Disaster Management Program
Open Platform for Emergency Networks

Instructions for Using the Emergency Data Exchange Language (EDXL) Distribution Element (DE) V1.0 Interface on the Open Platform for Emergency Networks (OPEN)

Draft as of November 6, 2008

Version 0.3

Revision History

Date	Version	Description	Author
August 25, 2006	0.1	Initial draft	Gary Ham
July 20, 2007	0.2	POC Updates	Gary Ham
Nov 6, 2008	0.3	POC updates and updates related to new FEMA Program Management	Gary Ham

Table of Contents

1. Introduction

The Disaster Management Open Platform for Emergency Networks (DM-OPEN) provides interoperability interfaces for sharing of alerts, situation reports, common operational picture snapshots, and other emergency related information. These interfaces provide data structures and rules of operations designed to enable information sharing between diverse systems, both commercial and government. In general, these interfaces conform to open messaging standards as defined through the Emergency Management Technical Committee sponsored by the OASIS standards organization and through the National Information Exchange Model (NIEM).

1.1. Purpose

This document is designed to help programmers and system designers to understand and use OPEN web service interfaces. This is primarily a technical "how to" document which will be updated whenever new releases are made within scope.

1.2. Scope

This document covers the interface that supports cross system information exchange using the Emergency Data Exchange Language (EDXL) Distribution Element (DE) Version 1.0, and differences between EDXL DE and CAP 1.1. Other interfaces are, or will be, covered separately.

1.3. Relationship to Other Documents

This is one of a set of documents that will describe connections to different web services interfaces. There will be a separate document written to cover each OPEN web service that is separately defined using different Web Service Definition Language (WSDL).

1.4. References

The following documents were used to obtain source information or are referenced in this document:

- *OASIS Emergency Data Exchange Language (EDXL) Distribution Element, v1.0, OASIS Standard EDXL-DE v1.0, 1 May 2006*

2. Before You Begin

[Note: If you have not connected to OPEN in the past, please email the DM-Open Interoperability Coordinator at OPEN@eyestreet.com) for an FAQ that explains the basics of OPEN. The FAQ will explain how to become approved for OPEN connection and where to find information on other OPEN offerings as well as this one.]

Location of WSDL for generating needed client side classes:

http://interopdev.cmiservices.org/axis/services/EDXLDistributionElement?wsdl

These instructions are Java specific information. If you are a .NET programmer, or use another language capable of consuming WSDL, please use the above WSDL as appropriate, following your normal methods for building a web service client. Read what follows below to understand the methods that should be available to you and parameters that you will need to employ.

This instruction also assumes that you understand the structure and usage of the EDXL Distribution Element message structure. For further reference please see the OASIS standard available from:

http://docs.oasis-open.org/emergency/edxl-de/v1.0/EDXL-DE_Spec_v1.0.pdf

3. Generating Service Classes and Beans from the WSDL

Using the Axis libraries, run the WSDL2Java tool on the WSDL to generate the 4 client classes.

```
java org.apache.axis.wsdl.WSDL2Java
    -v http://interopdev.cmiservices.org/axis/services/EDXLDistributionElement?wsdl
```

Running the above WSD2Java call will generate 4 client classes in package org.dmi_services:

```
EDXLDistributionElement.java
EDXLDistributionElementService.java
EDXLDistributionElementServiceLocator.java
EDXLDistributionElementSoapBindingStub.java
```

With these classes you can retrieve and post EDXL-DE V1.0 messages by executing following instructions:

4. Connecting to the Web Service

Use the following Code snippet to establish the connection (Note: This is code extracted from the available demo code. It was taken out of a larger class and is not intended to be a complete functioning class on its own.) :

```
// needed imports
import java.net.*;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import org.dmi_services.EDXLDistributionElementService;
import org.dmi_services.EDXLDistributionElementServiceLocator;
import org.dmi_services.EDXLDistributionElementSoapBindingStub;


// beginning of code snippet
EDXLDistributionElementService service = new
EDXLDistributionElementServiceLocator();
    URL svcURL;
    EDXLDistributionElementSoapBindingStub de;

        try {
            // set your end point
            svcURL = new URL
("http://interopdev.cmiservices.org/axis/services/EDXLDistributionElement");
            System.out.println("Invoking service at " + svcURL.toString());
            de =(EDXLDistributionElementSoapBindingStub)
service.getEDXLDistributionElement(svcURL);
            //set cog id and user id
            de.setUsername(cogId+"/"+userId);
            //set your password
            de.setPassword(password);
           } catch (MalformedURLException e) {
              // TODO Auto-generated catch block
            e.printStackTrace();
           } catch (ServiceException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

5. Brief Explanations of Service Methods

Including the getSeverTime() method, you now have access to six EDXL Distribution Element service methods:

a. To retrieve the current OPEN server time in Calendar format:

```
String getServerTime()throws java.rmi.RemoteException
```

Retrieves the current time on the server as an ISO 8601 compliant String; useful for time-synching to ensure that use of methods below based on posted date are handled correctly. This method is also useful for verifying that the service is up and running and that your user id and password have been successfully authenticated.

b. To post an EDXL DE message:

```
String postEdxlDEMessage(long[] cogIds, String EDXLDEmessage)
throws java.rmi.RemoteException
```

This method posts a DE for retrieval by one or more COGs as identified by their CogId. CogIds is an array of long where each long is the identifier of an OPEN or DMIS Collaborative Operations Group (COG). If your are posting from the Interoperability COG (1737) you will only be able to post to your COG and to two other COGs (2 – a second interoperability COG, and 1000 – the DMIS project Team COG). This allows testing and development without disrupting operational DMIS COGS. Once a system is used operationally at a customer (or custom production) site, you will have the ability to post to more than 2000 COGS in the DM OPEN environment.

Within DM-OPEN the EDXL message is treated as a simple (often very large) string. The string must validate against the EDXL DE schema (.xsd file), otherwise an exception may be returned. The interface does not validate content elements beyond the DE schema. Connectors at both ends are relied upon to handle any content processing that is required. The OPEN interfaces are "content agnostic."

c. To retrieve a particular EDXLDE message with known identifier and sender from the OPEN interface:

```
String getEdxlDEMessage(String identifierAndSender) throws
java.rmi.RemoteException
```

This method returns a specific EDXL DE message based upon the combined value of the <identifier> and <sender> tags. The String argument is space delimited 2 value string of identifier and sender (e.g., "DM001567" + " " + "hamg@battelle.org"). Remember the EDXL DE spec requires that identifier and sender contain no spaces. A space delimiter should therefore work for retrieval of valid EDXL DE messages.

d. To retrieve all DE messages posted to your COG:

```
String[] getAllEdxlDEMsgs(long cogId) throws
java.rmi.RemoteException;
```

This method returns an array Strings representing all EDXL Messages posted to the COG represented by the cogId parameter. Each string is an EDXLDEMessage in its entirety. The EDXL message is complete and unformatted (no spaces or carriage returns) other than its tags and content. It should be in perfect condition parsing using any appropriate XML parsing method (e.g., Xpath query, SAX, or DOM). The current interface does not yet offer querying capability based upon content other than the value of the <sent> tag (see below). Additional query capability will be gradually introduced in later interfaces and in production. In the meantime, connecting systems will need to filter duplicates and unwanted items from the returned array using Xpath or similar techniques.

e. To retrieve all EDXL-DE messages posted to your COG later than the date and time specified by the DateTime parameter value:

```
String[] getEdxlDEMsgsByPostedDate(String postDateTime) throws
java.rmi.RemoteException;
```

The getEdxlMsgByPostedDate method returns an array of Strings where each string is a complete EDXL DE message with its content. It returns all messages that were posted to your COG after the time represented by the postDateTime parameter. Note that the postDateTime parameter is a string in ISO8601 format (e.g., "YYYY-MM-DDTHH:MM:SS").

This method allows retrieving systems to choose not to retrieve older messages, making the retrieval process somewhat less onerous. It is designed to work in tandem with getServerTime() as a way of getting only relevant postings. Because perfect synchronization of servers may be difficult across the distributed architecture, user systems may still want to allow some overlap in time from most recent retrieval when choosing the 8601 value for subsequent retrievals. Of course, this also means duplicate message retrievals should be checked for and discarded as appropriate.

f. **To retrieve all EDXL-DE messages posted with a <sent> field value later than the date and time specified by the DateTime parameter value::**

```
String[] getEdxlDEMsgsBySentDate(String sentDateTime) throws
      java.rmi.RemoteException;
```

The getEdxlMsgBySentDate method returns an array of Strings where each string is a complete EDXL DE message with its content. It returns all messages that were posted to your COG where the value of the <sent> tag is later than the time represented by the sentDateTime parameter. Note that the sentDateTime parameter (like the value of the <sent> tag) is a string in ISO8601 format (e.g., "YYYY-MM-DDTHH:MM:SS").

User systems should allow some overlap in time from most recent retrieval when choosing the <sent> value for subsequent retrievals. This overlap is needed because relevant messages can be posted to OPEN at some time after the original message creation time and OPEN does not alter the originating <sent>tag value, regardless of the actual post time. Of course, this also means duplicate message retrievals should be checked for and discarded as appropriate.

6. Interface differences between the EDXL DE and CAP1.1 as implemented in DM-OPEN

While the basic method of connectivity is the same for all DM-OPEN interfaces, there are some functional differences between the CAP 1.1 interface and the EDXL DE interface which should be understood:

a. The CAP1.1 interface is an object-oriented interface that uses SOAP encoded Alert objects. While Alert objects are more efficient in some ways, .NET clients may encounter a "jagged arrays" problem with their retrieval paradigm. A separate HTTP retrieval protocol was built for this purpose. The EDXL DE interface returns an array of String and is fully compatible with .NET clients using Microsoft's client generation tools for consumption of its WSDL.

b. CAP1.1 uses an object as its date and time retrievals argument. The .NET DateTime class does not store time zone in the same fashion as the Java Calendar class. To assure

consistency across applications, the EDXL DE implementation uses a specifically formatted ISO8601 string (the same format as specified in the CAP and EDXL standards for data inside the message) for its parameters and does all appropriate calculations internally.

c. The CAP 1.1 interface does not allow a COG to send an alert to itself but provides the ability to retrieve alerts it sent out as well as alerts posted to it in a variety of ways. The DE interface allows a DE to be self-posted, but only retrieves messages that have been posted to the login COG.

d. The CAP 1.1 posts to an Array of SimpleCOG object and provides separate retrieval access to allowed COGs. The DE is posted to an array of cogid, which must be known in advance. The cogids used have the same values as found in corresponding SimpleCog objects. DE builders needing dynamic access will need to access methods on the CAP1.1 or Tactical Information Exchange (TIE) interfaces to access SimpleCOG interface.

e. The string for retrieving a specific alert in CAP is the <identifier> and <sender> delimited by a colon. The string for retrieving a specific DE instance is <identifier> and <sender> delimited by a space.

7. More Examples and Contact Info

Sample code is available. E-mail the OPEN Systems interoperability Coordinator at OPEN@eyestreet.com for some WSDL generated classes, appropriate Axis libraries, and some test drive code to get you started. This e-mail address is also your first point-of-contact for any problems related to programming issues related to OPEN.

8. Coming Attractions

a. Content specific and/or tag specific query capability. This requirement has not yet begun formal analysis. User input is welcome.
b. COG Profile – More complete COG information is needed if user systems wish any form of dynamic creation of COG posting pick lists. This requirement is still in the analysis phase.

9. Known Issues

a. Circular post bug – A posted alert with identical <sender> and <identifier> is rejected as duplicate and an exception. In most case, this is appropriate. However, if the DE message is already in the system for COG A where COG A represents a rule-based distribution activity, COG A is prevented from using those rule to post the same message back to COG B because the alert is already in the DM OPEN system. A temporary work around would be for COG B to slightly alter the identifier and repost. A future solution will be to add COG B to the list of COGs posted to without adding a new message to the system. This would allow COG B to retrieve the alert in an unaltered state, without duplicating the data in the database.

10. Suggestions, Recommendations and Assistance

a. Please send your suggestions, recommendations for improvement, etc. by e-mail to help@cmiservices.org. Be sure to specify that you are referencing the OPEN Interop CAP1.1 interface. Be specific. Remember that resources are limited and that the DM Program emphasizes the use of open standards for data and uses a directed distribution (user chooses recipients) sharing paradigm.

b. Initial programming and connection assistance can be provided through our OPEN Systems Interoperability Coordinator at OPEN.eyestreet.com.