Disaster Management Program
Open Platform for Emergency Networks

Instructions for Using the NOAA HazCollect Interface on the Open Platform for Emergency Networks (OPEN)

December 31, 2008

Version 0.4

Revision History

Date	Version	Description	Author
September 15, 2006	0.1	Initial draft	Gary Ham
July 20, 2007	0.2	POC updates	Gary Ham
November 6, 2008	0.3	POC updates and updates related to new FEMA Program Management	Gary Ham
December 31, 2008	0.4	Correct minor document errors and identify issues with Area Block identification that require programmer "work around" until new release is available.	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

The All Hazards Emergency Collection System (HazCollect) was developed to collect and efficiently distribute non-weather emergency messages (NWEM). NWEMs, commonly known as Civil Emergency Messages (CEMs), will be sent through the NWS dissemination infrastructure, other national systems, and to the Emergency Alert System (EAS). HazCollect will provide emergency responders and government officials a more efficient means to distribute alerts and warning information to an affected population in the event of an emergency.

This document covers the DM-OPEN interfaces that support this function through conversion from specially formatted Common Alerting Protocol (CAP) Version 1.1 messages to Non Weather Emergency Messages (NWEM). 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 Web Service Definition Language (WSDL).

1.4. References

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

- *OASIS Common Alerting Protocol, v1.1,* OASIS Standard CAPV-1.1, October 2005
- NWEM permission procedures for DM COGs (to be published).
- National Weather Service Instruction 10-1701, Text Product Formats and Codes, February 12, 2003.

2. Before You Begin

If you have not connected to OPEN in the past, please email OPEN@eyestreet.com for an FAQ that explains the basics of OPEN. The FAQ explains how to become approved for OPEN connection and where to find information on other OPEN offerings as well as this one.

The DM-OPEN environment operates on the basis of what the DM Program calls a Collaborative Operations Group or COG. A COG is a virtual organization that holds membership in DM-OPEN and manages individual user names and passwords within that membership. All DM COGs are sponsored by a vetted Emergency Management Organization. The FAQ explains the basics of how this works.

Operational NWEM capability requires additional COG level permission from the National Weather Service to submit NWEM messages. Initial testing COGs have already been defined and will not require additional permissions. Commercial and other Government system operational instances will require separate specific permission for each COG that wishes to post NWEM messages. (See NWEM approval procedures – published separately.)

The location of WSDL for generating needed client side classes is:
http://interopdev.cmiservices.org/axis/services/NWEM?wsdl

[IMPORTANT TEMPORARY NOTE: the NWEM interface is currently available for development/test/prototyping on http://interopdev.cmiservices.org/. It has not been moved to https://interop.cmiservices.org (its eventual production environment). Please substitute the "interopdev" reference for the "interop" in your development activities until the move is complete. You will be notified in the DM-OPEN Special Interest Group (SIG) mail list as soon as the move is complete.]

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 a CAP 1.1 message. For further reference please see the OASIS standard available from:

http://www.oasis-open.org/committees/download.php/14759/emergency-CAPv1.1.pdf

3. Defining a HazCollect NWEM Message in CAP 1.1 Format

A HazCollect Message is a valid CAP 1.1 message. Not all valid CAP 1.1 messages, however, have the data content and structure needed for HazCollect. So, HazCollect messages must be described as a specialization of CAP1.1 message. Because HazCollect Messages inherit, in an object-oriented sense, from CAP messaging, they can be sent over standard CAP networks and processed as a CAP message with no difficulty.

But a HazCollect message has specific needs for to allow downstream processing through NWS and EAS systems that require particular structuring and data content within a Valid CAP 1.1 format. Appendix A provides an annotated CAP1.1 sample message that identifies the structure and other data requirements needed for NWEM processing. Appendix A is normative, meaning that it the rules and structure found there MUST be followed in preparing you NWEM message. Once that is done, the processes and procedures needed to post NWEM messages are fairly simple. They are defined in the remainder of this document.

In addition to Appendix A, the following restriction (Paragraph 2.1 from NWS Instruction 10-1701) applies to all narrative data used in the message:

> "Characters, Case, and Punctuation for Narrative Text. Narrative text uses upper case and only the following punctuation marks in the text: the period (.), and the three dot ellipsis (...); the forward slash (/); the dash (-); and the plus (+). Use of other characters may inhibit the proper dissemination or automated processing by certain users' systems."

4. Generating Service Classes and Beans from the WSDL

Using the Axis libraries, run the WSDL2Java tool on the WSDL to generate the needed client side files. The following example call uses some additional parameters to create a package structure consistent with the OPEN server side. These parameters (NStoPkg) should not be necessary, but may be helpful.

```
java org.apache.axis.wsdl.WSDL2Java
        --NStoPkg http://dmi-services.org/beans=org.cmis.interopserver.beans
         --NStoPkg http://dmi-services.org=org.cmis.interopserver.services.nwem
        --NStoPkg urn:oasis:names:tc:emergency:cap:1.1=org.cmis.interopserver.beans.cap1_1
        -v https://interop.cmiservices.org/axis/services/NWEM?wsdl
```

Assuming your classpath was correctly set, running the above WSD2Java call will generate the following package and class structure:

```
org\cmis\interopserver\beans\cap1_1\Urgency.java
org\cmis\interopserver\beans\cap1_1\Info.java
org\cmis\interopserver\beans\cap1_1\Area.java
org\cmis\interopserver\beans\cap1_1\Parameter.java
org\cmis\interopserver\beans\cap1_1\Geocode.java
org\cmis\interopserver\beans\cap1_1\Resource.java
org\cmis\interopserver\beans\cap1_1\Status.java
org\cmis\interopserver\beans\cap1_1\ResponseType.java
org\cmis\interopserver\beans\cap1_1\Category.java
org\cmis\interopserver\beans\cap1_1\Severity.java
org\cmis\interopserver\beans\cap1_1\Scope.java
org\cmis\interopserver\beans\cap1_1\EventCode.java
org\cmis\interopserver\beans\cap1_1\Alert.java
org\cmis\interopserver\beans\SimpleCOG.java
org\cmis\interopserver\beans\cap1_1\Certainty.java
org\cmis\interopserver\beans\cap1_1\MsgType.java
org\cmis\interopserver\services\nwem\NWEM.java
org\cmis\interopserver\services\nwem\NWEMSoapBindingStub.java
org\cmis\interopserver\services\nwem\NWEMService.java
org\cmis\interopserver\services\nwem\NWEMServiceLocator.java
```

With these classes you can post NWEM messages to the HazCollect server for transmission to the NWS. Because the CAP specific classes are exactly the same as used in the CAP 1.1 general interface, these messages may also be posted to and retrieved by other DM-OPEN COGs using the general interface (see USING CAP1.1 on OPEN).

You may also access these classes without using WSDL2Java from the DM-OPEN Special Interest Group Web Site:

http://www.disasterhelp.gov/disastermanagement/library/documents/

A downloadable zip file contains the generated stubs needed for all DM-OPEN interfaces. A second zip file contains the axis jar files needed to use the interface. These axis jar files are required to support your Java application regardless of whether you generate directly from the WSDL or use the pre-generated stubs.

5. Connecting to the Web Service

Run the following class (call it from a main function with your user id, password and COG id) to prove that you can establish a connection. A return string of "pong" proves that you are using appropriate credentials and that the service is alive. It is always good to do a ping to check your connection before attempting other functionality. It isolates any connection errors for possible problems of another nature.

```
import java.net.*;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;

import org.cmis.interopserver.services.nwem.NWEMService;
import org.cmis.interopserver.services.nwem.NWEMServiceLocator;
import org.cmis.interopserver.services.nwem.NWEMSoapBindingStub;


public class SimplePingExample {


  public SimplePingExample (String userId, String pwd, String cogId ){
     URL svcURL;
     NWEMService service = new NWEMServiceLocator();
     NWEMSoapBindingStub nwem;

     try {
        // INTEROPDEV or INTEROP
        svcURL = new URL("http://interopdev.cmiservices.org/axis/services/NWEM");
//         svcURL = new URL("https://interop.cmiservices.org/axis/services/NWEM");
        System.out.println("Invoking service at " + svcURL.toString());
        nwem = (NWEMSoapBindingStub) service.getNWEM(svcURL);
        nwem.setUsername(cogId+"/"+userId);
        nwem.setPassword(pwd);
        try {
           System.out.println(nwem.ping());
        } catch (RemoteException e1) {
           e1.printStackTrace();
        }
       } catch (MalformedURLException e) {
        e.printStackTrace();
       } catch (ServiceException e) {
        e.printStackTrace();
     }
  }
}
```

6. Brief Explanations of Service Methods

Including the ping() method, you now have access to six proven CAP 1.1 service methods (There are a couple of other methods found in the WSDL that employ SOAP with attachments. These should be used only for special needs as they are, by definition, not compatible with the CAP 1.1 standard.):

a. To ping the service and ensure that it is up and running:

```
String ping() throws java.rmi.RemoteException;
```

The ping method returns the string “pong” if you are able to connect to the web service. It is used simply to verify that the service is up and running and that the user has been successfully authenticated.

b. **To get the COG you are logged into in SimpleCOG format (id as long, name as String):**

```
SimpleCOG getMyCog() throws java.rmi.RemoteException;
```

The getMyCog method returns an object of type SimpleCOG which contains the connection cogId as a long and the name of the COG as a String. (A SimpleCOG object is an instance of the SimpleCOG class generated from WSDL2Java). This is not a particularly useful function on operational basis, but its simplicity can be used to determine whether database connectivity is alive and well before attempting to diagnose more complicated system connection issues.

c. **To get all of the COGs that you are allowed to post to in an array of type SimpleCOG:**

```
SimpleCOG[] getCogs() throws java.rmi.RemoteException;
```

From the returned array, you may choose the target for your Alert posts. The getCogs method returns an array of SimpleCOG objects where each object includes a cogid as a long and the name of the COG as a String. Each SimpleCOG instance represents an organization that can retrieve an OPEN posted alert. For testing and initial development, OPEN interoperability partners are given passwords and ids in COG 1737, the OPEN Interoperability COG. For members of COG 1737, this method will return only two SimpleCOGs that have been set up expressly for test purposes (COG 1000 – the DMI Services Project Team and COG 2 – OPEN Interoperability COG 2). When testing is sufficient and a vendor or project has been sponsored by a vetted emergency organization, fully operational COGs can be set up for each sponsor. At that point, this method will retrieve an array of more than 2000 available COGs sponsored by organizations throughout the United States (and a couple of location in Canada). As a developer, you could make this a dynamically built pick list for your users. Since a list of this size may not be practical for your users, you can run this query and manually pick from those of interest to your users to build a static pick list for your users. (Note: See coming attractions below. There are plans to remedy this shortcoming.)

d. **To post an NWEM Alert to one or more OPEN COGs:**

```
void postNWEM(Alert alertObj, String timeZone, SimpleCOG[]
      simpleCOGs) throws java.rmi.RemoteException;
```

This method first sends the NWEM message to the HazCollect Server to be broadcast. After HazCollect Server broadcasts the message, the message will be posted to the DM COGs. Only valid NWEM messages can be posted to the HazCollect Server.
The first parameter is an instance of the Alert object generated from the WSDL. This instance must be populated according to CAP 1.1 schema and NWEM specialization instructions found in Appendix A. The second parameter is a string representing the time Zone of the sender in three digit, all capitalized format (e.g., EST for Eastern

Standard Time or PDT for Pacific Daylight Time). The third parameter is an array of SimpleCOG (see 3.c above) where each SimpleCOG represents an organization that you are allowing to retrieve the alert in addition to NOAA. Do NOT post to your own COG (i.e., do not include include the sign on COG in the array of SimpleCOG). It will raise errors. (Note: this differs from the EDXL DE implementation where posting to your own cog is required for retrieval of your own posted messages).

e. To retrieve a particular Alert with known identifier and sender from the OPEN interface:

```
Alert getAlert(String senderIdentifier) throws
        java.rmi.RemoteException;
```

The getAlert method returns a specific Alert object based upon the combined value of the <sender> and <identifier> tags. The String argument is colon delimited string consisting of the identifier and the sender (e.g., `"hamg@battelle.org" + ":"+ "DM001567" or "sender:identifier"`). (Note: This particular method fails when a colon (":") is an integral part of the identifier or sender. It is likely that the delimiter will be changed to a character that cannot be used as part of the identifier or sender fields in the next release). You can only retrieve an alert using this method if the alert was posted by your COG, the alert was expressly posted to your COG by a user member of a different COG, or the Alert was posted globally across the system. If the "senderIdentifier" does not correspond to an Alert in the DM-OPEN system, an empty value (NULL) will be returned.

f. To retrieve all Alerts posted to or from your COG with <sent> values greater than the time specified by the Calendar parameter:

```
Alert[] getAlertsBySentDate(java.util.Calendar dateTime)
        throws java.rmi.RemoteException;
```

The getAlertsBySentDate method returns an array of Alert objects where the time specified in each Alert object's <sent> tag is later than the Java Calendar argument. This includes Alerts that have been posted directly to the caller's COG, Alerts sent by the caller, as well as those that have been posted globally. The Alerts are returned in chronological order, from newest to oldest. If the date is invalid, an error is returned. If there are no alerts since that date-time, an empty list is returned.

This method allows retrieving systems to choose not to retrieve older alerts, making the retrieval process somewhat less onerous. Users of the getAlertsBySentDate (Calendar) method should still allow some overlap in time from most recent retrieval when choosing the Calendar 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 Alert retrievals should be checked for and discarded as appropriate.

g. To retrieve all Alerts posted to or from your COG later than the date and time specified by the Calendar value:

```
Alert[]getAlertsByPostedDate(Calendar dateTime) throws
```

The getAlertsByPostedDate method returns an array of Alert objects posted to the DM OPEN service after the time specified in the Java Calendar argument. This includes Alerts that have been posted directly to the caller's COG, Alerts sent by the caller, as well as those that have been posted globally. The Alerts are returned in chronological order, from newest to oldest. If the date is invalid, an error is returned. If there are no alerts since that date-time, an empty list is returned.

This method is almost identical to `getAlertsBySentDate(Calendar)` (paragraph 5f above) except that the date-time value is being compared to the actual time that Alert is posted to the DM-OPEN server instead of the <sent> field in the Alert.

h. To retrieve the current DM-OPEN server time in Calendar format:

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

Retrieves the current time on the server; useful for time-synching to ensure that getAlertsByPostedDate() is accurate. Note that current server time is also returned as a SOAP header with each request; this method is here only for convenience.

i. To retrieve the current operational mode of the HazCollect server :

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

Possible modes are Active, Training, and Test. Actual dissemination of an NWEM message through NWS systems is dependent on a combination of this value with the value of the <status> tag in the CAP 1.1 message. Appendix B provides specific details.

j. To determine whether a COG is authorized to post NWEM messages to the NWS:

```
boolean isCogAuthorized(String cogID)
        throws java.rmi.RemoteException;
```

Requires entry of a string representing a valid cogId as a parameter. Returns “true” if the COG has been authorized by the NWS to post NWEM messages. Returns "false" if permission has not been expressly been granted to that COG.

k. To retrieve list of geographic areas and associated codes to which a COG is authorized to broadcast:

```
CogGeoFipsCodeImpl[]getNWEMAuxData (String cogId) throws
     java.rmi.RemoteException
```

Each COG will be authorized by the NWS to request a broadcast of alerts to a pre-defined set of geographic areas. This function retrieves the code values and associated names (e.g. FIPS code and associated County Name) that have been authorized for the COG identified by the `cogId` as an array of `CogGeoFipsCodeImpl` objects. Developing systems can build dynamically populated pick lists of “warnable” areas for their user interfaces from this data. The codes chosen by users are then used to populate the <area> and <geocode> tags in the CAP message (as shown in Appendix A).

7. Steps in a Typical NWEM Post

The following steps illustrate the "happy path" (no errors, unexpected results, or exceptions) of a DM-OPEN connection and post to HazCollect:

a. Establish the connection.
b. Use either `ping()` or `getServerTime()` to ensure that you DM-Open credentials are accepted and that the service is running correctly.
c. Use `isCogAuthorized()` to ensure that your COG is currently authorized to send NWEM messages. Remember that COG establishment is done through DM-OPEN but approval of the COG for NWEM creation is a NWS action and is done separately.
d. Use `GetNWEMAuxData()` to get the geocode information that your COG is allowed to us in building the CAP area block of the NWEM specialized CAP message. This probably becomes a pick list for your users to choose from.
e. Use `getHCSMode()` to determine the operational status of the HazCollect Server. . Actual dissemination of an NWEM message through NWS systems is dependent on a combination of this value with the value of the <status> tag in the CAP 1.1 message. To send an actual alert through HazCollect to the public requires a <status>= "Actual" and HCSMode = "Active." System users should be specifically warned that ***this combination will result in public dissemination as an actual alert.*** Special care should be taken that such messages are real alerts requiring immediate public dissemination. Appendix B provides specific details on the results of each potential combination of HCSMode and CAP <status> tag.
f. Build your valid CAP 1.1 message such at all rules defined in Appendix A of this document are followed.
g. If you plan to post the message to other COGs (as well as to NWS), use `getCogs()` or a predetermined list of COGs to allow your users to select additional COGs for posting.
h. Finally use the `postNWEM()` method to send your message downstream.
i. You may wish to use `getAlert()` to retrieve a return copy of the message to ensure that your post was successful.

8. 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. Be sure to identify you client as Java 1.4 or 1.5 because the needed Axis libraries will be different. OPEN@eyestreet.com should also be your first point-of-contact for any problems related to programming issues when using OPEN.

9. Coming Attractions

- getCogs(geography) – There will soon be a version of getCogs using a circle structure (point and radius) or similar geographic reference that will limit the number of COGs retrieved by this method. This will make the creation of dynamic posting pick lists more effective. Date of release TBD (near future).
- 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.

10. Known Issues

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 Alert 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 alert 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. The proper solution will be to add COG B to the list of COGs posted to without adding a new alert to the system. This would allow COG B to retrieve the alert in an unaltered state, without duplicating the data in the database. [Note: this bug is not related in any way to NWEM processing directly. It only applies to CAP messages that are the basis for NWEM messages, but may also be transmitted on parallel CAP networks.

11. Suggestions, Recommendations and Assistance

Please send your suggestions, recommendations for improvement, etc. by e-mail to OPEN@eyestreet.com. Be sure to specify that you are referencing the OPEN Interop NWEM 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.

Initial programming and connection assistance can be provided through our OPEN Systems interoperability Coordinator, also at OPEN@eyestreet.com.

Acronyms

The following acronyms are used in this document or are relevant to its content.

Acronym	Description
CAP	Common Alerting Protocol
CEM	Civil Emergency Message
COG	Collaborative Operations Group
DM-OPEN	Disaster Management Open Platform for Emergency Networks
EAS	Emergency Alert System
HazCollect	All Hazards Emergency Message Collection System
HTTP	Hypertext Transfer Protocol
NIEM	National Information Exchange Model
NOAA	National Oceanic and Atmospheric Administration
NWEM	Non Weather Emergency Messages
NWS	National Weather Service
NWS I	National Weather Service Instruction
OASIS	Organization for the Advancement of Structured Information Standards
SOAP	Simple Object Access Protocol
WSDL	Web Service Definition Language

Appendix A. Annotated NWEM CAP Example

```
<!--
HazCollect Non-weather Emergency Messages are a specialization of CAP 1.1.
1.  There are some tags that are required for CAP, but not used in NWEM.  NWEM builders should
use these fields appropriately as NWEM Messages may also travel on Cap networks.
2.  There are some optional tags that are required for NWEM.  Some of these have special content
restrictions.  Those restrictions are needed in order to support NOAA's downstream systems.
3. Some cardinalities are more restrictive in NWEM than they are in CAP.  These cardinality
restrictions are also required to adequately support downstream system requirements.
4. Finally, there are some tags that are optional in CAP and not used by HazCollect. As in item 1
above, the OPEN interface will take these fields but will not process them thorough NWEM.
If the message is also posted though the general CAP network, these tags will be processed
appropriately.
The Comments in the following CAP message further define the specific requirements for processing
as an NWEM message.
-->
<alert xmlns="urn:oasis:names:tc:emergency:cap:1.1">

    <!-- Identifier must be a unique identifier across the system. Suggest using a system identifier in
    form of an actual identifier string.  The DM program affixes a DM in form of a randomly generated
    long integer turned to a string. You may use what you like, but it should carry a guarantee of
    uniqueness.
     -->
    <identifier>DM3117513862327615488gh_02</identifier>

    <!-- The sender field is required in CAP but not used in the corresponding NWEM. It is of value
    for messages that will travel on CAP networks as well as to HazCollect.  A common practice
    is to use an email identification type format (e.g., senderPerson@domainName.net). The DM
    program uses the COG name with underlines in place of any spaces found in that name.
    Either solution is acceptable.
    -->
    <sender>DMI-_Services_Project_Team</sender>

    <!-- Time at which the message is released.  Exactly as define in the CAP1.1 spec
    -->
    <sent>2006-04-20T10:12:29-04:00</sent>

    <!--  Enumerated Value Actual, Exercise, System, or Test. "Draft" from CAP1.1 is
    not allowed for NWEM posting.  The tag itself in not used in the actual
    NWEM message, but is used in pre-processing. For any message other than
    Actual the following text will automatically be added by the HazCollect Server to
    Resulting NWEM message translation: " ....THIS IS A <status> MESSAGE. DO NOT TAKE
    ACTION BASED ON THIS <status> message..." This additional text WILL NOT be added to
    the text of the CAP message itself. Messages broadcast on other CAP networks will need
    to take this into account and add specific additional warnings to the description field in the
    info block of the CAP message, where applicable.  Such additional warnings are OK with the
    HazCollect system as they re-emphasize the fact that a message is not an “actual” alert.
    -->
    <status>Test</status>

    <!--  msgType is not directly used in NWEM but is valid for NWEM messages passed
    in parallel through CAP channels. It is also used in NWEM pre-processing.
    The server will use an Error, Update or Cancel message to do exactly that
```

```
if the message references a previously sent NWEM Alert that has not yet
expired. The Error, Update or Cancel message must properly reference the originating
message in its references tag.
 -->
<msgType>Alert</msgType>

<!--
Last name of sender and initials.  This field is used in combination with the CAP <identifier>
to “sign” the bottom of the message in its NWEM format.
 -->
<source>HamGA</source>

<!-- All NWEM Messages must have a Public scope
-->
<scope>Public</scope>

<!-- <references> is not used directly in an NWEM message but is valid for
NWEM messages passed in parallel through CAP channels and is processed by
the NWEM server to update unexpired previous NWEM messages.  The server
will use an Error, Update or Cancel messageType message to do exactly that
if the message references a previously sent NWEM Alert that has not yet
 expired.
 -->
<references/>

<!-- An NWEM instance can have only one info block vice the many allowed in
regular CAP messaging.
-->
<info>

    <!--   Language is required. Only two values allowed: en-US or sp-US.
     -->
    <language>en-US</language>

    <!-- Category is a required CAP1.1 tag. It is not used in the outgoing NWEM but is
    used in parallel CAP channels.
     -->
    <category>Other</category>

    <!-- Event is a required field for HazCollect and is still a string. The specific
    String, however is specialized from CAP.  It must be one of an enumerated
    set of NWEM names and must relate to the Event Code tag below.  For NWEM,
    it is the text name of the SAME Code found below in alert/info/eventCode/value.
    Allowed values are found in Appendix C below.
     -->
    <event>Administrative Message/Follow up Statement</event>

    <!-- Urgency is a required CAP1.1 tag. It is not used in the outgoing NWEM,
    but is used in parallel CAP channels.
     -->
    <urgency>Future</urgency>

    <!-- Severity is a required CAP1.1 tag. It is not used in the outgoing NWEM,
    but is used in parallel CAP channels.
     -->
    <severity>Moderate</severity>
```

```
<!-- Certainty is a required CAP1.1 tag. It is not used in the outgoing NWEM,
but is used in parallel CAP channels.
 -->
<certainty>Unlikely</certainty>

<!-- Event Code is an optional, 1 to many, structure in CAP1.1.  For HazCollect
exactly one instance of the tag is required.
 -->
<eventCode>

        <!-- The only eventCode/ValueName name accepted by HazCollect
        is "SAME" or "same." All HazCollect CAP messages must be
        “SAME” messages or they will not be accepted for NWEM translation.
        -->
        <valueName>SAME</valueName>

        <!-- The eventCode/value must be one item from the list of SAME codes
        recognized by HazCollect.  It must be the code that corresponds to the
        name found in the <event> tag above.  Allowed values are defined in
        Appendix C below.
        -->
        <value>ADR</value>

</eventCode>

<!-- <effective> must be set to the same time as <sent> for HazCollect
-->
<effective>2006-04-20T10:12:29-04:00</effective>

<!-- <onset> is not used by HazCollect.  HazCollect assumes
sent = effective = onset
This makes sense in the NWEM scenario because NOAA radio warnings
are limited to immanent actual happenings.

<expires> is optional in CAP1.1 but required for HazCollect. Known in HazCollect as
"Product Purge Time”, it also has special requirements in terms
of value as follows:
1.  It must not exceed <sent> by more than 360 minutes
2.  It must differ from <sent> in exactly 15 minute intervals up to 120 minutes.
3. It must differ from <sent> in exactly 30 minute intervals between 120 and 360.
 -->
<expires>2006-04-20T10:27:29-04:00</expires>

<!--  Sender name is a required and specifically formatted field for HazCollect
 Format is <CogName>,<City>,<State>.
Most systems will want to generate this field for the user.
-->
<senderName>COG Name,Stafford,VA</senderName>

<!--
Headline requires a String <= 160 characters including spaces
-->
<headline>THIS MESSAGE IS FOR TEST PURPOSES ONLY. THIS IS TEST
        MESSAGE NUMBER 1.</headline>
```

```
<!-- The NWEM message concatenates the CAP description and the CAP
Instruction and is limited to a single field that must be <= 160 words. The
description field for NWEM includes what might also be in
the instruction field of a normal CAP message. The interface concatenates
for posting to NWEM but will use the original structure
for posting copies of the NWEM message to other CAP users and networks when
directed by your application. This allows the Call to action portion of the
CAP message to be properly placed within the instruction tag per the CAP 1.1 specification-->
<description>THIS IS A TEST MESSAGE. THIS IS A TEST OF THE CAPABILITY
TO RELAY EMERGENCY MESSAGES FROM NON-NATIONAL WEATHER SERVICE SOURCES
USING DEPARTMENT OF HOMELAND SECURITY AND NWS SYSTEMS. THIS TEST
MESSAGE IS NOT INTENDED TO ACTIVATE THE EMERGENCY ALERT SYSTEM.
</description>

<!-- The Call to action portion of the CAP message should be placed within the <instruction>
tag per the CAP 1.1 specification. The combined size of <description> and <instruction>
cannot exceed 160 words.
-->
<instruction>This is a place holder for evac instructions, etc.
It is where the “call to action” should be placed.</instruction>

<!--  Area:
1.Each FIPS,  ZONE, or STATE CODE gets it own <area> block in a HazCollect message
2.The current implementation of HazCollect does not use <polygon> or <circle>.
within <area>. Support for these structures is projected for a future release. For now, however
they are ignored. Only the <geocode> tag is processed.
```

Interim Note 1 concerning Area: While it is suggested that each FIPS, ZONE, or STATE CODE get its own separate area as shown below so that the proper name for the FIPS code can be associated with the area in <areaDesc>, the HazCollect server will allow multiple <geocode> entries within the same <area> block and will process all of them.

Interim Note 2 concerning Area: Parts of <area> other than <geocode> can be used for non-NWEM purposes in a NWEM CAP message because the HazCollect Server simply ignores them.

Interim Note 3 concerning Area: Currently the “state” or “STATE” value for geocode also ignored. This requires a statewide alert to include all of the fips codes applicable at the county level for the entire state. This is a bug which will be fixed in a future release. As a workaround, the single area block naming the state with multiple fips genocodes (as in Interim Note 1) might be appropriate.

```
-->

<area>

    <!--
    The <areaDesc> tag should contain the administrative name corresponding to the
    geocode value. Allowed values for a particular COG are retrieved using the
    getNWEMAuxData() method.
    -->
    <areaDesc>Burleigh</areaDesc>

    <!-- Exactly one geocode is required for each area block. Allowed values will be
    specified separately for each COG by NOAA.  They are dynamically
    retrievable using the GeNWEMAuxData() method on the DM-OPEN interface. The
    downloadable structure includes administrative names associated with the
    codes (for <areaDesc>) so that systems can build dynamic pick lists for  alert
```

```
                creators that include all "Zone", FIPS, and State codes allowed to the alert issuing
                COG by NWS.
                -->
                <geocode>
                        <valueName>fips</valueName>
                        <value>38015</value>
                </geocode>
        </area>
        <area>
                <areaDesc>Allegheny</areaDesc>
                <geocode>
                        <valueName>fips</valueName>
                        <value>42003</value>
                </geocode>
        </area>
        <area>
                <areaDesc>Pennsylvania</areaDesc>
                <geocode>
                        <valueName>state</valueName>
                        <value>PA</value>
                </geocode>
        </area>
    </info>
</alert>
```

Appendix B. Explanatory Information Concerning HazCollect Server Mode

Table B-1 below correlates HazCollect Server Mode and the value of the <status> tag in a submitted CAP 1.1 message. The NWEM format column refers to the additional data that is added to the final product submitted to the NWS as follows: " ....THIS IS A <status> MESSAGE. DO NOT TAKE ACTION BASED ON THIS <status> message...” This additional structure is attached to the outgoing NWEM only and is not maintained in the CAP message that may be transmitted on other CAP networks.

One key fact that is discoverable from this table is that only three of the possible combinations of HazCollect Server Mode and CAP <status> will result in downstream transmission of alerts through NWS systems. “Actual” alerts will be sent only when the HazCollect Server is in “active” mode. “Exercise” alerts also require that the HazCollect Server be in “active” mode. Finally, “Test” alerts may be passed down stream under special circumstances defined by the NWS with the HazCollect server in “Test” mode. All of the other combinations will be accepted, but will not be sent to downstream systems for processing.

Table B-1 – HazCollect Server Mode to CAP Status Relationship

DM User Activity	HazCollect Server Mode	CAP Status Field	NWEM Format	NWEM Disseminated
Normal	Active Ops	Actual	Normal	Yes
	Training Ops	Actual	Normal	no
	Test Ops	Actual	Normal	no
Training	Active Ops	System	[Training]	no
	Training Ops	System	[Training]	no
	Test Ops	System	[Training]	no
Exercise	Active Ops	Exercise	[Exercise]	Yes
	Training Ops	Exercise	[Exercise]	no
	Test Ops	Exercise	[Exercise]	no
Test	Active Ops	Test	[Test]	no
	Training Ops	Test	[Test]	no
	Test Ops	Test	[Test]	Yes if Authorized*

Appendix C. NWEM Event Codes

The following codes are usable by HazCollect as SAME codes within an NWEM compatible CAP message. The allowable CAP alert/info/eventCode/value entries make up the "Event Code" Column. The allowable CAP alert/info/event entries are shown in "Event (Product) Name" column. The Priority Column shows relative priority or processing upon receipt by NWS.

Table C-1. NWEM Event Codes

Event Code	Priority	Event (Product) Name
AVW	1	Avalanche Warning
CDW	1	Civil Danger Warning
CEM	1	Civil Emergency Message
EQW	1	Earthquake Warning
EVI	1	Immediate Evacuation Warning
FRW	1	Fire Warning
HMW	1	Hazardous Materials Warning
LEW	1	Law Enforcement Warning
NUW	1	Nuclear Power Plant Warning
RHW	1	Radiological Hazard Warning
SPW	1	Shelter In Place Warning
VOW	1	Volcano Warning
AVA	2	Avalanche Watch
CAE	2	Child Abduction Emergency
LAE	2	Local Area Emergency
TOE	2	911 Telephone Outage Emergency
ADR	3	Administrative Message/Follow up Statement
NIC	3	National Information Center
DMO	Configurable	Practice/Demo Warning
NPT	Configurable	National Periodic Test
RMT	Configurable	Routine Monthly Test
RWT	Configurable	Routine Weekly Test
NMN	Not Currently Disseminated	Network Message Notification