ERICSSON

Trace Event Guidelines

ADAPTATION DIRECTION

1/155 42-APR 901 0500/1 Uen Rev PA12

Copyright

Disclaimer

Trademark List

Contents

Revision History

Table 1 Revision History

Revision	Modifications	Date	Prepared
PA1	First draft of this document.	2011-11-07	eusgarc
PA2	Updated after internal review.	2011-11-09	eusgarc
PA3	Additional comments from lmcjidu.	2011-11-09	eusgarc
PA4	Comments from Mathieu Desnoyers.	2011-11-20	eusgarc
PA5	Domain names changed to lowercase.	2011-11-28	eusgarc
PA6	Updated domain structure based on comments from uabums.	2012-02-22	eusgarc
PA7	Updated with comments from uabums, lmcjidu, and ejusmiz. Added log levels, removed AXE references, and added the concept of parallel domains.	2012-02-29	eusgarc
PA8	Performed light edit and updated document structure.	2012-04-16	ejusmiz
PA9	Comments from uabums.	2012-04-17	eusgarc
PA10	New title and document number.	2012-04-20	ejusmiz
PA11	Removed version information from the document subtitle.	2012-04-23	ejusmiz

1 Introduction

This document provides guidelines and naming conventions to use when instrumenting applications for tracing with the Linux Trace Toolkit - next generation (LTTng) Userspace Tracer (UST).

1.1 Intended Audience

This document is intended for developers of software applications and common components.

2 Key Concepts

2.1 Trace Event

Trace Events are a C-function prototype based instrumentation mechanism meant to create a simple and flexible instrumentation API.

Note: When instrumenting the code, the macro is called TRACEPOINT_EVENT, but for simplicity this document uses the term “Trace Event”.

The developers define instrumentation prototypes in include (.h) files which are then used in the code. It is then possible to connect a probe (a function with an argument type list matching the Trace Event) to a Trace Event, and to activate the Trace Event. When an activated Trace Event is reached in the program, the connected probe is called and the execution continues when the probe returns.

2.2 Trace Domain

Trace Events are defined and grouped hierarchically following the architecture/functionality of the product/application that contains them. Therefore, Trace Domains constitute hierarchical groups of Trace Events and they can be seen as a “region” of tracing [Figure 1]. A Trace Domain can contain other Trace (sub) Domains depending on the product/application’s structure and level of complexity.

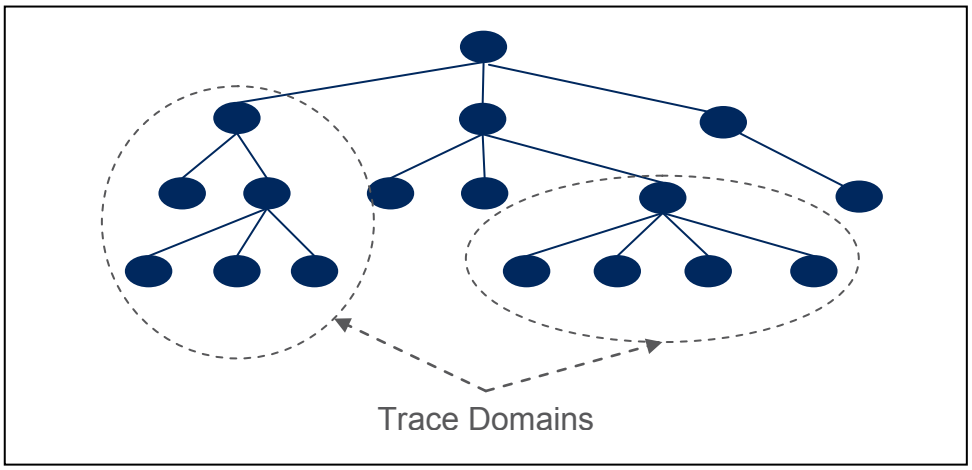

Figure 1 Hierarchical Trace Domain Structure

2.3 Trace Sub-domain

The concept of Trace Sub-domain is used in this document exclusively to emphasize the fact that a particular Trace Domain is contained within another (larger) Trace Domain. For example, “com_ericsson_common” is a Trace Sub-domain of “com_ericsson”, “com_ericsson_common_brfc” is a Trace Sub-domain of “com_ericsson_common”, and so on.

2.4 Log Level

Log Levels act as a filtering mechanism that can be used in conjunction with Trace Events in order to narrow down the Trace Events that are produced at any given time. Log levels have an order of priority (e.g. “Emergency” has the highest priority, “Debug” has the lowest), which allows the grouping of Trace Events into categories such as “Emergency”, “Alert”, “Critical”, “Warning”, etc. In a large software product with a significant number of Trace Events, log levels provide an efficient way to limit the trace output and to reduce the focus to a more pertinent set of Trace Events. In order to be used, log levels must be tied to Trace Events during instrumentation.

3 Instrumentation Framework

The purpose of instrumenting an application is to trace the application’s execution in order to facilitate troubleshooting of software faults. Therefore, as a first instrumentation step, designers must consider how to optimize the introduction of Trace Events in their code. Too few Trace Events might not provide enough clues to pinpoint the location of a fault. Too many Trace Events can worsen code readability and translate into very large trace log files that would overwhelm troubleshooters and occupy too much disk space unnecessarily.

Trace Events must be grouped in a hierarchical manner (i.e. Trace Domains) that allows a troubleshooter to navigate a product’s layers in a top-down fashion to narrow down the fault’s “neighborhood”. That is, the troubleshooter would activate traces in a higher Trace Domain and, if relevant trace logs are produced, activate a lower Trace Sub-domain, and so on, until the fault is found.

A “higher” Trace Domain is higher up the domain tree as shown in Figure 1. Therefore, a higher Trace Domain would yield a larger number of Trace Events. On the contrary, a lower Trace Domain would produce a smaller number of Trace Events, but closer and more related to the fault being investigated.

To this end, Trace Domains must be built efficiently and consistently across applications, so troubleshooters can know what to expect. It is up to the designers of each product to decide which approach optimally suits their purposes. The guidelines outlined in the following chapters provide a generic naming convention that can be adapted to different instrumentation strategies.

4 Basic Rules

The following are implementation-imposed restrictions:

- The maximum length of a Trace Domain name (i.e. the whole hierarchical structure including Trace Sub-domains) is 127 characters.

- The maximum length of a Trace Event name (i.e. not including the Trace Domain name) is 127 characters.
- Trace Domain names must be written in lowercase only.
- Trace Event names are case sensitive, so care must be taken not to specify the same Trace Event name with different cases. For instance, ImmInitialize and IMMinitialize are not the same and should not coexist in the same software product.

5 Trace Domain Guidelines

- Trace Domain names should be kept as short as possible while still being readable. This will help to keep log lines shorter and easier for troubleshooters to read.
- Trace Domains must have a hierarchical structure, which must be established by product architects and agreed upon by the committee with technical authority over the product.
- Trace Domains can be constructed using different approaches. For instance:
  - Follow the product architecture/structure. If a product is broken down into smaller parts (areas, modules, subsystems, subcomponents, etc.), each Trace Domain corresponds to one of those parts.
  - Follow functionality. Trace Events are grouped according to the role software units play in specific functions/features, e.g., IMS call setup, subscriber registration, restoring of a backup.
  - Mixed categories. Depending on the type of application, it may be desirable to have a combination of both architectural and functional elements as part of the Trace Domain.
  - Parallel domains. This strategy involves the use of both architecturally and functionally structured domains that coexist in the same software product. The two types of domains would complement each other and possibly allow more effective troubleshooting.
- The Trace Domain name structure must conform to one of the following two patterns:
  - Components:
    com_ericsson_common_<component name>_<sub-domain name>[_<sub-domain name>…]

    Examples:

com_ericsson_common_com
com_ericsson_common_brfc

- Applications:
  com_ericsson_<application name>_<sub-domain name>[_<sub-domain name>…]

  Examples:
  com_ericsson_apg
  com_ericsson_sapc

Where:

com: Standard prefix. Domains without the "com_" prefix are reserved for cross-companies, e.g., open-source projects.

common: prefix reserved for components.

component name: Name of component, e.g., brfc, lm, cmw, com.

sub-domain name: This represents part of the hierarchical structure of the Trace Domain in a component or application and it must contain at least one level. This structure can be architectural or functional, or a combination of both. The highest level has a wide scope and the lower levels narrow down to smaller parts of the product's structure or functionality. The different Trace Sub-domains in a Trace Domain are separated with underscores. There is no restriction in regards to the number of Trace Sub-domains, but keep in mind that the whole Trace Domain can have a maximum length of 127 characters. Trace Sub-domains can fit into the following categories:

functionality: Name of a feature/sub-feature in which one or more applications participate, e.g., locationupdate, callsetup, faultmgmt.

module: a software unit that is part of an application. It can contain Trace Events as well as other smaller software modules. The smallest expression of a module is a function (e.g., subroutine, procedure, method) where Trace Events are instrumented.

Examples:
com_ericsson_common_com_faultmgmt_protocol_netconf
com_ericsson_common_brfc_participant [see Figure 2]
com_ericsson_common_brfc_oaminterfacehandler_createbackuprequest
tinytek_tinyapp

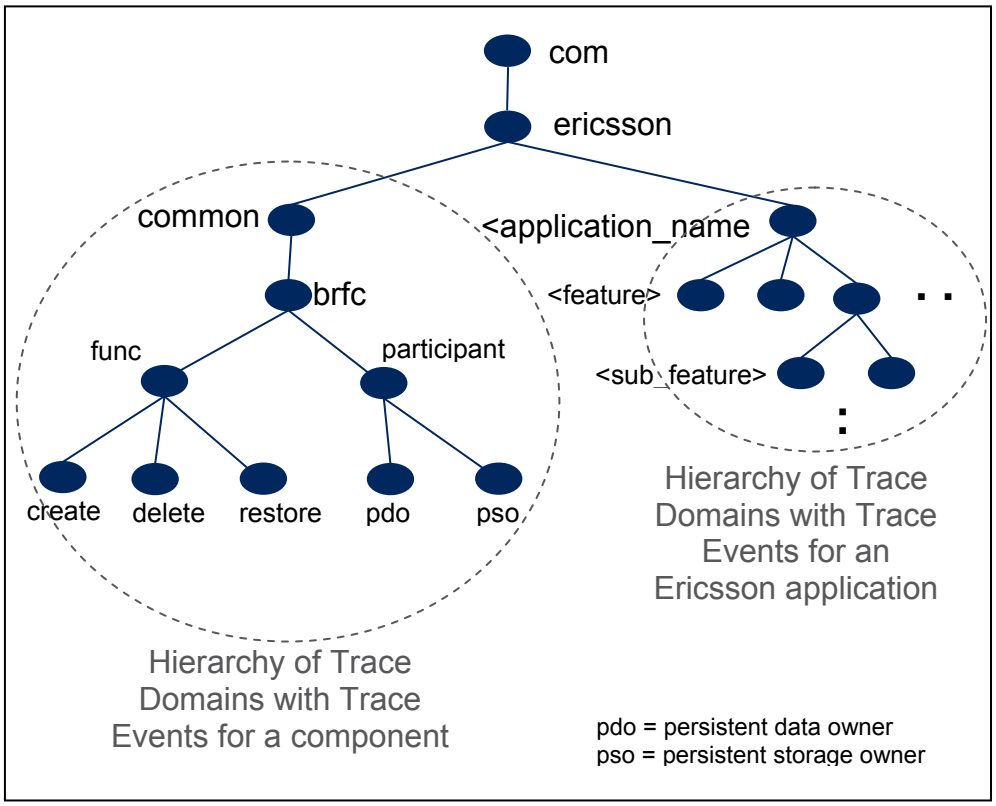

Figure 2 Example of a Trace Domain Hierarchical Structure

- Trace Domain name clashes must be prevented. That is, a component/application must have a unique name with respect to other components/applications.

6 Trace Event Guidelines

- Trace Event names should be kept as short as possible while still being readable. This will help to keep log lines shorter and easier for troubleshooters to read.
- When a Trace Event name consists of several parts, each part must be easily distinguishable from the others by using CamelCase. That is, each part of the name is written in lower case, with the exception of the first character, which is written in upper case.

  Example: VerifyCcbObjectModifyRequest
- Use common sense and avoid generic names (e.g., “TracePoint1”, “Temp”, “MyEvent”). Trace Event names must match the functionality of the code where they are located.

- Trace Event name clashes must be prevented to ensure that unrelated Trace Events do not end up in the same log just because they happen to have the same name.

7 Abbreviations

LTTng	Linux Trace Toolkit - next generation
UST	Userspace Tracer

8 References

[1] LTTng-UST