I/O Trace Common Semantics

Version 1.0 revision 9

Internal Draft

September 3, 2008

Revision History

Revision	Date	Sections	Originator	Comments
v1.0 rev 0	April 28, 2006	All		Initial version
v1.0 rev 1	August 29, 2006	All	Alistair Veitch	Added more commentary, new table format and tables for semantics entries, fixed (some) style entries, many other misc. formatting/editing changes
v1.0 rev 2	October 8, 2006	Posix System Call Traces	Michael Ernst	Added semantics entries for more system calls
v1.0 rev 3	November 6, 2006	Posix System Call Traces	Michael Ernst	Completed the semantics entries for all the system calls
v1.0 rev 4	April 11, 2007	Posix System Call Traces	Michael Ernst	Added some units entries and updated main system call section
v1.0 rev 5	November 29, 2007	Block Level Traces	Brad Morrey	Completed the semantics entries for the block level format
v1.0 rev 6	June 11, 2008	All	Geoff Kuenning	Added trace identification semantics; changed timestamp type in system call traces; rewrote block trace semantics
v1.0 rev 7	June 16, 2008	All	Geoff Kuenning	Make versioning compatible with DataSeries; add support for other formats.
v1.0 rev 8	July 22, 2008	All	Geoff Kuenning	Incorporate comments from Eric Anderson.
v1.0 rev 9	September 3, 2008	All	Geoff Kuenning	Split out trace-specific semantics

Suggestions for changes or modifications to this document should be sent to the IOTTA Technical Work Group at iottatwg-chair@snia.org.

1 Introduction

One general problem with trace files of any nature is determining the semantics of the various fields within the trace. While the meaning of a field is often obvious (e.g., an Ethernet trace with a field named "frame_size"), in other cases it is not (e.g., a disk trace with a "time" field—is it the time that the I/O operation was started or completed? Or something else again? Is the time relative or absolute?).

This document is one of a group of documents that together define semantics for various common types of I/O traces, specifically the names and types of fields, unambiguous definitions of their meanings, and their interrelationship. It is hoped that this information can then be used to enable easier understanding, manipulation, and analysis of traces from a wide variety of sources, without having to resort to having the originators of the traces around to explain them.

This document gives general information on trace semantics and notation, and defines fields that are common to all traces. Other documents specify fields that are specific to a certain kind of trace. For example, there are documents for Posix system-call traces, OS-independent block I/O traces, etc. The reader should consult both this document and the specification for a particular trace type to acquire a complete specification.

1.1 Notes on units

We prefer to use the International System of Units (SI) where possible. To avoid confusion, distinction should be made between base-10 prefixes (e.g., kilobytes) and base-2 units (e.g., kibibytes).

For time measurement, this document recommends the use of seconds as the base unit, with 64-bit fixed-point representation using 32 fractional bits. (Another interpretation of this representation is as a 64-bit integer measured in units of 2^{-32} seconds.) This representation supports a resolution of approximately 233 ps. For convenience of exposition, this representation will often be referred to as *Tfrac* (time fractional). A later version of this document may recommend an alternative representation with additional precision or range, but the current representation will continue to be supported for backwards compatibility.

Some traces represent time as an absolute value rather than a relative offset. In these cases, a zero point or "epoch" must also be given. This document recommends using the Unix epoch (00:00:00 UTC, January 1, 1970). A later version will address the question of epoch in a more general fashion.

For fields in which SI units are not applicable, we recommend using the smallest natural unit of measurement, and integer values of sufficient size (often 64 bits) to cover all possible values for the foreseeable future. A common example of this is in offset and size values, which we typically present as byte-sized units, although the raw version of the interface may use some other size (e.g., sectors, or KiB's).

1.2 Notes on the data model

This document takes the approach that all trace files consist of a set of records, where each record is comprised of a set of fields, with each field having a unique name (i.e., two fields within a single record cannot have the same name) and a data type. Data types consist of integers (with precisions from 8 to 64 bits), floating-point values, ASCII characters, strings of ASCII characters, and strings of UTF-8 characters. There is no constraint requiring distinct records to have common fields.

1.2.1 Interactions with trace syntax

1.2.1.1 DataSeries

Currently, the IOTTA TWG is recommending that the HP DataSeries trace format be used as a common syntax for all trace types. The choice of syntax is not entirely separate from that of semantics. Specifically, the choice of DataSeries restricts the range of available types to the following:

- bool: “true” or “false” values
- byte: signed 8-bit values
- int32: signed 32-bit values
- int64: signed 64-bit values
- double: IEEE 754 double-precision encoded values (64 bits)
- variable32: variable-length sequences of bytes with a maximum length of 2^{32}-1. Often used for string types, but could also represent any value larger than 64 bits in size.

DataSeries also supports several possibilities for grouping records together. In particular, it allows multiple types of records (i.e., records with different field names or types) to coexist within a single trace, through its Extents mechanism (an Extent is roughly analogous to a collection of rows in a database table). For some trace types, particularly those that consist of several very different record types (e.g. system-call traces) this raises the question of using either multiple extents (one record type per extent), a single extent (with lots of fields that may potentially be empty, called *nullable* in DataSeries) or variant record types. In general, variant types are preferred, although they have not yet been implemented within DataSeries, and a final decision on their adoption has been deferred till that time.

DataSeries also has various options available that control compression and determine the presence or absence of various fields. In this document, all of these options will be ignored, with the exception of “opt_nullable”, which specifies that a given field may (optionally) not be present (i.e. set to a special “null” value).

Finally, it should be noted that the definitions given in this document, are, as far as possible, meant to remain independent of the DataSeries encoding. In particular, if a particular trace references this semantics document, the user of that trace should be able to unambiguously infer the meaning of each of the trace fields from the field name (in the trace) and the description here.

1.2.1.2 SQL

Although the IOTTA TWG recommends DataSeries as the preferred trace format, other formats are possible. Because of the ubiquity of SQL databases, that format is also supported by this document. SQL-format traces use SQL tables instead of DataSeries Extents, and use slightly different type declarations. In SQL, the following types should be used:

- BOOLEAN: “true” or “false” values (DataSeries bool)
- SMALLINT: signed 8-bit values (DataSeries byte)
- INTEGER: signed 32-bit values (DataSeries int32)
- BIGINT: signed 64-bit values (DataSeries int64)
- DOUBLE PRECISION: IEEE 754 double-precision encoded values (64 bits) (DataSeries double)
- BLOB: variable-length sequences of bytes. (DataSeries variable32)

Although this document recommends BLOB as a representation for byte sequences, the trace collector may select a different format in specific situations. In particular, it may be preferable to use a format that is indexable by the database software being used, that accepts binary data without modification, or that stores strings more efficiently than BLOB. However, the chosen representation should always be such that it can be loaded into a BLOB without error.

In SQL, all fields must be declared as “NOT NULL” unless they are described in this document as “nullable”.

2 Types of traces

We have identified several broad categories of I/O traces for which we wish to define semantics. Typically, though not exclusively, these are defined by the measurement point at which the trace is taken.

The categories (or types) of I/O traces include but are not limited to:

- System call (subtypes will include standards such as POSIX, as well as per-operating system).
- Block-level (subtypes include at least logical volume manager, disk drivers, and block devices).

- Block protocol (e.g., SCSI or ATA). These differ from block level traces which only include I/O operations, whereas protocol traces include all disk level commands (e.g., formatting, connection establishment, etc.). This category may include traces made on networks (e.g., Fibre Channel or iSCSI).
- File system protocol (e.g., NFS or CIFS).
- Static snapshots of filesystems.

This document specifies only fields that are common to all trace types.

We plan to eventually specify standard semantics for all of the above types of traces. To begin with, we are concentrating upon only two types of traces, which comprise a large fraction of publicly available traces today. These are:

- System call (specifically POSIX)
- Block level (specifically captured within the operating system)

The semantics for each of these trace types are defined in separate documents.

3 Type-independent information

The following information appears in all traces, regardless of type.

3.1 Trace Identification

Every trace, regardless of type, must contain information that identifies the trace type and the version of this document with which it complies. In DataSeries, this information is kept in the XML ExtentType declaration. In SQL, it is kept in a table named “trace_identification”. The following table describes the fields in the ExtentType declaration and the trace_identification table:

Name	Definition
namespace	The string “iotta.snia.org”
name	A string defining the trace type, (see below). No trace may contain two identification rows with the same name. In SQL, the name should be declared UNIQUE.
version	A numeric string giving the version of this specification with which the trace conforms (see below).

Fields in the ExtentType declaration or trace_identification table

Note that a trace may contain more than one version identification, depending on the particular extents or tables that are included. However, all extents or tables of a given type in a given trace should be of the same version.

All traces should contain a "IOTTA:Common" version identification.

The trace type must be selected from the following list:

Type	Definition	Current Version
IOTTA:Common	Common fields	1.00
IOTTA:Trace:SystemCall:POSIX	POSIX system call trace.	*See trace-specific document*
IOTTA:Trace:Block	Block-level I/O trace.	*See trace-specific document*

The trace version is a string of the form "x.yy", where x is one or more digits giving the major version number of this document, and yy is two digits giving the minor version number. If two traces share the same major version number, they share the property that all fields in the trace with the smaller minor version also exist in the trace with the larger version, and no other definitions differ. In other words, incrementing the minor version number means that no fields were changed or removed; only new fields were added. All other changes require incrementing the major version.

3.2 General commentary

Every trace may contain an optional extent named "commentary". Each row in this extent contains a single variable32 field named "comments", which contains an arbitrary string with comments the trace collector wishes to provide with the trace.