XES Software Event Extension

M. Leemans C. Liu

June 22, 2017

Documents:	XES Software Event Extension
Authors:	M. Leemans, C. Liu
Email:	m.leemans@tue.nl
Date:	June 22, 2017
Version:	5.2

1 Introduction and Background

During the execution of software, execution data can be recorded. With the development of process mining techniques on the one hand, and the growing availability of software execution data on the other hand, a new form of software analytics comes into reach. That is, applying process mining techniques to analyze software execution data. To enable process mining for software, event logs should be capable of capturing software-specific data.

A software event log is typically recorded at the method call level during software execution. Events generated at this level reference a specific point in the software source code. The *Software Event* extension captures this event location information, together with some basic runtime information related to this location.

Extension definition

Name:	Software Event
Prefix:	swevent
URI:	http://www.xes-standard.org/swevent.xesext
XML:	`<extension name="Software Event" prefix="swevent" uri="http://www.xes-standard.org/swevent.xesext"/>`

The remainder of this extension is organized as follows. In Section 2 we explain some basic terminology. In Sections 3 and 4 we detail how a method execution is recorded. In addition, Section 5 covers how to record data associated with a method execution. Via Section 6 we allow the user to describe the executed software, making the event log more self-contained. Section 7 provides minimal support for runtime information, and Section 8 covers exception handling data. In Section 9, we touch upon the relation with the *micro event* extension. Section 10 provides an example, and the XES Extension definition is given in Section 11. Finally, in Section 12 provides a reference glossary.

2 Terminology

In this extension, we will use some software-specific terminology. We provided a reference glossary in Section 12. In this section, we will explain some key terminology using the example code snippet shown in Listing 1.

Listing 1: Example code snippet showing when which event types are triggered.

```
 1  class A {
 2      void f(int y) { -------------------------- call A.f
 3          try {
 4              ----------------------------- calling B.g
 5              int r = b.g(12, y);
 6              --------------------------- returning B.g
 7          } catch (Exception e) { -------------- handle in A.f
 8              ...
 9          }
10      } ------------------------------ return/throws A.f
11  }
12  class B {
13      int g(int x, int y) { --------------------- call B.g
14          return x / y;
15      } ----------------------------- return/throws B.g
16  }
```

When recording a software event at the method call level, we refer to the method being called or invoked as the *callee*. Optionally, we can track the context method where a method is triggered from. We call this context method the *caller*. That is, the *caller* method invokes the *callee* method. In Listing 1, we can see how the callee and caller role can change over the course of a software execution:

Line	Callee	Caller	Line	Callee	Caller	Line	Callee	Caller
2	A.f	–	4	B.g	A.f	13	B.g	–
10	A.f	–	6	B.g	A.f	15	B.g	–

3 Event Type and Lifecycle

Software events can be triggered at different states during a software execution. To indicate when an event was triggered, we record an indicative event type attribute, as defined below. In Listing 1, these event types are shown on the right in the context of an example code snippet.

Level	Key	Type	Description
event	type	string	Software event type, indicating at what point during execution this event was generated. The possible values are enumarated below.

The possible software event type values we recognize are:

call	The start of a method block
return	The normal end of a method block
throws	The end of a method block in case of an uncaught exception
handle	The start of an exception handle catch block
calling	The start of calling / invoking another method
returning	The end of returning a called method

The software event type values describe transitions in a transactional model for the lifecycle of a method execution. In Figure 1, the state machine for the method execution transactional model is given. As an example, these are the states for the two methods in Listing 1:

Line	Event Type	A.f	B.g
2	**call** A.f	InProcess	–
4	**calling** B.g	Suspended	–
13	**call** B.g	Suspended	InProcess
15	**return** B.g	Suspended	Completed
15	**throws** B.g	Suspended	Aborted
6	**returning** B.g	InProcess	Completed
7	**handle** in A.f	InProcess	Aborted
10	**return** A.f	Completed	Closed
10	**throws** A.f	Aborted	Closed

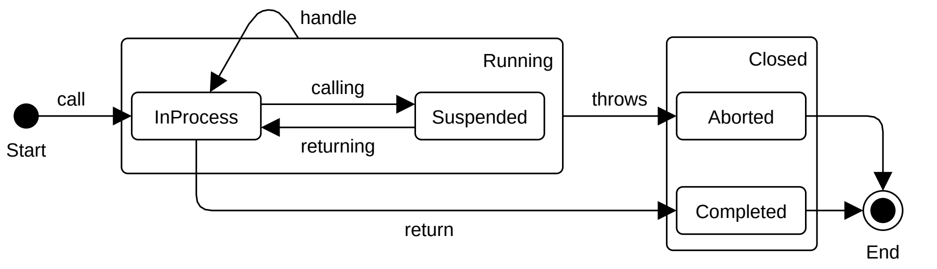

Figure 1: State machine for the method execution transactional model.

The software event type values can be related to values in the *standard lifecycle transactional model*, defined in the *lifecycle* extension. We suggest the following mapping (note that the standard lifecycle transactional model cannot correctly support the **handle** transition):

Type	Lifecycle
call	start
return	complete

Type	Lifecycle
calling	start
returning	complete

Type	Lifecycle
handle	reassign
throws	ate_abort

4 Event Location

Software events are triggered at a particular, uniquely identifiable location in the software source code. With this location, each event is traceable back to the source code location where it was generated.

Each event has a *callee* location. Events with the **calling** or **returning** event type also have an additional *caller* location. Each location is described by the following attributes, with a *callee*- or *caller*- key prefix:

Level	Key	Type	Description
event	package	string	The *package* in the software code architecture to which the method belongs.
event	class	string	The *class* to which the method belongs.
event	method	string	The referenced *method*.
event	paramSig	string	The *parameter signature* of the referenced method.
event	returnSig	string	The *return signature* of the referenced method.
event	isConstructor	boolean	If the referenced method *is a class constructor*.
event	instanceId	string	The *instance id* of the corresponding class instance. The absence of an instance id is represented by the value "0".
event	filename	string	The *file name* of the corresponding source code artifact.
event	lineNr	int	The *line number* of the executed source code statement.

The software event callee values can be related to the *concept name*, defined in the *concept* extension. We suggest the use the concatenation of callee package, class, method and paramSig as a concept:name (e.g., `demo.A.f(int)`). This way, the concept name is the unique, canonical name for the executed method. Example event location information for some events from Listing 1:

	Line 2 **call** A.f	Line 4 **calling** B.g
callee-package	demo	demo
callee-class	A	B
callee-method	f	g
callee-paramSig	(int)	(int,int)
callee-returnSig	void	int
callee-isConstructor	false	false
callee-instanceId	1074593562	660017404
callee-filename	source/A.java	source/B.java
callee-lineNr	2	13
concept:name	demo.A.f(int)	demo.B.g(int,int)

	Line 4 **calling** B.g
caller-package	demo
caller-class	A
caller-method	f
caller-paramSig	(int)
caller-returnSig	void
caller-isConstructor	false
caller-instanceId	1074593562
caller-filename	source/A.java
caller-lineNr	5

5 Method Data

At runtime, when a method is called or returns, data can be passed along. A **return** or **returning** event can define a return data value. A **call** or **calling** event can define multiple parameter data values.

Level	Key	Type	Description
log	hasData	boolean	If method data is recorded for this log.
event	returnValue	string	The *return value* for the returning method.
event	params	list	List of *parameters* for the called method.
meta	paramValue	string	A *parameter value* in the list params.
meta	valueType	string	The runtime *value type* for a return or parameter value.

6 Application Information

At runtime, the software events are generated by a particular application or process instance. Users can annotate events with application information, indicating which application instance generated the events. Not only makes this the event log more self-contained, it also can help in tracing events in a distributed or multi process setup.

Level	Key	Type	Description
event	appName	string	The user defined *application name.*
event	appTier	string	The user defined *application tier.*
event	appNode	string	The user defined *application node.*
event	appSession	string	The user defined *application session.*

7 Runtime Information

At runtime, the software events in an application process are triggered on a particular thread. Such a thread can be identified by its thread id.

The nano time attribute is used to measure elapsed time in software. It does not have to be related to any other notion of system or wall-clock time. The value represents nanoseconds since some fixed but arbitrary time (perhaps in the future, so values may be negative).

Level	Key	Type	Description
event	threadId	string	The *thread id* on which the event was generated.
event	nanotime	int	The elapsed *nano time* since some fixed but arbitrary time (see description above.)

8 Exception Information

An *exception* in software indicates an error. When an exception is *thrown* from a particular location / method, it can be caught / *handled* at a particular location.

Level	Key	Type	Description
log	hasException	boolean	If exception data is recorded for this log.
event	exThrown	string	The *thrown exception type* for a **throws** or **handle** event.
event	exCaught	string	The *caught exception type* for a **handle** event.

9 Relation to the Micro Event Extension

Note that the events software events generated on the same execution thread implicitly describe a call graph structure. For example in Listing 1, the event **calling** B.g on line 4 happens during the execution of A.f. That is, the event **calling** B.g on line 4 is a nested event of the events **call** A.f on line 2 and **return** A.f on line 10. This nested call grap relation can be explicitly logged using the *micro event* extension.

10 Example

In Listing 2 an example XES trace is given for the code snippet in Listing 1, called with f(0). In Figure 2, a message sequence diagram is given as a visual aid. We have annotated each event in Listing 2 with the corresponding even type, similar to the annotations in Listing 1. Note that, in this example, we assume that Line 14 in Listing 1 throws an ArithmeticException when y equals zero. This example log consists of 1 trace with 6 events.

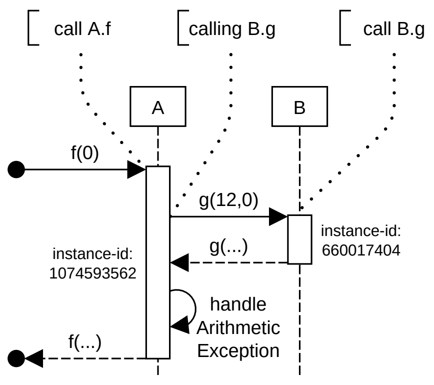

Figure 2: Message sequence diagram for the events in Listing 2.

Listing 2: Example XES log for the code snippet in Listing 1, called with f(0).

```
<log>
  <extension name="Concept" prefix="concept"
     uri="http://www.xes-standard.org/concept.xesext"/>
  <extension name="Lifecycle" prefix="lifecycle"
     uri="http://www.xes-standard.org/lifecycle.xesext"/>
  <extension name="Time" prefix="time"
     uri="http://www.xes-standard.org/time.xesext"/>
  <extension name="Software Event" prefix="swevent"
     uri="http://www.xes-standard.org/swevent.xesext"/>
  <boolean key="swevent:hasData" value="true" />
  <boolean key="swevent:hasException" value="true" />
  <trace>
    <event> - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - call A.f
      <string key="concept:name" value="demo.A.f(int)" />
      <string key="lifecycle:transition" value="start" />
      <date key="time:timestamp" value="2017-06-15T10:02:30.287Z" />
      <string key="swevent:type" value="call" />
      <string key="swevent:callee-package" value="demo" />
      <string key="swevent:callee-class" value="A" />
      <string key="swevent:callee-method" value="f" />
      <string key="swevent:callee-paramSig" value="(int)" />
      <string key="swevent:callee-returnSig" value="void" />
      <boolean key="swevent:callee-isConstructor" value="false" />
      <string key="swevent:callee-instanceId" value="1074593562" />
      <string key="swevent:callee-filename" value="source/A.java" />
      <int key="swevent:callee-lineNr" value="2" />
      <list key="swevent:params"> <values>
        <string key="swevent:paramValue" value="0">
          <string key="swevent:valueType" value="int" />
        </string>
      </values> </list>
      <string key="swevent:appName" value="Demo" />
      <string key="swevent:threadId" value="1" />
      <int key="swevent:nanotime" value="493674332622147" />
    </event>
    <event> - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - calling B.g
      <string key="concept:name" value="demo.B.g(int,int)" />
      <string key="lifecycle:transition" value="start" />
      <date key="time:timestamp" value="2017-06-15T10:02:30.287Z" />
      <string key="swevent:type" value="calling" />
      <string key="swevent:callee-package" value="demo" />
      <string key="swevent:callee-class" value="B" />
      <string key="swevent:callee-method" value="g" />
      <string key="swevent:callee-paramSig" value="(int,int)" />
      <string key="swevent:callee-returnSig" value="int" />
      <boolean key="swevent:callee-isConstructor" value="false" />
      <string key="swevent:callee-instanceId" value="660017404" />
      <string key="swevent:callee-filename" value="source/B.java" />
      <int key="swevent:callee-lineNr" value="13" />
      <string key="swevent:caller-package" value="demo" />
      <string key="swevent:caller-class" value="A" />
      <string key="swevent:caller-method" value="f" />
      <string key="swevent:caller-paramSig" value="(int)" />
      <string key="swevent:caller-returnSig" value="void" />
      <boolean key="swevent:caller-isConstructor" value="false" />
      <string key="swevent:caller-instanceId" value="1074593562" />
      <string key="swevent:caller-filename" value="source/A.java" />
      <int key="swevent:caller-lineNr" value="5" />
      <list key="swevent:params"> <values>
        <string key="swevent:paramValue" value="12">
          <string key="swevent:valueType" value="int" />
        </string>
        <string key="swevent:paramValue" value="0">
          <string key="swevent:valueType" value="int" />
        </string>
      </values> </list>
      <string key="swevent:appName" value="Demo" />
      <string key="swevent:threadId" value="1" />
      <int key="swevent:nanotime" value="493674332823571" />
    </event>
    <event>- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - call B.g
      <string key="concept:name" value="demo.B.g(int,int)" />
      <string key="lifecycle:transition" value="start" />
      <date key="time:timestamp" value="2017-06-15T10:02:30.287Z" />
      <string key="swevent:type" value="call" />
      <string key="swevent:callee-package" value="demo" />
      <string key="swevent:callee-class" value="B" />
      <string key="swevent:callee-method" value="g" />
      <string key="swevent:callee-paramSig" value="(int,int)" />
      <string key="swevent:callee-returnSig" value="int" />
      <boolean key="swevent:callee-isConstructor" value="false" />
      <string key="swevent:callee-instanceId" value="1074593562" />
      <string key="swevent:callee-filename" value="source/B.java" />
      <int key="swevent:callee-lineNr" value="13" />
      <list key="swevent:params"> <values>
        <string key="swevent:paramValue" value="12">
          <string key="swevent:valueType" value="int" />
        </string>
        <string key="swevent:paramValue" value="0">
          <string key="swevent:valueType" value="int" />
        </string>
      </values> </list>
      <string key="swevent:appName" value="Demo" />
      <string key="swevent:threadId" value="1" />
      <int key="swevent:nanotime" value="493674332878738" />
    </event>
    <event>- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - throws B.g
      <string key="concept:name" value="demo.B.g(int,int)" />
      <string key="lifecycle:transition" value="ate_abort" />
      <date key="time:timestamp" value="2017-06-15T10:02:30.287Z" />
      <string key="swevent:type" value="throws" />
      <string key="swevent:callee-package" value="demo" />
      <string key="swevent:callee-class" value="B" />
      <string key="swevent:callee-method" value="g" />
      <string key="swevent:callee-paramSig" value="(int,int)" />
      <string key="swevent:callee-returnSig" value="int" />
      <boolean key="swevent:callee-isConstructor" value="false" />
      <string key="swevent:callee-instanceId" value="1074593562" />
      <string key="swevent:callee-filename" value="source/B.java" />
      <int key="swevent:callee-lineNr" value="15" />
      <string key="swevent:appName" value="Demo" />
      <string key="swevent:threadId" value="1" />
      <int key="swevent:nanotime" value="493674332936044" />
      <string key="swevent:exThrown" value="ArithmeticException" />
    </event>
    <event>- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - handle in A.f
      <string key="concept:name" value="demo.A.f(int)" />
      <string key="lifecycle:transition" value="reassign" />
      <date key="time:timestamp" value="2017-06-15T10:02:30.287Z" />
      <string key="swevent:type" value="handle" />
      <string key="swevent:callee-package" value="demo" />
      <string key="swevent:callee-class" value="A" />
      <string key="swevent:callee-method" value="f" />
      <string key="swevent:callee-paramSig" value="(int)" />
      <string key="swevent:callee-returnSig" value="void" />
      <boolean key="swevent:callee-isConstructor" value="false" />
      <string key="swevent:callee-instanceId" value="1074593562" />
      <string key="swevent:callee-filename" value="source/A.java" />
      <int key="swevent:callee-lineNr" value="7" />
      <string key="swevent:appName" value="Demo" />
      <string key="swevent:threadId" value="1" />
      <int key="swevent:nanotime" value="493674333039536" />
      <string key="swevent:exThrown" value="ArithmeticException" />
      <string key="swevent:exCaught" value="Exception" />
    </event>
```

```
    <event> - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  return A.f
      <string key="concept:name" value="demo.A.f(int)" />
      <string key="lifecycle:transition" value="complete" />
      <date key="time:timestamp" value="2017-06-15T10:02:30.287Z" />
      <string key="swevent:type" value="return" />
      <string key="swevent:callee-package" value="demo" />
      <string key="swevent:callee-class" value="A" />
      <string key="swevent:callee-method" value="f" />
      <string key="swevent:callee-paramSig" value="(int)" />
      <string key="swevent:callee-returnSig" value="void" />
      <boolean key="swevent:callee-isConstructor" value="false" />
      <string key="swevent:callee-instanceId" value="1074593562" />
      <string key="swevent:callee-filename" value="source/A.java" />
      <int key="swevent:callee-lineNr" value="10" />
      <string key="swevent:returnValue" value="">
        <string key="swevent:valueType" value="void" />
      </string>
      <string key="swevent:appName" value="Demo" />
      <string key="swevent:threadId" value="1" />
      <int key="swevent:nanotime" value="493674333162700" />
    </event>
  </trace>
</log>
```

11 XES Extension

Listing 3: XES Extension - Software Event.

```
<extension name="Software Event" prefix="swevent"
  uri="http-//www.xes-standard.org/swevent.xesext">
  <log>
    <boolean key="hasData">
      <alias mapping="EN" name="Has method data"/>
    </boolean>
    <boolean key="hasException">
      <alias mapping="EN" name="Has exception data"/>
    </boolean>
  </log>
  <event>
    <string key="type">
      <alias mapping="EN" name="Event Type"/>
    </string>
    <string key="callee-package">
      <alias mapping="EN" name="Callee - Package"/>
    </string>
    <string key="callee-class">
      <alias mapping="EN" name="Callee - Class"/>
    </string>
    <string key="callee-method">
      <alias mapping="EN" name="Callee - Method"/>
    </string>
    <string key="callee-paramSig">
      <alias mapping="EN" name="Callee - Parameter signature"/>
    </string>
    <string key="callee-returnSig">
      <alias mapping="EN" name="Callee - Return signature"/>
    </string>
    <boolean key="callee-isConstructor">
      <alias mapping="EN" name="Callee - Is a class constructor"/>
    </boolean>
    <string key="callee-instanceId">
      <alias mapping="EN" name="Callee - Instance id of class instance"/>
    </string>
    <string key="callee-filename">
```

```
      <alias mapping="EN" name="Callee - File name source code artifact"/>
    </string>
    <string key="callee-lineNr">
      <alias mapping="EN" name="Callee - Line number in source code artifact"/>
    </string>
    <string key="caller-package">
      <alias mapping="EN" name="Caller - Package"/>
    </string>
    <string key="caller-class">
      <alias mapping="EN" name="Caller - Class"/>
    </string>
    <string key="caller-method">
      <alias mapping="EN" name="Caller - Method"/>
    </string>
    <string key="caller-paramSig">
      <alias mapping="EN" name="Caller - Parameter signature"/>
    </string>
    <string key="caller-returnSig">
      <alias mapping="EN" name="Caller - Return signature"/>
    </string>
    <boolean key="caller-isConstructor">
      <alias mapping="EN" name="Caller - Is a class constructor"/>
    </boolean>
    <string key="caller-instanceId">
      <alias mapping="EN" name="Caller - Instance id of class instance"/>
    </string>
    <string key="caller-filename">
      <alias mapping="EN" name="Caller - File name source code artifact"/>
    </string>
    <string key="caller-lineNr">
      <alias mapping="EN" name="Caller - Line number in source code artifact"/>
    </string>
    <string key="returnValue">
      <alias mapping="EN" name="Return value for the returning method"/>
    </string>
    <list key="params">
      <alias mapping="EN" name="List of parameters for the called method"/>
    </list>
    <string key="appName">
      <alias mapping="EN" name="User defined application name"/>
    </string>
    <string key="appTier">
      <alias mapping="EN" name="User defined application tier"/>
    </string>
    <string key="appNode">
      <alias mapping="EN" name="User defined application node"/>
    </string>
    <string key="appSession">
      <alias mapping="EN" name="User defined application session"/>
    </string>
    <string key="threadId">
      <alias mapping="EN" name="Thread id for generated event"/>
    </string>
    <int key="nanotime">
      <alias mapping="EN" name="Elapsed nano time"/>
    </int>
    <string key="exThrown">
      <alias mapping="EN" name="Thrown exception type"/>
    </string>
    <string key="exCaught">
      <alias mapping="EN" name="Caught exception type"/>
    </string>
  </event>
  <meta>
    <string key="paramValue">
      <alias mapping="EN" name="A parameter value in the list params"/>
    </string>
    <string key="valueType">
```

```
      <alias mapping="EN" name="A runtime value type for a return or parameter value"/>
    </string>
  </meta>
</extension>
```

12 Glossary

callee The *callee* is the method being called or invoked.

caller The *caller* is the context/method where a method is triggered from.

exception An *exception* in software indicates an error.

throw An exception can be *thrown* from a particular location / method, signaling an error.

handle An exception can be caught / *handled* at a particular location, indicating we are handling an error.

class/object A *class* / object is a conceptual unit in software, defining a combination of methods and data.

(object) instance For each class, at runtime, multiple *instances* can exist, be created and destroyed. Each instance has its own lifecycle during the execution of a software program.