

EDEX Data Plug-in Development for ADE 1.0 Addendum

AWIPS-II ADE Plug-in Development Guide Addendum

Abstract

This guide provides additional steps to assist a new ADE1.0 Software Developer in implementing an edex server data plug-in for AWIPS-II. The instructions are an addendum to the AWIPS-II Training Material contained in the document “awp.trg.swctrto6.ade-cave_all.pdf”. This document will focus on the National Weather Service Software Developer, but the material is relevant to ADE Software Developers in other organizations.

Revision History

revision	description	author	date
0.1	Initial, additional steps for ade1.0 development	Tom Kretz	10/30/07

EDEX Data Plug-in Development for ADE 1.0 Addendum

The following are additional steps for creating an edex data Plug-in using the Plug-in Creator Tool for ADE1.0 in AWIPS-II.

The following procedure assumes that you are using the instructions to create a Plug-in beginning on page 146 of the ADE1.0 training document “awp.trg.swctrto6.ade-cave_all.pdf”. Also, you should be familiar with the material in the Plug-in Architecture section of the training document beginning on page 108 and going through page 164 to implement an edex server data Plug-in, and with the Eclipse Java development environment.

Follow the steps on page 148 of the training document to install the Plug-in Creator tool. You may need to increase the display resolution to at least 1024 x 768 to see all of the Plug-in Creator tool editor buttons.

After installing and opening the Plug-in Creator in Eclipse following the steps on pages 148 and 149, you are ready to create a Plug-in.

Starting on page 150 of the ADE training document, you can follow the training instructions to create a Plug-in with the addition of the below exceptions to the procedure.

For NWS developed Plug-ins use the following parameters on page 150:

- 1 Set the “**Organization Domain**” field to “gov”
- 2 Set the “**Organization Name**” field to “noaa”
- 3 The “**Writer Necessary**” check box is selected for data being stored in the HDF5 repository and creates a *Plug-in_nameWriter* class. You are required to add the data mapping and storage code including any import statements
- 4 The “**Separator Necessary**” check box is selected for multi-record input files/messages and creates a *Plug-in_nameSeparator* class. You are required to add the data mapping code including any import statements
- 5 The Field Name and Type represent the data elements for the product contained in the *Plug-in_nameRecord* class and stored in Postgres and HDF5 repositories
- 6 The “Data URI” check box is selected for fields contained in the unique identifier, so make sure that the selected fields create a unique URI parameter
- 7 Set the “**Output Directory**” field to “.../edex-cave-src/edex/extensions”; see page 153
- 8 After the Plug-in is generated (page 155), edit the configuration files as follows:
 - 8.1 Remove any quotes on the end of the class name for the Decoder, Writer, and Separator in the conf/plugin.xml file. The xml elements and class names should be similar to the following:

```

<properties>
  <Name>Plug-in_name</Name>
  <Plug-in>true</Plug-in>
  <Decoder>com.raytheon.edex.plugin.radar.Plug-in_nameDecoder</Decoder>
  <SEPARATOR>com.raytheon.edex.plugin.radar.Plug-in_nameSeparator</SEPARATOR>
  <Writer>com.raytheon.edex.plugin.radar.Plug-in_nameWriter</Writer>
</properties>

```

8.1.1 If you did not select the Writer Necessary and/or Separator Necessary check boxes in the Plug-in Creator, then the Writer and/or Separator will appear as follows:

```

<SEPARATOR>com.raytheon.edex.plugin.RecordSeparatorImpl</SEPARATOR>
<Writer>com.raytheon.edex.plugin.RecordWriterImpl</Writer>

```

8.2 Add the following xml elements to the **binding.xml** file:

```

<value name="pluginName" field="pluginName" usage="optional" />
<value name="dataURI" field="dataURI" usage="optional" />

```

8.3 Add the following xml elements to the *plugin_name.db.xml* file:

```

<columnDefinition>
  <name>datauri</name>
  <columnType>varchar</columnType>
  <constraintType>PRIMARY KEY</constraintType>
  <precision>256</precision>
  <scale>0</scale>
  <dataURI>>false</dataURI>
</columnDefinition>

```

```

<columnDefinition>
  <name>insert_time</name>
  <columnType>timestamp with time zone</columnType>
  <constraintType>none</constraintType>
  <defaultValue>now()</defaultValue>
  <dataURI>>false</dataURI>
</columnDefinition>

```

8.3.1 If a datauri xml element exists, then replace it with the above datauri element.
If the key element exists, then delete the key xml element.

8.4 Add the following xml elements to the *plugin_name.hbm.xml* file:

```

<id name="dataURI" column="datauri" type="java.lang.String" />
<property name="insertTime" column="insert_time" type="java.util.Calendar"
update="false" insert="false" />

```

8.4.1 If the id xml element named key exists, then delete the key id element.

- 9 The Plug-in name has to be added to “**deployment.properties**” file in the “**.../edex-cave-src/edex/build/deployments**” directory. The following shows adding the message Plug-in:

```
comp=ingestSrv,productSrv,uEngine,AdapterSrv,notifySrv,AutoBldSrv,utilitySrv,plugin-grib,plugin-satellite,plugin-radar,plugin-message
install.dir=/opt/ade1.0/edex
```

- 10 In the *Plug-in_name*Decoder class decode() method, the Plug-in name and the cacheID have to be set, and add import statement “**import org.mule.util.UUID;**” as follows:

```
import org.mule.util.UUID;

Class {
    public AbstractDataRecord decode() throws DecoderException {

        Plug-in_nameRecord record = null;

        try {
            record = new Plug-in_nameRecord();

            //set the cache identifier, so the data can be
            //retrieved by other services

            record.setCacheIdentifier(UUID.getUUID());

            // Plug-inName is in the AbstractMessageDecoder class

            record.setPlug-inName(Plug-inName);

        } catch (Exception e) {
            throw new DecoderException("Unable to decode message", e);
        }
        return record;
    } // end of decode mehtod

} // end of class
```

- 11 In the *Plug-in_name*Record class verify that all variable names and references are correct. The code generated by the Plug-in Creator tool creates all lower case variable names in the getter and setter methods.
- 12 Continue with the ADE instructions on page 156 “Adding Plug-in to EDEX Ingest” section.

For ascii type products where you did not select a writer component and you are using the “com.raytheon.edex.plugin.RecordWriterImpl” class, you should configure the outbound-router xml element to use the “vm://indexVMQueue” endpoint. The product data is stored in Postgres.

For binary type products where you created a writer class, you need to configure the outbound-router xml element to use the “vm://persistVMQueue” endpoint. The product data is stored in the HDF5 repository.