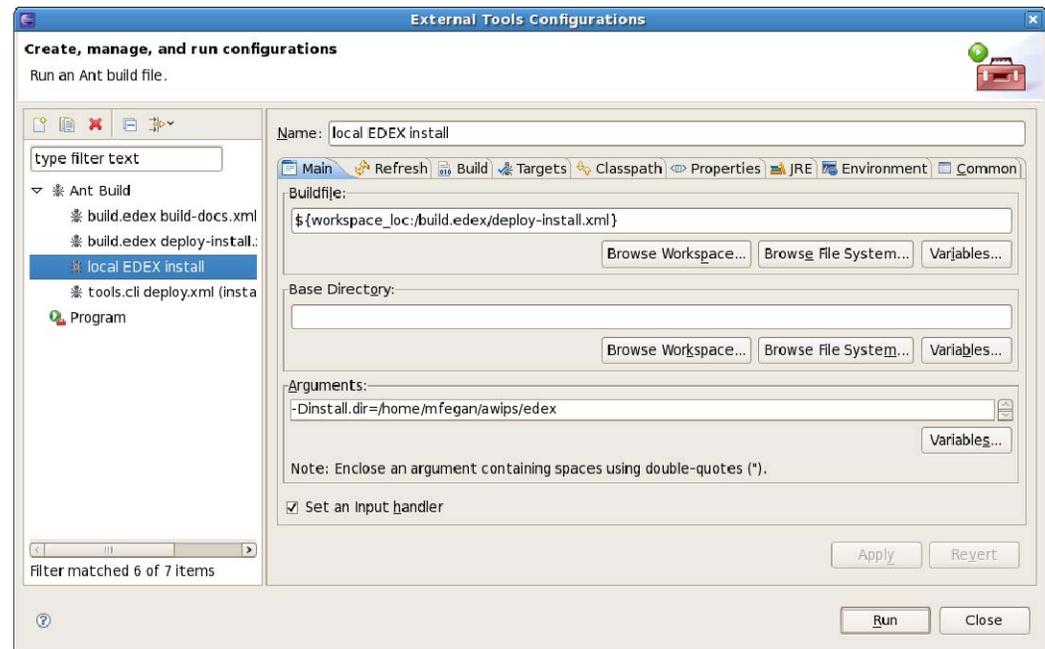
- Locate and expand the *build.edex* project in the Navigator view.
- Right click on *deploy-install.xml* and select *Run As=External Tools Configurations*.
- This opens the *External Tools Configurations* dialog.



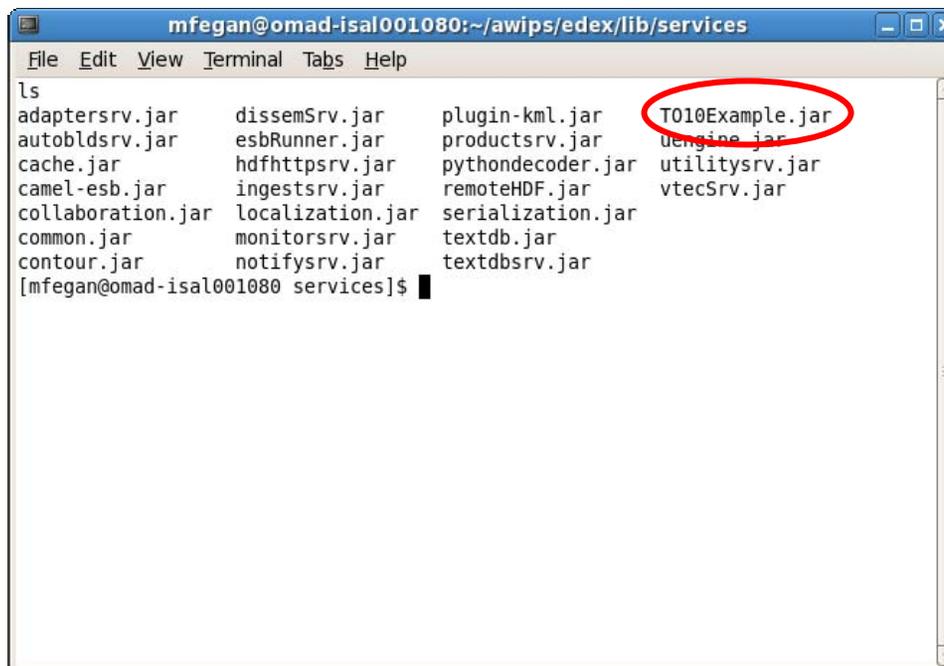
# EDEX Code Reorganization – Example (cont'd)

- Change the Name entry to something meaningful.
- In the Arguments text box, enter:  
     –*Dinstall.dir=<EDEX install directory>*.  
     Normally, this will be something like  
     */home/mfegan/awips/edex* or */awips/edex*.
- Click Apply to save the configuration.
- Click Run to deploy EDEX.

Note: Once the deploy script has been run once, it can be executed via the *External Tools* button on Eclipse's toolbar.



# EDEX Code Reorganization – Example (cont'd)

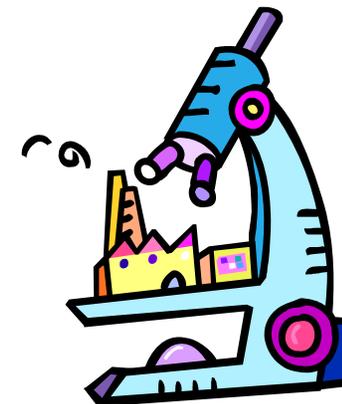


A terminal window titled "mfegan@omad-isal001080:~/awips/edex/lib/services" displays the output of the "ls" command. The output lists various JAR files in a grid format. The file "TO10Example.jar" is circled in red. The terminal prompt is "[mfegan@omad-isal001080 services]\$".

```
ls
adaptersrv.jar      dissemSrv.jar      plugin-kml.jar      TO10Example.jar
autobldsrv.jar     esbRunner.jar      productsrv.jar      engine.jar
cache.jar           hdfhttpsrv.jar     pythondcoder.jar   utilitysrv.jar
camel-esb.jar       ingestsrv.jar      remoteHDF.jar       vtecSrv.jar
collaboration.jar  localization.jar   serialization.jar
common.jar          monitorsrv.jar     textdb.jar
contour.jar         notifiysrv.jar    textdbsrv.jar
[mfegan@omad-isal001080 services]$
```

Validating the deploy:

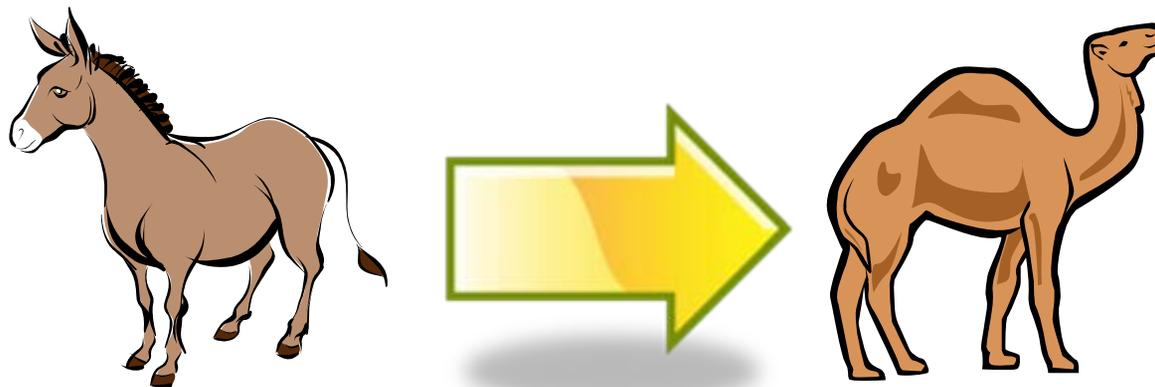
- Open the EDEX services directory.
  - location /awips/edex/lib/services
- List the directory contents.
- Verify that TO10Example.jar is in the directory.



# Questions?



# Transition from Mule to Camel



# Why Camel?

- Mule 1.4.x no longer viable solution
  - Mule 1.4 branch full of show-stopping bugs that are not being fixed in Mule 1.4.x product line
  - Mule 1.4.3 introduced a new licensing model that indicates a potential move to closed source
  - Mule 1.4.3 included library changes that forced EDEX recoding
  - Even with suggested fixes applied, the software was deemed to be highly unstable for the processing load required by EDEX
    - MTBF dropped to under an hour.
  - Major EDEX redesign indicated to support continuing with Mule!

Note: Even though Mule has been eliminated from the EDEX code baseline, some non-code files still contain references to Mule. The primary example of these Mule references is the obsolete Mule deployment descriptor files. These files will be removed as part of a general code cleanup scheduled for TO 11.



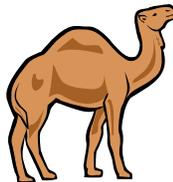
# Alternatives Analyzed

- Closed-source commercial ESB
  - Not considered an option due to requirements
- Open-source ESB Alternatives
  - Mule 2.0 ESB
    - 2.0 version has more restrictive licensing (“Red Flagged” by RME Aurora), Mule has gone to more commercial model
    - No telling whether next version of Mule will have an even more restrictive license and go closed-source
  - Apache ESB
    - Camel on steroids, unnecessary for our purposes, should stay lightweight if we don't need its features
  - JBoss ESB
    - Runs inside Jboss, which we don't want
  - Apache Camel Integration Framework



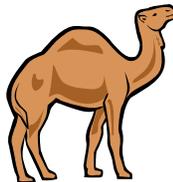
# Camel Benefits

- Apache project – quality, reliable, open source
- Very lightweight
  - Camel bills itself as a “Spring based Integration Framework”
  - Spring (XML) configurable
  - Easy-to-configure services and routes
- Camel may be embedded in ActiveMQ
  - Facilitates running ESB as a single process
    - ActiveMQ, EDEX, and the test driver all run from within the same Java VM
    - Can be separated and deployed separately if necessary
- Camel is an “Integration Framework”
  - Designed to work with POJO classes rather than Camel-specific classes
  - This provides for a more flexible runtime environment



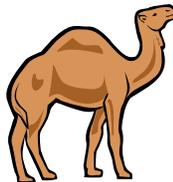
# Camel Benefits (cont'd)

- Facilitates migration to a new processing model
  - More fault tolerant
    - Less synchronicity, more transaction-style processing (with ability to migrate to full transactions in future)
    - Less data read into memory queues
    - Pipeline style processing
      - ▶ Ingest, persist, and decode occur in a single thread, more data not read in until finished
  - More customizable
    - Don't force one-size-fits-all
    - Separators, etc. are now optional



# Camel Benefits (cont'd)

- Camel runs as an integrated part of ActiveMQ
  - ActiveMQ uses a journal approach to queue management
    - The journal is saved at shutdown; it is restored at startup
  - There is generally no loss of data when EDEX is stopped and restarted!
- Simplified EDEX startup
  - EDEX is started using a single script, start.sh
  - EDEX mode set by passing a parameter to the script
    - Standalone: Single server running EDEX and optionally PostgreSQL
    - Server: Master server in a cluster of 2 or more servers
    - Client: Slave server in a cluster of 2 or more servers
  - Additional optional argument allows runtime debugging



# Running EDEX

## Start PostgreSQL

- In a console window, execute  
`<<awips-home>>/bin/start_developer_postgres.sh`

## Start EDEX

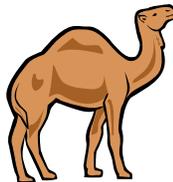
- In a second console window, execute  
`<<awips-home>>/edex/bin/start.sh standalone`

Hint: Replace <<awips-home>> with the location where EDEX is installed.



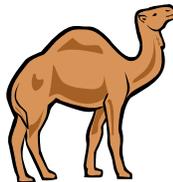
# Camel-Specific Code Required

- All Camel-specific code has been isolated into to plug-ins:
  - com.raytheon.uf.edex.esb.camel
    - Provides “executor” class for ESB launch
    - Implements EDEX specific “components”
    - Provides several EDEX specific utilities
  - com.raytheon.uf.edex.esb.camel.launcher
    - Provides the required a “main” class for system startup
- This isolation generally eliminates ESB dependencies
  - Code not isolated into these plug-ins may be used elsewhere without requiring Camel
- Any additional Camel-specific code will be isolated into the first plug-in



# Camel Configuration

- All Camel deployment descriptor (configuration) files are bundled into the appropriate EDEX jars
  - Located in res/spring
- File names must be unique within EDEX
  - Typically file name pattern: {function}-spring.xml or {function}-server.xml
  - Deployments that should run on only one server in the clustered environment must use {function}-server.xml
- Examples:
  - Purge Service: purge-server.xml
  - TAF Ingest: taf-spring.xml
  - Subscription Service: subscription-spring.xml
  - Subscription Script Runners: runners-spring.xml



# Camel Configuration: Basics

```

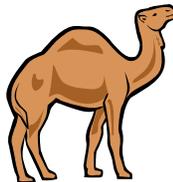
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:amq="http://activemq.apache.org/schema/core"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
    http://activemq.apache.org/schema/core
    http://activemq.apache.org/schema/core/activemq-core.xsd
    http://activemq.apache.org/camel/schema/spring
    http://activemq.apache.org/camel/schema/spring/camel-spring.xsd">
  <bean id="<<unique Bean ID>>" class="<<implementing class>>">
    <property name="<<Property Name>>" value="<<property Value" />
  </bean>
  <bean id="<<unique Bean ID" class="<<implementing class"/>

  <camelContext id="unique context ID"
    xmlns="http://activemq.apache.org/camel/schema/spring"
    errorHandlerRef="errorHandler">

    <route id="<<unique route ID">
      <!-- route implementation -->
    </route>
  </camelContext>
</beans>

```

- In EDEX, the main document tag in a Camel deployment descriptor is the <beans /> tag
- Embedded tags:
  - <bean /> tag defines classes for later use by Camel
    - May set properties using nested <property /> tag
  - <camelContext /> tag defines processing to be managed by Camel by defining one or more routes
    - A route is declared using the <route /> tag
    - Exact processing to perform is specified by additional nested tags within each route



# Camel Configuration: - camelContext

- In EDEX, a <camelContext /> tag typically defines one or more routes
  - each route is defined in a separate <route /> tag
- a <camelContext /> tag may also define one or more endpoints
  - each endpoint is defined in a separate <endpoint /> tag
    - endpoint URI may include values from environment

```

<camelContext id="<<unique Context ID>>"
  xmlns="http://activemq.apache.org/camel/schema/spring"
  errorHandlerRef="errorHandler">

  <endpoint id="<<unique ID>>"
    uri="<<endpoint URI>>" />

  <!-- define routes -->
  <route id="<<unique Route ID>>">
    <from ref="<<endpoint ID>>" />
    <bean ref="<<bean ID>>" />
    <to uri="<<destination URI>>" />
  </route>

  <route id="<<unique Route ID>>">
    <from uri="<<endpoint URI>>" />
    <bean ref="<<bean ID>>" />
    <bean ref="<<bean ID>>" method="<<method name>>" />
  </route>
</camelContext>

```

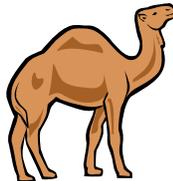
Note: Specific examples of Camel deployment descriptors will follow.



# Camel Configuration: Camel Routes

- Routes are paths through services for various tasks
  - Routes are defined using the `<route/>` tag
  - Following SOA, can chain routes together
  - Routes can include transformers
- Embedded tags:
  - `<from/>` and `<to/>` are basic route tags
  - `<bean/>` tag defines processing

```
<route id="uEngineHttpThrift">  
  <from uri="jettyedex:http://0.0.0.0:9581/services/pyproductthrift" />  
  <bean ref="uEngine" method="executePython" />  
  <bean ref="serializationUtil" method="transformToThrift" />  
</route>
```

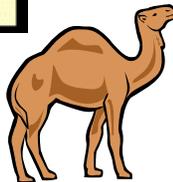


# Camel Configuration: Java Beans

- Java Beans are used to define basic units of work
  - Uses POJO's; don't have to implement interfaces or extend anything
  - Ensures no dependencies on Camel for services and transformers
- Java Beans are declared and referenced using the `<bean/>` tag
  - Java Beans are declared outside the `<camelContext/>` tag
  - Java beans are referenced within the `<camelContext/>` tag
- `<bean/>` tag syntax:
  - Use **id** and **class** attributes when declaring the bean
  - Use **ref** and **method** attributes when using the bean

```
<bean id="uEngine" class="com.raytheon.edex.productsrv.ProductSrv" />

<route id="uEngineHttpThrift">
  <from uri="jettyedex:http://0.0.0.0:9581/services/pyproductthrift" />
  <bean ref="uEngine" method="executePython" />
  <bean ref="serializationUtil" method="transformToThrift" />
</route>
```

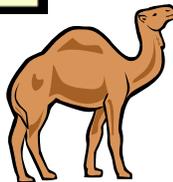


# Camel Configuration: Java Beans (cont'd)

- Within a route, the return value from one component is passed to the method called in the next method
- In the example below:
  - The HTTP request obtained from the `<from/>` tag,
  - Is passed to the `executePython` method of the `uEngine` bean,
  - Its return value is passed to the `transformToThrift` method of the `SerializationUtil` bean,
  - Its return value is the response to the HTTP request

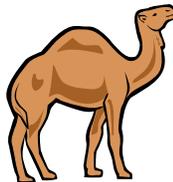
```
<bean id="uEngine" class="com.raytheon.edex.productsrv.ProductSrv" />

<route id="uEngineHttpThrift">
  <from uri="jettyedex:http://0.0.0.0:9581/services/pyproductthrift" />
  <bean ref="uEngine" method="executePython" />
  <bean ref="serializationUtil" method="transformToThrift" />
</route>
```



# Camel Configuration: Useful Routing

- **Multicast.** Send one message to multiple endpoints
- **Splitter.** Split a message into multiple messages
- **Try/Catch.** Catch exceptions from sections of a route
- **Filter.** Filter what messages reach the endpoint



# Camel Documentation

- <http://activemq.apache.org/camel/overview.html>
- Mailing list/forum is very helpful
- Search the camel site (the table of contents is not so great)

It's not documentation, but...

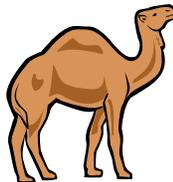
- The EDEX code base provides a number of examples of Camel descriptors



# EDEX Camel Deployment Issues

- All EDEX components are deployed into the library directory
  - Deployment is managed by the installer
    - “awips” is replaced by a path provided
- When EDEX starts up, it looks into two directories for jars containing deployment descriptors
- All plug-ins (data decoders) are deployed into the *plugins* directory
  - Plug-ins support additional data oriented features such as purging
- Services and libraries are generally deployed into the services directory

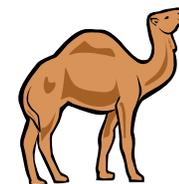
```
/awips/edex
|-- activemq-data
|-- bin
|-- conf
|-- data
|   |-- hdf5
|   |-- processing
|   |-- sbn
|   |-- static
|   |-- uEngine
|   |-- utility
|-- lib
|   |-- core
|   |-- dependencies
|   |-- native
|   |-- plugins
|   |-- services
|-- logs
|-- webapps
|   |-- admin
|   |-- uEngineWeb
```



# EDEX Camel Deployment Issues (cont'd)

- Deployment of components is determined by project name
- All AWIPS II components are deployed into the plug-ins and services directories under the edex/lib directory.
  - As a default, a service is a component having “raytheon” in its name; a plug-in has both “raytheon” and “plugin” in its name
  - This default can be modified by editing the deploy scripts
- Non-AWIPS II components used by EDEX (e.g., Camel) are deployed into the dependencies directory

```
/awips/edex
|-- activemq-data
|-- bin
|-- conf
|-- data
|   |-- hdf5
|   |-- processing
|   |-- sbn
|   |-- static
|   |-- uEngine
|   |-- utility
|-- lib
|   |-- core
|   |-- dependencies
|   |-- native
|   |-- plugins
|   |-- services
|-- logs
|-- webapps
|   |-- admin
|   |-- uEngineWeb
```



# Camel Configuration – Example

## Problem:

Create an EDEX component, *YAHW*, that logs messages to the EDEX system log. In this example, we add wiring to have *YAHW* available as a Camel managed service endpoint.

## Solution:

Add the deployment descriptor to the *TO10Example* project to enable HTTP interaction with the endpoint.

## To Do:

Sending a message to the endpoint will be provided in a later example.



# Camel Configuration – Example (cont'd)

- Once the component has been coded (see previous example), it needs to be wired into EDEX (configured as a Camel endpoint)
  - The Camel configuration file for this endpoint is shown below.
- The next few slides cover the file creation in detail

Note: The Camel descriptor file must be in res/spring in the EDEX project!

```
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:amq="http://activemq.apache.org/schema/core"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
    http://activemq.apache.org/schema/core
    http://activemq.apache.org/schema/core/activemq-core.xsd
    http://activemq.apache.org/camel/schema/spring
    http://activemq.apache.org/camel/schema/spring/camel-spring.xsd">

  <bean id="to.10.example"
    class="org.noaa.gov.edex.to10.example.ExampleMain" />
  <camelContext id="example-camel"
    xmlns="http://activemq.apache.org/camel/schema/spring"
    errorHandlerRef="errorHandler">
    <route id="to.10.example.http.route">
      <from uri="jettyedex:http://0.0.0.0:9581/services/example" />
      <try>
        <bean ref="to.10.example" method="execute" />
        <catch>
          <exception>java.lang.Throwable</exception>
          <to uri="log:dispatcher?level=ERROR&showBody=false" />
        </catch>
      </try>
    </route>
  </camelContext>
</beans>
```



# Camel Configuration – Example (cont'd)

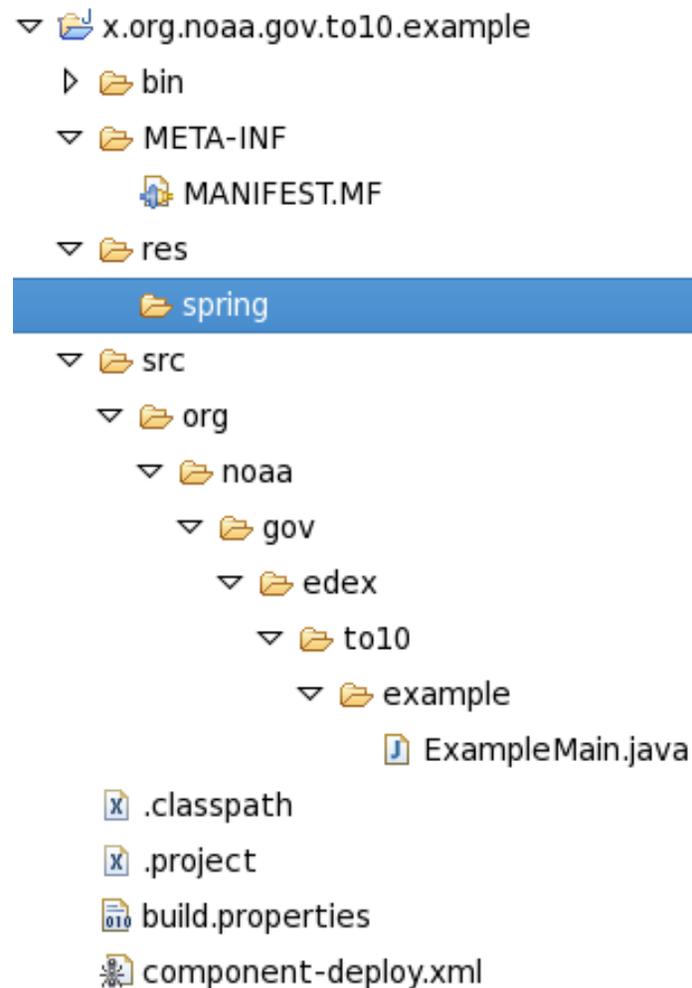
Create the resource folder:

- Locate *org.noaa.gov.to10.example* in the Navigator.
- In the Navigator, right click on *org.noaa.gov.to10.example*.
- Select *New*→*Folder*.

The *New Folder* dialog will open

- Enter *res/spring* for *Folder name*.
- Click *Finish*.

The new folder will be created (see listing at right).



# Camel Configuration – Example (cont'd)

Add *example-spring.xml*:

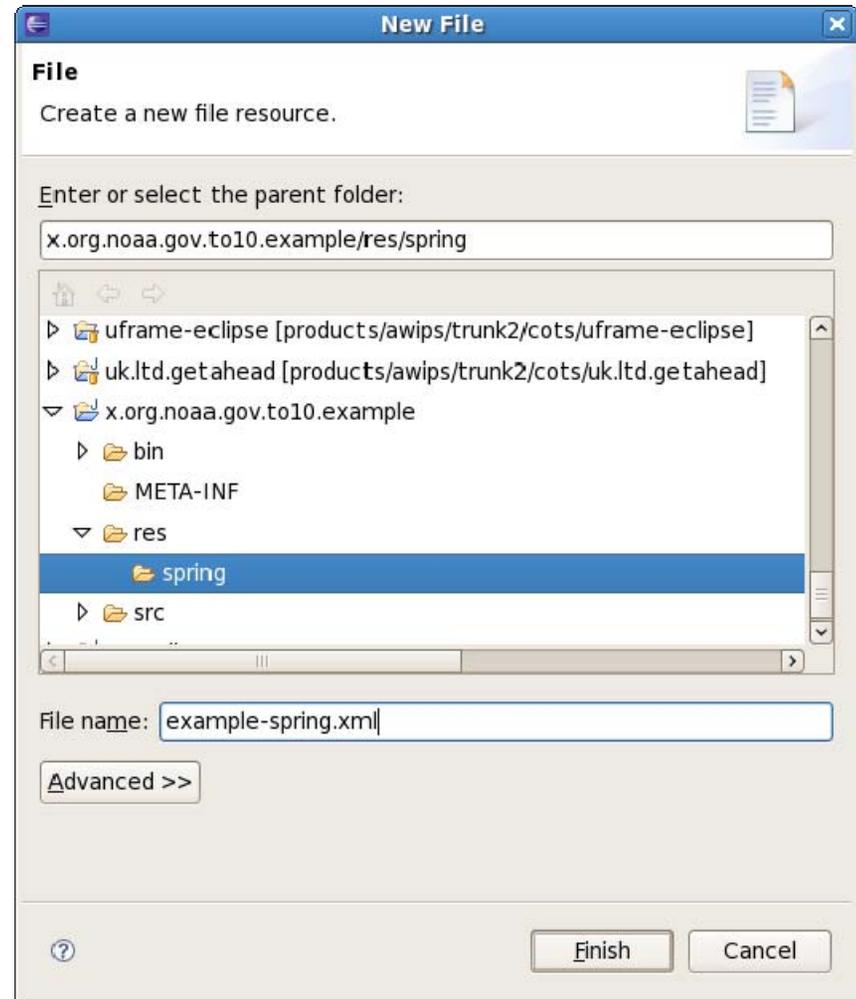
- In the Navigator, right click on *res/spring* in *org.noaa.gov.to10.example*.
- Select *New*→*File*.

The *New File* dialog will open.

- Enter *example-spring.xml* as the *File name*.
- Click *Finish*.

Eclipse creates the new file.

- *example-spring.xml* is open in the XML editor.
  - Note that the new file is empty!



# Camel Configuration – Example (cont'd)

Add the basic descriptor XML:

- Enter the XML below; this is the `<beans/>` tag.
  - Hint: This XML tag is the same in every descriptor; it may be copied from another EDEX component.

```
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:amq="http://activemq.apache.org/schema/core"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
    http://activemq.apache.org/schema/core
    http://activemq.apache.org/schema/core/activemq-core.xsd
    http://activemq.apache.org/camel/schema/spring
    http://activemq.apache.org/camel/schema/spring/camel-spring.xsd">

</beans>
```

**CRITICAL!!** This is the Spring `<beans/>` tag, which defines the XML document in the file. It must be entered exactly as shown. Unless it is entered correctly, the new bean will not be available.



# Camel Configuration – Example (cont'd)

Add the `<camelContext/>` tag:

- Add the `<camelContext/>` tag shown below to `example-spring.xml`.
- Hint: This XML tag is the same in every descriptor; it may be copied from another EDEX component.



```
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:amq="http://activemq.apache.org/schema/core"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
    http://activemq.apache.org/schema/core
    http://activemq.apache.org/schema/core/activemq-core.xsd
    http://activemq.apache.org/camel/schema/spring
    http://activemq.apache.org/camel/schema/spring/camel-spring.xsd">

  <camelContext id="example-camel"
    xmlns="http://activemq.apache.org/camel/schema/spring"
    errorHandlerRef="errorHandler">

  </camelContext>
</beans>
```

**CRITICAL!!** This tag defines an XML sub-document that specifies the Camel endpoint. It must be entered exactly as shown. Unless it is entered correctly, the new bean will not be available.



# Camel Configuration – Example (cont'd)

Add the bean definition

- Add the `<bean/>` tag shown below to `example-spring.xml`

New XML

```
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:amq="http://activemq.apache.org/schema/core"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
    http://activemq.apache.org/schema/core
    http://activemq.apache.org/schema/core/activemq-core.xsd
    http://activemq.apache.org/camel/schema/spring
    http://activemq.apache.org/camel/schema/spring/camel-spring.xsd">

  <bean id="to.10.example"
    class="org.noaa.gov.edex.to10.example.ExampleMain"/>

  <camelContext id="example-camel"
    xmlns="http://activemq.apache.org/camel/schema/spring"
    errorHandlerRef="errorHandler">

    </camelContext>
</beans>
```

Note: The `<bean/>` tag defines a Java Bean that will be used in defining the Camel endpoint.



# Camel Configuration – Example (cont'd)

Complete the Route definition

- Add the route definition as show below

```
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:amq="http://activemq.apache.org/schema/core"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
    http://activemq.apache.org/schema/core
    http://activemq.apache.org/schema/core/activemq-core.xsd
    http://activemq.apache.org/camel/schema/spring
    http://activemq.apache.org/camel/schema/spring/camel-spring.xsd">

  <bean id="to.10.example"
    class="org.noaa.gov.edex.to16.example.ExampleMain"/>

  <camelContext id="example-camel"
    xmlns="http://activemq.apache.org/camel/schema/spring"
    errorHandlerRef="errorHandler">

    <route id="to.10.example.http.route">
      <from uri="jettyedex:http://0.0.0.0:9581/services/example" />
      <try>
        <bean ref="to.10.example" method="execute" />
      <catch>
        <exception>java.lang.Throwable</exception>
        <to uri="log:dispatcher?level=ERROR&showBody=false"/>
      </catch>
    </try>
  </route>

</camelContext>
</beans>
```

New XML

Hint: The new XML is detailed on the next slide.



# Camel Configuration – Example (cont'd)

Exception handling,  
errors are logged

```
<route id="to.10.example.http.route">
  <from uri="jettyedex:http://0.0.0.0:9581/services/example" />
  <try>
    <bean ref="to.10.example" method="execute" />
    <catch>
      <exception>java.lang.Throwable</exception>
      <to uri="log:dispatcher?level=ERROR&showBody=false" />
    </catch>
  </try>
</route>
```

Triggered by HTTP  
request

Request processed  
by example class



# Testing the Example

## Build:

- Build and deploy EDEX as previously described

## Run:

- Start PostgreSQL and EDEX as previously described

## Test:

- Open a browser (Firefox) and enter the following URL:  
`http://localhost:9581/services/example#This is a test`
- Hit the go button; if everything works correctly, you will get an empty page



## Testing the Example (cont'd)

Check the log:

- Open a console window.
- Change directory to <<awips-home>>/edex/logs.
- Examine the latest log in a text viewer such as *less* or *view*.
- Near the bottom of the log, you should see an entry similar to this:

```
INFO 2009-01-29 14:22:22,873 [btpool2-0] ExampleMain:  
org.mortbay.HttpParser$Input@6e36ff
```

Note: The log entry means the message was delivered and logged. We really need to do some processing to make the logging more meaningful. This will be covered later.



# Questions?



# Database Updates



# Database Updates: Table Structure

- Metadata database has a new schema
  - Has an additional schema: subscription
  - The tables in this schema are used by the EDEX subscription service
- Metadata database awips schema has been reorganized
  - The most visible change is the elimination of partitions in the non-static tables
  - Partitions were used to facilitate time based data purges, however ...
    - This locked in a single purge strategy
      - ▶ Single strategy not appropriate for all data types
    - Main table is essentially a view combining the partitions
      - ▶ PostgreSQL treats each partition as a separate table
      - ▶ Indices and constraints existed on the partition
      - ▶ Causes problems with searches and inserts



# Database Updates: DAO

- As previously briefed (TO9), Data Access Object (DAO) pooling has been eliminated in EDEX
  - Has been replaced by the lower level caching and pooling provided by PostgreSQL and Hibernate
- This will only affect developers writing code that accesses data
  - In most applications, you will be able to use the CoreDao class for data retrieval
  - In most cases, the code that utilizes a DAO has been simplified



# Database Updates: Purging Tables

- Database purging is now managed by the individual data decoders
  - Each decoder specifies a purger in its plugin.xml file
    - Located in res/conf in the plug-in
- A *default purge* strategy is provided (DefaultPurgerImpl) that implements a simple time-based purge
  - Purges table entries that are older than 24 hours
- A no-purge strategy is provided (NoOpPurger) that implements a non-purging strategy
  - Use only if alternative purging is implemented

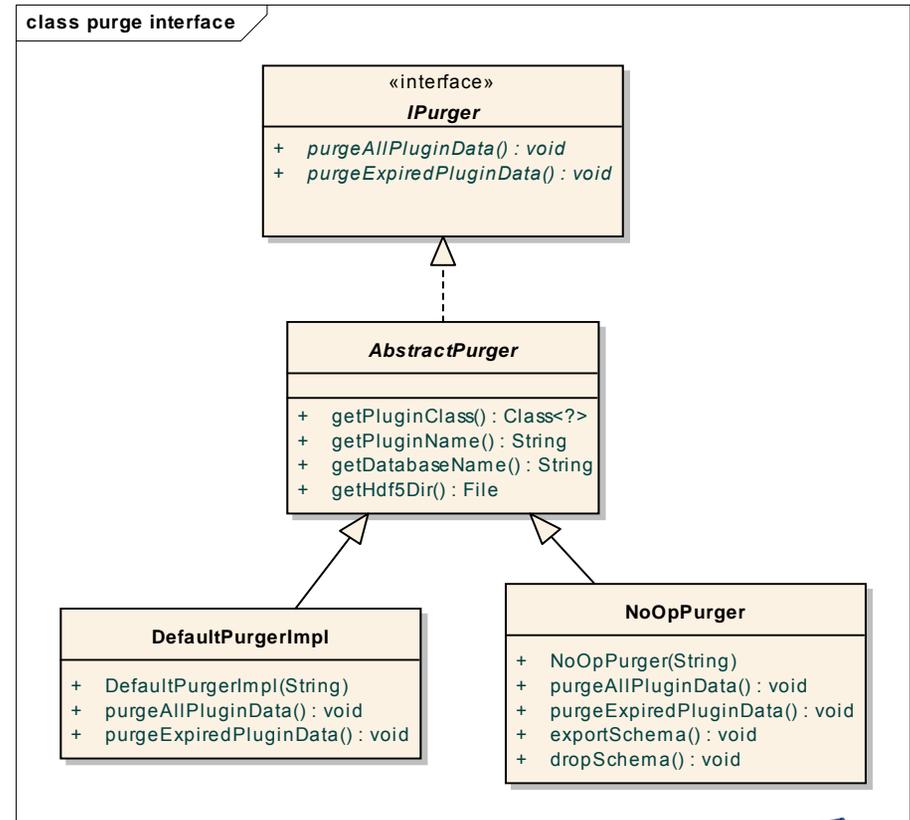
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<properties>
  <Name>GRIB</Name>
  <Plugin>>true</Plugin>
  <Database>metadata</Database>
  <Record>com.raytheon.edex.plugin.grib.GribRecord</Record>
  <Decoder>com.raytheon.edex.plugin.grib.GribDecoder</Decoder>
  <Purger>com.raytheon.edex.db.purge.DefaultPurgerImpl</Purger>
  <SEPARATOR>com.raytheon.edex.plugin.grib.GribSeparator</SEPARATOR>
</properties>
```

**CAUTION!!** In TO10, EDEX purging is limited to components deployed in the plug-ins directory.



# Database Updates: Purging Tables (cont'd)

- A decoder plug-in may implement a purge strategy
  - The purge strategy implements the *IPurger* interface
  - Package: `com.raytheon.edex.db.purge`
  - Part of the `com.raytheon.edex.common` component
- Examples:
  - `TextPurger` in `com.raytheon.edex.plugin.txt`



Note: Custom purgers *implement* *IPurger*; they *extend* *AbstractPurger*, which provides functionality needed by the purge service.



# Questions?



# Database Updates: Hibernate Mapping

- Hibernate Mapping is now implemented using Java 5.0 style annotations
  - Annotations allow the Hibernate mapping information to be maintained in the code rather than in a separate file
    - no more \*.hbm.xml files!
- Basic annotations are provided by the Java Persistence API
  - Class level annotations identify the class as persistable
- Hibernate specific annotations are provided by Hibernate
  - Class level annotations map the Java class to the database table
  - Field level attributes map the class' attributes to database fields
  - Some support for foreign key relationships are provided
- Additional, custom annotations have been implemented
  - Field level annotation defining elements of the data URI
- Classes containing Hibernate mappings are listed in the component
  - listed in `com.raytheon.uf.common.serialization.ISerializableObject`
  - located in META-INF/services



# Database Updates: Hibernate Mapping (cont'd)

## Hibernate Annotations

### ■ Class level annotations

#### – *@Entity*, *@Table*

- Specifies the DB table to class mapping for Hibernate
- *@Table* has two arguments, name and schema, schema is optional

### ■ Attribute level annotations

#### – *@Id*, *@GeneratedValue*

- Specifies the attribute as the table's primary key, value to be generated

#### – *@Column*

- Maps the attribute to a DB column
- Optional argument, length, is used to specify column size

#### – *@ManyToOne*, *@JoinColumn*, *@OneToMany*

- Used to specify foreign key relationships



# Hibernate Mapping – Example

## Problem:

Add database persistence to YAHW. Specifically, create a Hibernate mapped class that will store messages received by YAHW to a database table. The table will be in the *example* schema of the *metatadata* database. The table will be named *example* and will include fields for message and its receipt time.

## Solution:

Create a POJO with three fields: id; time; and message. Add annotations mapping to the desired database table. Hook the POJO into YAHW for future use.

## To Do:

We will add additional capability to this class later.



## Hibernate Mapping – Example (cont'd)

Add new dependencies to the YAHW project:

- Open *org.noaa.gov.to10.example/META-INF/MANIFEST.MF*.
- Select the *Dependencies* tab in the editor.
- Add dependencies to the following:
  - com.raytheon.edex.common
  - javax.persistence

Hint: Eclipse operations previously covered in detail will only be summarized from here out.



# Hibernate Mapping – Example (cont'd)

Create a new package for the data object:

- Using eclipse, create a package:  
*org.noaa.gov.edex.to10.example.data*
- In the newly created package, create a class: *ExampleData*
  - ExampleData will *extend* PersistableDataObject.
  - ExampleData will *implement* ISerializableObject.
- Once Eclipse has generated ExampleData, add the field shown.

```
package org.noaa.gov.edex.to10.example.data;  
  
import com.raytheon.edex.db.objects.PersistableDataObject;  
import com.raytheon.uf.common.serialization.ISerializableObject;  
  
public class ExampleData extends PersistableDataObject implements  
    ISerializableObject {  
  
    private static final long serialVersionUID = 1L;  
  
    public ExampleData() {  
        // TODO Auto-generated constructor stub  
    }  
  
}
```



# Hibernate Mapping – Example (cont'd)

Add data fields:

- Add the class attributes shown.
  - *id* will be a unique identifier.
  - *message* will contain the text.
  - *time* will save the receipt time.

Add accessor methods:

- Add standard getters and setters for the class attributes.
- Use the standard “bean” pattern.

Hint: Eclipse will generate accessors.

- Select Source → Generate Getters and Setters...

```
package org.noaa.gov.edex.to10.example.data;

import java.util.Calendar;

import com.raytheon.edex.db.objects.PersistableDataObject;
import com.raytheon.uf.common.serialization.ISerializableObject;

public class ExampleData extends PersistableDataObject implements
    ISerializableObject {

    private static final long serialVersionUID = 1L;

    private long id;

    private String message;

    private Calendar msgTime;

    public ExampleData() {
        // TODO Auto-generated constructor stub
    }

}
```

Note: getters and setters are not shown.



# Hibernate Mapping – Example (cont'd)

Add the Hibernate annotations at the class level:

- Add package imports.
- Add the class level annotations as shown.

Note: Annotation functions

@Entity -- Identifies the class as being an object to persist

@Table -- identifies the table (and schema) the class is mapped to

- The schema is optional, by default it is set to “awips”.

```
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
@Table(name="example", schema="example")
public class ExampleData extends PersistableDataObject implements
    SerializableObject {
}
```

Note: This listing shows only the new code and the existing code needed to establish its context.



# Hibernate Mapping – Example (cont'd)

Add the Hibernate annotations at the attribute level:

- Add package imports as shown.
- Add attribute annotations as shown.

Note: Annotation functions

*@Id* -- identifies the field as the primary key

*@GeneratedValue* -- identifies that the database should generate the value

*@Column* -- specifies the size of the field  
length attribute is optional

Note: This listing shows only the new code and the existing code needed to establish its context.

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
@Table(name="example", schema="example")
public class ExampleData extends PersistableDataObject implements
    SerializableObject {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue
    private long id;

    @Column(length=512)
    private String message;

    @Column
    private Calendar msgTime;

}
```



## Hibernate Mapping – Example (cont'd)

Link the new class into the framework:

- Using Eclipse, create a directory called *services* in the project's META-INF directory.
- Using Eclipse, create a file in META-INF/services. The name of the file:

*com.raytheon.uf.common.serialization.ISerializableObject*

- Enter the full name of the new data object in this file. The name to enter:

*org.noaa.gov.edex.to10.example.data.ExampleData*

**CAUTION!!** Remember: In Linux, all names are case sensitive.



# Hibernate Mapping – Example (cont'd)

Modify YAHW's main class to write to the database:

- Add the imports as shown (in bold).

These imports provided access to our ExampleData class and EDEX database access code. EdexException is a standard Exception class used within EDEX.

Note: This listing shows only the new code and the existing code needed to establish its context.

```
package org.noaa.gov.edex.tol0.example;  
  
import java.util.Calendar;  
  
import org.apache.commons.logging.Log;  
import org.apache.commons.logging.LogFactory;  
import org.noaa.gov.edex.tol0.example.data.ExampleData;  
  
import com.raytheon.edex.db.dao.CoreDao;  
import com.raytheon.edex.db.dao.pool.DaoConfig;  
import com.raytheon.edex.exception.EdexException;
```



# Hibernate Mapping – Example (cont'd)

Modify YAHW's main class to write to the database:

- Add the *saveMessage(...)* method shown to our ExampleMain class.

This new method handles all communication with the database. It throws an EdexException when the database insert fails.

```
public class ExampleMain {  
  
    private void saveMessage(String message) throws EdexException {  
        Calendar now = Calendar.getInstance();  
        ExampleData ed = new ExampleData();  
        ed.setMessage(message);  
        ed.setMsgTime(now);  
        DaoConfig cfg = DaoConfig.forClass(ExampleData.class);  
        CoreData dao = new CoreData(cfg);  
        try {  
            dao.saveOrUpdate(ed);  
        } catch (Exception e) {  
            String msg = "Unable to perform database insert";  
            logger.error(msg,e);  
            throw new EdexException(msg,e);  
        }  
    }  
}
```

Note: This listing shows only the new code and the existing code needed to establish its context.



# Hibernate Mapping – Example (cont'd)

Modify YAHW's main class to write to the database

- Add the new code (in bold) to ExampleMain's *execute()* method.

This code calls the newly created method. A try...catch block is used to trap any exception thrown by the method. In case of an error, the exception's message is returned rather than echoing the message.

Note: This listing shows only the new code and the existing code needed to establish its context.

```
public class ExampleMain {  
    private transient Log logger = LogFactory.getLog(getClass());  
  
    public ExampleMain() {  
        super();  
    }  
  
    public String execute(String message) {  
        logger.info(message);  
        try {  
            saveMessage(message);  
        } catch (EdexException e) {  
            return e.getMessage();  
        }  
        return message;  
    }  
}
```



# Hibernate Mapping – Example (cont'd)

Testing the newly created functionality:

- Because we are working in Eclipse, we know the code compiles.
- Beyond that, we are not ready to test the new code. First, we need to get the database set up.



# Questions?



# Database Updates: Table Generation

- As before, dynamic database tables may be auto-generated at system startup
  - With the TO10 AWIPS II release, dynamic table generation is enabled via Hibernate mappings for the data object classes
    - Note: Only the tables are generated; this process doesn't generate a new database or a schema
  - Classes containing Hibernate mappings are listed in the component
    - Listed in `com.raytheon.uf.common.serialization.ISerializableObject`
    - Located in META-INF/services

```
com.raytheon.edex.subscription.data.ReplacementRecord  
com.raytheon.edex.subscription.data.SubscriptionRecord
```

- Only plug-ins may auto generate dynamic tables



# Database Updates: Table Generation (cont'd)

- New in TO10: Plug-ins may include DDL to generate/populate/modify static tables at EDEX start up
  - The DDL for generating the tables must be in *res/scripts* in the component
- For example, the GRIB decoder plug-in uses this mechanism to populate several static tables and to create indices
- Again, this feature is only supported in data decoder plug-ins



# Database Updates: Table Generation (cont'd)

- In general, auto-generation of tables is limited to data decoder plug-ins.
- Other components requiring table generation/population have two options:
  - Running PostgreSQL DDL scripts under control of the EDEX installer
  - Running PostgreSQL DDL scripts manually after the EDEX install
- For TO10 EDEX, the installer runs any DDL needed to create tables that can't be auto generated by EDEX
- Examples:
  - The subscription and VTEC tables are generated by the installer.



# Database Updates – Example

## Problem:

Generate the tables required for YAHW.

## Solution:

Because YAHW is not a data decoder plug-in, we will create DDL to generate the tables and use a PostgreSQL admin tool to create the database.

## To Do:

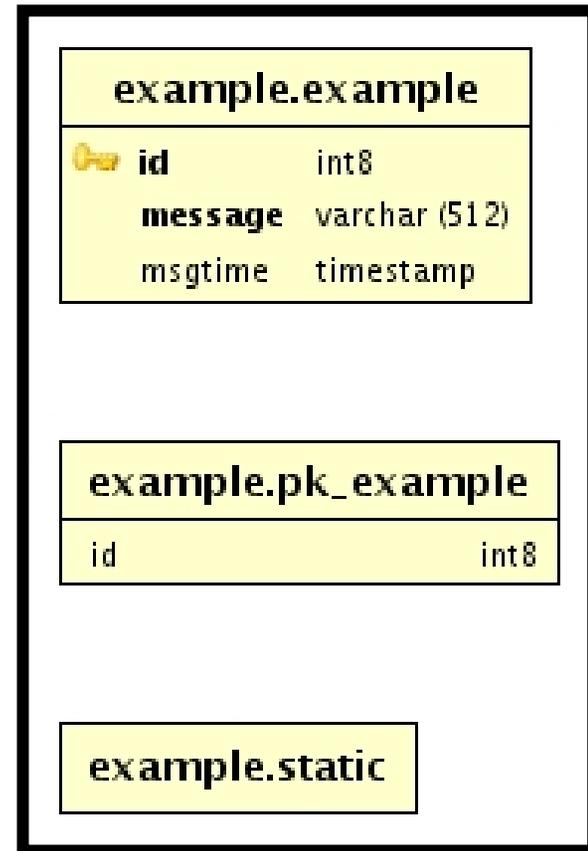
We will add additional capability to this example later.



# Database Updates – Example (cont'd)

Design the database schema:

- This diagram shows the basic structure of the schema for YAHW.
- The main data table is *example*.
  - *id* is the primary key.
  - *message* contains the message text.
  - *msgtime* contains the insert time.
- *static* is a table used internally by PostgreSQL.
- *pk\_example* is the primary key constraint for the *example* table.



# Database Updates – Example (cont'd)

Code the DDL

- The DDL shown below right is used to generate our table structure
  - In this DDL, we:
    - Create the *example* schema
    - Create the *example.static* table
    - Create the *example.example* table
    - Create the primary key on the *example.example* table
  - For both tables, we set ownership to “awips”

Hint: The DDL can be executed using a standard db client such as PG Admin.

```
create schema example authorization awips;

create table example.static();
alter table example.static owner to awips;
create or replace rule rule_static as
    on insert to example.static do also delete from example.static;

create table example.example (
    id bigint not null,
    message varchar(512) not null,
    msgtime timestamp without time zone
);

alter table example.example owner to awips;
alter table example.example add constraint PK_example primary key (id);
```



# Questions?



# JAXB/Thrift Serialization



# JAXB/Thrift Serialization

- JAXB (Java Architecture for XML Binding) is a standard Java API and tool set intended to automate the mapping between XML documents and Java Objects
  - AWIPS II uses JAXB 2.1, which works with Java 5.0 and newer
  - See <http://java.sun.com/javase/6/docs/technotes/guides/xml/jaxb/index.html> for additional information
- Thrift is an Apache Incubator Project providing a software framework for scalable cross-language services development.
  - Thrift is open standard. Developed by Facebook and open sourced in 2007, it is currently an Apache Incubator project.
  - See <http://incubator.apache.org/thrift/> for more information



# JAXB/Thrift Serialization (cont'd)

- Starting with TO3, EDEX/CAVE has used JiBX as its main serialization mechanism
- At runtime, JiBX is generally fast
- JiBX simplifies coding:
  - Eliminates the need to write code to generate XML for a class
  - Eliminates the need to write code to parse XML into a class
- Special JiBX marshaling routines do need to be written, but they can be placed in a utility (library) class as static methods



# JAXB/Thrift Serialization (cont'd)

JiBX has certain “issues” that affect AWIPS II

- JiBX supports serialization by modifying the compiled “byte code” of a Java class
  - Requires a separate, post-build pass through the system to compile the JiBX bindings
  - Makes incremental builds problematic – unable to build consistently against modified classes
- JiBX uses a separate binding file, usually binding.xml, to specify bindings
  - This is an additional file required in the plug-in project
  - Tends to be error prone as changes to Java code must be reflected a separate file
- JiBX does not work well with component-based applications
  - Both CAVE and EDEX are component (plug-in) based
  - Specifically, JiBX (out of the box) has problems with binding that bridge components
    - Special coding was required to support binding that bridge components



# JAXB/Thrift Serialization (cont'd)

## Why JAXB?

- From the Sun web site: “JAXB simplifies access to an XML document from a Java program by presenting the XML document to the program in a Java format.”
  - essentially, this means most of the program is unaware of XML
- XML binding are combined with the Java code
  - AWIPS II uses XML binding annotations to specify the bindings
  - Code and binding are in one place, no separate files to specify binding
- Binding are finalized at runtime rather than compile time
  - Binding do not require a separate build step
  - Bindings between components work well



# JAXB/Thrift Serialization (cont'd)

## Why Thrift?

- Thrift is fast and flexible
- Thrift supports “self describing” data
  - Eliminates separate descriptors and separate build steps
- Thrift is used in conjunction with JAXB
  - Generally transparent to the developer



# JAXB/Thrift Serialization (cont'd)

- Preliminary performance benchmarks
  - Test #1: Serialize and Deserialize 1000 “typical” MetarRecords
    - JiBX: 752ms Serialize / 1090ms Deserialize
    - Dynamic Thrift: 550ms Serialize / 801ms Deserialize
  - Test #2: Serialize and Deserialize Object with float[] of size of grid218.
    - JiBX: 240ms Serialize / 310ms Deserialize
    - Dynamic Thrift: 42ms Serialize / 38ms Deserialize



# JAXB/Thrift Serialization (cont'd)

## Programming Issues

- Both JAXB and Thrift require custom marshaling code
  - These have been added to the AWIPS II base line in a utility class
  - Utility methods are in `com.raytheon.uf.common.serialization.SerializationUtil`
- Serializable classes must be implemented using the Java “Bean” pattern
  - The class must have a default, no-arg constructor
  - All attributes must have accessors that follow the bean pattern



# JAXB/Thrift Serialization (cont'd)

## Serialization Annotations

### ■ Class level annotations:

- *@XmlRootElement*, *@XmlAccessorType*, *@DynamicSerialize*
  - *@XmlAccessorType* has single argument. use *XmlAccessType.NONE*

### ■ Attribute annotations

- *@XmlAttribute*, *@DynamicSerializeElement*
- use both annotations with each serializable attribute

### ■ AWIPS II includes custom serialization annotations to support combined JAXB/Thrift serialization

- *@DynamicSerialize*, *@DynamicSerializeElement*



# JAXB/Thrift Serialization – Example

## Problem:

Add JAXB serialization to the data class in YAHW. Modify the main service class to return the serialized data object on successful data insertion.

## Solution:

Add the JAXB annotations needed to make ExampleData serializable. Add a serialization to ExampleMain. Wire in any dependencies for the component.

## To Do:

We still need a client that can communicate with YAHW.



# JAXB/Thrift Serialization – Example (cont'd)

Add import statements to ExampleData:

- Open ExampleData in Eclipse.
- Add the import statements shown in bold at right.

```

package org.noaa.gov.edex.to10.example.data;

import java.util.Calendar;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlRootElement;

import com.raytheon.edex.db.objects.PersistableDataObject;
import com.raytheon.uf.common.serialization.ISerializableObject;
import com.raytheon.uf.common.serialization.annotations.DynamicSerialize;
import com.raytheon.uf.common.serialization.annotations.DynamicSerializeElement;

@Entity
@Table(name="example", schema="example")
public class ExampleData extends PersistableDataObject implements
    ISerializableObject {

```

Note: This listing shows only the new code and the existing code needed to establish its context.



# JAXB/Thrift Serialization – Example (cont'd)

Add annotations to ExampleData:

- Add the class level annotations shown below (in **bold**).
  - @XmlRootElement, @XmlAccessorType, @DynamicSerialize
- Add element level annotations as shown (in **bold**).
  - @XmlAttribute, @DynamicSerializeElement
  - add to every attribute

```

@Entity
@Table(name="example", schema="example")
@XmlRootElement
@XmlAccessorType(XmlAccessorType.NONE)
@DynamicSerialize
public class ExampleData extends PersistableDataObject implements
    SerializableObject {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue
    @XmlAttribute
    @DynamicSerializeElement
    private long id;

    @Column(length=512)
    @XmlAttribute
    @DynamicSerializeElement
    private String message;

    @Column
    @XmlAttribute
    @DynamicSerializeElement
    private Calendar msgTime;

}

```

Note: This listing shows only the new code and the existing code needed to establish its context.



# JAXB/Thrift Serialization – Example (cont'd)

Modify ExampleMain to include new capability:

- Open ExampleMain in Eclipse.
- Add the new import statements shown below (in **bold**).

```
package org.noaa.gov.edex.to10.example;  
  
import java.util.Calendar;  
  
import javax.xml.bind.JAXBException;  
  
import org.apache.commons.logging.Log;  
import org.apache.commons.logging.LogFactory;  
import org.noaa.gov.edex.to10.example.data.ExampleData;  
  
import com.raytheon.edex.db.dao.CoreDao;  
import com.raytheon.edex.db.dao.pool.DaoConfig;  
import com.raytheon.edex.exception.EdexException;  
  
import com.raytheon.uf.common.serialization.SerializationUtil;
```

Note: This listing shows only the new code and the existing code needed to establish its context.



# JAXB/Thrift Serialization – Example (cont'd)

Add a new method to manage serialization:

- Add an attribute, *exampleData*, to hold data object.
- Add a method, *serializeMessage()*, to perform serialization.

```
public class ExampleMain {  
  
    private ExampleData exampleData = null;  
  
    private String serializeMessage() throws EdexException {  
        try {  
            return SerializationUtil.marshalToXml(exampleData);  
        } catch (JAXBException e) {  
            String msg = "Encountered problems marshalling data";  
            logger.error(msg,e);  
            throw new EdexException(msg,e);  
        }  
    }  
}
```

Note: This listing shows only the new code and the existing code needed to establish its context.



# JAXB/Thrift Serialization – Example (cont'd)

Modify existing method, *saveData()* to use the *exampleData* attribute:

- Locate the *saveMessage()* method and modify the lines (in **bold**).
  - Basically, replace the existing variable, *ed*, with the new attribute, *exampleData*.

```
public class ExampleMain {  
  
    private void saveMessage(String message) throws EdexException {  
        Calendar now = Calendar.getInstance();  
        exampleData = new ExampleData();  
        exampleData.setMessage(message);  
        exampleData.setMsgTime(now);  
        DaoConfig cfg = DaoConfig.forClass(ExampleData.class);  
        CoreData dao = new CoreData(cfg);  
        try {  
            dao.saveOrUpdate(exampleData);  
        } catch (Exception e) {  
            String msg = "Unable to perform database insert";  
            logger.error(msg, e);  
            throw new EdexException(msg, e);  
        }  
    }  
}
```

Note: This listing shows only the new code and the existing code needed to establish its context.



# JAXB/Thrift Serialization – Example (cont'd)

Modify the *execute()* method to use the new capability:

- Add/modify the lines shown below (in **bold**).
  - In the try block, we call the new method.
  - In the catch, we save the exception message to return later.
  - the (*new*) finally block is used to release the exampleData object

```
public class ExampleMain {  
  
    public String execute(String message) {  
        String retVal;  
        logger.info(message);  
        try {  
            saveMessage(message);  
            retVal = serializeMessage();  
        } catch (EdexException e) {  
            retVal = e.getMessage();  
        } finally {  
            exampleData = null;  
        }  
        return retVal;  
    }  
}
```

Note: This listing shows only the new code and the existing code needed to establish its context.



# Testing the Example

## Build:

- Build and deploy EDEX
  - As previously described on slides 36 – 39

## Run:

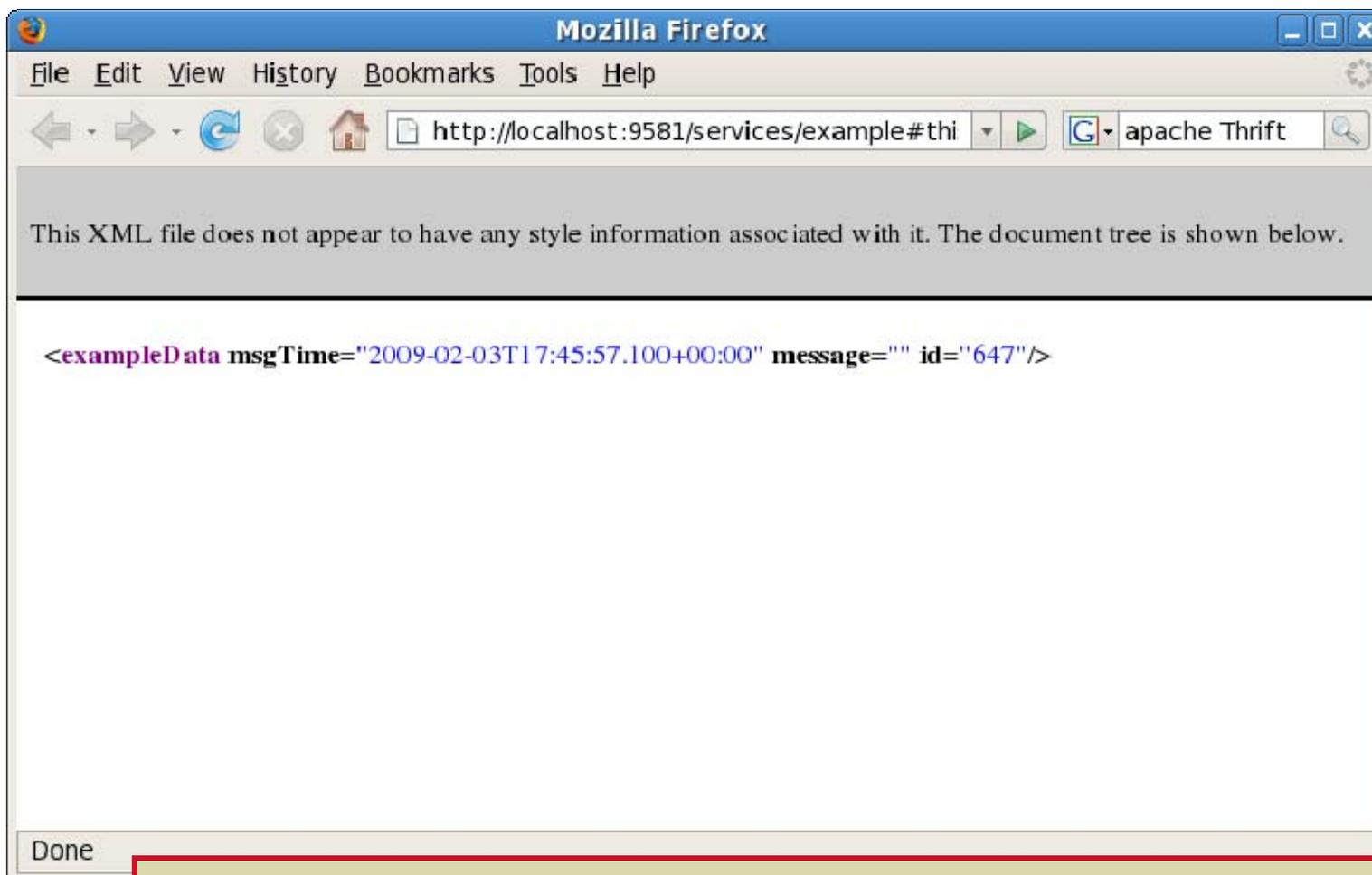
- Start PostgreSQL and EDEX
  - As previously described on slide 47

## Test:

- Open a browser (Firefox) and enter the following URL:  
`http://localhost:9581/services/example#This is a test`
- Hit the go button; if everything works correctly, the browser will display the page shown on the next slide



# Testing the Example (cont'd)



Note: This works because we modified YAHW to return an XML document rather than just the string. The message is missing because the browser is not correctly sending it to EDEX.



## Testing the Example (cont'd)

Check the log:

- Open a console window.
- Change directory to <<awips-home>>/edex/logs.
- Examine the latest log in a text viewer such as *less* or *view*.
- Near the bottom of the log, you should see an entry similar to:

```
INFO 2009-01-29 14:22:22,873 [btpool2-0] ExampleMain:
```

Note: The log entry means the message was delivered and logged. We really need is a client that can send a simple text message to EDEX.

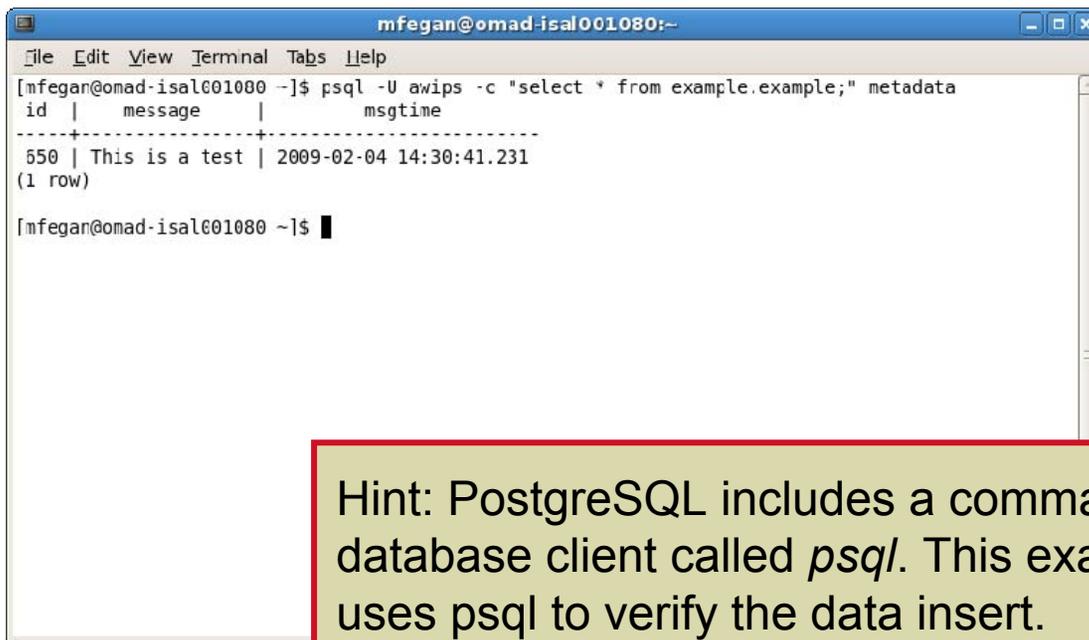


# Testing the Example (cont'd)

Check the Database:

PostgreSQL includes a command line tool for executing SQL.

- Open a console window.
- Execute the following command:  
***psql -U awips -c "select \* from example.example;" metadata***
- You should see a result similar to the screen capture below.



```
mfeagan@omad-isal001080:~  
File Edit View Terminal Tabs Help  
[mfeagan@omad-isal001080 ~]$ psql -U awips -c "select * from example.example;" metadata  
id | message | msgtime  
-----  
550 | This is a test | 2009-02-04 14:30:41.231  
(1 row)  
[mfeagan@omad-isal001080 ~]$
```

Hint: PostgreSQL includes a command line database client called *psql*. This example uses *psql* to verify the data insert.



# Questions?



# Command Line Interface

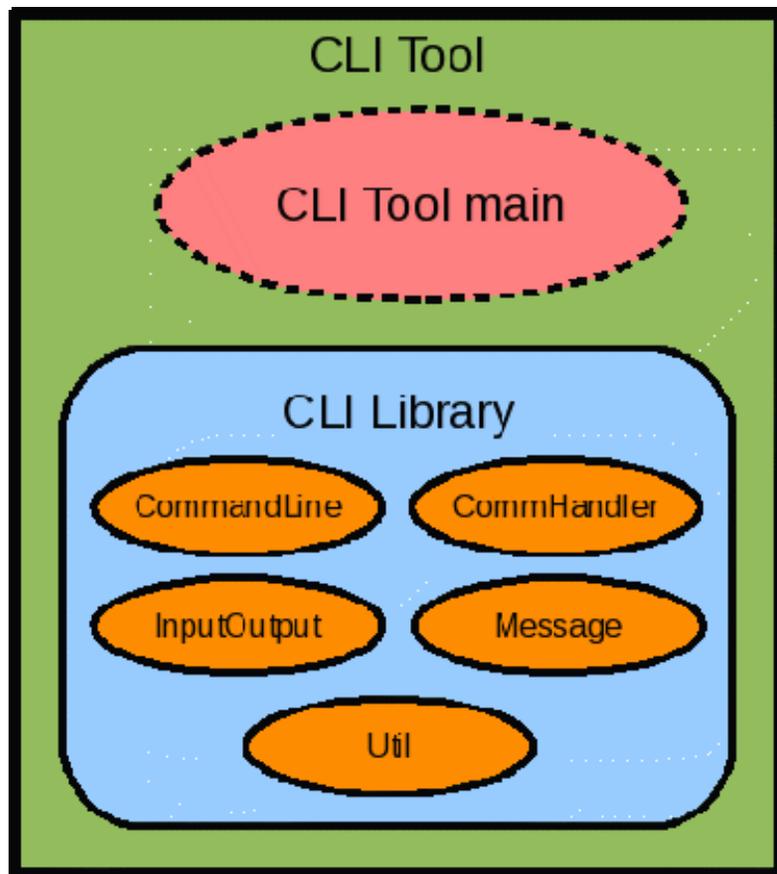


# Command Line Interface

- TO10 introduces a new AWIPS II component, the Command Line Interface (CLI) tools
  - Three CLI tools are provided:
    - **textdb.** A rehost of the existing (AWIPS I) textdb tool
      - ▶ existing textdb command line flags and responses have been preserved
    - **uengine.** A command line tool that interacts with the EDEX Product Server to run Micro Engine scripts
    - **subscription.** A command line tool that interacts with the EDEX Subscription Service to manage product subscriptions
  - CLI tools are packaged in a separate installer provided with the ADE
    - Included in the *com.raytheon.uf.tools.cli* project



# Command Line Interface: Design



- Each CLI tool follows a standard design
- The tool itself is a shell script that sets the appropriate environment and executes the tools main
  - The CLI Tool main is a Python class.
    - The class is implemented as a main
    - The action method is execute()
  - The CLI Tool main utilizes various library classes
    - Each library class is implemented in Python
    - Library classes provide reusable code



# Command Line Interface: Design (cont'd)

## CLI Library Packages

- **CommandLine.py.** Provides standard support for reading and parsing the command line
- **CommHandler.py.** Provides an HTTP communication handler
- **InputOutput.py.** Provides standard methods for reading and writing file streams
- **Message.py.** Provides standard methods for creating and decoding AWIPS II Canonical XML messages
- **Util.py.** Provides several utility methods

## Exceptions:

- All packages other than Util.py provide function-related exceptions for error propagation



# Command Line Interface: Configuration

Each CLI tool is supported by two configuration files:

- Configuration files are written as importable Python code
  - Once installed, these files generally should not be modified
- All tools use SiteConfig.py – contains site-specific information
  - Currently contains connection information for the target EDEX server
  - EDEX server DNS address is set by the installer
- Each tool has a tool specific configuration file
  - Contains data structures providing specific data



# Command Line Interface: Features

- The CLI tool set is designed to be extensible
- The CLI library is designed to promote writing of new tools
  - A new tool can use the library for common functionality
  - The new tool contains just the desired new functionality
- Existing CLI tools can be accessed by new tools
  - Example: Using Python-TK, a GUI-based interface could be added to the subscription tool without modifying the core tool code
  - Example: The textdb tool uses the subscription tool to manage (add, list, delete) AFOS PIL trigger scripts



# Command Line Interface – Example

## Problem:

Create a command line tool that provides a client for YAHW. The tool should allow the user to specify the message at the command line.

## Solution:

Following the CLI tool pattern to implement the tool. This will include a top level shell script, a main class, and tool specific configuration.

## To Do:

Configure to deploy into the CLI deployment directory.



# Command Line Interface – Example (cont'd)

Existing CLI directory structure:

- CLI tools deploy into the directory structure shown at right.
- We need to emulate parts of this structure in our project; conf and a new directory for the Python main class.

Create the CLI directory structure:

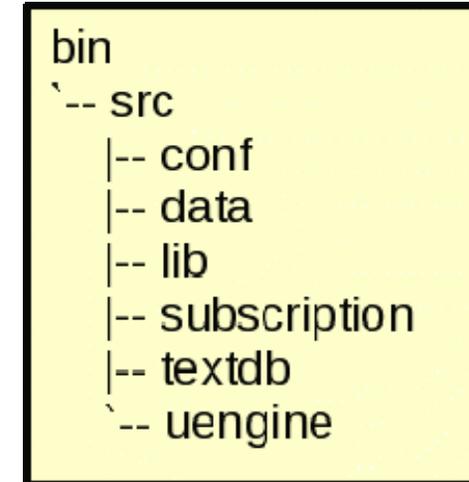
- Using Eclipse, create the following directories inside the TO 10 example project.

cli/src/conf

cli/src/example

Create the Python package files:

- Using Eclipse, create a file named `__init__.py` in each of these directories.



# Command Line Interface – Example (cont'd)

Create the top level shell script:

- Using Eclipse, create a file named *example* in the *cli/src* directory in the TO10 example project.
- After the new file opens, add the code shown below and save the file.

```
#!/bin/bash

# setup the environment needed to run the the Python
export LD_LIBRARY_PATH=${HOME}/awips/lib
export LD_PRELOAD=${HOME}/awips/lib/libpython2.5.so
export PYTHONPATH=./src:$PYTHONPATH

# execute the textdb Python module
_PYTHON="${HOME}/awips/bin/python"
_MODULE="./src/example/Example.py"

# quoting of '$@' is used to prevent command line interpretation
_PYTHON $_MODULE "$@"
```

**Note:** assumes EDEX is installed in the user's home directory

Note: We will set the permissions on this file and copy it into place later.



# Command Line Interface – Example (cont'd)

Create the Python main class:

- Using Eclipse, create a file called *Example.py* in the *cli/src/example* directory in the TO10 example project.
- Once the file opens, add the code shown at right and save the file.

Note: This example illustrates how to create a new CLI tool that leverages the existing CLI library to provide some basic functionality.

```
import sys
import httpLib

import lib.CommHandler as CH

import conf.SiteConfig as site
import conf.ExConfig as config

class Example:
    def __init__(self):
        self.connection = None
        self.service = None

    def process(self):
        status = 0
        message = ' '.join(sys.argv[1:])
        service = config.endpoint.get('example')
        ch = CH.CommHandler(site.connection, service)
        ch.process(message)
        if not ch.isGoodStatus():
            print ch.formatResponse()
            return 1
        print ch.getContents()
        return 1

if __name__ == "__main__":
    ex = Example()
    status = ex.process()
    exit(status)
```



# Command Line Interface – Example (cont'd)

Create the tool specific configuration:

- Using Eclipse, create a file called *ExConfig.py* in the *cli/src/config* directory in the TO10 example project.
- Once the file opens in Eclipse, add the following line of code  

```
endpoint = {'example': '/services/example'}
```
- Save the file.

Note: This is a rather trivial example and in fact is not required to implement the example CLI tool. It is included to illustrate a CLI tool-specific configuration. The CLI tools that shipped with TO10 have more realistic tool configurations.



# Command Line Interface – Example (cont'd)

Setting permissions on the shell script:

- Open a terminal window and change directory to the *cli* directory in the TO10 example project
- Execute: ***chmod a+x example***

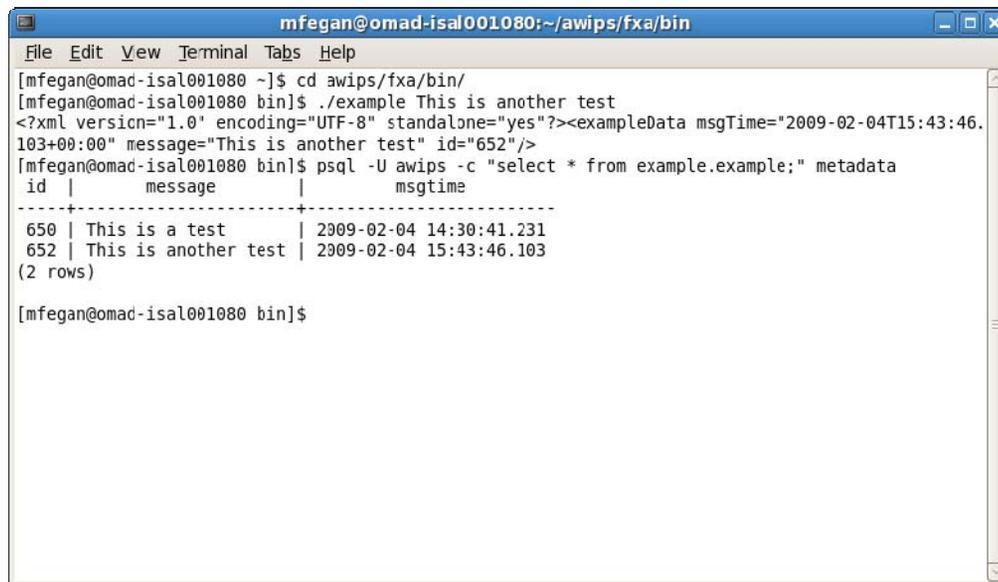
Deploying the new tool (use the same terminal)

- Execute ***cp -Rf . \${HOME}/awips/fixalbin***

Note: This procedure assumes EDEX and CLI have been installed as recommended in the ADE flow tag.



# Command Line Interface – Example (cont'd)



```

mfegan@omad-isal001080:~/awips/fxa/bin
File Edit View Terminal Tabs Help
[mfegan@omad-isal001080 ~]$ cd awips/fxa/bin/
[mfegan@omad-isal001080 bin]$ ./example This is another test
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><exampleData msgTime="2009-02-04T15:43:46.103+00:00" message="This is another test" id="652"/>
[mfegan@omad-isal001080 bin]$ psql -U awips -c "select * from example.example;" metadata
 id | message | msgtime
-----+-----+-----
 650 | This is a test | 2009-02-04 14:30:41.231
 652 | This is another test | 2009-02-04 15:43:46.103
(2 rows)

[mfegan@omad-isal001080 bin]$

```

Testing the Tool:

- Open a terminal window.
- Change directory to **`${HOME}/awips/fxa/bin`**.
- Execute **`./example This is another test`**.
- Execute **`psql -U awips -c "select * from example.example;" metadata`**.

Your terminal display will be similar the screen capture shown above



# Questions?



# Data Decoder Plug-ins



# Data Decoder Plug-ins

- To some extent, modifications to the data decoder plug-in model have already been covered.
- These include:
  - Basic component structure – covered in slides 12 – 16
  - Decoder data object considerations
    - Hibernate object/table mapping – covered in slides 77 – 79
    - Database table creation – covered in slides 91 – 93
    - Database purging – covered in slides 74 – 75
    - Data serialization – covered in slides 99 – 106
  - Camel configuration – covered in slides 49 – 59
- Several other features of data decoder plug-in components are discussed on the next few slides

Note: The existing **obs** data decoder, *com.raytheon.edex.plugin.obs*, is used to illustrate these additional features.

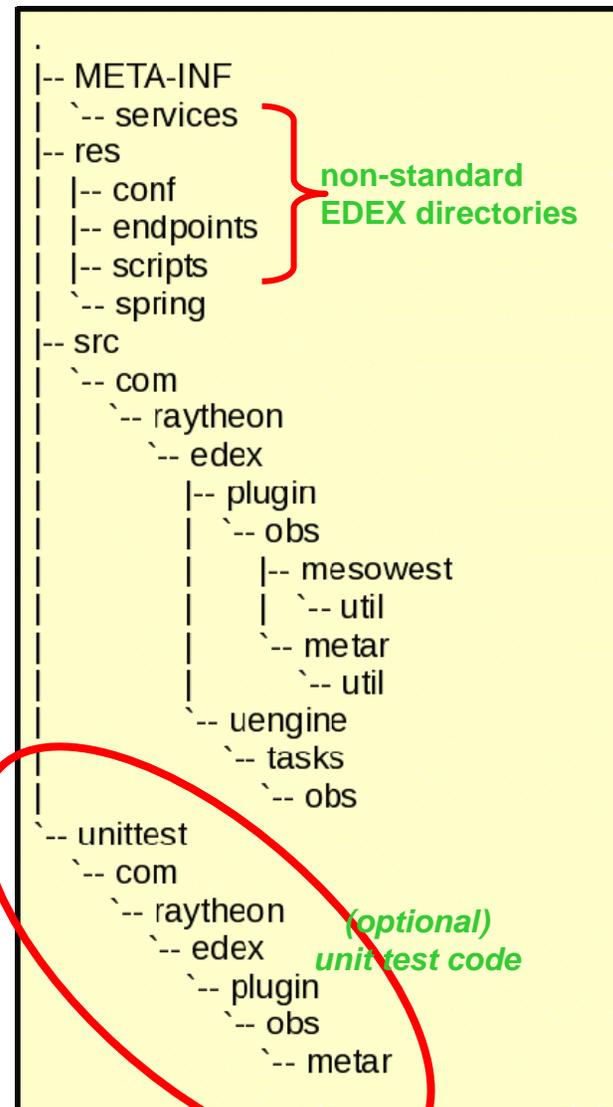


# Data Decoder Plug-ins (cont'd)

## Basic Directory Structure

- All directories under src are created as classes are created
- *META-INF/services* contains a single file which identifies files that either map to database tables or are serializable
- *res/conf* contains component configuration files
- *res/endpoints* is obsolete; it is being removed
- *res/scripts* contains PostgreSQL DDL scripts that are executed when the plug-in creates its database tables

The next few slides provide additional information on each of these directories. Each slide includes examples from the OBS data decoder plug-in.



# Data Decoder Plug-ins: META-INF/services

```
com.raytheon.edex.plugin.obs.metar.MetarRecord  
com.raytheon.edex.plugin.obs.metar.util.SkyCover  
com.raytheon.edex.plugin.obs.metar.util.WeatherCondition  
com.raytheon.edex.plugin.obs.mesowest.MesowestRecord
```

- META-INF/services contains a single file:  
com.raytheon.uf.common.serialization.ISerializableObject
- This file lists all classes in the data decoder plug-in that are either serializable or are mapped to database tables
  - The actual classes must implement ISerializableObject
  - The actual classes must use either JAXB or Hibernate annotations
- In most data decoder plug-ins, all data objects are listed here

**CRITICAL!!** This file must be created and/or updated manually as new data classes are created in the plug-in.



# Data Decoder Plug-ins: res/conf

```
<?xml version="1.0" encoding="UTF-8"?>
<properties>
  <Name>OBS</Name>
  <Plugin>true</Plugin>
  <Database>metadata</Database>
  <Record>com.raytheon.edex.plugin.obs.metar.MetarRecord</Record>
  <Decoder>com.raytheon.edex.plugin.obs.ObsDecoder</Decoder>
  <Purger>com.raytheon.edex.db.purge.DefaultPurgerImpl</Purger>
  <SEPARATOR>com.raytheon.edex.plugin.obs.ObsSeparator</SEPARATOR>
</properties>
```

- res/conf contains one or more component configuration files
  - It should include a file named *plugin.xml* that contains the basic configuration for the component
  - It may include other XML configuration files
- plugin.xml for the OBS data decoder is shown here



# Data Decoder Plug-ins: res/conf (cont'd)

```
<?xml version="1.0" encoding="UTF-8"?>
<properties>
  <Name>OBS</Name>
  <Plugin>true</Plugin>
  <Database>metadata</Database>
  <Record>com.raytheon.edex.plugin.obs.metar.MetarRecord</Record>
  <Decoder>com.raytheon.edex.plugin.obs.ObsDecoder</Decoder>
  <Purger>com.raytheon.edex.db.purge.DefaultPurgerImpl</Purger>
  <SEPARATOR>com.raytheon.edex.plugin.obs.ObsSeparator</SEPARATOR>
</properties>
```

- Tags in plugin.xml define the basic configuration of the plug-in
- Required tags
  - <Name /> identifies the configuration name for the plug-in
  - </Plugin /> identifies this component as needing auto creation of database tables
  - <Database /> identifies the database used by the plug-in
  - <Purger /> identifies the data purge class used by the plug-in
  - <Record /> identifies the data record used by the plug-in
- Optional tags
  - <Decoder /> identifies the data decode class used by the plug-in
  - <SEPARATOR /> identifies the record separator tags
  - other tags may be included to provide additional configuration for the decoder

Note: Some of the information in this file is being moved to the Camel descriptor in TO11.



# Data Decoder Plug-ins: res/scripts

res/scripts contains DDL for the plugin

- The DDL is run after the first time the tables are created by the purge server
- This example shows the contents of *obsIndices.sql* which is in the OBS data decoder plug-in
  - This DDL creates indices on the previously created tables
- Some plug-ins use this capability to populate previously created tables

obsIndices.sql

```
CREATE INDEX
sky_cover_parent_idx
ON obs_sky_cover
USING btree
(parentmetar);

CREATE INDEX weather_parent_idx
ON obs_weather
USING btree
(parentmetar);
```



# Data Decoder Plug-ins: Hibernate

- Hibernate annotations covered previously (slides 77 – 78)
- Annotations not covered in detail:
  - *@DataURI* – identifies the field as an element of the Data URI
    - The Data URI provides a unique identifier for the data, made up of the values of one or more fields from the data record
    - Has one argument, `position`, which identifies the position the data occupies in the URI
    - Example:  
`@DataURI(position=2)`  
result data URI is `/<plug-in name>/.../<data value>/...`



# Data Decoder Plug-ins: Hibernate (cont'd)

## ■ Annotations not covered in detail:

*@ManyToOne* – Identifies the field as a foreign key to another table

- No arguments, but data attribute must match class of foreign table
- Used with the *@JoinColumn* annotation

*@JoinColumn* – Identifies the column containing the foreign key value

- Two arguments: name and nullable
  - ▶ Name: identifies the name of the field to map
  - ▶ Nullable: specifies if the field can be null – generally set to false

### • Example:

```
@ManyToOne
```

```
@JoinColumn(name="parentMetar", nullable=false)
```

```
private MetarRecord parentMetar;
```



# Data Decoder Plug-ins: Hibernate (cont'd)

## ■ Annotations not covered in detail:

*@OneToMany* –Identifies the field as providing a mapping to a secondary table

- Three arguments: cascade; mappedBy; and fetch
  - ▶ cascade: identifies operations to cascade to secondary table
  - ▶ mappedBy: identifies the field of the secondary table to map
  - ▶ fetch: identifies loading strategy (EAGER or LAZY)
- Attribute must be a Set of objects representing the target table
- Example:

```
@ OneToMany( cascade=CascadeType .ALL ,  
              mappedBy="parentMetar" ,  
              fetch=FetchType .EAGER )
```

```
private Set<WeatherCondition> weatherCondition  
    = new HashSet<WeatherCondition>( );
```



# Data Decoder Plug-ins: Hibernate (cont'd)

## Additional Information

- Additional information on the annotations used for Hibernate mapping is available on line at
  - Basic persistence annotations (javax.persistence):  
[http://www.hibernate.org/hib\\_docs/ejb3-api/](http://www.hibernate.org/hib_docs/ejb3-api/)
  - Hibernate annotations (org.hibernate.annotations):  
<http://www.hibernate.org/5.html#A9>  
[http://www.hibernate.org/hib\\_docs/annotations/api/org/hibernate/annotations/package-summary.html](http://www.hibernate.org/hib_docs/annotations/api/org/hibernate/annotations/package-summary.html)



# Data Decoder Plug-ins: Naming Convention

- EDEX deployment
  - Deployment is managed by the *deploy-install.xml* ANT script in the *build.edex* project
- EDEX data decoder plug-ins
  - EDEX data decoder plug-ins are deployed in the EDEX plug-ins directory; `${AWIPS_HOME}/edex/lib/plugins`
  - By default, only projects containing both “raytheon” and “plugin” in the project directory name are deployed to the plug-in directory
- EDEX services and libraries
  - EDEX services and libraries are deployed in the EDEX services directory; `${AWIPS_HOME}/edex/lib/services`
  - By default, only projects containing “raytheon” but not “plugin” in the project directory name are deployed to the plug-in directory



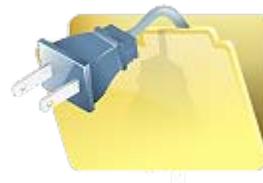
# Data Decoder Plug-ins: Naming Convention (cont'd)

## ■ EDEX support libraries

- EDEX support libraries are deployed in the EDEX dependencies directory; `${AWIPS_HOME}/edex/lib/dependencies`
- Only prepackaged JavaARchives (JAR) files are deployed to the dependencies directory
- By default, only projects containing neither “raytheon” nor “plugin” in the project directory name are deployed to the plug-in directory

## ■ Additional considerations

- As EDEX starts up, only the plug-ins and services directories are scanned for Camel deployment descriptors
- In addition, only the plug-ins directory is scanned for data decoders
- All components to be deployed must be listed as plug-in dependencies in the EDEX feature



# Data Decoder Plug-ins: Naming Convention (cont'd)

## ■ Modifying the defaults

- The *deploy-install.xml* ANT script uses a custom task to determine which component projects to deploy
  - The task is `GenerateIncludesFromFeature`, which is aliased as “includegen” in the ANT script
- includegen accepts several attributes in its XML tag, the critical ones controlling deployment are *providerfilter* and *pluginfilter*
  - **providerfilter**. A pipe separated list of names of component providers
    - ▶ Components containing one of these names are deployed to either the plug-ins or services directory
    - ▶ If specified, “raytheon” must be one of the names
  - **pluginfilter**. A pipe separated list of words that identify plugins
    - ▶ If specified, “plugin” must be one of the words

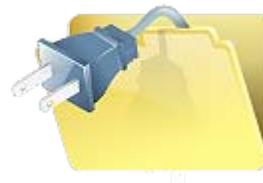


# Data Decoder Plug-ins: Naming Convention (cont'd)

## ■ Recommendation

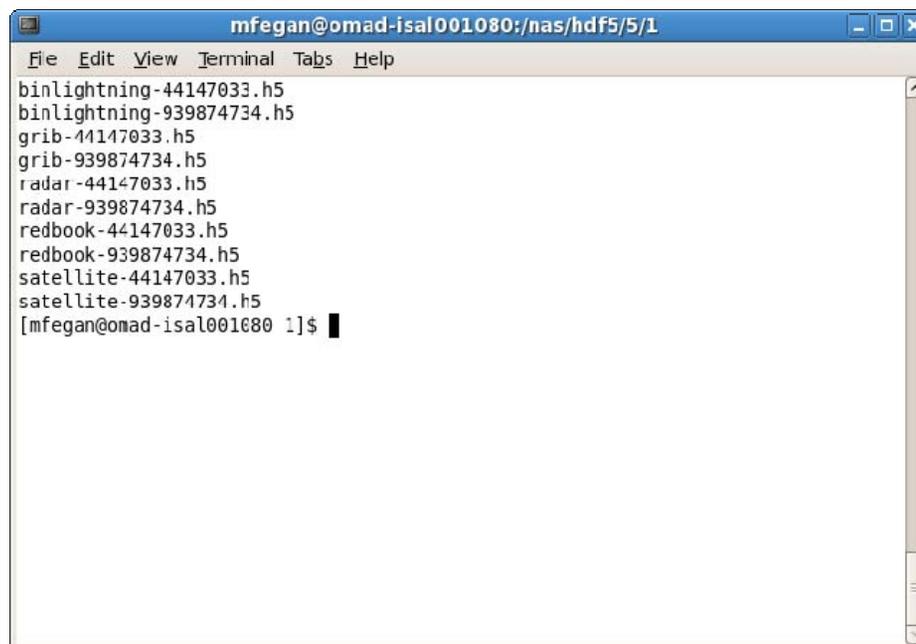
- Name new components following the domain-based naming convention
  - example: `com.raytheon.edex. ...`
  - example: `gov.nws.noaa.edex. ...`
- For data decoder based components include “plugin” in the name
- Modify the *includegen* task in *deploy.install.xml* to include both “raytheon” and the appropriate provider name
  - Example: `providerfilter="raytheon|noaa"`

Note: For a comprehensive example on naming components, see slides 17 – 39.



# Data Decoder Plug-ins: Data Storage

- HDF5 storage of BLOB (binary) data has been modified to improve storage and access efficiency
- A separate file is maintained for each server decoding the data
  - Naming scheme: {data type}-{server code}.h5
  - Server code is the hash code of the server's name

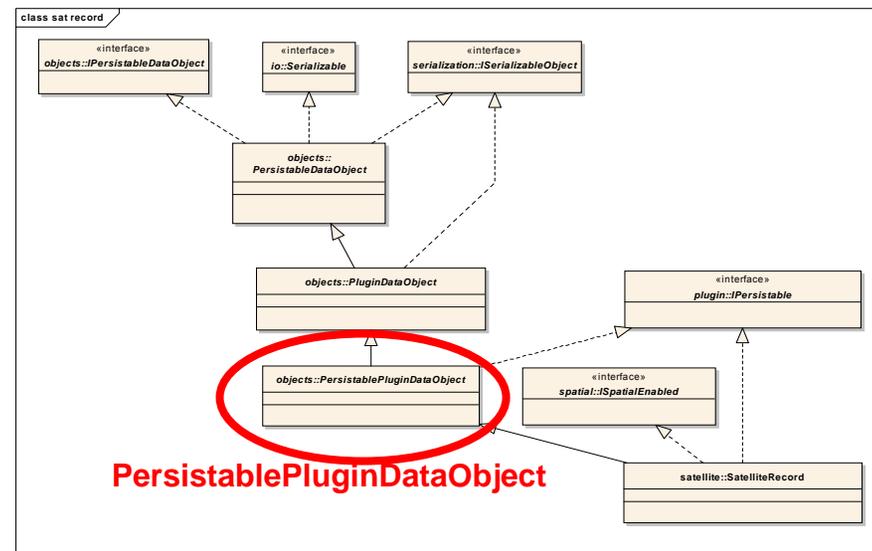


```
mfeagan@omad-isal001080:/nas/hdf5/5/1
File Edit View Terminal Tabs Help
binlightning-44147033.h5
binlightning-939874734.h5
grib-44147033.h5
grib-939874734.h5
radar-44147033.h5
radar-939874734.h5
redbook-44147033.h5
redbook-939874734.h5
satellite-44147033.h5
satellite-939874734.h5
[mfeagan@omad-isal001080 1]$
```



# Data Decoder Plug-ins

- The server code value is included in the data type's metadata table
  - Defined in `PersistablePluginDataObject`
  - Value is set by the HDF5 DAO; decoder code does not set this value
- All HDF5 data access (read/write) should go through the HDF5 Dao



# Data Decoder Plug-ins: Camel Wiring

- Each data decoder represents an ingest path
- There are four main components to wiring the component
  - Specify decoder bean(s)
  - Register the threading router
  - Define a file sniffing route
  - Define the ingest route
- Standard routes are configured to assist in the wiring
  - **persistIndexAlert**. Used for BLOB data; persists to HDF5, inserts into the database, alerts clients of data receipt
  - **indexAlert**. Used for text data; inserts into the database, alerts clients of data receipt



# Data Decoder Plug-ins: Camel Wiring (cont'd)

Basic document structure:

- Basic structure of the Camel descriptor is shown at right
- Each endpoint should provide a unique name (id) for its Camel Context
- File names must be unique
- Multiple beans may be declared
- Multiple routes may be declared

```

<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:amq="http://activemq.apache.org/schema/core"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
    http://activemq.apache.org/schema/core
    http://activemq.apache.org/schema/core/activemq-core.xsd
    http://activemq.apache.org/camel/schema/spring
    http://activemq.apache.org/camel/schema/spring/camel-spring.xsd">
  <bean id="<<unique Bean ID>>" class="<<implementing class>>">
    <property name="<<Property Name>>" value="<<property Value />
  </bean>
  <bean id="<<unique Bean ID" class="<<implementing class"/>

  <camelContext id="unique context ID"
    xmlns="http://activemq.apache.org/camel/schema/spring"
    errorHandlerRef="errorHandler">

    <route id="<<unique route ID">
      <!-- route implementation -->
    </route>
  </camelContext>
</beans>

```

Note: The next few slides examine the Camel descriptor for the satellite data decoder endpoint. Other decoder endpoints are similar.



# Data Decoder Plug-ins: Camel Wiring (cont'd)

```
<bean id="satDecoder"  
      class="com.raytheon.edex.plugin.satellite.SatelliteDecoder" />
```

Specify decoder beans:

- Define the beans that will be called in one of the endpoint routes
- The satellite decoder uses a single decoder bean



# Data Decoder Plug-ins: Camel Wiring (cont'd)

```
<bean id="satRegistry"  
      class=" com.raytheon.uf.edex.esb.camel.BasicThreadPoolRouter"  
      factory-method="getInstance">  
  <constructor-arg>  
    <value>Generic</value>  
  </constructor-arg>  
  <constructor-arg>  
    <value>satellite</value>  
  </constructor-arg>  
  <constructor-arg>  
    <value>directvm:satelliteIngest</value>  
  </constructor-arg>  
</bean>
```

## Register threading router

- This snippet is similar for most data decoders
- Constructor arguments are passed to the getInstance() method
  - 1<sup>st</sup> argument identifies the pool to use (Generic, AlphaNumeric or BUFR)
  - 2<sup>nd</sup> argument identifies the plugin
  - 3<sup>rd</sup> argument provides the route name



# Data Decoder Plug-ins: Camel Wiring (cont'd)

```
<endpoint id="satFileEndpoint"
          uri="fileedex:${edex.home}/data/sbn/sat?noop=true" />

<route id="satFileConsumerRoute">
  <from ref="satFileEndpoint" />
  <bean ref="fileToString" />
  <setHeader headerName="pluginName">
    <constant>satellite</constant>
  </setHeader>
  <to uri="ingest-activemq:queue:Ingest.Generic" />
</route>
```

Define a file sniffing route

- Replaces the TO9 staging service
- The endpoint tag defines pickup location for the data
  - Environment replace (e.g., \${edex.home}) may be used here
- The “fileToString” bean moves the file to the processing directory and passes along the new file location
- The file name is passed to the appropriate ingest router



# Data Decoder Plug-ins: Camel Wiring (cont'd)

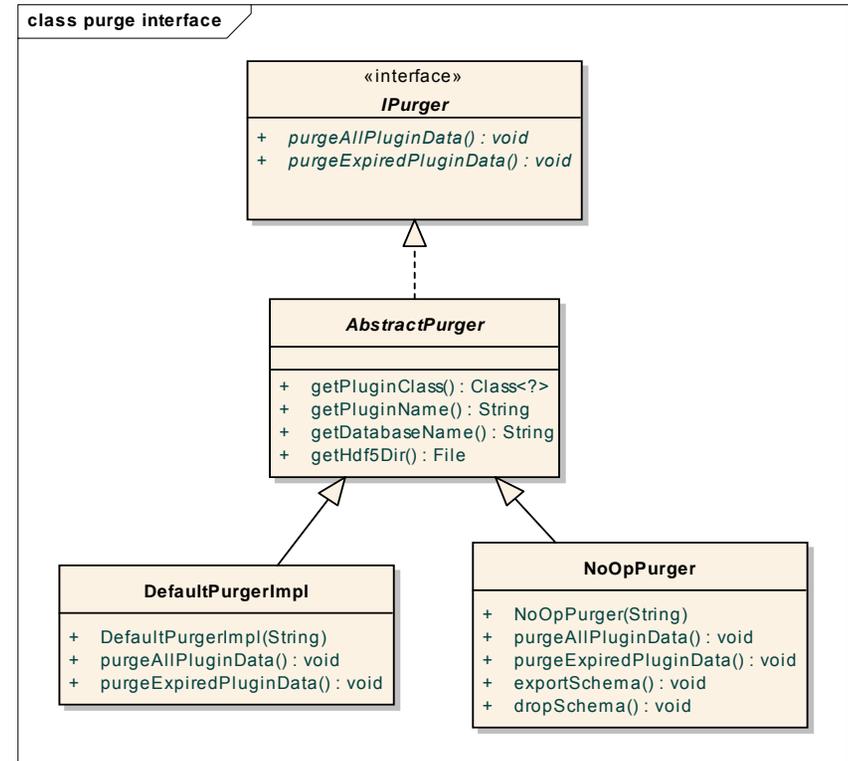
```
<route id="satIngestRoute">
  <from uri="directvm:satelliteIngest" />
  <try>
    <pipeline>
      <bean ref="stringToFile" />
      <bean ref="satDecoder" method="decode" />
      <to uri="directvm:persistIndexAlert" />
    </pipeline>
    <catch>
      <exception>java.lang.Throwable</exception>
      <to uri="log:sat?level=ERROR&showBody=false" />
    </catch>
  </try>
  <bean ref="processUtil" method="delete" />
</route>
```

- Define the data ingest route
  - Defines the decoding process
  - Receives a file name form the ingest router
  - The “stringToFile” bean converts the file name to a File object
  - The “satDecoder” bean decodes the file
    - Note that Camel automatically reads the file and passes it to the decoder
  - After decoding, the image is passed to the persistence component
  - The last action performed is to delete the original file
- Note the <try /> and nested <catch /> tags that implement basic error handling



# Data Decoder Plug-ins: Data Purging

- Data purging is currently limited to components deployed the EDEX plug-ins directory (slides 74 & 75)
  - Data purging is declared in the component's *plugin.xml* file
  - Data decoders requiring custom purging
    - Provide a purger that extends AbstractPurger (or an existing purger such as DefaultPurgerImpl)
    - Overrides the appropriate methods



# Data Decoder Plug-ins: Data Purging – Example

## Problem:

Create a custom purger that deletes data older than 12 hours rather than the 24 hours provided by the default purger.

## Solution:

Extend the default purger and override the *initializePlugin()* method (this is where the retention time is set).



# Data Decoder Plug-ins: Data Purging – Example (cont'd)

Implementation requires two methods:

- The constructor simply delegates to the base class
- InitializePlugin():
  - First delegates to the base class
  - Creates a new PluginVersion object
    - Represents the database table that tracks purge information
  - Updates the database

```
package org.noaa.gov.edex.to10.example.data;

import com.raytheon.edex.db.objects.supporting.PluginVersion;
import com.raytheon.edex.db.purge.DefaultPurgerImpl;
import com.raytheon.edex.exception.PluginException;
import com.raytheon.edex.plugin.PluginFactory;

public class ExamplePurger extends DefaultPurgerImpl {

    public ExamplePurger(String pluginName) throws PluginException {
        super(pluginName);
    }

    @Override
    protected void initializePlugin() throws PluginException {
        super.initializePlugin();
        String tableName = PluginFactory.getInstance().getPrimaryTable(
            pluginName);
        String database = PluginFactory.getInstance().getDatabase(
            pluginName);
        PluginVersion pv = new PluginVersion(pluginName, true, 12,
            tableName, database, 1.0f);
        pluginVersionDao.saveOrUpdate(pv);
    }
}
```



# Questions?



# Service Endpoints



# Service Endpoints

- Most changes/improvements relating to EDEX service endpoints have been covered already
  - The example in slides 17 – 39 generally covers component structure, additional files, deployment issues, etc., of an EDEX service endpoint
  - Slides 49 – 58 and the example on slides 59 – 69 cover Camel configuration and wiring an endpoint into EDEX
  - Slides 77, 78, and 140 – 143 cover Hibernate topics, and the example on slides 79 – 89 covers adding database persistence to an EDEX service



# Service Endpoints

## Camel Usage

- All services have been refactored as Camel endpoints

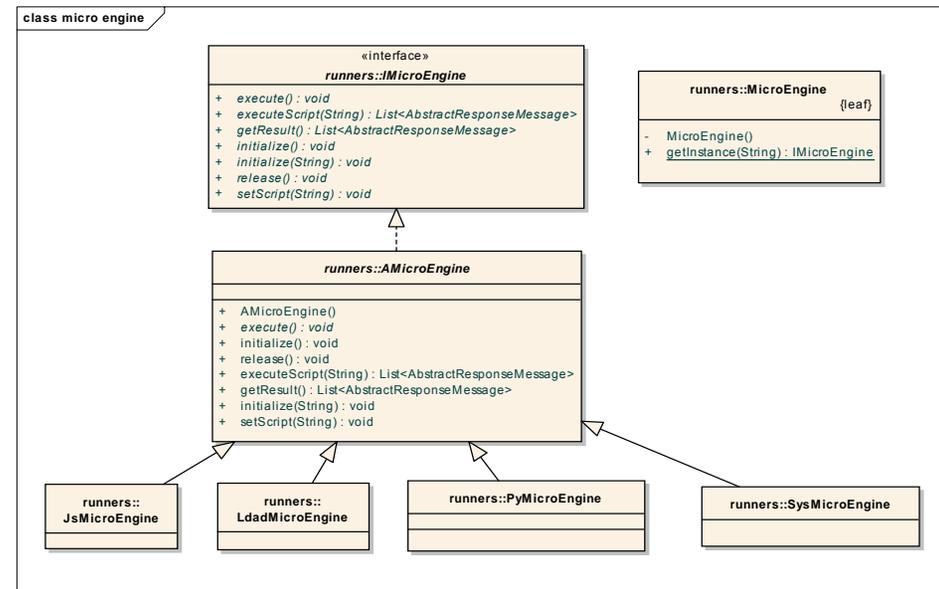
## Notable New Services

- TextDBSrv implements the back end of the of the textdb tool
- AutoBldSrv has be revamped to implement basic subscription management and script running capabilities
- ProductSrv no longer supports JavaScript micro engine scripts



# MicroEngine

- MicroEngine has been refactored somewhat
  - MicroEngine is now a factory class that generates script runners
    - Has a single method, getInstance(...) to get the runner
  - MicroEngine script runners are now pluggable
    - Script runners for Python, JavaScript, system calls are provided
    - The script runner for text database triggers is being worked in TO 11
  - Basic class design is shown at right
    - Note that new script runners can be created
  - Underlying scripting concepts have not been changed



Note: MicroEngine provides safe implementations of all methods except `execute()`.



# Questions?



# Wrap-Up



# Summary

- Covered EDEX platform updates
- Covered EDEX Code Reorganization
- Covered move to Camel as the Integration Framework
- Covered database improvements
- Covered data serialization improvements
- Covered new Command Line Interface tools package
- Covered improvements to data decoder plug-ins
- Covered improvements to EDEX service endpoints



# Resources

- On the ADE TO9 DVD
  - Current code available for examination in the ADE baseline
  - JavaDoc documentation available
- Also available
  - TO9 Training updates
  - TO-T1 Training materials

