Web Services Events (WS-Events) Version 2.0

16 July 2003

Authors:

Nicolas Catania (editor), Hewlett-Packard Company
Pankaj Kumar, Hewlett-Packard Company
Bryan Murray, Hewlett-Packard Company
Homayoun Pourhedari, Hewlett-Packard Company
William Vambenepe, Hewlett-Packard Company
Klaus Wurster, Hewlett-Packard Company

Copyright Notice

Abstract

This document describes Web Services Events (WS-Events) Version 2.0, an XML syntax and a set of processing rules for advertising, subscribing, producing and consuming Web Services Events. An Event is an abstract concept that is physically represented by a Notification. Notifications flow from Event producer to Event consumer using asynchronous or synchronous delivery modes (push/pull).

Status of this Document

This WS-Events Specification is an initial public draft release and is provided for review and evaluation only. Hewlett-Packard Company hopes to solicit your contributions and suggestions in the near future. Hewlett-Packard Company makes no warranties or representations regarding the specification in any manner whatsoever.

Table of Contents

1. Introduction

This document describes Web Services Events (WS-Events) Version 2.0, an XML syntax and a set of processing rules for advertising, subscribing, producing and consuming Web Services Events using a push and pull mode. In the push mode, the event notification producer calls a method (callback) on the event consumer passing one or more notifications as parameter. In effect the producer pushes asynchronously notifications to the consumer. In the pull mode, the consumer invokes methods on the producer the retrieve the buffered notifications. WS-Events defines the following:

- An extensible XML representation of event notifications.

- A simple subscription protocol between notification consumers, producers and brokers.
- Mechanisms to advertise and discover events.
- Support for both asynchronous and synchronous communication of notifications from the producer to the consumer through a push and pull mode.
- A lightweight XML syntax to describe event filters.

WS-Events does not define a general-purpose event mechanism, rather, it defines one suitable for the Web services infrastructure.

WS-Events builds upon the existing Web services framework. It uses Web Service Defnition Language (WSDL 1.1) [WSDL], and the XML schema specifications [XML Schema: Structures] and [XML Schema: Datatypes] to describe management interfaces and message structure. WS-Events is applicable to any version of SOAP and its message processing model, as well as to other protocols that may be described in WSDL documents.A mapping to WSDL 1.1 is included as part of this specification. However, there is ongoing work in the definition of WSDL 1.2 [WSDL 1.2] which may be useful in future versions of WS-Events. It is expected that a mapping of WS-Events to WSDL 1.2 will be made when that specification is released.

1.1 Notational Conventions

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 [KEYWORDS].

WS-Events is designed to work with the general Web Services framework including WSDL service descriptions and the [SOAP] message structure and message processing model. WS-Events should be applicable to any version of [SOAP]. The SOAP 1.1 namespace URI [URI] is used herein to provide detailed examples, but there is no intention to limit the applicability of this specification to a single version of SOAP.

All parts of this specification are normative, with the exception of examples and sections explicitly marked as "Non-Normative". In cases where this document and the WSDL or XML Schema Definitions differ, the WSDL and XSD files are considered normative.

1.2 Versions, Namespaces and Identifiers

This specification does not provision for an explicit version number in this syntax. Any future version of this specification will use a different namespace identifier. The namespace that MUST be used by implementations of this (dated) specification is:

http://devresource.hp.com/drc/specifications/wsmf/2003/07/events/

This URI designates a namespace that is under the control of this specification. The following namespaces are also used in this document:

Prefix	Namespace
xs	http://www.w3.org/2001/XMLSchema
wsdl	http://schemas.xmlsoap.org/wsdl/
soap	http://schemas.xmlsoap.org/wsdl/soap/
evt	http://devresource.hp.com/drc/specifications/wsmf/2003/07/events/

1.3 Compliance

An implementation is not WS-Events compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements defined in this specification.

A SOAP receiver MUST comply with this specification in order to process a WS-Events message successfully. A SOAP sender MUST NOT use the **http://devresource.hp.com/drc/specifications/wsmf/2003/07/events/** XML Namespace identifier

unless it is WS-Events compliant. A Web service provider MUST NOT use the **http://devresource.hp.com/drc/specifications/wsmf/2003/07/events/** XML Namespace identifier in its WSDL description unless it is WS-Events compliant.

1.4 Terminology

We introduce the following terms which are used throughout this document:

Event

An *event* is a change in the state of a resource or request for processing.

Event Producer

An *event producer* is an entity which generates notifications.

Event Consumer

An *event consumer* is a receiver of notifications.

Event Broker

An *event broker* is an entity which routes notifications. Brokers typically aggregate and publish events from other producers. An event broker can also apply some transformation to the notifications it processes.

Notification

A *notification* is an XML element representing an event. One or more notifications are emitted by an event producer and received or retrieved by one or more event consumers possibly through a broker.

Resource

A *resource* is defined in [URI] as:

"A resource can be anything that has identity. Familiar examples include an electronic document, an image, a service (e.g., 'today's weather report for Los Angeles'), and a collection of other resources. [...] The resource is the conceptual mapping to an entity or set of entities, not necessarily the entity which corresponds to that mapping at any particular instance in time. Thus, a resource can remain constant even when its content---the entities to which it currently corresponds---changes over time, provided that the conceptual mapping is not changed in the process."

A resource is identified by one or more URIs. If there is more than one URI, the most specific should be used (e.g. *http://www.example.com/news.html.en* instead of *http://www.example.com/news*).

Web Service

This specification uses a slightly modified definition of the one from the W3C WS-Architecture group. Namely it make *WSDL* the standard way to define and describe interfaces:

"A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by internet protocols."

This specification restricts this definition by making **WSDL** mandatory to describe a Web Service.

2. Examples

2.1 Event discovery example

This examples shows how an event producer such as a managed service communicates the list of events accessible by an event consumer like a management console.

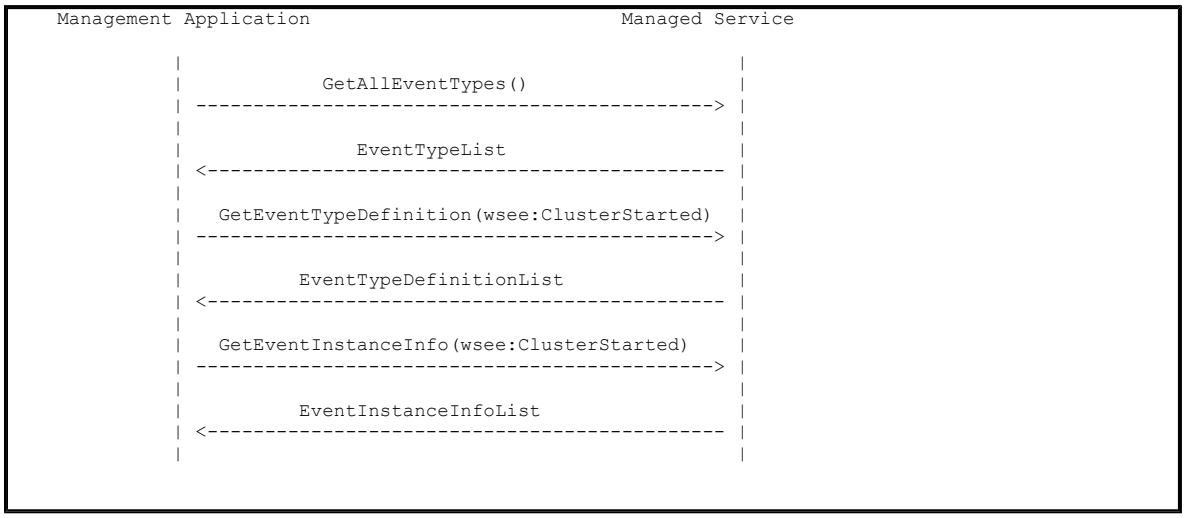
Usually, this discovery step happens before the subscription set-up using the methods

`GetAllEventTypes` and `GetEventTypeDefinition` to discover which events it is interested in.

An event notification consumer may be interested in events that have happened before it started its subscription if, for instance, it is the first time it monitors the service. By analogy, if a file were used to store events, the event notification consumer would read from either:

1. Last file position.
2. From the beginning of the file
3. From a random position in the file

This is done using the `GetEventInstanceInfo`. The management application would then know how much historical information it could retrieve assuming that this past events could shed some light on a future problem.



The management application would query the managed service for all the event notifications it can subscribe to. If one (say *wsee:ClusterStarted*) is of particular interest, it would ask for a more detailled definition and how many past instances can still be retrieved.

Below is an example of what the `EventTypeDefinition` and `EventInstanceInfo` could look like:

```
<EventTypeDefinition>
  <EventType>
    ns:ClusterStarted
  </EventType>
  <SchemaLocation>
    http://schemas.examples.com/2003/04/ClusterStarted.xsd
  </SchemaLocation>
  <SubscriptionURL>
    http://a.service.examples.com/
  </SubscriptionURL>
  <Description>
    Indicates that the WS execution environment cluster was started
  </Description>
  <SubscriptionMode>
    push
  </SubscriptionMode>
</EventTypeDefinition>

<EventInstanceInfo>
  <EventType>
    ns:ClusterStarted
  </EventType>
  <Available>
    200
  </Available>
</EventInstanceInfo>
```

2.2 Event Notification Subscription example

To achieve scalability, WS-Events uses a subscription mechanism by which an event consumer informs the producer that it is interested in receiving notifications.

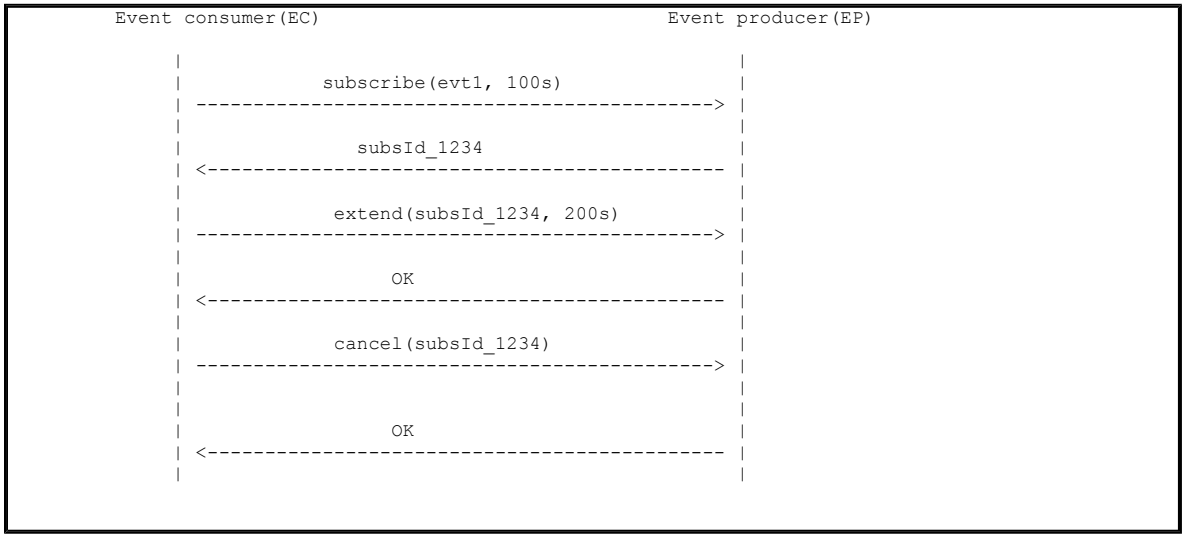
Subscriptions allow the notification producer to plan and allocate resources depending on the number of subscribers, event notification types and access modes and length of subscriptions.

Subscriptions have a limited duration in time. For instance, it might last for an hour. This ensures that if a subscriber went away and forgot to cancel its subscription, resources would not be held indefinitely and recovered by the event producer. Subscription can be renewed before it expires.

The diagram below illustrates an Event Consumer (*EC*) that subscribe to an event *evt1* published by a Event Producer (*EP*).

EC sends a request to EP indicating for which event it is interested to receive notification from as well as a suggested duration for the subscription. Since no callback URL was given it is a pull mode subscription.

If EP accepted the subscription request, it replies with a unique subscription ID. This ID is used by EC later when it invokes operations to extend or cancel the subscription.



2.3 Example of a Notification message

`Notifications` are used to communicate events. Typically zero or more notification are packed in a `NotificationList` and sent using operations described using WSDL. In this example the Notify method of the consumer is invoked by the producer to push a list of 1 notification over SOAP.

This example also demonstrates how the basic Notification format can be extended by a 3rd party (mngmt) to include other application specific information.

```
      <SOAP-ENV:Envelope
          xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
        <SOAP-ENV:Header>
        ...
        </SOAP-ENV:Header>
        <SOAP-ENV:Body>
[e01]     <evt:Notify xmlns:evt="http://devresource.hp.com/drc/specifications/wsmf/2003/07/events/"
                      xmlns:mngmt="http://samples.com/management">
[e02]       <evt:NotificationList>
[e03]         <evt:Notification Id="notification-1">
[e04]           <evt:Source>http://myservice.com</evt:Source>
[e05]           <evt:Type>http://openview.hp.com/shutdown</evt:Type>
[e06]           <evt:Timestamp>2001-04-02T13:20:00Z</evt:Timestamp>
[e07]           <evt:Duration>20S</evt:Duration>
[e08]           <evt:ExpiresOn>2001-04-02T15:20:00Z</evt:ExpiresOn>
[e09]           <evt:UUID>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d8</evt:UUID>
```

```
[e10]           <mngmt:Severity>warning</mngmt:Severity>
[e11]           <mngmt:Message>A message</mngmt:Message>
[e12]           <mngmt:CorrectiveMessage>Try this</mngmt:CorrectiveMessage>
[e13]         </evt:Notification>
[e14]       </evt:NotificationList>
[e15]     </evt:Notify>
        </SOAP-ENV:Body>
      </SOAP-ENV:Envelope>
```

`[e01-e15]` **Notify** call to push a notification to a subscriber.

`[e02-e14]` A list of notifications. In this case, there is only one notification in the list.

`[e03]` Start of a notification. The **Id** attribute identifies the notification in the list. In this example there is only one notification so the `Id` is optional. However when there is more than one notification in the list, the `Id` attribute SHOULD be used to make potential XML transformation easier (e.g. XML digital signature).

`[e04]` **Source** is the URI of the source of the notification.

`[e05]` The **Type** element identifies the kind of notification.

`[e06]` Each notification has a creation date captured by the **Timestamp** element. This time should be equal to the time of the associated event.

`[e07]` Events can be instantaneous but they also can have a duration. The optional element **Duration** contains the time period of the associated event.

`[e08]` A notification may expire. In that case a **ExpiresOn** element contains the date. This element is optional but if present, the MUST be greater than the date and time specified by the `Timestamp` element. A event producer could make use of this element to implement the policy to cache unretrieved notifications in the pull mode.

`[e09]` A notification has a identifier that is unique across time and space also called **UUID**. An element named `UUID` contains the identifier value. The UUID value is used to create causality chains and hierarchy of events.

`[e10-e12]` These are examples of extra elements from a foreign namespace that carry some extra data related to this notification. It demonstrates how the notification mechanism can be extended.

3. Event Notification Advertising and Discovery

WS-Events provides a publish/subscribe mechanism for a notification consumer to discover the list of events the producer exposes. This information can be retrieved in different ways including:

1. A standard API to support dynamic/run-time discovery.
2. An XML fragment published in a well-known registry (e.g. UDDI, event broker).
3. Some out-of-band mechanism (e.g. mail).

Currently, this specification addresses only the first method since it truly enables dynamic discovery of published events. The event producer can use the programmatic method to limit the scope of the events published based on the requesters identity and/or some internal policies. Solution based on a global registry to disclose events are more static and the producer loses access control rights on the information published this way.

This specification uses three operations and two reserved events to discover information about event notifications exposed by a resource.

3.1 Operations overview

The following methods are grouped in the `DiscoveryInterface` of the WSDL associated with this specification.

1. **GetAllEventTypes** takes no argument and returns a list of all the event types URI that can be subscribed to.
2. **GetEventTypeDefinition** takes a list (possibly empty) of event types and returns a list of `EventTypeDefinition` elements describing the corresponding types. When invoked without any argument, all subscribable event definitions are returned.
3. **GetEventInstanceInfo** takes a list (possibly empty) of event types and returns a list of `EventInstanceInfo` elements. `EventInstanceInfo` contains information on notifications of past events that the producer still holds. These past events may be of interest to a potential subscriber and could be retrieved using the pull mode.

To indicate runtime changes in the list of event definitions exposed by a producer, two reserved events are defined and sent over the implicit control subscription:

1. **NewEventType** indicates that a new event type is available for subscription.
2. **EventTypeUpdated** is used to signal a change in an event type definition. This is sent when any element or attribute of the `EventTypeDefinition` has changed. In particular this can be used to indicate when an event type becomes obsolete.

3.2 The `GetAllEventTypes` operation

This operation is used to retrieve all the event types (encoded as URIs) that can be subscribed to. The caller may or may not be able to associate a meaning to all the event type returned. In that case, further information on one or more type can be obtained by using the `GetEventTypeDefinition` operation.

Below is the structure returned by the call. It uses the common `EventTypeList` to encapsulate the event types. The list may be empty.

```
Schema Definition:

<element name="GetAllEventTypesResponse">
  <complexType>
    <sequence>
      <element ref="evt:EventTypeList"/>
    </sequence>
  </complexType>
</element>
```

3.3 The `GetEventTypeDefinition` operation

This method is used to provide detailed and meaningful information about an event type. As we have seen above, the type of an event is a simple unique string. There is no additional information associated with it. To enable true dynamic discovery, one should be able to get more complete info on an event type to make a decision whether the event should be subscribed to or not. `GetEventTypeDefinition` takes a list (possibly empty) of event types and returns a list of `EventTypeDefinition` elements describing the corresponding type.

When invoked without any argument, all event definitions that can be subscribed to are returned.

3.3.1 The `EventTypeDefinitionList` Element

This is the root element that encapsulates zero or more event type info elements.

```
Schema Definition:

<element name="EventTypeDefinitionList" type="evt:EventTypeDefinitionListType"/>
<complexType name="EventTypeDefinitionListType">
  <sequence>
    <element ref="evt:EventTypeDefinition" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

3.3.1.1 The `EventTypeDefinition` element

This is the main element that carries all the information describing an event type. To make the discovery process as exhaustive as possible, both human and machine targeted information are included in an `EventTypeDefinition` element:

```
Schema Definition:

<complexType name="EventTypeDefinitionType">
  <sequence>
    <element name="EventType" type="xs:anyURI"/>
    <element name="Obsolete" type="xs:anyURI" minOccurs="0"/>
    <element name="SchemaLocation" type="xs:anyURI"/>
    <element name="Description" type="string" minOccurs="0"/>
    <element name="SubscriptionURL" type="xs:anyURI"/>
    <element name="SubscriptionMode">
      <simpleType>
        <restriction base="token">
          <enumeration value="push"/>
          <enumeration value="pull"/>
          <enumeration value="pushAndPull"/>
          <enumeration value="none"/>
        </restriction>
      </simpleType>
    </element>

    <choice minOccurs="0">
      <element name="Aggregates" type="evt:EventTypeListType"/>
      <element name="CausalityExpression" type="string"/>
    </choice>
    <element name="Causes" type="evt:EventTypeListType" minOccurs="0"/>

    <any minOccurs="0" maxOccurs="unbounded" namespace="##other" processContents="lax"/>
  </sequence>
  <anyAttribute namespace="##other" processContents="skip"/>
</complexType>
```

Below is an explaination of the various attributes and elements:

1. **EventType** is the URI representing the event type and its definition. This information is used in a subscription call.
2. **Obsolete** is an optional element to indicate that the event type has been obsoleted. If non nil, the content of this element is the new type that superseeded this one. All the other infos contained in the subsequents elements were correct when this event type was active but may not longer be (e.g. URIs may not be dereferencable anymore).
3. **SchemaLocation** is the URI to the notification schema. It should point to an XML document that the subscriber can use to validate notifications against.
4. **Description** is an optional description of the purpose and semantic of the event. It should be explicit enough so that the requester can make a decision whether or not it should subscribe to the event.
5. **SubscriptionURL** is a dereferencable URL that points to a WSDL document of the service implementing the subscription interface.
6. **SubscriptionMode** is an indication if this event can be retrieved or not. If it is retrievable, the **pull and/or push mode** is indicated. The producer can chose the allowed retrival mode as needed, depending for instance on the identity of the requester, the network bandwith available, the frequency of the event.
7. Any other events that this type is dependant upon **aggregation** or **causality** (optional). This can be used to create high-level events from low-level ones and reduces the number of notificatins sent.
8. Any known event that depend on this type (optional).
9. `any` and `anyAttribute` can be used to extend this base `EventTypeDefinition` to create new

definitions.

3.4 The `GetEventInstanceInfo` operation

An event producer may keep some event instances stored for some time. A potential event consumer may be interested in retrieving events that occured before subscription time using the synchronous calls.

The `GetEventInstanceInfo` call gives a way for the potential subscriber to discover if the producer has stored some past events and if that is the case, how many and during what period of time.

If some notification can be retrieved, the producer gives an estimate of the time left before it will discard the event.

The data returned by that call is only informational and may have changed when the subscriber actually retrieve the notification depending on the policy used by the producer to manage its resources.

3.4.1 The `EventInstanceInfoList` Element

This is the root element that encapsulates zero, or more event instance info elements.

```
Schema Definition:

<element name="EventInstanceInfoList" type="evt:EventInstanceInfoListType"/>
<complexType name="EventInstanceInfoListType">
  <sequence>
    <element ref="evt:EventInstanceInfo" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

3.4.1.1 The `EventInstanceInfo` element

This element contains information about the set of notification instances that may be retrieved for a specific event type.

```
Schema Definition:

<complexType name="EventInstanceInfoType">
 <sequence>
   <element name="EventType" type="xs:anyURI"/>
   <element name="Available" type="xs:integer"/>
   <element name="LastInstance" minOccurs="0">
     <attribute name="UUID" type="xs:anyURI"/>
     <attribute name="Date" type="xs:dateTime"/>
     <attribute name="AvailableUntil" type="xs:dateTime"/>
   </element>
   <element name="FirstInstance"  minOccurs="0">
     <attribute name="UUID" type="xs:anyURI"/>
     <attribute name="Date" type="xs:dateTime"/>
     <attribute name="AvailableUntil" type="xs:dateTime"/>
   </element>
   <any minOccurs="0" maxOccurs="unbounded" namespace="##other" processContents="lax"/>
 </sequence>
 <anyAttribute namespace="##other" processContents="skip"/>
</complexType>
```

Below is an explaination of the various attribute and elements:

1. **EventType** is the URI representing the event type and its definition. This information is used in a subscription call.
2. **Available** is the number of instances that can be retrieved at the time of the call.
3. **LastInstance** is an optional element that contains info on the oldest instance that can be retrieved.

4. **FirstInstance** is an optional element that contails info on the newest notification that can be retrieved.
5. **UUID** Is the UUID of the event.
6. **Date** is the time when the event was created.
7. **AvailableUntil** is the time when notifications associated with the event will be discarded.

3.5 Notifications For Event Type Discovery

To indicate changes in the event definitions exposed by a producer, *two reserved* events are defined and can be subscribed to in order to get notifications when event type gets created, deleted or updated:

1. **NewEventType** indicates that a new event type is available for subscription.
2. **EventTypeUpdated** is used to signal a change in an event type definition. This is sent when any element or attribute of the event type info has changed. In particular this can be used to indicate when an event type becomes obsolete.

3.6 The `NewEventTypeNotification`

This event is used by a producer to inform a consumer that one or more new events are available for subscription. The URI for this event type is **http://devresource.hp.com/drc/specifications/wsmf/2003/07/events/new-event-type**. The base Notification type is restricted so that the value of the `Type` element is fixed and equal to the URI above. Also the new `EventTypeDefinition` has been append at end of the base notification.

```
Schema Definition:

  <complexType name="NewEventTypeNotification">
    <complexContent>
      <restriction base="evt:NotificationType">
        <sequence>
          <element ref="evt:Source"/>
          <element name="Type" type="xs:anyURI"
            fixed="http://devresource.hp.com/drc/specifications/wsmf/2003/07/events/new-event-type"/>
          <element ref="evt:Timestamp"/>
          <element ref="evt:ExpiresOn" minOccurs="0" maxOccurs="1"/>
          <element ref="evt:Duration" minOccurs="0"/>
          <element name="UUID" type="xs:anyURI"/>
          <element ref="evt:EventTypeDefinition"/>
        </sequence>
      </restriction>
    </complexContent>
  </complexType>
```

3.7 The `EventTypeUpdatedNotification`

This event is used by a producer to inform a consumer that one or more events definition have been updated. The URI for this event type is **http://devresource.hp.com/drc/specifications/wsmf/2003/07/events/event-type-updated**. The base Notification type is restricted so that the value of the `Type` element is fixed and equal to the URI above. Also the new `EventTypeDefinition` has been append at end of the base notification.

```
Schema Definition:

  <complexType name="EventTypeUpdatedNotification">
    <complexContent>
      <restriction base="evt:NotificationType">
        <sequence>
          <element ref="evt:Source"/>
          <element name="Type" type="xs:anyURI"
            fixed="http://devresource.hp.com/drc/specifications/wsmf/2003/07/events/event-type-update
          <element ref="evt:Timestamp"/>
```

```
          <element ref="evt:ExpiresOn" minOccurs="0" maxOccurs="1"/>
          <element ref="evt:Duration" minOccurs="0"/>
          <element name="UUID" type="xs:anyURI"/>
          <element ref="evt:EventTypeDefinition"/>
        </sequence>
      </restriction>
    </complexContent>
  </complexType>
```

4. Event Notification Subscription

4.1 Push/Pull Mode Overview

This specification describes two modes to retrieve events: synchronous and asynchronous also known as pull and push modes. In the push mode, the event notification producer calls a method (callback) on the event consumer passing one or more notifications as parameter. In effect the producer pushes asynchronously notifications to the consumer. In the pull mode, the consumer invokes methods on the producer the retrieve the buffered notifications.

While the push mode is what people associate naturally with an event subsystem, WS-Event also specifies a pull version to retrieve events. In that case, the consumer initiates the notifications retrieval by first calling the event producer. Notifications are returned by that call.

There is no general rule to chose between the two modes however the following should be considered to make a decision:

- Network bandwidth and connectivity: The push mode tends to be more network intensive since the notifications are typically sent as soon as they are created. Sometimes the network topology (firewalls) does not allow asynchronous traffic. In that case, the pull mode is preferable.
- Latency: The push mode is more suitable for real-time or near real time systems while the pull one does not, the granularity being the period used to do the pulling calls.
- Producer resources: The pull mode requires more resources on the producer side. It needs to buffer notifications between calls to retrieve them.

4.2 Subscription Request: The `Subscribe` message

An event notificaion consumer subscribes to events using the `Subscribe` message. It contains the information needed to set up the subscription: a selector for the types, a proposed duration for the subscription, optionally a filter and a callback for the asynchronous mode.

In the pull mode, the subscription request message contains enough information to create a pull based subscription:

1. An event selector that could be an EventType (URI), a regular expression matching event types or the string token **all** to match all the events exposed by the producer.
2. The expiration time is either an absolute time expressed in the producer timezone or the 'infinite' string token to indicates that the subscription should last forever..
3. An optional filter represented by a QName or a well formed XML document. A QName value can be used to reference a well known filter. It is expected that both the events producer and consumer have a common understanding of the filter capabilities. Alternatively any XML document representing a filter can be passed to the producer. The description of the syntax and sematic of such a filter is out of the scope of this document.

   When a filter is present, the producer will silently discard notifications that match the filter
4. A nil callback element. Because the callback is nil, this subscription is interpreted as a pull one.

```
 WSDL subscription message type

<!-- Subscription -->
<xs:element name="Subscribe">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="EventSelector" type="evt:EventSelectorType"/>
      <xs:element name="ExpirationTime"  type="xs:dateTime"/>
      <xs:element name="Filter"  type="evt:FilterType" minOccurs="0"/>
      <xs:element name="CallbackUrl" type="evt:CallbackType" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

In the push mode, the subscription to a set of event types is similar to the pull mode, except that an extra callback element is given. The callback contains a URL where `Notify` messages should be sent.

4.2.1 The `Notify` Message and `CallbackUrl` Element

In the push mode, notifications are still packed in a `NotificationList` element but the delivery mechanism is different: the producer sends the `NotificationList` to the consumer in an asynchronous fashion using a `Notify` message sent to the `CallbackUrl` specified at subscription time.

The `Notify` operation has a default SOAP over HTTP binding but more bindings will be added to future version of this specification as necessary.

WS-Events cannot standardize the WSDL port used by asynchronous event consumers because the WSDL port describes how a binding is associated with protocol-specific address.

The `CallbackType` is used by the event notification consumer to communicate one or more port(s) it wants the producer to push the Notification to. `CallbackType` leverages the port type definition from WSDL 1.1.

```
Schema Definition:

<xs:element name="Callback" type="evt:CallbackType"/>
  <xs:complexType name="CallbackType">
    <xs:sequence>
      <xs:element name="port" type="wsdl:tPort" maxOccurs="unbounded"/>
      <xs:any minOccurs="0" maxOccurs="unbounded" namespace="##other" processContents="lax"/>
    </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="skip"/>
</xs:complexType>
```

For instance this is what the callback of a service using the soap over http (see WS-Event.wsdl) binding could look like:

```
<evt:CallbackUrl>
  <evt:port binding="evt:NotifySoapHttpBinding">
    <soap:address location="http://example.org/aservice/notify"/>
  </evt:port>
</evt:CallbackUrl>
```

The event notification producer would forward notifications associated with this subscription to the given HTTP URL using the SOAP over HTTP binding.

4.3 Subscription Response

The response to the a subscription request is either an opaque string that is used as a subscription ID or a Fault element containing the fault message.

```
  WSDL subscription response message type


  <xs:element name="SubscribeResponse">
    <xs:complexType>
```

```
        <xs:choice>
          <xs:element name="SubscriptionId" type="xs:string"/>
          <xs:element name="Fault" type="xs:string"/>
        </xs:choice>
      </xs:complexType>
    </xs:element>
```

If the expiration time is not acceptable for the producer. An error message is returned with a suggested expiration time.

If the filter is unknown to the producer, an error message should be returned.

4.4 Extending Subscription

Subscription MAY have a finite duration. If that is the case, a subscriber could extend a subscription before it expires. This is done using the `ExtendSubscription` call to provide a new `ExpirationTime` for the given `SubscriptionId`. The new expiration time should be greater than the current one.

If the subscription ID is invalid, the producer MUST return a fault message.

4.5 Cancelling Subscription

The message `CancelSubscription` containing the `SubscriptionId` to be cancelled can be sent to the producer to end a subscription. If the subscription ID is invalid, the producer MUST return a fault message.

5. Event Notification Retrieval (Pull Mode)

In the push mode, the callback entry point is used to forward event notification to the consumer. In the pull mode, some standard getters are defined to do so. Currently three operations are defined:

- GetEventsSinceUUID: Retrieve all events that happened after the one with a given UUID.
- GetEventsSinceDate: Retrieve all events that occured on or after the given date.
- GetEventsRangeByDate: Retrieve all events that occured on or after the start date up to the end date (included).

For these 3 calls, the first element in the request document is named `SubscriptionId`. It is a string which value should be the one returned by a successful subscription call. All the selectors apply to a specific subscription. Events that are not part of the original subscription will not be matched.

These 3 calls return a `NotificationList` that contains zero or more `Notification` elements matching the request.

These 3 calls MUST return a fault if the `SubscriptionId` did not exist or if it does not belong to the caller or if it refers to a push subscription.

The next sections cover the specifics for each call

5.1 GetEventsSinceUUID

The `EventId` element specifies the event uuid from which events will be returned. If this a valid UUID and if the producer still has a copy of the event, the list returned start with this event followed by any events that occured after if any.

If the UUID is wrong or if the producer does not hold a copy of the event anymore, a fault message MUST be returned. This could be the indication that the consumer has missed some events.

```
        WSDL:
```

```
<!-- Get since Id -->
<xs:element name="GetEventsSinceUUID">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="SubscriptionId" type="xs:string"/>
      <xs:element name="EventId" type="xs:anyURI"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

5.2 GetEventsSinceDate

The `Date` element specifies the date and time from which events will be returned.

If the date is in the future a fault message MUST be returned.

```
WSDL:


<!-- Get since Id -->
<xs:element name="GetEventsSinceDate">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="SubscriptionId" type="xs:string"/>
      <xs:element name="Date" type="xs:dateTime"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

5.3 GetEventsRangeByDate

The `BeginDate` element specifies the date and time from which events will be returned. The `GetEventInstanceInfo` call can be used to get a sample of the producer clock and timezone indication.

The `EndDate` element specifies the ending date and time from which events will be returned. The date should be expressed in the producer timezone. It should be greater than the value specified in the `BeginDate` element.

If a date is in the future or if the end date is smaller than the begin date value, a fault message MUST be returned.

```
WSDL:


<!-- Get since Id -->
<xs:element name="GetEventsRangeByDate">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="SubscriptionId" type="xs:string"/>
      <xs:element name="BeginDate" type="xs:dateTime"/>
      <xs:element name="EndDate" type="xs:dateTime"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

6. Event Notification Syntax

A notification is the XML representation of one state change, also called an event in the producer.

6.1 The `NotificationList` Element

This is the root element that encapsulates zero or more Notifications. It is a named complex type used

in the WSDL to define the message that carries notifications. There are no constraints on how the `Notification` elements are grouped in a `NotificationList`. For instance, a `NotificationList` can contain `Notification` elements of various types or from different sources in any order.

```
Schema Definition:

<element name="NotificationList" type="evt:NotificationListType"/>
<complexType name="NotificationListType">
  <sequence>
    <element ref="evt:Notification" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

6.1.1 The `Notification` Element

The `Notification` element is the basic building block used to convey information about events. It contains the minimal set of information to:

- Be able to order event according to time but also causality. This allows representation of chain of events.
- Support aggregation of events.
- Provide for extension to support more complex notification syntax.

```
Schema Definition:

<element name="Notification" type="evt:NotificationType"/>
<complexType name="NotificationType">
  <sequence>
    <element ref="evt:Source"/>
    <element ref="evt:Type"/>
    <element ref="evt:Timestamp"/>
    <element ref="evt:Duration" minOccurs="0"/>
    <element ref="evt:ExpiresOn" minOccurs="0"/>
    <element ref="evt:UUID"/>
    <any minOccurs="0" maxOccurs="unbounded" namespace="##other" processContents="lax"/>
  </sequence>
  <anyAttribute namespace="##other" processContents="skip"/>
</complexType>
```

While a notification is uniquely identified by its `UUID`, a notification is also uniquely identified by the result of the concatenation of the content of the **Source**, **Type** and **Timestamp** elements. That is to say that 2 events of the same type cannot happen at a same time in a same event producer. It is the responsability of the source to make sure that the time granularity used to timestamp notifications is small enough so that two subsequent occurences of the same event have different `Timestamp` values. This is equivalent to saying that given a source, no two events of the same type can happen at the same time.

6.1.1.1 The `Source` Element

This element contains the URI of the event producer. It MAY be different from the URL of the Notification sender. For instance an event consumer MAY subscribe to a broker to receive notifications when a particular event happens. In this case, the address of the event source and the notification source are different.

```
Schema Definition:

<element name="Source" type="xs:anyURI"/>
```

The notification source URL MAY be different from the one of the sender of the notification if, for instance, there is a broker acting as an intermediary between the event producer and consumer.

6.1.1.2 The `Type` Element

AnyURI describes the type of the notification. Unlike the `Type` element used in subscription and advertising, wildcard are not allowed and any kind of structure of the type space is ignored.

```
Schema Definition:

<element name="Type" type="xs:anyURI"/>
```

6.1.1.3 The `Timestamp` Element

The date and time that uniquely identify the instant when the notification was created. This time should be equal to the time the event or events which generated this notification happened. This is not always the same as the time the notification itself was created.

The `dateTime` XML Schema type can have an arbitrary number of decimal after the second field. Howver when the granularity of the source clock is too big, it may happen that two or more event notifications are created with the same `Timestamp` value. In this case it is not possible to order the nofication according to time.

```
Schema Definition:

<element name="Timestamp" type="xs:dateTime"/>
```

6.1.1.4 The `Duration` Element

Events can be instantaneous but they also can last in time. The time the event lasts can be set in the optional **Duration** element. If this element is absent, it means that the associated event had a duration time of 0s.

```
Schema Definition:

<element name="Duration" type="evt:DurationType"/>
```

6.1.1.5 The `ExpiresOn` Element

An optional element to indicate how long the notification is valid for. Events happen at a single point in time but notification can last longer since they are a physical representation of the event that propagates through systems.

If `ExpiresOn` is present, the values MUST be greater than one in `Timestamp`.

If omitted, the notification will never expire.

A system SHOULD not propagate any notification that have expired though the mechanisms described in this specification.

```
Schema Definition:

<element name="ExpiresOn" type="evt:DateTimeType"/>
```

6.1.1.6 The `UUID` Element

The UUID element uniquely identify an event across time and space. The implementation is free to use any algorithm to generate UUID. However, this specification recommends that this well known algorithm and format (See UUIDType) should be used. The recommended UUID generation algorithm does not require any centralized authority and supports high generation rate.

7. Common Types

7.1 The `EventTypeList` and `EventType` Element Types

`EventType` is a URI that is not intended to be dereferencable. Even when that is the case, there is currently no way to retrieve the event definition from an event type directly. Instead the client must use the `GetEventTypeDefinition` call on the event producer.

`EventTypeList` is used to encapsulate 0 or more `EventType`.

```
Schema Definition:

<element name="EventTypeList" type="evt:EventTypeListType"/>
<complexType name="EventTypeListType">
  <sequence>
    <element name="EventType" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

7.2 The `UUID` Element Type

[Definition: UUID] The UUID element uniquely identify an event across time and space.

This document recommends the following UUID generation algorithm and format [UUID]. The generation does not require any centralized authority and support high generation rate. UUIDs are statistically unique.

The UUID format is made of the concatenation of the `uuid:` scheme [uuid Scheme]and a pseudo random number. The random number part is made of hexadecimal digits arranged in the common **8-4-4-4-12** format pattern.

Here is an example of such a UUID:

```
uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d8
```

7.3 The `dateTime` Type

The XML Schema type is used many time in this specification. Eventhough the time is part of the value of an `xs:dateTime` attribute or element, it is RECOMMENDED to use the UTC timezone for consistency. However, local timezone MAY be used.

8. Transport Considerations

8.1 Message encoding

The Web Services community has defined two XML encoding styles that map between SOAP data models and SOAP messages: RPC style with SOAP encoding (*RPC/encoded*) and Document style with literal encoding (*document/literal*). Having two accepted styles creates a degree of ambiguity that can produce potential conflicts. This specification recognizes both styles, but recommends standardizing on the document/literal style in order to guarantee interoperability.

With RPC/SOAP encoded style, the method calls, parameters and return values are mapped to XML elements according to the schema described in Section 5 of the SOAP [SOAP] specification. Section 5 also describes how the RPC/SOAP encoded style handles the request/response sequence and exception serialization. This style constrains each element in the message Body. However, the message as a whole is not constrained by any schema. As a consequence, schema validation cannot happen on the `Body` of the SOAP message. Additionally SOAP encoding allows the use of `id` and `href` attributes to reference elements and avoid duplication.

On the other hand, the Document style with literal encoding means that the `Body` of the SOAP message conforms to an XML schema typically available as `message` elements from the target WSDL. With literal encoding, there is no standard rule for encoding a method invocation as a SOAP message.

Even though RPC/SOAP encoded style was historically developed first, the web services industry is adopting and recommending the document/literal style. As a result, the major web services standardization bodies (WS-I, W3C WSDL) recommend against the use of the RPC/SOAP encoding style. To be compatible with the rest of the industry this specification also adopts the document/literal style to describe defined interfaces and messages.

8.2 Default Binding

> *Messages SHOULD be sent using HTTP/1.1.*

SOAP 1.1 defines only one binding: HTTP 1.1. To comply with the WS-I basic profile all messages SHOULD be sent over HTTP/1.1 and comply with section 4.3 of the basic profile 1.0 [WS-I-basic-profile]. In the future other transports MAY be supported.

8.3 Reliability

Reliability is a key feature for event systems. Today Web services are widely relying on HTTP, a non-reliable point-to-point protocol. However, Web services may use other transport protocols to provide reliability, or one of the SOAP-based reliability mechanisms, such as WS-ReliableMessaging [WS-ReliableMessaging]. To address reliability needs, the appropriate protocol should be chosen. Nevertheless, all WS-Events implementations MUST support SOAP HTTP binding.

9. Security

Security for Web services is a big concern for the industry. Since WS-Events is built on top of WS technology, some recommendations with regard to security should be provided to prevents attacks. Here is a non-exhaustive list of examples of potential threats:

1. A malicious party could spoof a subscription packet and use the ID to cancel the subscription, impersonating the original subscriber.
2. An attacker could generate extra meaningless event notifications, in the push mode.

WS-Events require some confidentiality and/or integrity at the transport level, as well as some access control mechanisms.

9.1 Transport

> *When integrity and/or confidentiality of the SOAP payload are required at the transport level, HTTPS SHOULD be used.*

Some attempts have been made to provide transport level security for XML payloads with WS-Security and XML firewalls. However, in their current state, these solutions fall short of creating a usable framework to create secure sessions for the transport of SOAP messages. Furthermore, how WS-Security affects performance is a concern. There are other concerns also, such as the slow rate WS-Security is being adopted and the fact that WS-I current basic profile was left out of WS-Security because of its shortcomings. The current recommendation is to only use HTTPS as a transport level security.

> *When the integrity and/or confidentiality of the SOAP payload is required at the transport level in a multi-hops path, WS-Security MAY be used.*

HTTPS has been around for a while now, is widely accepted and provides a robust way of securing the transport of event notifications. However, because HTTPS cannot be relayed, security applies only to the first hop. If the peer is not the final destination (e.g, a broker), the sender cannot assume anything with regard to the security used by the peer to forward the data to the final recipient. In that scenario, WS-security COULD be used to provide end-to-end security.

9.2 Authentication and Authorization

Access control MAY be applied to the subscription and notification operations.

When authentication is needed, HTTPS with mutual authentication is RECOMMENDED.

For the access control part, most of the modern execution environments support client side authentication and provide a way to map SSL client side certificates to their proprietary access control mechanisms. Until the Web services community comes up with a standard solution, client side certificates should be used to perform authentication and the access control rights should be derived from the assessed identity.

HTTP Basic authentication over SSL MAY be used.

Even though Web services is a machine-to-machine framework, HTTP basic authentication (username/password) MAY be used when the security requirements are very light or when XML cryptography is not suitable for the environement (heavyweight). SSL MUST be used to perform the HTTP authetication process.

When going across firewalls, HTTPS with client side authentication SHOULD be used.

The adoption of HTTPS and a firewall-friendly transport will facilitate adoption of WS-Events as a cross-enterprise solution. However, this flexibility comes at a cost: firewalls cannot apply security policies on an encrypted stream, in effect by-passing corporate security. While this has been a concerned for some time now, it is a current practice. Since the authentication is now done at the end-point, HTTPS with client side authentication is recommended to carry WS-Events between two distinct security domains.

Access control SHOULD be implemented at the Interface level.

Since all the operations of the same type (e.g, monitoring, configuration) are grouped in Interfaces, one should be able to specify its access level down to that level since these operations belong to the same management domain. A more coarse granularity at the Service level COULD also be achieved, as well as, a finer one at the Operation level.

10. Future Directions (Non-Normative)

There are several areas which WS-Events may address in the future. Some of these include the following:

- In the pull mode, a generic `GetEvents` call which would take an expression defined in a grammar to select events. These expressions would be able to select events according to time and/or causality.
- A filtering syntax and semantic to indicate at subscription time a complex condition under which event notification should be sent. For instance if there is an event that is generated when a CPU load changes, the subscriber should be able to specify that it wishes to receive the associated notification when the CPU load reaches 95% and above.
- A new call to retrieve the EventTypes associated with a given subscription. The producer would return the list of event types subscribed. This would be useful when the 'all' selector or a regular expression was used at subscription time.
- Brokering is an important concept in events systems. The current version of this specifications touches on that but needs to explain this concept in more details. More specifically, how a broker can relay event notification advertising and subscription and how notification are transmitted through one or more broker in both the push and pull modes.
- Fault messages need more details.
- The callback element is a URL and assumes SOAP over HTTP. However to support multiple binding, the URL could be a WSDL URL, forcing the producer to retrieve the consumer WSDL which would be inefficient. A better solution would be for the consumer to send a fragment of its WSDL containing the `service` element and all the endpoints (`ports`) for the `bindings` defined by this specification.

- EventTypeDefinition for the `NewEventType` and `EventTypeUpdated` is missing.

11. Appendix A: Extended notification examples

This appendix contains examples of nofitication schemas extending the basic notification.

11.1 Notification for a management plate form

Below is an example of a notification type that can be used to notify a management console. The schema defines a `EventSeverityType` and a `RelationType` types. `EventSeverityType` is used to indicates the severity of the event according the source rating policy. `RelationType` is used to indicate what was the change in the managed object relationships that triggered the notification.

```
<?xml version="1.0" encoding="utf-8"?>

<s:schema xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://devresource.hp.com/drc/specifications/wsmf/2003/07/events/notification-info"
  xmlns:core="http://devresource.hp.com/drc/specifications/wsmf/2003/07/foundation/"
  targetNamespace="http://devresource.hp.com/drc/specifications/wsmf/2003/07/events/notification-info
  version="2.0" elementFormDefault="qualified">

  <s:simpleType name="EventSeverityType">
    <s:annotation>
      <s:documentation>
        This type represents enum used to indicate the severity of an
        event. This is a standard list of severity values used in
        management software.
      </s:documentation>
    </s:annotation>


    <s:restriction base="s:string">
      <s:enumeration value="Critical" />
      <s:enumeration value="Major" />
      <s:enumeration value="Minor" />
      <s:enumeration value="Warning" />
      <s:enumeration value="Normal" />
      <s:enumeration value="Unknown" />
    </s:restriction>
  </s:simpleType>

  <s:complexType name="NotificationInfoType">
    <s:sequence>
      <s:element name="Severity" type="tns:EventSeverityType"/>
      <s:element name="Message" type="s:string"/>
      <s:element name="ResourceHostName" type="s:string" />
      <s:element name="CorrectiveMessage" type ="s:string" nillable="true" />
      <s:element name="ApplicationName" type="s:string" nillable="true" />
    </s:sequence>
  </s:complexType>

  <!-- NotificationInfo -->
  <s:annotation>
    <s:documentation>
      This type represents the operational notification information
      returned by the managed object.
    </s:documentation>
  </s:annotation>


  <s:element name="NotificationInfo" type="tns:NotificationInfoType"/>


  <s:simpleType name="RelationType">
    <s:annotation>
      <s:documentation>
        This type represents the relational notification information
        returned by the managed object.
      </s:documentation>
    </s:annotation>

    <s:restriction base="s:string">
```

```
        <s:enumeration value="RelationCreate" />
        <s:enumeration value="RelationDelete" />
      </s:restriction>
    </s:simpleType>

  </s:schema>
```

12. References

[URI]
RFC2396. *Uniform Resource Identifiers (URI): Generic Syntax*, T. Berners-Lee, R. Fielding, L. Masinter, Authors. Internet Engineering Task Force, August 1998. Available at http://www.ietf.org/rfc/rfc2396.txt. (See http://www.ietf.org/rfc/rfc2396.txt.)

[KEYWORDS]
RFC2119. *Key words for use in RFCs to Indicate Requirement Levels* , S. Bradner, Author. Internet Engineering Task Force, June 1999. Available at http://www.ietf.org/rfc/rfc2119.txt. (See http://www.ietf.org/rfc/rfc2119.txt.)

[SOAP]
Simple Object Access Protocol (SOAP) 1.1 , D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S. Thatte, D. Winer, Editors. World Wide Web Consortium, 8 May 2000. This version of the Simple Object Access Protocol 1.1 Note is http://www.w3.org/TR/2000/NOTE-SOAP-20000508. The latest version of Simple Object Access Protocol 1.1 is available at http://www.w3.org/TR/SOAP. (See http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.)

[WSDL]
Web Services Description Language (WSDL) 1.1, E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, Authors. World Wide Web Consortium, 15 March 2002. This version of the Web Services Description Language Note is http://www.w3.org/TR/2001/NOTE-wsdl-20010315. The latest version of Web Services Description Language is available at http://www.w3.org/TR/wsdl. (See http://www.w3.org/TR/2001/NOTE-wsdl-20010315.)

[WSDL 1.2]
Web Services Description Language (WSDL) Version 1.2 , R, Chinnici, M. Gudgin, J. Moreau, and S. Weerawarana, Editors. World Wide Web Consortium. The latest version of Web Services Description Language, Version 1.2 is available at http://www.w3.org/TR/wsdl12. (See http://www.w3.org/TR/2003/WD-wsdl12-20030303.)

[UUID]
UUIDs and GUIDs , Paul J. Leach, Rich Salz, Authors. Internet Engineering Task Force, February 1998. Available at latest version of UUIDs and GUIDs is available at http://hegel.ittc.ukans.edu/topics/internet/internet-drafts/draft-l/draft-leach-uuids-guids-01.txt. (See http://hegel.ittc.ukans.edu/topics/internet/internet-drafts/draft-l/draft-leach-uuids-guids-01.txt.)

[uuid Scheme]
The uuid: URI scheme , Charlie Kindel, Author. Internet Engineering Task Force, February 1997. Available at latest version of the uuid: scheme is available at http://www.globecom.net/ietf/draft/draft-kindel-uuid-uri-00.html. (See http://www.globecom.net/ietf/draft/draft-kindel-uuid-uri-00.html.)

[WS-ReliableMessaging]
Web Services Reliable Messaging Protocol (WS-ReliableMessaging) , BEA, IBM, Microsoft, TIBCO, March 2003. Available at latest version of the WS-ReliableMessaging specification is available at http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-reliablemessaging.asp. (See http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-reliablemessaging.asp.)

[XML 1.0]
Extensible Markup Language (XML) 1.0 (Second Edition), T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler, Editors. World Wide Web Consortium, 10 February 1998, revised 6 October 2000. This version of the XML 1.0 Recommendation is http://www.w3.org/TR/2000/REC-xml-20001006. The latest version of XML 1.0 is available at http://www.w3.org/TR/REC-xml. (See http://www.w3.org/TR/2000/REC-xml-20001006.)

[XML Namespaces]
Namespaces in XML, T. Bray, D. Hollander, and A. Layman, Editors. World Wide Web Consortium, 14 January 1999. This version of the XML Information Set Recommendation is http://www.w3.org/TR/1999/REC-xml-names-19990114. The latest version of Namespaces in XML is available at http://www.w3.org/TR/REC-xml-names. (See http://www.w3.org/TR/1999/REC-xml-names-19990114.)

[XML Schema: Structures]
XML Schema Part 1: Structures, H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn, Editors. World Wide Web Consortium, 2 May 2001. This version of the XML Schema Part 1 Recommendation is http://www.w3.org/TR/2001/REC-xmlschema-1-20010502. The latest version of XML Schema Part 1 is available at http://www.w3.org/TR/xmlschema-1. (See http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.)

[XML Schema: Datatypes]
XML Schema Part 2: Datatypes, P. Byron and A. Malhotra, Editors. World Wide Web Consortium, 2 May 2001. This version of the XML Schema Part 2 Recommendation is http://www.w3.org/TR/2001/REC-xmlschema-2-20010502. The latest version of XML Schema Part 2 is available at http://www.w3.org/TR/xmlschema-2. (See http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.)

[IETF RFC 2616]
Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Authors. Internet Engineering Task Force, June 1999. Available at http://www.ietf.org/rfc/rfc2616.txt. (See http://www.ietf.org/rfc/rfc2616.txt.)