TMF Event Structure

Version 0.3
François Chouinard

1. Introduction

The fundamental entity of the Tracing and Monitoring Framework (TMF) is the "event", the report of an occurrence of some phenomenon at some point in time.

This section deals with the basic framework event. The goal is to provide an efficient event structure that can be easily extended for application specific needs and yet be handled generically by the framework.

The TMF Event class structure is sketched in Figure 1.

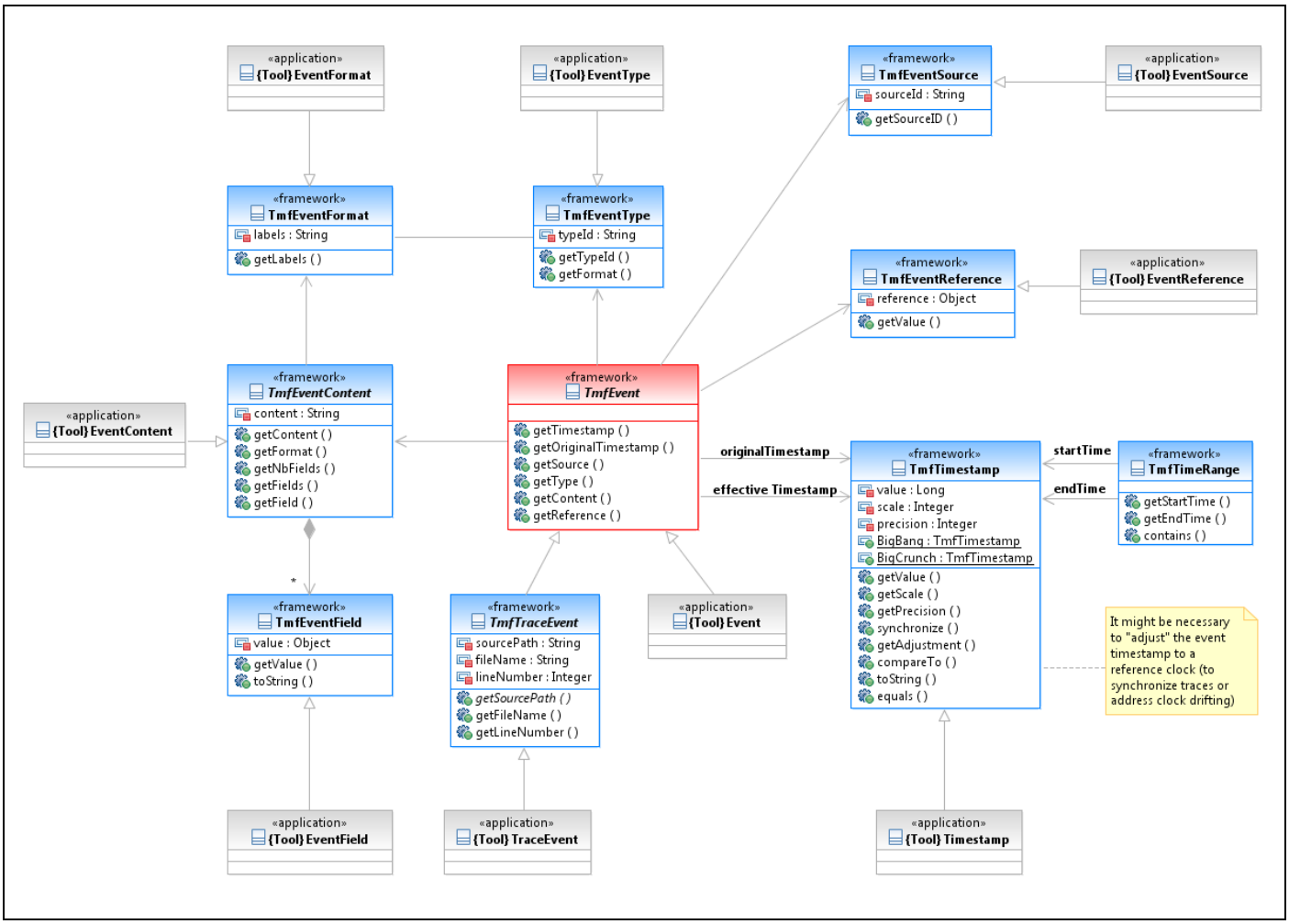

Figure 1 TMF Event Class Diagram

2. TmfEvent and TmfTraceEvent

In its canonical form, an event (*TmfEvent*) has a normalized timestamp (*TmfTimestamp*), a reporter or source (*TmfEventSource*), a type (*TmfEventType*) and content (*TmfEventContent*).

For convenience, a generic reference field (*TmfEventReference*) is also provided. This could be used as e.g. a location marker in the event stream to distinguish between otherwise identical events.

It might be argued that the basic event structure could be reduced to a timestamp and content i.e. the source, type and reference could be represented as fields within the content part. We think that they should be provided as basic fields because (1) it makes sense to distinguish event origin and type from actual content, (2) although not mandatory in the model, these informations are usually available, and (3) it eases the implementation of some meaningful generic operations (e.g. filtering on event types).

A *TmfTraceEvent* is an extension of *TmfEvent* that associates an event with a line of source code (for application tracing). This extension permits source lookup.

If needed, both *TmfEvent* and *TmfTraceEvent* could be extended by the application to represent more elaborate concrete events.

3. TmfTimestamp and TmfTimeRange

The *TmfTimestamp* provides a timestamp implementation in its most basic form: an unstructured *value*, a *scale* and a *precision*.

By representing the *value* with a signed 64-bits integer, most practical timestamp granularities should be accommodated (from e.g. wall clock time for admin logs to nanoseconds for kernel traces). In practice, it might make sense to define an application timestamps relative to some reference (e.g. the Epoch).

The *scale* refers to the magnitude of the *value* i.e. the base 10 exponent with respect to the base unit (e.g. the second). E.g. for seconds, the resolution would be 0; for nanoseconds, it would be -9.

The *precision* refers to the error on the measure. This value is useful to determine if 2 timestamps are equal within measurement tolerance (used by the *compareTo()* method).

A timestamp might need to be “adjusted” with respect to a reference clock (to compensate for clock drifting, event synchronization, change of epoch, etc…). The class provides a method to compute this value (*getAdjustment*()) .

In order to define open ranges, two special *TmfTimestamp* static values are provided to represent respectively the Beginning and End of Time: *BigBang* and *BigCrunch*.

If an application requires a more exotic timestamp structure, *TmfTimestamp* can easily be extended.

TmfTimeRange is provided as a convenience class to represent a time range. *BigBang* and *BigCrunch* can be used to define open-ended ranges.

4. TmfEventSource, TmfEventType and TmfEventReference

TmfEventSource and *TmfEventType* provide simple generic implementations of the corresponding concepts that could be extended by the application if more sophisticated structures are needed.

However, instantiating the framework object with a simple string identifier should be enough for most implementations.

As mentionned above, *TmfEventReference* is simply a convenience field that could be used to store application specific data.

5. TmfEventFormat, TmfEventContent and TmfEventField

The *TmfEventFormat* provides a format-specific event parser that creates instances of *TmfEventField* used to populate the *TmfEventContent. TmfEventFormat* (1) implements a type specific content parser and (2) provides the list of column labels for the *TmfEventContent* fields.

By default, there is only 1 field, the whole content expressed as a string. However, the application would typically create one immutable instance of this class for each event type and implement the *parse()* function.

This scheme works well for flat event structures. *TmfContentFormat* and *TmfEventContent* would need to be extended for more sophisticated event structures.

Note that this scheme allows late content parsing (i.e. only when actually needed).

6. Self-Defining Traces

In the case of self-defining traces, event attributes (source, type and format) might be redefined dynamically. In the worst case, this would prevent *a priori* declaration of the events structure.

In the cases studied, there is always a definition scheme involved (hence the name…) and the structural changes can usually be restricted to the content format. In other words, it should be sufficient to provide an extension to *TmfContentFormat* that implements the trace self-definition scheme.

However, we recognize that this is a gray area that needs to be addressed by the framework itself (in a later release).

7. Life Cycle

Coming soon to a wiki near you.