Technische Universiteit
Eindhoven
University of Technology

Den Dolech 2, 5612 AZ Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
www.tue.nl

XES

Standard Definition

Author
Christian W. Günther and Eric Verbeek

Date
October 2, 2012

Version
1.2

1 Introduction

Event logs, as they occur in practice and research, can take a plethora of different forms and instantiations. Every system architecture that includes some sort of logging mechanism has so far developed their own, insular solution for this task.

XES is an XML-based standard for event logs. Its purpose is to provide a generally-acknowledged format for the interchange of event log data between tools and application domains. Its primary purpose is for process mining, i.e. the analysis of operational processes based on their event logs. However, XES has been designed to also be suitable for general data mining, text mining, and statistical analysis.

When designing the XES standard, the following goals have been used as guiding principles.

Simplicity Use the simplest possible way to represent information. XES logs should be easy to parse and to generate, and they should be equally well human-readable. In designing this standard, care has been taken to take a pragmatic route wherever that benefits an ease of implementation.

Flexibility The XES standard should be able to capture event logs from any background, no matter what the application domain or IT support of the observed process. Thus, XES aims to look beyond process mining and business processes, and strives to be a general standard for event log data.

Extensibility It must be easy to add to the standard in the future. Extension of the standard should be as transparent as possible, while maintaining backward and forward compatibility. In the same vein, it must be possible to extend the standard for special requirements, e.g. for specific application domains, or for specific tool implementations.

Expressivity While striving for a generic format, event logs serialized in XES should encounter as little loss of information as possible. Thus, all information elements must be strongly typed, and there must be a generic method to attach human-interpretable semantics to them.

Since XES strives to be a generic interchange format, only those elements which can be identified in virtually any setting are explicitly defined by the standard. All further information is deferred to optional attributes, which may be standardized (in terms of their semantics) by external extensions.

2 The XES Meta-model

The UML 2.0 class diagram shown in Figure 2.1 describes the complete meta-model for the XES standard. In the following, the basic components and concepts of XES will be introduced in more detail.

2.1 Basic Structure

The basic hierarchy of an XES document follows the universal structure of event log information.

2.1.1 Log

On the top level there is one *log* object, which contains all event information that is related to one specific process. Examples for processes are:

- Handling insurance claims
- Using a complex x-ray machine
- Browsing a website

Tag name for the log object in the XML serialization of XES: `<log>`

Attributes of the XML `<log>` tag:

Attribute key	Attribute type	Required	Description
xes.version	xs:decimal	Yes	The version of the XES standard the document conforms to (e.g., “1.0”).
xes.features	xs:token	Yes	A whitespace-separated list of optional XES features this document makes use of (e.g., “nested-attributes”). If no optional features are used, this attribute must have an empty value.

Example:

```
<log xes.version="1.1" xes.features="nested-attributes">
```

2.1.2 Trace

A log contains an arbitrary number of *trace* objects. Each trace describes the execution of one specific instance, or case, of the logged process. Examples of a trace are:

- One specific insurance claim

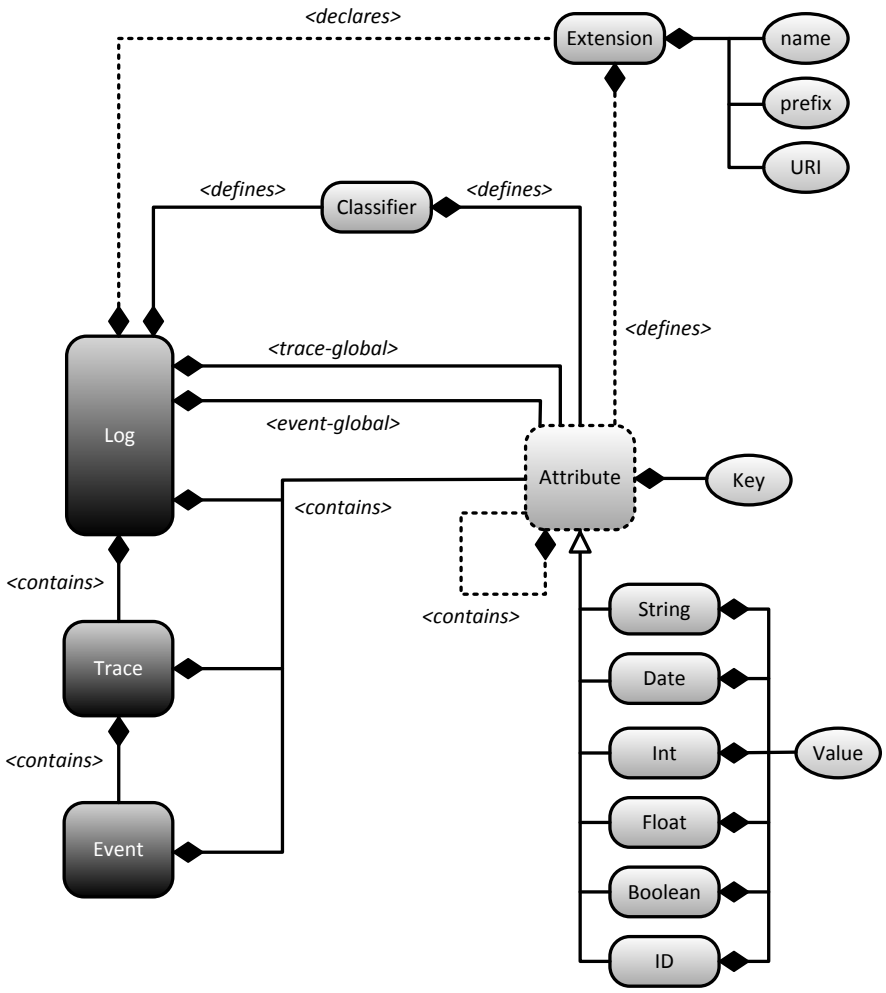

Figure 2.1: The UML 2.0 class diagram for the complete meta-model for the XES standard

- One examination in which the x-ray machine is employed
- One visit of the website, by one specific user

Tag name for the trace object in the XML serialization of XES: `<trace>`
No XML attributes are defined for the `<trace>` tag.

2.1.3 Event

Every trace contains an arbitrary number of *event* objects. Events represent atomic granules of activity that have been observed during the execution of a process. As such, an event has no duration. Examples of an event are:

- Recording the client's personal information in the database has been completed
- One picture is taken by the x-ray machine
- An image has been downloaded by the web browser

Tag name for the event object in the XML serialization of XES: `<event>`

No XML attributes are defined for the `<event>` tag.

2.2 Attributes

The log, trace, and event objects contain no information themselves. They only define the structure of the document. All information in an event log is stored in *attributes*. Attributes describe their parent element (log, trace, etc.).

All attributes have a string-based *key*. The XES standards requires for each attribute key the following:

- Attribute keys must *contain no whitespace characters*.
- Keys must be *unique within their enclosing container* (e.g., only one attribute with key “id” per trace).

Logs, traces, and events each contain an arbitrary number of attributes. There are six types of attribute, each defined by the type of data value they represent.

2.2.1 String

String attributes hold literal information which is generally untyped and of arbitrary length.

In the XML representation of XES, string attribute values are stored as `xs:string` data type.

Example:

```
<string key="name" value="Tom"/>
```

2.2.2 Date

Date attributes hold information about a specific point in time (with milliseconds precision).

In the XML representation of XES, string attribute values are stored as `xs:dateTime` data type.

Example:

```
<date key="name" value="2009-11-25T19:45:32.345+02:00"/>
```

2.2.3 Int

Int attributes hold a discrete integer number (with 64bit long precision).

In the XML representation of XES, string attribute values are stored as `xs:long` data type.

Example:

```
<int key="counter" value="236366"/>
```

2.2.4 Float

Float attributes hold a continuous floating-point number (with 64bit double precision).

In the XML representation of XES, float attribute values are stored as `xs:double` data type.

Example:

```
<float key="percentage" value="75.68"/>
```

2.2.5 Boolean

Boolean attributes hold a boolean value which can be either true or false.

In the XML representation of XES, boolean attribute values are stored as `xs:boolean` data type.

Example:

```
<boolean key="success" value="true"/>
```

2.2.6 ID

ID attributes hold id information which is generally a UUID.

In the XML representation of XES, string attribute values are stored as `xs:string` data type.

Example:

```
<id key="customer" value="f81d4fae-7dec-11d0-a765-00a0c91e6bf6"/>
```

2.3 Nested Attributes

For providing maximum flexibility in data storage, XES allows nested attributes, i.e. attributes can themselves have child attributes. While this feature is necessary for encoding certain types of information efficiently, it is *optional for tools to implement nested attributes*, i.e. this feature is not strictly required in order to be compliant to the XES standard.

If a document features nested attributes, it should announce this fact to the parser via the `xes.features` attribute of the `<log>` tag, by containing the token `nested-attributes`.

If an XES parser implementation does not support nested attributes, it must nevertheless be able to parse documents which feature nested attributes. These implementations should transparently ignore and discard any nested attributes, and, where feasible, alert the user to the fact that some information may not be available.

Example:

```
<string key="city" value="Bearlin">
    <boolean key="spellchecked" value="false"/>
</string>
```

2.4 Global Attributes

The log object holds two lists of *global attributes for the trace level and for the event level*. Global attributes are attributes that are understood to be *available* and *properly defined* for *each element*

on their respective level throughout the document. This means, a global attribute on the event level must be available for every event in every trace. A global attribute on the trace level, on the other hand, must be properly defined for each trace in the log.

Global attributes are defined within respective `<global>` tags, which are child elements of the <log> tag in a document.

Example:

```
<global scope="event">
    <string key="name" value=""/>
</global>
```

The above example would define that every event in the document has a valid string attribute with key “name”. In the definition, the value of the attribute is *only significant* in case a trace or event *needs to be created* for which *no value is provided* for the global attribute. In that case, the value of the definition will be used as the value for the global attribute. In all other cases, the value of the attribute is insignificant, and can thus be discarded.

Global attributes are a required features for XES standard compliance. Nevertheless, a defensive approach is recommended with respect to global attributes: All XES serialization implementations should attempt to define all global attributes in the log, and not define global attributes where complete coverage cannot be ensured. On the other hand, XES parser implementations should take a healthy distrust towards defined global attributes, and double-check their validity and completeness while parsing the document.

2.5 Event Classifiers

In XES, there are per se no predefined attributes with any well-understood meaning. Contrast this with the previous MXML format, where the WorkflowModelElement of each event would point to its event class, i.e. the higher level concept the event refers to, and which makes it comparable to other events. *Event classifiers are a mandatory feature of the XES standard*.

The XES format makes event classification configurable and flexible, by introducing the concept of event classifiers. An event classifier *assigns to each event an identity*, which makes it comparable to other events (via their assigned identity).

Classifiers are *defined via a set of attributes*, from which the class identity of an event is derived. In its simplest form, an event classifier is defined by one attribute, and the value of that attribute would yield the class identity of an event.

Event classifiers are defined for the log, and there may be an arbitrary number of classifiers for each document. Note that, *the set of attributes used to define event classifiers should be a subset of the global event attributes for that log*, i.e., event classifiers should only be defined over event-global attributes.

Event classifiers are defined within respective `<classifier>` tags, which are child elements of the <log> tag in a document.

Example:

```
<classifier name="Activity classifier" keys="name status"/>
```

The above example defines a classifier with the given name (for identification purposes) over the events’ attributes with keys “name” and “status”. Any two events who have the same values for both these attributes are considered to be equal by that classifier.

2.6 Extensions

The XES standard does not define a specific set of attributes per log, trace, or event. As such, the semantics of the attributes these elements do contain must necessarily be ambiguous, hampering the interpretation of that data.

This ambiguity is resolved by the concept of *extensions* in XES. An extension defines a set of attributes on any levels of the XES log hierarchy (log, trace, event, and meta for nested attributes). In doing so, it provides points of reference for interpreting these attributes (and, thus, their parent elements). Extensions therefore are *primarily a vehicle for attaching semantics to a set of defined attributes per element*.

Extensions have many possible uses. One important use is to introduce a set of commonly understood attributes which are vital for a specific perspective or dimension of event log analysis (and which may even not have been foreseen at the time of designing the XES standard). The current set of standard extensions is introduced further below in this document.

Other uses include the definition of generally-understood attributes for a specific application domain (e.g., medical attributes for hospital processes), or for supporting special features or requirements of a specific analysis application.

In the XML serialization of XES, extensions are declared with a corresponding `<extension>` tag, which is a child tag of the `<log>` tag. Example:

```
<extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.
    xesext"/>
```

The above statement would declare that the log features attributes defined by the “Concept” extension.

The “prefix” attribute declares the prefix of all attributes defined by this extension in the log. This means, the keys of all attributes defined by this extension will be prepended by “concept”, and the colon separation character.

Example:

```
<string key="concept:name" value="Initialization"/>
```

In this way, attributes refer back to their respective extension.

The “uri” attribute contains a unique URI which points to the definition of the extension in XESEXT format. XES implementations can download the XESEXT definition file from that URI, to query information about extension-defined attributes, or to generate stub implementations for internal use.

An example of an XESEXT definition is included in Appendix A, and the XSD stylesheet definition for XESEXT is included in Appendix C.

3 XML Serialization of XES

The composition of an XES document follows the meta-model introduced earlier. An example is given below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<log xes.version="1.1" xes.features="arbitrary-depth" xmlns="http://www.xes-standard.org
    /">
    <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.
        xesext"/>
    <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
    <global scope="trace">
        <string key="concept:name" value=""/>
    </global>
    <global scope="event">
        <string key="concept:name" value=""/>
        <date key="time:timestamp" value="1970-01-01T00:00:00.000+00:00"/>
        <string key="system" value=""/>
    </global>
    <classifier name="Activity" keys="concept:name"/>
    <classifier name="Another" keys="concept:name system"/>
    <float key="log attribute" value="2335.23"/>
    <trace>
        <string key="concept:name" value="Trace number one"/>
        <event>
            <string key="concept:name" value="Register client"/>
            <string key="system" value="alpha"/>
            <date key="time:timestamp" value="2009-11-25T14:12:45:000+02:00"/>
            <int key="attempt" value="23">
                <boolean key="triedhard" value="false"/>
            </int>
        </event>
        <event>
            <string key="concept:name" value="Mail rejection"/>
            <string key="system" value="beta"/>
            <date key="time:timestamp" value="2009-11-28T11:18:45:000+02:00"/>
        </event>
    </trace>
</log>
```

The state machine flow diagram in Figure 3.1 details the correct composition of an XES document. For a precise definition of the XML serialization of XES documents, please refer to the XSD stylesheet definition given in Appendix C.

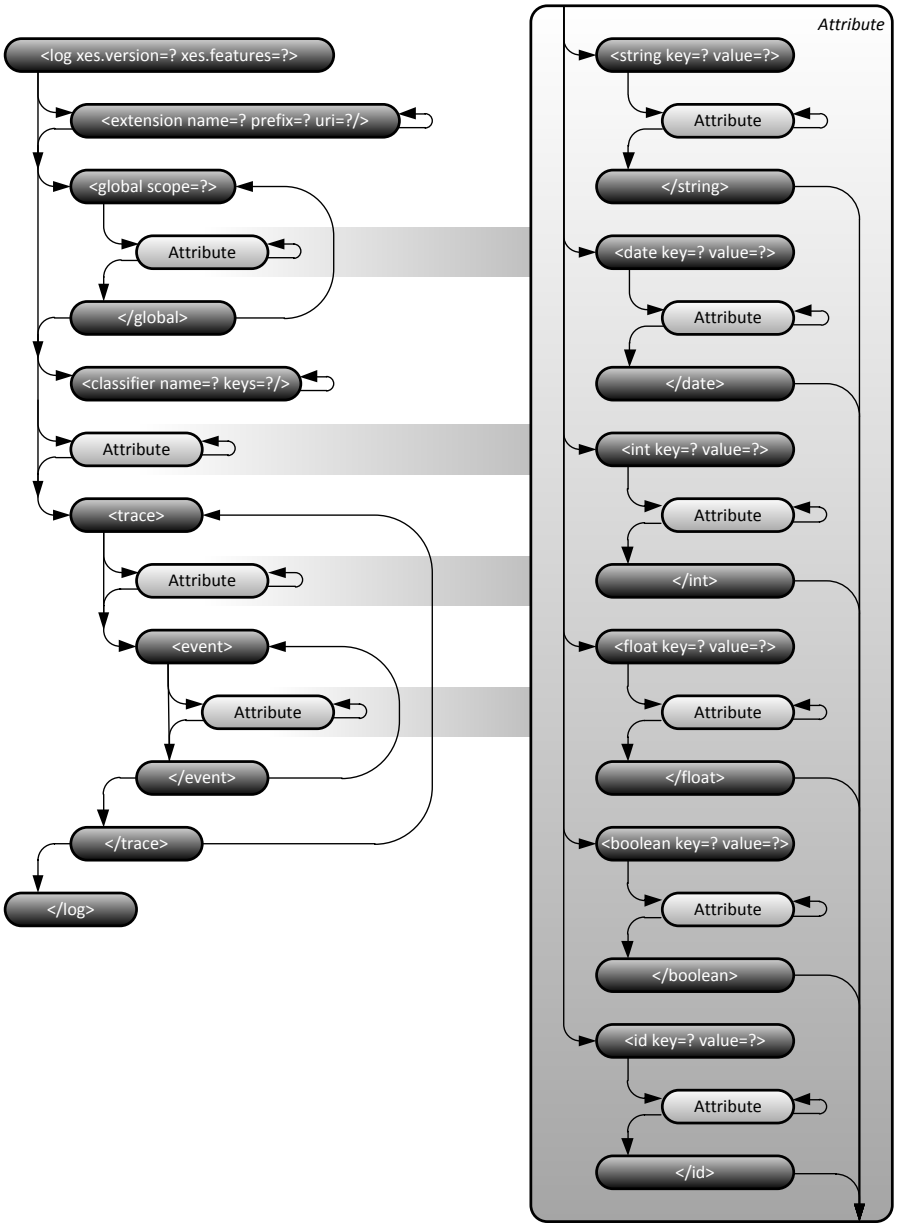

Figure 3.1: The state machine flow diagram for the XES standard

4 Standard Extensions

The XES meta-model recognizes and treats all extensions as equal, independent from their source. This allows users of the format to extend it, in order to fit any purpose or domain setting.

However, there are recurring requirements for information stored in event logs, which demand a fixed and universally understood semantics. For this purpose, a number of extensions have been standardized. When creating logs for a specific domain, or also when designing log-analyzing techniques, one should consider using these standardized extensions, since they allow for a wider level of understanding of the contents of event logs.

In the following, the currently standardized extensions to the XES formats are introduced.

4.1 Concept Extension

The Concept extension defines, for all levels of the XES type hierarchy, an attribute which stores the generally understood name of type hierarchy elements.

Extension prefix: `concept`

Extension URI: `http://www.xes-standard.org/concept.xesext`

Attribute Level	Key	Type	Description
log, trace, event	name	string	Stores a generally understood name for any type hierarchy element. For logs, the name attribute may store the name of the process having been executed. For traces, the name attribute usually stores the case ID. For events, the name attribute represents the name of the event, e.g. the name of the executed activity represented by the event.
event	instance	string	The instance attribute is defined for events. It represents an identifier of the activity instance whose execution has generated the event.

4.2 Lifecycle Extension

The Lifecycle extension specifies, for events, the lifecycle transition they represent in a transactional model of their generating activity. This transactional model can be arbitrary, however, the Lifecycle extension also specifies a standard transactional model for activities. Using this extension is appropriate in any setting, where events denote lifecycle transitions of higher-level activities.

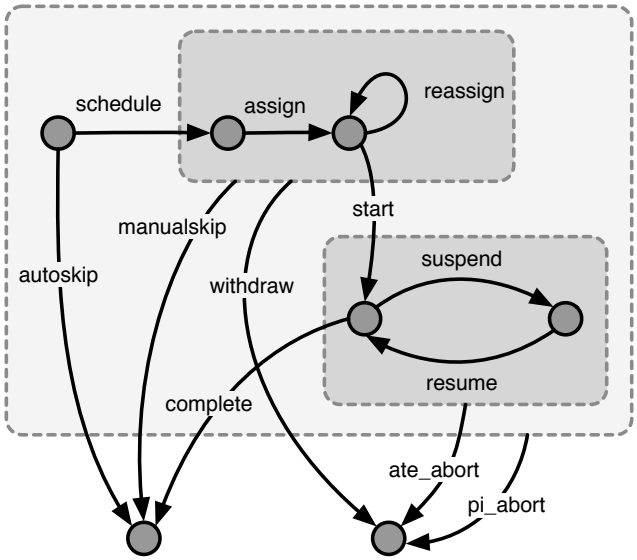

Figure 4.1: The state machine for the standard transactional model

Extension prefix: `lifecycle`

Extension URI: `http://www.xes-standard.org/lifecycle.xesext`

The standard transactional model is defined in the state machine as shown in Figure 4.1.

Attribute Level	Key	Type	Description
log	model	string	This attribute refers to the lifecycle transactional model used for all events in the log. If this attribute has a value of “standard”, the standard lifecycle transactional model of this extension is assumed.
event	transition	string	The transition attribute is defined for events, and specifies the lifecycle transition represented by each event. If the standard transactional model of this extension is used, the value of this attribute is one out of: **schedule -** The activity is scheduled for execution. **assign -** The activity is assigned to a resource for execution. **withdraw -** Assignment has been revoked. **reassign -** Assignment after prior revocation. **start -** Execution of the activity commences. **suspend -** Execution is being paused. **resume -** Execution is restarted. **pi_abort -** The whole execution of the process is aborted for this case. **ate_abort -** Execution of the activity is aborted. **complete -** Execution of the activity is completed. **autoskip -** The activity has been skipped by the system. **manualskip -** The activity has been skipped on purpose. **unknown -** Any lifecycle transition not captured by the above categories.

4.3 Organizational Extension

The organizational extension is useful for domains, where events can be caused by human actors, who are somewhat part of an organizational structure. This extension specifies three attributes for events, which identify the actor having caused the event, and his position in the organizational structure.

Extension prefix: `org`

Extension URI: `http://www.xes-standard.org/org.xesext`

Attribute Level	Key	Type	Description
event	resource	string	The name, or identifier, of the resource having triggered the event.
event	role	string	The role of the resource having triggered the event, within the organizational structure.
event	group	string	The group within the organizational structure, of which the resource having triggered the event is a member.

4.4 Time Extension

In almost all applications, the exact date and time at which events occur can be precisely recorded. Storing this information is the purpose of the time extension. Recording a timestamp for events is important, since this constitutes crucial information for many event log analysis techniques.

Extension prefix: `time`

Extension URI: `http://www.xes-standard.org/time.xesext`

Attribute Level	Key	Type	Description
event	timestamp	date	The date and time, at which the event has occurred.

4.5 Semantic Extension

Depending on the view on a process, type hierarchy artifacts may correspond to different concepts. For example, the name of an event (as specified by the Concept extension) may refer to the activity whose execution has triggered this event. However, this activity may be situated on a low level in the process meta-model, and be a part of higher-level, aggregate activities itself.

Besides events, also other elements of the XES type hierarchy may refer to a number of concepts at the same time (e.g., a log may refer to different process definitions, on different levels of abstractions). To express the fact, that one type artifact may represent a number of concepts in a process meta-model, the semantic extension has been defined.

It is assumed that there exists an ontology for the process meta-model, where every concept can be identified by a unique URI. The semantic extension defines an attribute, which allows to store a number of model references, as URIs, in any element of the XES type hierarchy.

Extension prefix: `time`

Extension URI: `http://www.xes-standard.org/time.xesext`

Attribute Level	Key	Type	Description
log, trace, event, meta	modelReference	string	References to model concepts in an ontology. Model references are stored in a literal string, as comma-separated URIs identifying the ontology concepts.

4.6 ID Extension

Provides unique identifiers (UUIDs) for elements.

Extension prefix: `identity`

Extension URI: `http://www.xes-standard.org/identity.xesext`

Attribute Level	Key	Type	Description
log, trace, event, meta	id	id	Unique identifier (UUID) for an element.

4.7 Cost Extension

The cost extension defines a nested element to store information about the cost associated with activities within a log. The objective of this extension is to provide semantics to cost aspects that can be associated with events in a log. The definition associates three data elements with a particular cost element: the amount associated with the cost element as well as the cost driver that is responsible for incurring that cost and the cost type. As it is possible for more than one cost element to be associated with an event, the cost incurred per event is summarized using the total attribute. The currency element is also recorded once per event. Cost information can be recorded at the trace level (for instance, to be able to say that it costs $20 when a case is started). Cost information can also be recorded at the event level (for instance, for certain event types such as complete or canceled events) to capture the cost incurred in undertaking the activity by a resource.

Extension prefix: `cost`

Extension URI: `http://www.xes-standard.org/cost.xesext`

Attribute Level	Key	Type	Description
trace, event	total	float	Total cost incurred for a trace or an event. The value represents the sum of all the cost amounts within the element.
trace, event	currency	string	The currency of all costs of this element in any valid currency format.
meta	amount	float	The value contains the cost amount for a cost driver.
meta	driver	string	The value contains the id for the cost driver used to calculate the cost.
meta	type	string	The value contains the cost type (e.g., Fixed, Overhead, Materials).

Please note that the amount, driver, and type attributes are meta-attributes, that is, they need an attribute as parent. These parent attributes are used to separate these three attributes, as the XES standard does not allow to have multiple attributes with the same key in a single container. Nevertheless, these parent attributes do not belong to this extension, as their keys typically all need to be different. As a result, a typical structure for this extension looks like follows:

```
<trace>
    <string key="cost:currency" value="AUD"/>
    <float key="cost:total" value="20.00"/>
    <string key="xyz123" value="">
```

```
        <float key="cost:amount" value="20.00"/>
        <string key="cost:driver" value="xyz123"/>
        <string key="cost:type" value="Fixed Overhead"/>
    </string>
    <event>
        <string key="cost:currency" value="AUD"/>
        <float key="cost:total" value="123.50"/>
        <string key="d2f4ee27" value="">
            <float key="cost:amount" value="21.40"/>
            <string key="cost:driver" value="d2f4ee27"/>
            <string key="cost:type" value="Labour"/>
        </string>
        <string key="abc124" value="">
            <float key="cost:amount" value="102.10"/>
            <string key="cost:driver" value="abc124"/>
            <string key="cost:type" value="Variable Overhead"/>
        </string>
    </event>
</trace>
```

A XESEXT Example

We have chosen the Semantic Extension (see 3.5) to exemplify the XESEXT format for XES extensions. This extension defines, on each level of abstraction (log, trace, event, and meta), the same string-based attribute “modelReference”. Attributes can be defined on all four levels of abstraction, similar to attribute declarations in XES (while omitting the value attribute). For every defined attribute, the XESEXT document may feature an arbitrary number of alias mappings as child elements. These mappings define a human-readable alias for the attribute within a given namespace (typically a country code, used for localization).

For a more detailed definition of the XESEXT format, the reader is referred to Appendix C, which contains the XSD stylesheet definition for XESEXT.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xesextension name="Semantic" prefix="semantic" uri="http://www.xes-standard.org/
    semantic.xesext">
    <log>
        <string key="modelReference">
            <alias mapping="EN" name="Ontology Model Reference"/>
            <alias mapping="DE" name="Ontologie-Modellreferenz"/>
            <alias mapping="FR" name="Référence au Modèle Ontologique"/>
            <alias mapping="ES" name="Referencia de Modelo Ontológico"/>
            <alias mapping="PT" name="Referência de Modelo Ontológico"/>
        </string>
    </log>
    <trace>
        <string key="modelReference">
            <alias mapping="EN" name="Ontology Model Reference"/>
            <alias mapping="DE" name="Ontologie-Modellreferenz"/>
            <alias mapping="FR" name="Référence au Modèle Ontologique"/>
            <alias mapping="ES" name="Referencia de Modelo Ontológico"/>
            <alias mapping="PT" name="Referência de Modelo Ontológico"/>
        </string>
    </trace>
    <event>
        <string key="modelReference">
            <alias mapping="EN" name="Ontology Model Reference"/>
            <alias mapping="DE" name="Ontologie-Modellreferenz"/>
            <alias mapping="FR" name="Référence au Modèle Ontologique"/>
            <alias mapping="ES" name="Referencia de Modelo Ontológico"/>
            <alias mapping="PT" name="Referência de Modelo Ontológico"/>
        </string>
    </event>
    <meta>
        <string key="modelReference">
            <alias mapping="EN" name="Ontology Model Reference"/>
            <alias mapping="DE" name="Ontologie-Modellreferenz"/>
            <alias mapping="FR" name="Référence au Modèle Ontologique"/>
            <alias mapping="ES" name="Referencia de Modelo Ontológico"/>
            <alias mapping="PT" name="Referência de Modelo Ontológico"/>
        </string>
    </meta>
</xesextension>
```

B XES XML Serialization Schema Definition (XSD)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <!-- This file describes the XML serialization of the XES format for event log data.
      -->
  <!-- For more information about XES, visit http://www.xes-standard.org/ -->

  <!-- (c) 2012 by IEEE Task Force on Process Mining (http://www.win.tue.nl/ieetfpm) -->

  <!-- Date: June 12, 2012 -->
  <!-- Version: 1.1 -->
  <!-- Author: Christian Guenther (christian@fluxicom.com) -->
  <!-- Author: Eric Verbeek (h.m.w.verbeek@tue.nl) -->
  <!-- Change: Added AttributableType (list of attribute types now occurs only once) -->
  <!-- Change: Added id type -->
  <!-- Change: Made xes.features and openxes.version optional -->

  <!-- Date: November 25, 2009 -->
  <!-- Version: 1.0 -->
  <!-- Author: Christian Guenther (christian@fluxicom.com) -->

  <!-- Every XES XML Serialization needs to contain exactly one log element -->
  <xs:element name="log" type="LogType"/>

  <!-- Attributables -->
  <xs:complexType name="AttributableType">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="string" minOccurs="0" maxOccurs="unbounded" type="
          AttributeStringType"/>
      <xs:element name="date" minOccurs="0" maxOccurs="unbounded" type="
          AttributeDateType"/>
      <xs:element name="int" minOccurs="0" maxOccurs="unbounded" type="AttributeIntType"
          />
      <xs:element name="float" minOccurs="0" maxOccurs="unbounded" type="
          AttributeFloatType"/>
      <xs:element name="boolean" minOccurs="0" maxOccurs="unbounded" type="
          AttributeBooleanType"/>
      <xs:element name="id" minOccurs="0" maxOccurs="unbounded" type="AttributeIDType"/>
    </xs:choice>
  </xs:complexType>

  <!-- String attribute -->
  <xs:complexType name="AttributeStringType">
    <xs:complexContent>
      <xs:extension base="AttributeType">
            <xs:attribute name="value" use="required" type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- Date attribute -->
  <xs:complexType name="AttributeDateType">
    <xs:complexContent>
      <xs:extension base="AttributeType">
            <xs:attribute name="value" use="required" type="xs:dateTime"/>
```

```
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- Integer attribute -->
<xs:complexType name="AttributeIntType">
  <xs:complexContent>
    <xs:extension base="AttributeType">
          <xs:attribute name="value" use="required" type="xs:long"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- Floating-point attribute -->
<xs:complexType name="AttributeFloatType">
  <xs:complexContent>
    <xs:extension base="AttributeType">
          <xs:attribute name="value" use="required" type="xs:double"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- Boolean attribute -->
<xs:complexType name="AttributeBooleanType">
  <xs:complexContent>
    <xs:extension base="AttributeType">
          <xs:attribute name="value" use="required" type="xs:boolean"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- ID attribute -->
<xs:complexType name="AttributeIDType">
  <xs:complexContent>
    <xs:extension base="AttributeType">
          <xs:attribute name="value" use="required" type="xs:string"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- Extension definition -->
<xs:complexType name="ExtensionType">
  <xs:attribute name="name" use="required" type="xs:NCName"/>
  <xs:attribute name="prefix" use="required" type="xs:NCName"/>
  <xs:attribute name="uri" use="required" type="xs:anyURI"/>
</xs:complexType>

<!-- Globals definition -->
<xs:complexType name="GlobalsType">
  <xs:complexContent>
    <xs:extension base="AttributableType">
      <xs:attribute name="scope" type="xs:NCName" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- Classifier definition -->
<xs:complexType name="ClassifierType">
  <xs:attribute name="name" type="xs:NCName" use="required"/>
  <xs:attribute name="keys" type="xs:token" use="required"/>
</xs:complexType>

<!-- Attribute -->
<xs:complexType name="AttributeType">
  <xs:complexContent>
    <xs:extension base="AttributableType">
          <xs:attribute name="key" use="required" type="xs:Name"/>
```

```
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- Elements may contain attributes -->
  <xs:complexType name="ElementType">
    <xs:complexContent>
      <xs:extension base="AttributableType"/>
    </xs:complexContent>
  </xs:complexType>

  <!-- Logs are elements that may contain traces -->
  <xs:complexType name="LogType">
    <xs:complexContent>
      <xs:extension base="ElementType">
            <xs:sequence>
              <xs:element name="extension" minOccurs="0" maxOccurs="unbounded" type="
                  ExtensionType"/>
              <xs:element name="global" minOccurs="0" maxOccurs="2" type="GlobalsType"/>
              <xs:element name="classifier" minOccurs="0" maxOccurs="unbounded" type="
                  ClassifierType"/>
              <xs:element name="trace" maxOccurs="unbounded" type="TraceType"/>
            </xs:sequence>
            <xs:attribute name="xes.version" type="xs:decimal" use="required"/>
            <xs:attribute name="xes.features" type="xs:token"/>
            <xs:attribute name="openxes.version" type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- Traces are elements that may contain events -->
  <xs:complexType name="TraceType">
    <xs:complexContent>
      <xs:extension base="ElementType">
            <xs:sequence>
              <xs:element name="event" maxOccurs="unbounded" type="EventType"/>
            </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <!-- Events are elements -->
  <xs:complexType name="EventType">
    <xs:complexContent>
      <xs:extension base="ElementType">
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

</xs:schema>
```

C XESEXT Extension Format Schema Definition (XSD)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <!-- This file describes the serialization for extensions of the XES format for event
       log data. -->
  <!-- For more information about XES, visit http://www.xes-standard.org/ -->

  <!-- (c) 2012 IEEE Task Force on Process Mining (http://wwww.win.tue.nl/ieeetfpm) -->

  <!-- Date: June 12, 2012 -->
  <!-- Version: 1.1 -->
  <!-- Author: Christian Guenther (christian@fluxicom.com) -->
  <!-- Author: Eric Verbeek (h.m.w.verbeek@tue.nl) -->
  <!-- Change: Added AttributableType (list of attribute types now occurs only once) -->
  <!-- Change: Added id type -->

  <!-- Date: November 25, 2009 -->
  <!-- Version: 1.0 -->
  <!-- Author: Christian Guenther (christian@fluxicom.com) -->

  <!-- Any extension definition has an xesextension root element. -->
  <!-- Child elements are containers, which define attributes for -->
  <!-- the log, trace, event, and meta level of the XES -->
  <!-- type hierarchy. -->
  <!-- All of these containers are optional. -->
  <!-- The root element further has attributes, defining: -->
  <!--  * The name of the extension. -->
  <!--  * A unique prefix string for attributes defined by this -->
  <!--    extension. -->
  <!--  * A unique URI of this extension, holding the XESEXT -->
  <!--    definition file. -->
  <xs:element name="xesextension">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="log" minOccurs="0" maxOccurs="1" type="AttributableType"/>
        <xs:element name="trace" minOccurs="0" maxOccurs="1" type="AttributableType"/>
        <xs:element name="event" minOccurs="0" maxOccurs="1" type="AttributableType"/>
        <xs:element name="meta" minOccurs="0" maxOccurs="1" type="AttributableType"/>
      </xs:sequence>
      <xs:attribute name="name" use="required" type="xs:NCName"/>
      <xs:attribute name="prefix" use="required" type="xs:NCName"/>
      <xs:attribute name="uri" use="required" type="xs:anyURI"/>
    </xs:complexType>
  </xs:element>

  <!-- Attributes -->
  <xs:complexType name="AttributableType">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="string" type="AttributeType"/>
      <xs:element name="date" type="AttributeType"/>
      <xs:element name="int" type="AttributeType"/>
      <xs:element name="float" type="AttributeType"/>
```

```
      <xs:element name="boolean" type="AttributeType"/>
      <xs:element name="id" type="AttributeType"/>
    </xs:choice>
  </xs:complexType>

  <!-- Attribute -->
  <xs:complexType name="AttributeType">
    <xs:sequence>
      <xs:element name="alias" minOccurs="0" maxOccurs="unbounded" type="AliasType"/>
    </xs:sequence>
    <xs:attribute name="key" use="required" type="xs:Name"/>
  </xs:complexType>

  <!-- Alias definition, defining a mapping alias for an attribute -->
  <xs:complexType name="AliasType">
        <xs:attribute name="mapping" use="required" type="xs:NCName"/>
        <xs:attribute name="name" use="required" type="xs:string"/>
  </xs:complexType>

</xs:schema>
```

D Changes

D.1 Version 1.2

Eric Verbeek Added ID extension.

Eric Verbeek Added cost extension.