WEBDAV Working Group
Internet Draft
draft-reddy-enp-protocol-00.txt
Expires December 12, 1998

Surendra Reddy(Oracle)
Mark L Fisher(TCE)
June 12,1998

# Event Notification Protocol - ENP

## Status of this Memo



## Abstract

As the complexity of distributed applications increases, an increasing amount of processing is done using distributed processes, which typically execute without the direct supervision of an end user. The user must poll these processes periodically to check if they are completed successfully or not. This polling results in unnecessary wastage of network bandwidth as well as computing resources. The user generally cannot see intermediate results or progress reports for long running processes, they must wait till the process is completely finished before viewing the results.

Thus the problem of monitoring events is central in distributed

applications and protocols. A repeated need in such applications is receive notifications when a resource property value changes or event state changes. Current database systems provides mechanisms like constraints, triggers and active database rules. These facilities provides an automated means to ensure the database integrity or perform specific action when data changes. Need for such kind of requirement is fundamental is network applications.

Event Notification Protocol(ENP) abstracts the notification requirements from the applications. ENP provides a lean and mean protocol with a client side semantics for processing notifications. The goal of ENP is to provide a service which allows users to select resources or events for which they wish to be notified in case changes of property values or state values occur. The Event Notification Protocol will also allow users to define what events or state changes they are interested in.

This document describes the Event Notification Protocol. The objective is to provide a simple, scalable and highly efficient notification protocol while also providing the appropriate flexibility to meet the needs of both the internet and enterprise environments.

Table of Contents

## 1. Introduction

As the complexity of distributed applications increases, an increasing amount of processing is done using distributed processes, which typically execute without the direct supervision of an end user. The user must poll these processes periodically to check if they are completed successfully or not. This polling results in unnecessary waste of network bandwidth as well as computing resources. The user generally cannot see intermediate results or progress reports for long running processes, they must wait till the process is completely finished before viewing the results.

Thus the problem of monitoring event states is central in distributed applications and protocols. A repeated need in such applications is receive notifications when a resource property value changes. Current database systems provides mechanisms like constraints, triggers and active database rules. These facilities provides an automated means to ensure the database integrity or perform specific action when data changes. Need for such kind of requirement is fundamental is network applications.

There is already a multitude of applications that requires notification mechanisms. Some of these applications include:

- Internet Printing Protocol
- Workflow
- Distributed Authoring
- Email

Event Notification protocol(ENP) abstracts the notification requirements from the applications. ENP provides a lean and mean protocol with a client side semantics for processing notifications. The goal of ENP is to enable a generic event notification to do routing to all these applications domains.

The main objective of this protocol is to provide a simple, scalable and highly efficient notification protocol while also providing the appropriate flexibility to meet the needs of both the internet and enterprise environments. This document describes a  a set of methods, headers, request entity body formats, and response entity body formats anciliary to HTTP/1.1 to provide operations for:

Properties: The ability to create, remove, and query information about events.

triggers: The ability to register, unregister, define  events or resources that need to be monitored.

rules: The ability to notify subscribe when the requested

criteria defined on event satisfied.

Requirements for these operations are described in a companion document, "Requirements for Event Notification Protocol" [S.Reddy,1998].

ENP employs the property mechanism to store information about the current state of the event, rules and triggers associated with of these events.

## 2. Notational Conventions

**Since this document describes a set of extensions to the HTTP/1.1** protocol, the augmented BNF used herein to describe protocol elements is exactly the same as described in section 2.1 of [Fielding et al., 1997]. Since this augmented BNF uses the basic production rules provided in section 2.2 of [Fielding et al., 1997], these rules apply to this document as well.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [Bradner, 1997].

## 3. Event Notification Protocol

### 3.1. Overview

**The event notification protocol(ENP) defines two roles for the** events: the supplier role and the consumer role. Event supplier produce event data and event consumers process event data. Event data are communicated between suppliers and consumers through Event Notification Protocol(ENP). Event Notification Protocol uses push and pull model to initiates communication. The push model allows a supplier of events to initiate the transfer of the event data to consumers. The pull model allows a consumer of events to request the event data from a supplier.

ENP decouples the communication between producers and consumers. Event Channel is central to the idea of decoupling event producers and consumers. The most important of these is subscribe, which tells the event channel that the consumer is interested in receiving notifications matching with constraints. The event channel evaluates the constraints against each notification it receives and routes these notifications to all subscribed consumers matching with their constraints.

The consumer may use either a blocking or non-blocking mechanism for receiving notifications. The consumer can periodically poll the

channel for events.

One of the major goals of this protocol is to leverage on the existing protocols and provide this service as extensions rather than developing a whole new protocol. By closely looking at the WebDAV efforts, Event Notification Protocol can easily be achived with no extensions to methods and fewer extensins to request and response headers. Following are the major protocol elements:

- o Notification Server
- o Event Producer interface
- o Event Consumer interface

## 3.2. Notification Server

**The notification server will be responsible for managing the** notification content database and to perform notification delivery. This server will respond to two kinds of responses as explained in section 3.1. Event producers sends requests to notification servers to register the state changes or any event related attributes with the server. Notification server queues these notifications for delivery to subscribed consumers. Event consumers request for various notifications of their interest. Event producers will have a writer role into the event database maintained by the notification server where as event consumers will have read only role.

## 3.3. Event Consumer interface

**The event consumers subscribe to specific events with the** notification server or event consumer can also request notification server to start the specific process instance and notify them or defined set of users or group upon completion of the event.

## 3.4. Event Producer interface

**The event producers are the process instances which modify the event** states, or produce event data. However, notification server point of view these process instances are black boxes and notification server doesnot need to know the internal semantics of these execution steps. Instead, notification server is only interested in maintaining the published attributes and observer the changes that occurs on these values, deliver them to the subscribed consumers.

## 3.5. How does ENP Work?

**ENP acts as a event channel which propages event notifications to** event producers and consumers. Event Producers and Consumers need to subscribe their services with the ENP so that event consumers can query ENP to find out what services are avialable.

### 3.6. Event Consumer requesting for a service

ENP based print service registers the event data with the ENP.

```
     >> Request

     <advertise>
        <einfo id="http://www.printspooler.com/lpservice",
               name="distributed printer service">
        <eattributes>
                <attribute name="printername", type="string">
                        printer-200
                </attribute>
                <attribute name="job", type="string">
                        printer-200-1001
                </attribute>
        </eattributes>
        <estates>
          <vstate>
                <state>aborted</state>
                <state>canceled</state>
                <state>completed</state.
          </vstate>
          <cstate>
                <state>completed</state>
          </cstate>
        </estates>
     </advertise>

     >> Response

     <multiresponse>
        <enpresponse>
                <response> HTTP/1.1 100 Successful </response>
        </enpresponse>
     </multiresponse>
```

In the above example, ENP complaint print servce, notifies the ENP server that print event identified by http://www.printspooler.com/lpservice has <aborted, completed,canceld> as valid states and current state of this event is <completed> for the job=printer-200-1001 and printer=printer-200.

### 3.7. Event Consumer Querying on Event State

ENP client queries the ENP to figure out the status of the print service.

```
     >> Request
```

```
        <enprequest>
         <query>
           <einfo id="http://www.printspooler.com/lpservice">
              <erule>
                 <eterm>
                   <eprop>job</eprop> <eop="eq"/>
                    <evalue type=string>printer-200-1001</evalue>
                  </eterm><and/>
                  <eterm>
                   <eprop>printer</eprop> <eop="eq"/>
                    <evalue type=string>printer-200</evalue>
                  </eterm>
               </erule>
             <enotify type="mail">skreddy@us.oracle.com</enotify>
         </query>
        </enprequest>

        >> Response

        <multi-response>
           <enpresponse>
              <response> HTTP/1.1 100 Successful </response>
           </enpresponse>
        </multiresponse>
```

In the above example, event consumer sends a query to ENP to find out the print event with job=printer-200-1001 AND printer=printer-200. This request tells the ENP to notify the event consumer through email whenever the print job is completed. In this case, notification request is transient which means that it is valid only for one notification. Event consumer can also indicate persistent notifications, which means ENP keep notifying the event consumer as long as these events are advertised with the ENP.

## 3.8. ENP Notifying the Consumer

**When the ENP print service(event producer) advertises the event** state with the ENP, it will validate the notification attributes to match with the subscribers constraints and routes them accordingly to consumers.

```
        <notification>
           <eventid> 00.0.22.33.111.wewewwe </eventid>
           <message> Print Job  printer-200-1001 Completed Successfully </
message>
        </notification>
```

## 3.9. Simple Workflow Example

**In this scenario, consumer subscribes with the ENP to notify users** accounts and purchases on successful completion of the purchase

order processing. Various protocol interactions are described below:

```
(1). User Agent to ENP (subscribing for event notifications)
     >> Request

     <enprequest>
      <subscribe>
           <einfo id = http://www.foo.com/porder>
           <erule>
              <eterm>
                <eprop>ponumber</eprop> <eop="eq"/>
                 <evalue type=string>L234567/evalue>
               </eterm><and/>
               <eterm>
                <state>completed</state>
               </eterm>
           </erule>
          <enotify type="mail">accounts@us.oracle.com</enotify>
          <enotify type="mail">purchase@cs.oracle.com</enotify>
      </subscribe>
     </enprequest>

     >> Response

     <multi-response>
           <enpresponse>
                <response> HTTP/1.1 100 Successful </response>
           </enpresponse>
        </multi-response>

(2). ENP Producer to ENP(Advertising event state and data)

     "porder" process instance completes the purchase order processing,
     sends the result data payload to the ENP.
   >> Request

     <advertise>
        <einfo id="http://www.foo.com/porder",
               name="purchase order processing">
        <eattributes>
                <attribute name="ponumber", type="string">
                        L234567
                </attribute>
                <attribute name="amount", type="real">
                        1000.20
                </attribute>
                <attribute name="vendor", type="string">
                        Sun Microsystems, Inc.
                </attribute>
        </eattributes>
```

```
            <estates>
              <vstate>
                    <state>aborted</state>
                    <state>canceled</state>
                    <state>completed</state.
              </vstate>
              <cstate>
                    <state>completed</state>
              </cstate>
            <edata content-type=base64 length=2000>
                    BASE64ENCODEDDATAHERE
            </edata>
          </advertise>

        >> Response

            <multi-response>
               <enpresponse>
                    <response> HTTP/1.1 100 Successful </response>
               </enpresponse>
            </multi-response>
```

(3). ENP to ENP Consumer(Notification)

ENP notifies the "porder" completion notification and result data to the subscriber through the known notification route.

```
            <notification>
               <einfo id = http://www.foo.com/porder>
               <estate>
                    <cstate>completed</cstate>
               </estate>
            <edata content-type=base64 length=2000>
                    BASE64ENCODEDDATAHERE
            </edata>
            </notification>
```

## 4. Data Model for Event Notification Protocol

### 4.1. The Event Property Model

**Properties are meta data that describe the state of a** event/resource, subscribers, monitorable events, triggers, and rules.

The ENP property model is similar to WebDAV property model and consists of name/value pairs. The name of a property identifies the property's syntax and semantics, and provides an address by which to refer to its syntax and semantics.

### 4.2. Property Values

**The value of a property is, at minimum, well formed XML.**

### 4.3. Property Names

**A property name is a universally unique identifier that is** associated with a schema that provides information about the syntax and semantics of the property.

Because a property's name is universally unique, clients can depend upon consistent behavior for a particular property across multiple resources.

### 4.4. Notification Attributes

**The notification attribute data contains the of the process** instance(URI), triggers(changes for which subscribers should be notified), filters(set of conditions that need to be applied to notifications before delivering them to the subscribed users), and the list of addresses to which the notification is sent.

### 4.5. Notification Content

**The notification content data contains additional information that** need to delivered along with the notifications.

### 4.6. Triggers

**Defines what actions to be performed.**

### 4.7. Rules

**Defines what triggers need to be activated on what property values.** Rules provides a mechanism to filter unwanted notifications.

### 4.8. Client Based Semantics

**ENP protocol doesnot know anything about property value semantics** and it doesn't interpret these values. Client process SHOULD be responsible for interpreting the meaning of these notifications. Client based semantics are essential for scalable protocol.

## 5. HTTP Method Definition Extensions

**This protocol is completely defined around the WebDAV. Following** sections describe who existing methods in WebDAV can be used to define additional requirements for Event Notification Protocol.

### 5.1. PROPFIND

**The PROPFIND method retrieves all pending events from the event** queue identified by the Request-URI. All events are identified uniquely using UUIDs[P.Leach].

A client MUST submit a eventquery XML element in the body of the

request method describing what information is being requested. It is possible to request particular event attribute, all property values, or a list of the names of the events's properties.

All servers MUST support returning a response of content type text/xml that contains a multistatus XML element that describes the results of the attempts to retrieve the pending event notifications from the queue.

If there are no pending notifications available from the queue then not found result MUST be included in the response.

The results of this method SHOULD NOT be cached.

### 5.1.1. Example - Retrieving Queued Notifications

```
>>Request

PROPFIND  /enpqueue HTTP/1.1
Host: www.foo.com
enprequest-type: Query
Content-type: text/xml
Content-Length: xyz

<?xml version="1.0" ?>
<?xml:namespace ns="ENP:" prefix="D" ?>
<D:eventquery>
     <D:allevents/>
</D:eventquery>

>>Response

HTTP/1.1 207 Multi-Status
Content-Type: text/xml
Content-Length: xxxxx

<?xml version="1.0" ?>
<?xml:namespace ns="ENP:" prefix="D" ?>
<D:multistatus>
  <D:response>
          <D:eventinfo>
             <eventid>abc0001</eventid>
             <eventprop>DELETED</eventprop>
             <eventsource>Surendra Reddy</eventsource>
             <eventdate> May 10, 1998 </eventdate>
             <resref>http://www.foo.com/container/res1.html</resref>
          </D:eventinfo>
          <D:eventinfo>
             <eventid>xyz0002</eventid>
```

```
                <eventprop>UPDATED</eventprop>
                <eventsource>unknown</eventsource>
                <eventdate>May 10, 1998</eventdate>
                <resref>http://www.foo.com/container/test.html</resref>
              </D:eventinfo>
    </D:response>

  </D:multistatus>
```

In this example, PROPFIND is executed on the event queue identified by http://www.foo.com/skreddy. The eventquery XML element specifies the name of the event properties whose values are being requested.

## 5.2. PROPPATCH

**The PROPPATCH method processes instructions specified in the request** body to define, set and/or remove properties defined on the events identified by the Request-URI. The PROPATCH method is also used to subscribe, unsubscribe for event notifications.

The request message body of a PROPPATCH method MUST contain the eventrequest XML element. Instruction processing MUST occur in the order instructions are received (i.e., from top to bottom). Instructions MUST either all be executed or none executed. Thus if any error occurs during processing all executed instructions MUST be undone and a proper error result returned.

### 5.2.1. Example - PROPPATCH

```
  >>Request

  PROPPATCH /bar.html HTTP/1.1
  Host: www.foo.com
  enprequest-type: Create
  Content-Type: text/xml
  Content-Length: xxxx

  <?xml version="1.0" ?>
  <?xml:namespace ns="DAV:" prefix="D" ?>
  <D:eventrequest>
       <subscribe reqref="001">
              <href>http://www.foo.com/enp.html</href>
              <eventprop>update</eventprop>
              <eventprop>delete</evenrprop>
              <notifyroute>mailto:skreddy@us.oracle.com</notifyroute>
       </subscribe>

  </D:eventrequest>

  >>Response
```

```
HTTP/1.1 207 Multi-Status
Content-Type: text/xml
Content-Length: xxxxx

<?xml version="1.0" ?>
<?xml:namespace ns="DAV:" prefix="D" ?>
<D:multistatus>
  <D:response reqref="001>
       <D:status>HTTP/1.1 100 Successful </D:status>
       <D:eventref>abc001<eventref>
  </D:response>
  <D:response reqref="002">
       <D:status>HTTP/1.1 424 Method Failure</D:status>
  </D:response>
</D:multistatus>
```

In this example, the client requests the server to subscribe for state change on resource http://www.foo.com/enp.html to update or delete. In this case, ENP will notify the client through notification route mailto:skreddy@us.oracle.com whenever the resource update or delete property changes.

In the second request, client registers for an event validate for state chane to complete. Upon successful completion of this event, ENP triggers http://www.foo.com/cgi-bin/generate event and on completion of this trigger it notifies the client at URL identified by http://www.foo.com/.

## 6. HTTP Headers for Event Notification Protocol

### 6.1. ENP Header

**ENP = "ENP" ":" "1.0"**

This header indicates that the server supports the ENP schema and protocol as specified. All ENP compliant servers MUST return the ENP header on all OPTIONS responses.

### 6.2. Depth Header

**For instance, user may want to set a notification on a collection,** but not all descendants. In this case, the user should be given the option of not letting its changes propagate. Depth header provides the user with this functionality to control of the depth of the notifications. If Depth header is set to 0, then changes trigger on the collection only, if it is 1 it applies to all immediate descendants of the collection. If depth header is set to infinitity all changes performed on all descendants of the collection.

### 6.3. If Header

**If = "If" ":" ( 1*No-tag-list | 1*Tagged-list)**
No-tag-list = List
Tagged-list = Resource 1*List
Resource = Coded-url
List = "(" 1*(["Not"](State-token ")"
State-token = Coded-url
Coded-url = "<" URI ">"

The If header is intended to have similar functionality to the If-Match header defined in section 14.25 of [Fielding et al., 1997]. However the If header is intended for use with any URI which represents state information, referred to as a state token, about an event as well as e-tags.

All ENP compliant resources MUST honor the If header.
The If header's purpose is to define a filter with aseries of state lists. If the state of the event to which the header is applied does not match any of the specified state lists then the request MUST fail with a 412 Precondition Failed. If one of the described state lists matches the state of the resource then the request may succeed and respond with all events that satisfies the given filter.

#### 6.3.1. No-tag-list Production

**The No-tag-list production describes a series of state tokens. If** multiple No-tag-list productions are used then only one needs to match the state of the resource for the method to be allowed to continue.

If a method, due to the presence of a Depth or Destination header, is applied to multiple resources then the No-tag-list production MUST be applied to each resource the method is applied to.

#### 6.3.2. Tagged-list Production

**The tagged-list production scopes a list production. That is, it** specifies that the lists following the resource specification only apply to the specified resource. The scope of the resource production begins with the list production immediately following the resource production and ends with the next resource production, if any.

When the If header is applied to a particular resource, the Tagged-list productions MUST be searched to determine if any of the listed resources match the operand resource(s) for the current method. If none of the resource productions match the current resource then the header MUST be ignored. If one of the resource productions does match the name of the resource under consideration then the list productions following the resource production MUST be applied to the

resource in the manner specified in the previous section.

The same URI MUST NOT appear more than once in a resource production in an If header.

### 6.3.3. not Production

**Every state token is either current, and hence describes the state** of a resource, or is not current, and does not describe the state of a resource. The boolean operation of matching a state token to the current state of a resource thus resolves to a true or false value. The not production is used to reverse that value. The scope of the not production is the state-token immediately following it.

If: (Not <loctoken:write1> <locktoken:write2>)

When submitted with a request, this If header requires that all operand resources must not be locked with locktoken:write1 and must be locked with locktoken:write2.

## 6.4. Matching Function

**When performing If header processing, the definition of a matching** state token is as follows.

Matching state token: Where there is an exact match between the state token in the If header and any state token on the resource.

## 7. Status Code Extensions to HTTP/1.1

The following status codes are added to those defined in HTTP/1.1 [Fielding et al., 1997].

### 7.1. 207 Multi-Status

**The response provides status for multiple independent operations.**

### 7.2. 422 Unprocessable Entity

**The server understands the content type of the request entity, but** was unable to process the contained instructions.

### 7.3. 424 Method Failure

**The method was not executed on a particular resource within its** scope because some part of the method's execution failed causing the entire method to be aborted. For example, if a command in a PROPPATCH method fails then, at minimum, the rest of the commands will also fail with 424 Method Failure.

## 8. Multi-Status Response

**The default 207 Multi-Status response body is a text/xml HTTP entity** that contains a single XML element called multistatus, which contains a set of XML elements called response which contain 200, 300, 400, and 500 series status codes generated during the method invocation. 100 series status codes SHOULD NOT be recorded in a response XML element.

## 9. XML Element Definitions

**In the section below, the final line of each section gives the** element type declaration using the format defined in [Bray, Paoli, Sperberg-McQueen, 1998]. The "Value" field, where present, specifies futher restrictions on the allowable contents of the XML element using BNF (i.e., to further restrict the values of a PCDATA element).

### 9.1. eventrequest XML Element

```
<!ELEMENT eventrequest((subscribe | unsubscribe | eventquery | advertise)
+)>
```

### 9.2. subscribe XML element

```
<!ELEMENT subscribe(einfo, enotify, erule*, eauth*)>
<!ATTLIST subscribe
          sref  ID      #REQUIRED >
```

### 9.3. einfo XML element

```
<!ELEMENT einfo(edata*, eattributes*, estates*)>
<!ATTLIST einfo
          eid           ID      #REQUIRED
          ename         PCDATA  #IMPLIED
          etimestamp    CDATA   #IMPLIED
          eref          CDATA   #IMPLED>
```

Attributes:

| | |
|---|---|
| eid | An identifier which uniquely identiies the event. |
| ename | A short description for the event. |
| etimestamp | It is set to the time at which event notifications is sent and it is in UTC format. |
| eref | This points to URI which contains the data for this event. |

Content:

| | |
|---|---|
| edata | Pay load of event data. |

eattributes Event properties. This information is advertised by the event producer.

estates Valid states of the event. This information is advertised by the event producer. These properties are queryable and selectable for the event consumers.

### 9.4. edata XML element in 4

```
<!ELEMENT edata ANY >
<!ATTLIST edata
          content-type ID #REQUIRED
          content-length ID #REQUIRED >
```

Attributes:

content-type It defines the content format of the edata element. Valid values are:
xml the data is structured using XML
mime the data consists of mime message
base64 the data consists of binary information using base64 encoding

content-length It defines the content legth in bytes.

Content:

ANY pay load of the event data encoded in the format specified in content-type attribute.

### 9.5. eattributes XML element

```
<!ELEMENT eattributes(attribute+)>
```

Content:

attribute defines event properties as name/value pairs.

### 9.6. attribute XML element

```
<!ELEMENT attribute ANY >
<!ATTLIST attribute
          name    #PCDATA #REQUIRED
          type    #PCDATA #REQUIRED >
```

Attributes:

name identifies the attribute name. This attribute refers to the event identified by the einfo element.
type identifies the type of the event attribute value.

```
            Valid values are:
                    string  the attribute value is a string value
                    int     the attribute value is an integer value.
                    real    the attribute value is a decimal value
                    date    the attribute valie is a date value
                    xml     the data is structured using XML
                    mime    the data consists of mime message
                    base64  the data consists of binary information using
                            base64 encoding

    Content:

    ANY             pay load of the event attribute value encoded in the format
                    specified in type attribute.
```

### 9.7. estates XML element

```
    <!ELEMENT estates (vstate,cstate)>
    <!ELEMENT vstate  (state+)>
    <!ELEMENT cstate  (state)>
    <!ELEMENT state   (#PCDATA)>
```

### 9.8. enotify XML element

```
    <!ELEMENT enotify ( einfo+)>
```

### 9.9. erule XML element

```
    <!ELEMENT erule   ( term, (( and | or ),term)+)
    <!ELEMENT and     EMPTY>
    <!ELEMENT or      EMPTY>

    <!ELEMENT term   ( propname, propop, propvalue)>
    <!ELEMENT propname (#PCDATA)>
    <!ELEMENT propvalue ANY
    <!ATTLIST propvalue
              content-type ID #REQUIRED
              content-length ID #REQUIRED>

    <!ELEMENT propop (eq,lt,le,ge,gt)
    <!ELEMENT eq     EMPTY>
    <!ELEMENT le     EMPTY>
    <!ELEMENT lt     EMPTY>
    <!ELEMENT gt     EMPTY>
```

### 9.10. eauth XML element

```
<!ELEMENT eauth  #PCDATA
<!ELEMENT eauth
          content-type ID #REQUIRED
          content-length ID #REQUIRED >
```

### 9.11. unsubscribe XML element

```
<!ELEMENT unsubscribe(einfo, enotify, erule*, eauth*)>
<!ATTLIST unsubscribe
          sref  ID      #REQUIRED >
```

### 9.12. eventref XML Element

**Name: eventref**
Namespace: ENP:
Purpose: Identifies the event as a URI.
Value: URI ; See section 3.2.1 of [Fielding et al., 1997]

```
<!ELEMENT eventref (#PCDATA)>
```

### 9.13. eventid XML Element

**Name: eventid**
Namespace: ENP:
Purpose: Identifies the event uniquely. When the event is subscribed, servers returns the unique id using UUIDs.
Description:

```
<!ELEMENT eventid (src+, dst+) >
```

### 9.14. eventstatus XML Element

**Name: eventstatus**
Namespace: ENP:
Purpose: Specifies the status of the event. Valid values are COMPLETED, IN-PROGERSS, KILLED, MODIFIED, DELETED.

```
<!ELEMENT eventid (#PCDATA) >
```

## 10. Access Controls

**ENP provides three levels of access control mechanisms. The first** level of access is for all process related meta data which is owned by individual process i.e. event producers as per the access control rights assigned my these processes. Notification server will challenges the each process supply authentication information before it gives access to this data. Second level of access is read-only access to the notification content and data to event consumers. Third level of access controls are for the data genrated by the notification server.

## 11. Security Considerations

**The ENP will provide a level of security identical to the WebDAV or** HTTP/1.1 protocol. However, within the limits of the delivery system, security will be provided by the following principles:

(1). The right to create the notification requests will be protected by ACLs.

(2). A clear separation between self-notifications and group notifications will be enforced.

## 12. Author's Address

**Surendra Reddy**
Oracle Corporation
500 Oracle Parkway
M/S 6op3
Redwoodshores, CA 94065

Phone: +1(650) 506 5441
Fax: +1(650) 654 6205
Email: skreddy@us.oracle.com

Mark Leighton Fisher
Thomson Consumer Electronics
Indianapolis, IN
email: fisherm@indy.tce.com