Trace Event Format - ENSC 351 reduced

Original authors: nduca@, dsinclair@

JSON Array Format

The format accepted by the trace viewer is the JSON array format. Essentially, an array of event objects. The events do not have to be in timestamp-sorted order.

```
[ {"name": "Asub", "cat": "PERF", "ph": "B", "pid": 22630, "tid": 22630, "ts": 829},
  {"name": "Asub", "cat": "PERF", "ph": "E", "pid": 22630, "tid": 22630, "ts": 833}
]
```

JSON Array Format

When provided as a string to the importer the **]** at the end of the JSON Array Format is optional. The Trace Viewer importer will automatically add a **]** if needed to turn the string into valid JSON. This is to support tracing systems that can not cleanly finish writing the trace. For example, when tracing the exit of a program.

Event Descriptions

There are a common set of fields for each of the events.

```
{
  "name": "myName",
  "cat": "category,list",
  "ph": "B",
  "ts": 12345,
  "pid": 123,
  "tid": 456,
  "args": {
    "someArg": 1,
    "anotherArg": {
      "value": "my value"
    }
  }
}
```

General Event Structure

- **name:** The name of the event, as displayed in Trace Viewer
- **cat**: The event categories. This is a comma separated list of categories for the event. The categories can be used to hide events in the Trace Viewer UI.
- **ph:** The event type. This is a single character which changes depending on the type of event being output. The valid values are listed in the table below. We will discuss each phase type below.
- **ts:** The tracing clock timestamp of the event. The timestamps are provided at microsecond

granularity.
- **pid**: The process ID for the process that output this event.
- **tid**: The thread ID for the thread that output this event.
- **args:** Any arguments provided for the event. Some of the event types have required argument fields, otherwise, you can put any information you wish in here. The arguments are displayed in Trace Viewer when you view an event in the analysis section.

Optional

- **cname:** A fixed color name to associate with the event. If provided, cname must be one of the names listed in trace-viewer's base color scheme's reserved color names list

The following table lists all event types and their associated phases:

Event type	Event phases
Duration Events	**B** (begin), **E** (end)
Instant Events	**i**,
Counter Events	**C**
Object Events	**N** (created), **O** (snapshot), **D** (destroyed)
Metadata Events	**M**

Duration Events

Duration events provide a way to mark a duration of work on a given thread. The duration events are specified by the **B** and **E** phase types. The **B** event must come before the corresponding **E** event. You can nest the **B** and **E** events. This allows you to capture function calling behaviour on a thread. The timestamps for the duration events *must* be in increasing order for a given thread. Timestamps in different threads do not have to be in increasing order, just the timestamps within a given thread.

The only required fields for the **E** events are the **pid**, **tid**, **ph** and **ts** fields, all others are optional. If you provide **args** to both the **B** and **E** events then the arguments will be merged. If there is a duplicate argument value provided the **E** event argument will be taken and the **B** event argument will be discarded.

```
{"name": "myFunction", "cat": "foo", "ph": "B", "ts": 123, "pid": 2343, "tid": 2347,
 "args": {
   "first": 1
 }
},
{"ph": "E", "ts": 145, "pid": 2343, "tid": 2347,
 "args": {
   "first": 4,
   "second": 2
 }
}
```

Duration Event Example

In this example we will create a single slice in the Trace Viewer. The slice is named **myFunction** and has a single category **foo**. The slice starts at time **123**µs and will have a duration of **22**µs. The event will be displayed as part of process **2343** and thread **2347**. There will be two arguments attached to the slice, **first: 4** and **second: 2**.

```
{ ..., "ts": 1.0, "tid": 1, "ph": "B", "name": "A"},
{ ..., "ts": 1.1, "tid": 1, "ph": "B", "name": "Asub"},
{ ..., "ts": 3.9, "tid": 1, "ph": "E"},
{ ..., "ts": 4.0, "tid": 1, "ph": "E"}
```

Nested Duration Event Example

In the above example, there will be two slices created. One for function **A** and one for function **Asub**. When drawn in Trace Viewer the **Asub** slice will be nested under the **A** slice. The **A** function will be recorded as having a duration of **3**µs. **Asub** will have a recorded duration of **2.8**µs.

```
  { ..., "ts": 1.0, "tid": 1, "ph": "B", "name": "A"},
  { ..., "ts": 0.9, "tid": 2, "ph": "B", "name": "B"},
  { ..., "ts": 1.1, "tid": 1, "ph": "E"},
  { ..., "ts": 4.0, "tid": 2, "ph": "E"}
```

Thread Interleaving of Duration Events

In the above, even though the timestamps are out of order, everything will work correctly because within a given thread they are strictly increasing.

While duration events allow you to trace the function flow, they must be nested. It is not possible to have non-nested duration events for a given thread.

Instant Events

The instant events correspond to something that happens but has no duration associated with it. For example, vblank events are considered instant events. The instant events are designated by the **i** phase type.

There is an extra parameter provided to instant events, **s**. The **s** property specifies the scope of the event. There are 3 scopes available global (**g**), process (**p**) and thread (**t**). If no scope is provided we default to thread scoped events. The scope of the event designates how tall to draw the instant event in Trace Viewer. A thread scoped event will draw the height of a single thread. A process scoped event will draw through all threads of a given process. A global scoped event will draw a time from the top to the bottom of the timeline.

```
{"name": "OutOfMemory", "ph": "i", "ts": 1234523.3, "pid": 2343, "tid": 2347, "s":
"g"}
```

Instant Event Example

Counter Events

The counter events can track a value or multiple values as they change over time. Counter events are specified with the **C** phase type. Each counter can be provided with multiple series of data to display. When multiple series are provided they will be displayed as a stacked area chart in Trace Viewer. Please note that counters are process-local.

```
{..., "name": "ctr", "ph": "C", "ts":  0, "args": {"cats":  0}},
{..., "name": "ctr", "ph": "C", "ts": 10, "args": {"cats": 10}},
{..., "name": "ctr", "ph": "C", "ts": 20, "args": {"cats":  0}}
```

Counter Event Example

In the above example the counter tracks a single series named **cats**. The cats series has a value that goes from 0 to 10 and back to 0 over a 20µs period.

```
{..., "name": "ctr", "ph": "C", "ts":  0, "args": {"cats":  0, "dogs": 7}},
{..., "name": "ctr", "ph": "C", "ts": 10, "args": {"cats": 10, "dogs": 4}},
{..., "name": "ctr", "ph": "C", "ts": 20, "args": {"cats":  0, "dogs": 1}}
```

Multi Series Counter Example

In this example we have a single counter named **ctr**. The counter has two series of data, **cats** and **dogs**. When drawn, the counter will display in a single track with the data shown as a stacked graph.

Object Events

Objects are a building block to track complex data structures in traces. Since traces go for long periods of time, the formative concept for this is a way to identify objects in a time varying way. To identify an object, you need to give it an **id**, usually its pointer, e.g. id: "0x1000". But, in a long running trace, or just a trace with a clever allocator, a raw pointer 0x1000 might refer to two different objects at two different points in time. Object events do not have an **args** property associated with them.

In the trace event format, we think in terms of object instances. e.g. in C++, given class `Foo` , `new Foo()` at `0x1000` is an instance of `Foo`. There are three trace phase types associated with objects they are: **N**, **O**, and **D**, object created, object snapshot and object destroyed respectively.

```
{"name": "MyObject", "ph": "N", "id": "0x1000", "ts": 0, "pid": 1, "tid": 1},
{"name": "MyObject", "ph": "D", "id": "0x1000", "ts": 20, "pid": 1, "tid": 1}
```

Object Events Example

This sequence of events shows **MyObject**, identified by `0x1000` that is alive from 0µs to 20µs.

Objects obey the convention of the start time being inclusive and the deletion time being exclusive.

```
{"name": "MyOtherObject", "ph": "N", "id": "0x1000", "ts": 20, "pid": 1, "tid": 1},
{"name": "MyOtherObject", "ph": "D", "id": "0x1000", "ts": 25, "pid": 1, "tid": 1}
```

Reusing Object IDs

So, if we also had this set of events in addition to the **MyObject** events, then, a **MyOtherObject** reference at 20µs points at a **MyOtherObject** instance, not a **MyObject** instance.

Snapshots

As mentioned above, object instance commands do not have an **args** property. Meaning, you can not associate data with object instances. This may seem silly, until you think about how traces are time varying. In software, very few things stay constant for their lifetime -- rather they vary over time. Object snapshots allow you to output object state at a given instant in time. Object snapshots are designated with the **O** phase type.

```
{"name": "MyObject", "id": "0x1000", "ph": "O", "ts": 10, "pid": 1, "tid": 1,
 "args": {
  "snapshot": {...}
 }
}
```

Object Snapshot Example

The ellipsis inside the snapshot field can be anything you want, as much as you want. You can, for instance, put a complete dump of your rendering system into the snapshot.

Metadata Events

Metadata events are used to associate extra information with the events in the trace file. This information can be things like process names, or thread names. Metadata events are denoted by the **M** phase type. The argument list may be empty.

There are currently 5 possible metadata items that can be provided:

- **process_name**: Sets the display name for the provided pid. The name is provided in a **name** argument.
- **process_labels**: Sets the extra process labels for the provided pid. The label is provided in a **labels** argument.
- **process_sort_index**: Sets the process sort order position. The sort index is provided in a **sort_index** argument.
- **thread_name**: Sets the name for the given tid. The name is provided in a **name** argument.
- **thread_sort_index**: Sets the thread sort order position. The sort index is provided in a **sort_index** argument.

For the two sort_index items the **sort_index** argument value specifies the relative position you'd like the item to be displayed. Lower numbers are displayed higher in Trace Viewer. If multiple items all have the

same sort index then they are displayed sorted by name and, given duplicate names, by id.

```
{"name": "thread_name", "ph": "M", "pid": 2343, "tid": 2347,
 "args": {
  "name" : "RendererThread"
 }
}
```

Metadata Event Example

In this example we are setting the **thread_name** for tid **2347** to be **RenderThread**.