Open Trace Format 2

User Manual

3.0.3 (revision 7f6882e3)

Fri Mar 15 2024 06:50:00

OTF2 LICENSE AGREEMENT

Page

Chapter 1

Open Trace Format 2

1.1 Introduction

The OTF2 library provides an interface to write and read trace data.

OTF2 is developed within the Score-P project. The Score-P project is funded by the German Federal Ministry of Education and Research.

1.2 Getting started

OTF2 usage examples

Chapter 2

OTF2 INSTALL

```
For generic installation instructions see below.
When building for an Intel MIC platform, carefully follow the
platform-specific instructions below.

Configuration of OTF2
*********************

‘configure’ configures OTF2 to adapt to many kinds of systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...

To assign environment variables (e.g., CC, CFLAGS...), specify them as
VAR=VALUE.  See below for descriptions of some of the useful variables.

Defaults for the options are specified in brackets.

Configuration:
  -h, --help              display this help and exit
      --help=short        display options specific to this package
      --help=recursive    display the short help of all the included packages
  -V, --version           display version information and exit
  -q, --quiet, --silent   do not print ‘checking ...’ messages
      --cache-file=FILE   cache test results in FILE [disabled]
  -C, --config-cache      alias for ‘--cache-file=config.cache’
  -n, --no-create         do not create output files
      --srcdir=DIR        find the sources in DIR [configure dir or ‘..’]

Installation directories:
  --prefix=PREFIX         install architecture-independent files in PREFIX
                          [/opt/otf2]
  --exec-prefix=EPREFIX   install architecture-dependent files in EPREFIX
                          [PREFIX]

By default, ‘make install’ will install all the files in
‘/opt/otf2/bin’, ‘/opt/otf2/lib’ etc.  You can specify
an installation prefix other than ‘/opt/otf2’ using ‘--prefix’,
for instance ‘--prefix=$HOME’.

For better control, use the options below.

Fine tuning of the installation directories:
  --bindir=DIR            user executables [EPREFIX/bin]
  --sbindir=DIR           system admin executables [EPREFIX/sbin]
  --libexecdir=DIR        program executables [EPREFIX/libexec]
  --sysconfdir=DIR        read-only single-machine data [PREFIX/etc]
  --sharedstatedir=DIR    modifiable architecture-independent data [PREFIX/com]
  --localstatedir=DIR     modifiable single-machine data [PREFIX/var]
  --libdir=DIR            object code libraries [EPREFIX/lib]
  --includedir=DIR        C header files [PREFIX/include]
  --oldincludedir=DIR     C header files for non-gcc [/usr/include]
  --datarootdir=DIR       read-only arch.-independent data root [PREFIX/share]
  --datadir=DIR           read-only architecture-independent data [DATAROOTDIR]
```

```
  --infodir=DIR           info documentation [DATAROOTDIR/info]
  --localedir=DIR         locale-dependent data [DATAROOTDIR/locale]
  --mandir=DIR            man documentation [DATAROOTDIR/man]
  --docdir=DIR            documentation root [DATAROOTDIR/doc/otf2]
  --htmldir=DIR           html documentation [DOCDIR]
  --dvidir=DIR            dvi documentation [DOCDIR]
  --pdfdir=DIR            pdf documentation [DOCDIR]
  --psdir=DIR             ps documentation [DOCDIR]

Program names:
  --program-prefix=PREFIX            prepend PREFIX to installed program names
  --program-suffix=SUFFIX            append SUFFIX to installed program names
  --program-transform-name=PROGRAM   run sed PROGRAM on installed program names

System types:
  --build=BUILD     configure for building on BUILD [guessed]
  --host=HOST       cross-compile to build programs to run on HOST [BUILD]

Optional Features:
  --disable-option-checking  ignore unrecognized --enable/--with options
  --disable-FEATURE       do not include FEATURE (same as --enable-FEATURE=no)
  --enable-FEATURE[=ARG]  include FEATURE [ARG=yes]
  --enable-silent-rules          less verbose build output (undo: 'make V=1')
  --disable-silent-rules         verbose build output (undo: 'make V=0')
  --disable-dependency-tracking  speeds up one-time build
  --enable-dependency-tracking   do not reject slow dependency extractors
  --enable-platform-mic   Force build for Intel Xeon Phi co-processors
                          [no].  This option is only needed for Xeon
                          Phi co-processors, like the Knights Corner
                          (KNC). It is not needed for self-hosted Xeon
                          Phis, like the Knights Landing (KNL); for these
                          chips no special treatment is required.
  --enable-debug          activate internal debug output [no]
  --enable-backend-test-runs
                          Run tests at make check [no]. If disabled, tests are
                          still build at make check. Additionally, scripts
                          (scorep_*tests.sh) containing the tests are
                          generated in <builddir>/build-backend.
  --enable-shared[=PKGS]  build shared libraries [default=yes]
  --enable-static[=PKGS]  build static libraries [default=yes]
  --enable-fast-install[=PKGS]
                          optimize for fast installation [default=yes]
  --disable-libtool-lock  avoid locking (might break parallel builds)

Optional Packages:
  --with-PACKAGE[=ARG]    use PACKAGE [ARG=yes]
  --without-PACKAGE       do not use PACKAGE (same as --with-PACKAGE=no)
  --with-sionlib[=<sionlib-bindir>]
                          Use an already installed sionlib. Provide path to
                          sionconfig. Auto-detected if already in $PATH.
  --with-pic              try to use only PIC/non-PIC objects [default=use
                          both]
  --with-gnu-ld           assume the C compiler uses GNU ld [default=no]
  --with-sysroot=DIR Search for dependent libraries within DIR
                        (or the compiler's sysroot if not specified).

Some influential environment variables:
  CC_FOR_BUILD
              C compiler command for the frontend build
  CXX_FOR_BUILD
              C++ compiler command for the frontend build
  F77_FOR_BUILD
              Fortran 77 compiler command for the frontend build
  FC_FOR_BUILD
              Fortran compiler command for the frontend build
  CPPFLAGS_FOR_BUILD
              (Objective) C/C++ preprocessor flags for the frontend build,
              e.g. -I<include dir> if you have headers in a nonstandard
              directory <include dir>
  CFLAGS_FOR_BUILD
              C compiler flags for the frontend build
  CXXFLAGS_FOR_BUILD
              C++ compiler flags for the frontend build
```

```
  FFLAGS_FOR_BUILD
              Fortran 77 compiler flags for the frontend build
  FCFLAGS_FOR_BUILD
              Fortran compiler flags for the frontend build
  LDFLAGS_FOR_BUILD
              linker flags for the frontend build, e.g. -L<lib dir> if you
              have libraries in a nonstandard directory <lib dir>
  LIBS_FOR_BUILD
              libraries to pass to the linker for the frontend build, e.g.
              -l<library>
  CC          C compiler command
  CFLAGS      C compiler flags
  LDFLAGS     linker flags, e.g. -L<lib dir> if you have libraries in a
              nonstandard directory <lib dir>
  LIBS        libraries to pass to the linker, e.g. -l<library>
  CPPFLAGS    (Objective) C/C++ preprocessor flags, e.g. -I<include dir> if
              you have headers in a nonstandard directory <include dir>
  CXX         C++ compiler command
  CXXFLAGS    C++ compiler flags
  CPP         C preprocessor
  CXXCPP      C++ preprocessor
  PYTHON      The Python interpreter to be used for the Python bindings. Use
              PYTHON=: to disable Python support.
  PYTHON_FOR_GENERATOR
              The Python interpreter used for the generator. Not a build
              requirement, only needed for developing. Python version 2.5 or
              above, but no support for Python 3. Use PYTHON_FOR_GENERATOR=:
              to disable Python support.

Use these variables to override the choices made by 'configure' or to help
it to find libraries and programs with nonstandard names/locations.

Please report bugs to <support@score-p.org>.


Platform-specific instructions
******************************

Intel Xeon Phi (aka. MIC) co-processors
---------------------------------------

   [Note: The following instructions only apply to Intel Xeon Phi
 co-processors, like the Knights Corner (KNC). They do not apply to
 self-hosted Xeon Phis, like the Knights Landing (KNL); for these
 chips no special treatment is required.]

   Building OTF2 for Intel Xeon Phi co-processors requires some extra care,
and in some cases two installations into the same location.  Therefore, we
strongly recommend to strictly follow the procedure as described below.

  1. Ensure that Intel compilers are installed and available in $PATH, and
     that the Intel Manycore Platform Software Stack (MPSS) is installed.

  2. Configure OTF2 to use the MIC platform:

        ./configure --enable-platform-mic [other options, e.g., '--prefix']

  3. Build and install:

        make; make install

   On non-cross compiling systems (e.g., typical Linux clusters), that's it.
On cross-compiling systems (e.g., Cray XC30 with Xeon Phi daughter board), a
second installation of OTF2 *on top* of the just installed one is required to
provide a single installation serving login nodes, compute nodes, and MIC:

  4. Remove MIC program binaries, object files, and configure-generated files
     from the source code directory:

        make distclean

  5. Reconfigure for login/compute nodes using *identical directory options*
     (e.g., '--prefix' or '--bindir') as in step 2:
```

```
       ./configure [other options as used in step 2]

     This will automatically detect the already existing native MIC build and
     enable the required support in the login node tools.

  6. Build and install:

       make; make install

   Note that this approach also works with VPATH builds (even with two
separate build directories) as long as the same options defining directory
locations are passed in steps 2 and 5.

Python bindings
---------------

  1. Requirements:
    + python 2.7 or later or
    + python 3.5 or later
    + Earlier versions will probably work, but are not currently tested.
    + Required packages are "six" (>= 1.4.0) and "future" (providing the "builtins" module)
    + sphinx to build the python documentation
    + Ubuntu package names: python python-future python-six python-sphinx


Installation Instructions
*************************

Copyright (C) 1994, 1995, 1996, 1999, 2000, 2001, 2002, 2004, 2005,
2006, 2007, 2008, 2009 Free Software Foundation, Inc.

   Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.  This file is offered as-is,
without warranty of any kind.

Basic Installation
==================

   Briefly, the shell commands `./configure; make; make install' should
configure, build, and install this package.  The following
more-detailed instructions are generic; see the `README' file for
instructions specific to this package.  Some packages provide this
`INSTALL' file but do not implement all of the features documented
below.  The lack of an optional feature in a given package is not
necessarily a bug.  More recommendations for GNU packages can be found
in *note Makefile Conventions: (standards)Makefile Conventions.

   The `configure' shell script attempts to guess correct values for
various system-dependent variables used during compilation.  It uses
those values to create a `Makefile' in each directory of the package.
It may also create one or more `.h' files containing system-dependent
definitions.  Finally, it creates a shell script `config.status' that
you can run in the future to recreate the current configuration, and a
file `config.log' containing compiler output (useful mainly for
debugging `configure').

   It can also use an optional file (typically called `config.cache'
and enabled with `--cache-file=config.cache' or simply `-C') that saves
the results of its tests to speed up reconfiguring.  Caching is
disabled by default to prevent problems with accidental use of stale
cache files.

   If you need to do unusual things to compile the package, please try
to figure out how `configure' could check whether to do them, and mail
diffs or instructions to the address given in the `README' so they can
be considered for the next release.  If you are using the cache, and at
some point `config.cache' contains results you don't want to keep, you
may remove or edit it.

   The file `configure.ac' (or `configure.in') is used to create
`configure' by a program called `autoconf'.  You need `configure.ac' if
```

you want to change it or regenerate ‘configure’ using a newer version
of ‘autoconf’.

   The simplest way to compile this package is:

  1. ‘cd’ to the directory containing the package’s source code and type
     ‘./configure’ to configure the package for your system.

     Running ‘configure’ might take a while. While running, it prints
     some messages telling which features it is checking for.

  2. Type ‘make’ to compile the package.

  3. Optionally, type ‘make check’ to run any self-tests that come with
     the package, generally using the just-built uninstalled binaries.

  4. Type ‘make install’ to install the programs and any data files and
     documentation. When installing into a prefix owned by root, it is
     recommended that the package be configured and built as a regular
     user, and only the ‘make install’ phase executed with root
     privileges.

  5. Optionally, type ‘make installcheck’ to repeat any self-tests, but
     this time using the binaries in their final installed location.
     This target does not install anything. Running this target as a
     regular user, particularly if the prior ‘make install’ required
     root privileges, verifies that the installation completed
     correctly.

  6. You can remove the program binaries and object files from the
     source code directory by typing ‘make clean’. To also remove the
     files that ‘configure’ created (so you can compile the package for
     a different kind of computer), type ‘make distclean’. There is
     also a ‘make maintainer-clean’ target, but that is intended mainly
     for the package’s developers. If you use it, you may have to get
     all sorts of other programs in order to regenerate files that came
     with the distribution.

  7. Often, you can also type ‘make uninstall’ to remove the installed
     files again. In practice, not all packages have tested that
     uninstallation works correctly, even though it is required by the
     GNU Coding Standards.

  8. Some packages, particularly those that use Automake, provide ‘make
     distcheck’, which can by used by developers to test that all other
     targets like ‘make install’ and ‘make uninstall’ work correctly.
     This target is generally not run by end users.

Compilers and Options
=====================

   Some systems require unusual options for compilation or linking that
the ‘configure’ script does not know about. Run ‘./configure --help’
for details on some of the pertinent environment variables.

   You can give ‘configure’ initial values for configuration parameters
by setting variables in the command line or in the environment. Here
is an example:

     ./configure CC=c99 CFLAGS=-g LIBS=-lposix

   *Note Defining Variables::, for more details.

Compiling For Multiple Architectures
====================================

   You can compile the package for more than one kind of computer at the
same time, by placing the object files for each architecture in their
own directory. To do this, you can use GNU ‘make’. ‘cd’ to the
directory where you want the object files and executables to go and run
the ‘configure’ script. ‘configure’ automatically checks for the
source code in the directory that ‘configure’ is in and in ‘..’. This
is known as a "VPATH" build.

```
   With a non-GNU ‘make’, it is safer to compile the package for one
architecture at a time in the source code directory.  After you have
installed the package for one architecture, use ‘make distclean’ before
reconfiguring for another architecture.

   On MacOS X 10.5 and later systems, you can create libraries and
executables that work on multiple system types--known as "fat" or
"universal" binaries--by specifying multiple ‘-arch’ options to the
compiler but only a single ‘-arch’ option to the preprocessor.  Like
this:

     ./configure CC="gcc -arch i386 -arch x86_64 -arch ppc -arch ppc64" \
                 CXX="g++ -arch i386 -arch x86_64 -arch ppc -arch ppc64" \
                 CPP="gcc -E" CXXCPP="g++ -E"

   This is not guaranteed to produce working output in all cases, you
may have to build one architecture at a time and combine the results
using the ‘lipo’ tool if you have problems.

Installation Names
==================

   By default, ‘make install’ installs the package’s commands under
‘/usr/local/bin’, include files under ‘/usr/local/include’, etc.  You
can specify an installation prefix other than ‘/usr/local’ by giving
‘configure’ the option ‘--prefix=PREFIX’, where PREFIX must be an
absolute file name.

   You can specify separate installation prefixes for
architecture-specific files and architecture-independent files.  If you
pass the option ‘--exec-prefix=PREFIX’ to ‘configure’, the package uses
PREFIX as the prefix for installing programs and libraries.
Documentation and other data files still use the regular prefix.

   In addition, if you use an unusual directory layout you can give
options like ‘--bindir=DIR’ to specify different values for particular
kinds of files.  Run ‘configure --help’ for a list of the directories
you can set and what kinds of files go in them.  In general, the
default for these options is expressed in terms of ‘${prefix}’, so that
specifying just ‘--prefix’ will affect all of the other directory
specifications that were not explicitly provided.

   The most portable way to affect installation locations is to pass the
correct locations to ‘configure’; however, many packages provide one or
both of the following shortcuts of passing variable assignments to the
‘make install’ command line to change installation locations without
having to reconfigure or recompile.

   The first method involves providing an override variable for each
affected directory.  For example, ‘make install
prefix=/alternate/directory’ will choose an alternate location for all
directory configuration variables that were expressed in terms of
‘${prefix}’.  Any directories that were specified during ‘configure’,
but not in terms of ‘${prefix}’, must each be overridden at install
time for the entire installation to be relocated.  The approach of
makefile variable overrides for each directory variable is required by
the GNU Coding Standards, and ideally causes no recompilation.
However, some platforms have known limitations with the semantics of
shared libraries that end up requiring recompilation when using this
method, particularly noticeable in packages that use GNU Libtool.

   The second method involves providing the ‘DESTDIR’ variable.  For
example, ‘make install DESTDIR=/alternate/directory’ will prepend
‘/alternate/directory’ before all installation names.  The approach of
‘DESTDIR’ overrides is not required by the GNU Coding Standards, and
does not work on platforms that have drive letters.  On the other hand,
it does better at avoiding recompilation issues, and works well even
when some directory options were not specified in terms of ‘${prefix}’
at ‘configure’ time.

Optional Features
=================
```

If the package supports it, you can cause programs to be installed with an extra prefix or suffix on their names by giving 'configure' the option '--program-prefix=PREFIX' or '--program-suffix=SUFFIX'.

Some packages pay attention to '--enable-FEATURE' options to 'configure', where FEATURE indicates an optional part of the package. They may also pay attention to '--with-PACKAGE' options, where PACKAGE is something like 'gnu-as' or 'x' (for the X Window System). The 'README' should mention any '--enable-' and '--with-' options that the package recognizes.

For packages that use the X Window System, 'configure' can usually find the X include and library files automatically, but if it doesn't, you can use the 'configure' options '--x-includes=DIR' and '--x-libraries=DIR' to specify their locations.

Some packages offer the ability to configure how verbose the execution of 'make' will be. For these packages, running './configure --enable-silent-rules' sets the default to minimal output, which can be overridden with 'make V=1'; while running './configure --disable-silent-rules' sets the default to verbose, which can be overridden with 'make V=0'.

Particular systems

On HP-UX, the default C compiler is not ANSI C compatible. If GNU CC is not installed, it is recommended to use the following options in order to use an ANSI C compiler:

```
./configure CC="cc -Ae -D_XOPEN_SOURCE=500"
```

and if that doesn't work, install pre-built binaries of GCC for HP-UX.

On OSF/1 a.k.a. Tru64, some versions of the default C compiler cannot parse its '<wchar.h>' header file. The option '-nodtk' can be used as a workaround. If GNU CC is not installed, it is therefore recommended to try

```
./configure CC="cc"
```

and if that doesn't work, try

```
./configure CC="cc -nodtk"
```

On Solaris, don't put '/usr/ucb' early in your 'PATH'. This directory contains several dysfunctional programs; working variants of these programs are available in '/usr/bin'. So, if you need '/usr/ucb' in your 'PATH', put it _after_ '/usr/bin'.

On Haiku, software installed for all users goes in '/boot/common', not '/usr/local'. It is recommended to use the following options:

```
./configure --prefix=/boot/common
```

Specifying the System Type

There may be some features 'configure' cannot figure out automatically, but needs to determine by the type of machine the package will run on. Usually, assuming the package is built to be run on the _same_ architectures, 'configure' can figure that out, but if it prints a message saying it cannot guess the machine type, give it the '--build=TYPE' option. TYPE can either be a short name for the system type, such as 'sun4', or a canonical name which has the form:

```
CPU-COMPANY-SYSTEM
```

where SYSTEM can have one of these forms:

```
OS
KERNEL-OS
```

```
   See the file 'config.sub' for the possible values of each field.  If
'config.sub' isn't included in this package, then this package doesn't
need to know the machine type.

   If you are _building_ compiler tools for cross-compiling, you should
use the option '--target=TYPE' to select the type of system they will
produce code for.

   If you want to _use_ a cross compiler, that generates code for a
platform different from the build platform, you should specify the
"host" platform (i.e., that on which the generated programs will
eventually be run) with '--host=TYPE'.

Sharing Defaults
================

   If you want to set default values for 'configure' scripts to share,
you can create a site shell script called 'config.site' that gives
default values for variables like 'CC', 'cache_file', and 'prefix'.
'configure' looks for 'PREFIX/share/config.site' if it exists, then
'PREFIX/etc/config.site' if it exists.  Or, you can set the
'CONFIG_SITE' environment variable to the location of the site script.
A warning: not all 'configure' scripts look for a site script.

Defining Variables
==================

   Variables not defined in a site shell script can be set in the
environment passed to 'configure'.  However, some packages may run
configure again during the build, and the customized values of these
variables may be lost.  In order to avoid this problem, you should set
them in the 'configure' command line, using 'VAR=value'.  For example:

     ./configure CC=/usr/local2/bin/gcc

causes the specified 'gcc' to be used as the C compiler (unless it is
overridden in the site shell script).

Unfortunately, this technique does not work for 'CONFIG_SHELL' due to
an Autoconf bug.  Until the bug is fixed you can use this workaround:

     CONFIG_SHELL=/bin/bash /bin/bash ./configure CONFIG_SHELL=/bin/bash

'configure' Invocation
======================

   'configure' recognizes the following options to control how it
operates.

'--help'
'-h'
     Print a summary of all of the options to 'configure', and exit.

'--help=short'
'--help=recursive'
     Print a summary of the options unique to this package's
     'configure', and exit.  The 'short' variant lists options used
     only in the top level, while the 'recursive' variant lists options
     also present in any nested packages.

'--version'
'-V'
     Print the version of Autoconf used to generate the 'configure'
     script, and exit.

'--cache-file=FILE'
     Enable the cache: use and save the results of the tests in FILE,
     traditionally 'config.cache'.  FILE defaults to '/dev/null' to
     disable caching.

'--config-cache'
'-C'
```

```
     Alias for '--cache-file=config.cache'.

'--quiet'
'--silent'
'-q'
     Do not print messages saying which checks are being made.  To
     suppress all normal output, redirect it to '/dev/null' (any error
     messages will still be shown).

'--srcdir=DIR'
     Look for the package's source code in directory DIR.  Usually
     'configure' can determine that directory automatically.

'--prefix=DIR'
     Use DIR as the installation prefix.  *note Installation Names::
     for more details, including other options available for fine-tuning
     the installation locations.

'--no-create'
'-n'
     Run the configure checks, but stop before creating any output
     files.

'configure' also accepts some other, not widely useful, options.  Run
'configure --help' for more details.
```

Chapter 3

Attribute Conventions

While users have the liberty to select attribute names and formats as they like, the OTF2 project suggests certain conventions to enable interoperability between various OTF2 producers and consumers. The conventions used by Vampir and Score-P are as follows:

3.1 General Style and Formatting

Score-P and Vampir have historically used a variety of conventions for how attribute names should be formatted and whether they should be namespaced. Any OTF2 consumer that intends to read trace files produced by Score-P should read the following sections carefully and note the expected names and types of attributes. For future development, best practice is as follows:

- Place attributes in namespaces in the C++ style, with `::` as a separator
- Use namespaces to disambiguate overlapping attributes, such as thread IDs assigned by different paradigms
- Namespaces may also be used to identify the tool that produced an attribute
- If namespaces are used, they should descend from the parent `OTF2::` namespace
- Format attributes as `ALL_CAPS_WITH_UNDERSCORES`

As a reminder, tools that read OTF2 should gracefully handle the absence of any free-form optional attributes.

3.2 Memory Space Attributes

These attributes may be assigned to any data transfer operation. They should describe respectively the source and destination of the data transfer. They refer to an appropriate LocationGroup. In the case of unified memory, it should generally be possible to determine whether the source and target addresses are actually in main memory (corresponding to the location group of the associated CPU process) or in accelerator memory (corresponding to the location group of the accelerator context).

- *OTF2_TYPE_LOCATION_GROUP* `OTF2::MEMORY_SPACE_SOURCE`
- *OTF2_TYPE_LOCATION_GROUP* `OTF2::MEMORY_SPACE_DESTINATION`

3.3 Memory Allocation Attributes

These attributes are conventionally used to describe memory allocation performed within a region, as described by *Enter* and *Leave* events. They are conventionally of type *OTF2_TYPE_UINT64*. The Vampir tool uses the _ADDRESS suffix as a hint to format an attribute in hexadecimal style. The usage of these attributes is described with respect to standard memory allocation (e.g. `malloc` or `new`).

- *OTF2_TYPE_UINT64* `ALLOCATION_SIZE` Associated with the entry of an allocation region.
- *OTF2_TYPE_UINT64* `RESULT_ADDRESS` Associated with the exit of an allocation region.
- *OTF2_TYPE_UINT64* `DEALLOCATION_SIZE` Associated with the entry of a deallocation region.
- *OTF2_TYPE_UINT64* `ARGUMENT_ADDRESS` Associated with the entry of a deallocation region. Should typically correspond to a `RESULT_ADDRESS`.

More complex behavior may be modeled with combinations of these attributes: for instance, a `realloc` can be represented as a deallocation and allocation, as can a memory move.

3.4 I/O Attributes

This attribute is used in I/O recording to describe the starting point of an I/O operation in a file. This is conventionally of type *OTF2_TYPE_UINT64*. Score-P records this attribute for *IoOperationBegin* events.

- *OTF2_TYPE_UINT64* `Offset`

3.5 Process and Thread Attributes

These attributes are recorded by Score-P upon process or thread creation, as reflected by *ProgramBegin* events for processes and *ThreadBegin* or *ThreadTeamBegin* events for threads. These are conventionally of type *OTF2_TYPE_UINT64*. Note that *ProgramBegin* accepts not only a process ID, but also a thread ID for the initial thread. This allows OTF2 users to omit thread creation events for the main thread of the program while still recording the associated thread ID somewhere. This style is produced by Score-P.

- *OTF2_TYPE_UINT64* `ProcessId`
- *OTF2_TYPE_UINT64* `ThreadId`

Tools writing OTF2 should take particular note of the following:

- If multiple threading paradigms are recorded, it may be necessary to disambiguate which thread IDs map to which paradigm. This may be done through namespacing.
- If a tool creates thread events (begin/end and/or creation/destruction) for the program's main thread, it should either omit the thread ID attribute at program begin or ensure that the thread IDs used for creation of the program and its main thread are consistent.
- The IDs here are optional attributes which may be assigned semantics as desired (for example, values assigned by the operating system or the threading paradigm). This is distinct from the sequence number which is a mandatory attribute on OTF2 thread events. Note that Score-P uses the OS-provided process and thread IDs only; it does not consider any higher-level identifiers when writing these attributes.

Chapter 4

Deprecated List

Module records_definition

In version 2.0

Module records_event

In version 1.2

In version 1.2

In version 1.2

In version 1.2

In version 1.2

In version 1.2

In version 1.2

Chapter 5

Module Documentation

5.1 OTF2 usage examples

Listing of example code.

5.2 OTF2 records

Modules

- List of all definition records
- List of all event records
- List of all marker records
- List of all snapshot records

5.2.1 Detailed Description

OTF2 records

Listings of all OTF2 records.

5.3 OTF2 callbacks

OTF2 callbacks

Description of the non-records callbacks available in OTF2.

5.4 Usage of OTF2 tools

Modules

- OTF2 config tool
- OTF2 print tool
- OTF2 snapshots tool
- OTF2 marker tool
- OTF2 estimator tool

5.4.1 Detailed Description

Usage of OTF2 tools

OTF2 I/O recording

Appendices

Usage instructions of the OTF2 command line tools.

.1 OTF2 config tool

A call to otf2-config has the following syntax:

```
Usage: otf2-config [OPTION]... COMMAND

Commands:
  --cflags     prints additional compiler flags. They already contain
               the include flags
  --cppflags   prints the include flags for the OTF2 headers
  --libs       prints the required libraries for linking
  --ldflags    prints the required linker flags
  --cc         prints the C compiler name
  --features <FEATURE-CATEGORY>
               prints available features selected by <FEATURE-CATEGORY>.
               Available feature categories:
                * substrates
                * compressions
                * targets
  --help       prints this usage information

  --version    prints the version number of the OTF2 package
  --revision   prints the revision number of the OTF2 package
  --interface-version
               prints the interface version number
  --config-summary
               prints the configure summary of the OTF2 package
  --pythonpath
               prints the python path for the OTF2 modules

Options:
  --target <TARGET>
               displays the requested information for the given <TARGET>.
               On non-cross compiling systems, the 'backend' target is ignored.
  --backend    equivalent to '--target backend' (deprecated)
  --cuda       specifies that the required flags are for the CUDA compiler
               nvcc (deprecated)
```

.2 OTF2 print tool

A call to oft2-print has the following syntax:

```
Usage: otf2-print [OPTION]... [--] ANCHORFILE
Print selected content of the OTF2 archive specified by ANCHORFILE.

Options:
  -A, --show-all          print all output including definitions and anchor
                          file
  -G, --show-global-defs  print all global definitions
  -I, --show-info         print information from the anchor file
  -T, --show-thumbnails   print the headers from all thumbnails
  -M, --show-mappings     print mappings to global definitions
  -C, --show-clock-offsets
                          print clock offsets to global timer
      --timestamps=<FORMAT>
                          format of the timestamps. <FORMAT> is one of:
                           plain -  no formatting is done (default)
                           offset - timestamps are relative to the global offset
                                    (taken form the ClockProperties definition)
  -L, --location <LID>    limit output to location <LID>
  -s, --step <N>          step through output by steps of <N> events
      --time <MIN> <MAX>  limit output to events within time interval
      --system-tree       output system tree to dot-file
      --silent            only validate trace and do not print any events
      --unwind-calling-context
                          unwind the calling context for each calling context
                          event. Each calling context node is prefixed depending
                          on the unwind distance of the current event:
                           '?' - unwind distance is undefined
                           '+' - region was newly entered
                           '*' - region was not left
                           ' ' - region did not made any progress
  -Werror, --warnings-as-errors
                          all warnings are treated as errors
  -d, --debug             turn on debug mode
  -V, --version           print version information
  -h, --help              print this help information
```

.3 OTF2 snapshots tool

A call to oft2-snapshots has the following syntax:

```
Usage: otf2-snapshots [OPTION]... ANCHORFILE
Append snapshots to existing otf2 traces at given 'break' timestamps.

Options:
  -n, --number <BREAKS> Number of breaks (distributed regularly)
                        if -p and -t are not set, the default for -n is 10
                        breaks.
  -p <TICK_RATE>        Create break every <TICK_RATE> ticks
                        if both, -n and -p are specified the one producing
                        more breaks wins.
      --progress        Brief mode, print progress information.
      --verbose         Verbose mode, print break timestamps, i.e. snapshot
                        informations to stdout.
  -V, --version         Print version information.
  -h, --help            Print this help information.
```

.4 OTF2 marker tool

A call to oft2-marker has the following syntax:

```
Usage: otf2-marker [OPTION] [ARGUMENTS]... ANCHORFILE
Read or edit a marker file.

Options:
                      Print all markers sorted by group.
      --def <GROUP> [<CATEGORY>]
                      Print all marker definitions of group <GROUP> or of
                      category <CATEGORY> from group <GROUP>.
      --defs-only     Print only marker definitions.
      --add-def <GROUP> <CATEGORY> <SEVERITY>
                      Add a new marker definition.
      --add <GROUP> <CATEGORY> <TIME> <SCOPE> <TEXT>
                      Add a marker to an existing definition.
      --remove-def <GROUP> [<CATEGORY>]
                      Remove all marker classes of group <GROUP> or only the
                      category <CATEGORY> of group <GROUP>; and all according
                      markers.
      --clear-def <GROUP> [<CATEGORY>]
                      Remove all markers of group <GROUP> or only of category
                      <CATEGORY> of group <GROUP>.
      --reset         Reset all marker.
  -V, --version       Print version information.
  -h, --help          Print this help information.

Argument descriptions:
  <GROUP>, <CATEGORY>, <TEXT>
                      Arbitrary strings.
  <SEVERITY>          One of:
                       * NONE
                       * LOW
                       * MEDIUM
                       * HIGH
  <TIME>              One of the following formats:
                       * <TIMESTAMP>
                         A valid timestamp inside the trace range
                         'global offset' and 'global offset' + 'trace
                         length'.
                       * <TIMESTAMP>+<DURATION>
                         <TIMESTAMP> and <TIMESTAMP> + <DURATION> must be valid
                         timestamps inside the trace range 'global
                         offset' and 'global offset' + 'trace length'.
                       * <TIMESTAMP-START>-<TIMESTAMP-END>
                         Two valid timestamps inside the trace range 'global
                         offset' and 'global offset' + 'trace length', with
                         <TIMESTAMP-START> <= <TIMESTAMP-END>.
                      See the CLOCK_PROPERTIES definition with the help
                      of the 'otf2-print -G' tool.
  <SCOPE>[:<SCOPE-REF>]
                       The <SCOPE> must be one of:
                       * GLOBAL
                       * LOCATION:<LOCATION-REF>
                       * LOCATION_GROUP:<LOCATION-GROUP-REF>
                       * SYSTEM_TREE_NODE:<SYSTEM-TREE-NODE-REF>
                       * GROUP:<GROUP-REF>
                       * COMM:<COMMUNICATOR-REF>
                      <SCOPE-REF> must be a valid definition reference of
                      the specified scope. Use 'otf2-print -G' for a list of
                      defined references.
                      There is no <SCOPE-REF> for <SCOPE> 'GLOBAL'.
                      For a scope 'GROUP' the type of the referenced
                      group must be 'OTF2_GROUP_TYPE_LOCATIONS' or
                      'OTF2_GROUP_TYPE_COMM_LOCATIONS'.
```

.5 OTF2 estimator tool

A call to oft2-estimator has the following syntax:

```
Usage: otf2-estimator [OPTION]...
This tool estimates the size of OTF2 events.
It will open a prompt to type in commands.

Options:
  -V, --version       Print version information.
  -h, --help          Print this help information.

Commands:
   list definitions              Lists all known definition names.
   list events                   Lists all known event names.
   list types                    Lists all known type names.
   set <DEFINITION> <NUMBER>     Specifies the number of definitions of a
                                 type of definitions.
   get DefChunkSize              Prints the estimated definition chunk size.
   get Timestamp                 Prints the size of a timestamp.
   get AttributeList [TYPES...]  Prints the estimated size for an attribute
                                 list with the given number of entries and
                                 types.
   get <EVENT> [ARGS...]         Prints the estimated size of records for
                                 <EVENT>.
   exit                          Exits the tool.

This tool provides a command line interface to the estimator API of the OTF2
library. It is based on a stream based protocol.  Commands are send to the
standard input stream of the program and the result is written to the standard
output stream of the program.  All definition and event names are in there
canonical CamelCase form.  Numbers are printed in decimal.  The TYPES are in
ALL_CAPS.  See the output of the appropriate 'list' commands.  Arguments are
separated with an arbitrary number of white space.  The 'get' commands are using
everything after the first white space separator verbatim as a key, which is
then printed in the output line and appended with the estimated size.

Here is a simple example.  We have at most 4 region definitions and one metric
definition.  We want to know the size of a timestamp, enter, and leave event,
and a metric event with 4 values.

cat <<EOC | otf2-estimator
set Region 4
set Metric 1
get Timestamp
get Enter
get Leave
get Metric  4
exit
EOC
Timestamp 9
Enter 3
Leave 3
Metric  4 44
```

.6 OTF2 I/O recording

.6.0.1 Known OTF2 I/O paradigms

The introduction of I/O recording with OTF2 made it necessary to distinguish different I/O paradigms. Like it is done with the parallel paradigms, like MPI, OpenMP. Though instead of the usual enum used to identify the paradigm, the I/O paradigms are expressed in a dynamic way with the help of a definition record. While this has the advantage that the API does not need to be changed only to add new I/O paradigms, it also lacks confidence in the definition. To overcome this, OTF2 textually defines known I/O paradigms and their expected definition.

"POSIX" This is the I/O interface of the *POSIX standard*.

Required properties

Class *OTF2_IO_PARADIGM_CLASS_SERIAL*
Flags *OTF2_IO_PARADIGM_FLAG_OS*

"ISOC" This is the I/O interface of the *ISO C standard*.

Required properties

Class *OTF2_IO_PARADIGM_CLASS_SERIAL*

"MPI-IO" This is the I/O interface of the *Message Passing Interface*.

Required properties

Class *OTF2_IO_PARADIGM_CLASS_PARALLEL*
Flags *OTF2_IO_PARADIGM_FLAG_NONE*

"netCDF" This is the *Network Common Data Form*. The *class* depends on whether the NetCDF library was built with or without MPI support.

Required properties

Class *OTF2_IO_PARADIGM_CLASS_SERIAL* or *OTF2_IO_PARADIGM_CLASS_PARALLEL*
Flags *OTF2_IO_PARADIGM_FLAG_NONE*

"PnetCDF" This is the *Parallel netCDF*.

Required properties

Class *OTF2_IO_PARADIGM_CLASS_PARALLEL*
Flags *OTF2_IO_PARADIGM_FLAG_NONE*

"HDF5" This is the I/O interface of *The HDF Group*. The *class* depends on whether the HDF5 library was built with or without MPI support.

Required properties

Class *OTF2_IO_PARADIGM_CLASS_SERIAL* or *OTF2_IO_PARADIGM_CLASS_PARALLEL*
Flags *OTF2_IO_PARADIGM_FLAG_NONE*

"ADIOS" This is the *Adaptable IO System*.

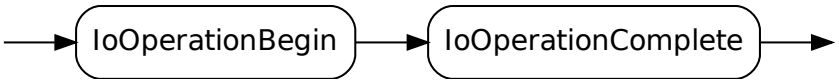
Required properties

Class *OTF2_IO_PARADIGM_CLASS_PARALLEL*
Flags *OTF2_IO_PARADIGM_FLAG_NONE*

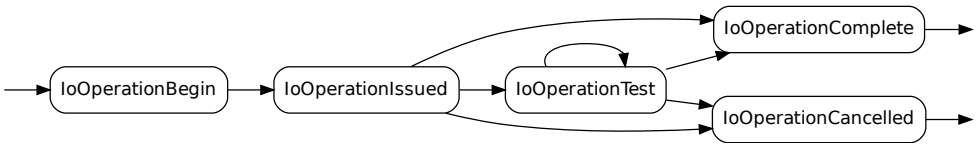
.6.0.2 Event order for I/O operation records

These diagrams show valid event orders of I/O operations, which also denotes the lifetime of the (*IoHandle*, `matchingId`) tuple.

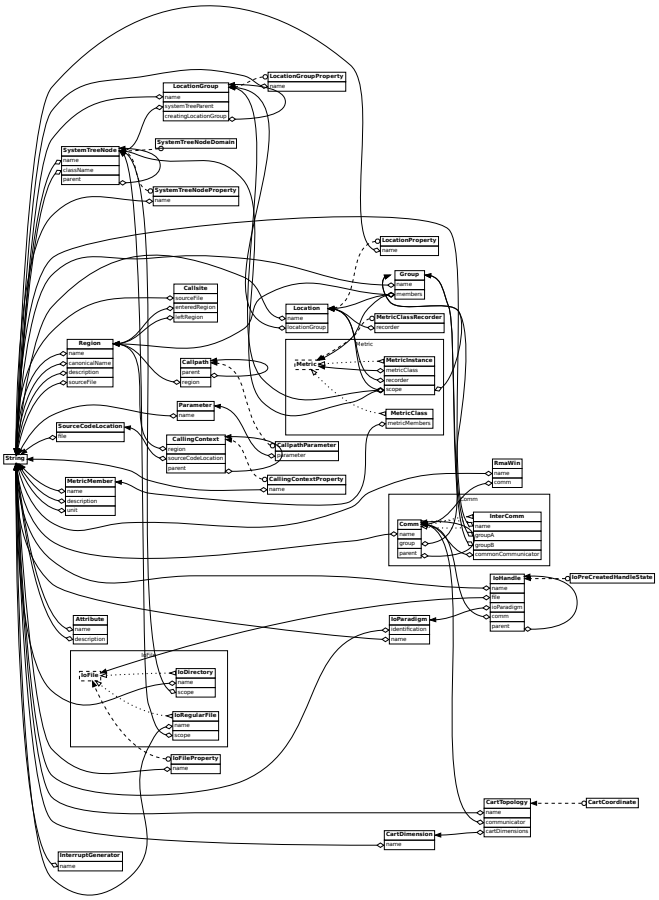
If the *OTF2_IO_OPERATION_FLAG_NON_BLOCKING* is not set in the *IoOperationBegin* record, then the event order must follow:



If the *OTF2_IO_OPERATION_FLAG_NON_BLOCKING* is set in the *IoOperationBegin* record, then the event order must follow:



.7 List of all definition records



.8 ClockProperties

Defines the timer resolution and time range of this trace. There will be no event with a timestamp less than `globalOffset`, and no event with timestamp greater than (`globalOffset` + `traceLength`).

This definition is only valid as a global definition.

Attributes

uint64_t	timerResolution	Ticks per seconds.
uint64_t	globalOffset	A timestamp smaller than all event timestamps.
uint64_t	traceLength	A timespan which includes the timespan between the smallest and greatest timestamp of all event timestamps.
uint64_t	realtime↩ Timestamp	A realtime timestamp of the `globalOffset` timestamp in nanoseconds since 1970-01-01T00:00 UTC. Use *OTF2_UNDEFINED_TIMESTAMP* if no such timestamp exists. Since version 3.0.

See also

OTF2_GlobalDefWriter_WriteClockProperties()
OTF2_GlobalDefReaderCallbacks_SetClockPropertiesCallback()

Since

Version 1.0

.9 Paradigm

Attests that the following parallel paradigm was available at the time when the trace was recorded, and vice versa. Note that this does not attest that the paradigm was used. For convenience, this also includes a proper name for the paradigm and a classification. This definition is only allowed to appear at most once in the definitions per *Paradigm*.

This definition is only valid as a global definition.

Attributes

OTF2_Paradigm	paradigm	The paradigm to attest.
OTF2_StringRef	name	The name of the paradigm. References a *String* definition.
OTF2_ParadigmClass	paradigmClass	The class of this paradigm.

See also

OTF2_GlobalDefWriter_WriteParadigm()
OTF2_GlobalDefReaderCallbacks_SetParadigmCallback()

Since

Version 1.5

.10 ParadigmProperty

Extensible annotation for the *Paradigm* definition.

The tuple (*paradigm*, *property*) must be unique.

This definition is only valid as a global definition.

Attributes

OTF2_Paradigm	paradigm	The paradigm to annotate.
OTF2_ParadigmProperty	property	The property.
OTF2_Type	type	The type of this property. Must match with the defined type of the *property*.
OTF2_AttributeValue	value	The value of this property.

See also

OTF2_GlobalDefWriter_WriteParadigmProperty()
OTF2_GlobalDefReaderCallbacks_SetParadigmPropertyCallback()

Since

Version 1.5

.11 *OTF2_IoParadigmRef* IoParadigm

Attests that the following I/O paradigm was available at the time when the trace was recorded, and vice versa. Note that this does not attest that the paradigm was used. For convenience, this also includes a proper name for the paradigm and a classification.

This definition is only valid as a global definition.

Attributes

OTF2_StringRef	identification	The I/O paradigm identification. This should be used programmatically to identify a specific I/O paradigm. For a human-readable name use the `name` attribute. If this identification matches one of the known I/O paradigms listed in the OTF2 documentation Known OTF2 I/O paradigms, then the attributes of this definition must match those specified there. References a *String* definition.
OTF2_StringRef	name	The name of the I/O paradigm. This should be presented to humans as the name of this I/O paradigm. References a *String* definition.
OTF2_IoParadigmClass	ioParadigmClass	The class of this I/O paradigm.
OTF2_IoParadigmFlag	ioParadigmFlags	Boolean properties of this I/O paradigm.
uint8_t	numberOfProperties	Number of properties.
OTF2_IoParadigmProperty	properties [numberOfProperties]	The property.
OTF2_Type	types [numberOfProperties]	The type of this property. Must match with the defined type of the *property*.
OTF2_AttributeValue	values [numberOfProperties]	The value of this property.

See also

OTF2_GlobalDefWriter_WriteIoParadigm()
OTF2_GlobalDefReaderCallbacks_SetIoParadigmCallback()

Since

Version 2.1

.12 MappingTable

Mapping tables are needed for situations where an ID is not globally known at measurement time. They are applied automatically at reading.

This definition is only valid as a local definition.

Attributes

OTF2_↩ Mapping↩ Type	mappingType	Says to what type of ID the mapping table has to be applied.
const OT↩ F2_IdMap∗	idMap	Mapping table.

See also

OTF2_DefWriter_WriteMappingTable()
OTF2_DefReaderCallbacks_SetMappingTableCallback()

Since

Version 1.0

.13 ClockOffset

Clock offsets are used for clock corrections.

This definition is only valid as a local definition.

Attributes

OTF2_↩ Time↩ Stamp	time	Time when this offset was determined.
int64_t	offset	The offset to the global clock which was determined at `time`.
double	standard↩ Deviation	A possible standard deviation, which can be used as a metric for the quality of the offset.

See also

OTF2_DefWriter_WriteClockOffset()
OTF2_DefReaderCallbacks_SetClockOffsetCallback()

Since

Version 1.0

.14 *OTF2_StringRef* String

The string definition.

Attributes

const char∗	string	The string, null terminated.

See also

OTF2_GlobalDefWriter_WriteString()
OTF2_GlobalDefReaderCallbacks_SetStringCallback()
OTF2_DefWriter_WriteString()
OTF2_DefReaderCallbacks_SetStringCallback()

Since

Version 1.0

.15 *OTF2_AttributeRef* Attribute

The attribute definition.

Attributes

OTF2_↩ StringRef	name	Name of the attribute. References a *String* definition.
OTF2_↩ StringRef	description	Description of the attribute. References a *String* definition. Since version 1.4.
OTF2_↩ Type	type	Type of the attribute value.

See also

OTF2_GlobalDefWriter_WriteAttribute()
OTF2_GlobalDefReaderCallbacks_SetAttributeCallback()
OTF2_DefWriter_WriteAttribute()
OTF2_DefReaderCallbacks_SetAttributeCallback()

Since

Version 1.0

.16 *OTF2_SystemTreeNodeRef* SystemTreeNode

The system tree node definition.

Attributes

OTF2_StringRef	name	Free form instance name of this node. References a *String* definition.
OTF2_StringRef	className	Free form class name of this node References a *String* definition.
OTF2_SystemTreeNodeRef	parent	Parent ID of this node. May be *OTF2_UNDEFINED_SYSTEM_TREE_NODE* to indicate that there is no parent. References a *SystemTreeNode* definition.

Supplements

SystemTreeNodeProperty
SystemTreeNodeDomain

See also

OTF2_GlobalDefWriter_WriteSystemTreeNode()
OTF2_GlobalDefReaderCallbacks_SetSystemTreeNodeCallback()
OTF2_DefWriter_WriteSystemTreeNode()
OTF2_DefReaderCallbacks_SetSystemTreeNodeCallback()

Since

Version 1.0

.17 *OTF2_LocationGroupRef* LocationGroup

The location group definition.

Attributes

OTF2_StringRef	name	Name of the group. References a *String* definition.
OTF2_LocationGroupType	locationGroupType	Type of this group.
OTF2_SystemTreeNodeRef	systemTreeParent	Parent of this location group in the system tree. References a *SystemTreeNode* definition.
OTF2_LocationGroupRef	creatingLocationGroup	The creating location group of this group. For type *OTF2_LOCATION_GROUP_TYPE_PROCESS* this may be another group of type *OTF2_LOCATION_GROUP_TYPE_PROCESS* or *OTF2_UNDEFINED_LOCATION_GROUP*. For type *OTF2_LOCATION_GROUP_TYPE_ACCELERATOR*, this must be a group of type *OTF2_LOCATION_GROUP_TYPE_PROCESS*. References a *LocationGroup* definition. Since version 3.0.

Supplements

LocationGroupProperty

See also

OTF2_GlobalDefWriter_WriteLocationGroup()
OTF2_GlobalDefReaderCallbacks_SetLocationGroupCallback()
OTF2_DefWriter_WriteLocationGroup()
OTF2_DefReaderCallbacks_SetLocationGroupCallback()

Since

Version 1.0

.18 *OTF2_LocationRef* Location

The location definition.

Attributes

OTF2_StringRef	name	Name of the location References a *String* definition.
OTF2_LocationType	locationType	Location type.
uint64_t	numberOfEvents	Number of events this location has recorded.
OTF2_LocationGroupRef	locationGroup	Location group which includes this location. References a *LocationGroup* definition.

Supplements

LocationProperty

See also

OTF2_GlobalDefWriter_WriteLocation()
OTF2_GlobalDefReaderCallbacks_SetLocationCallback()
OTF2_DefWriter_WriteLocation()
OTF2_DefReaderCallbacks_SetLocationCallback()

Since

Version 1.0

.19 *OTF2_RegionRef* Region

The region definition.

Attributes

OTF2_StringRef	name	Name of the region (demangled name if available). References a *String* definition.
OTF2_StringRef	canonicalName	Alternative name of the region (e.g. mangled name). References a *String* definition. Since version 1.1.
OTF2_StringRef	description	A more detailed description of this region. References a *String* definition.
OTF2_RegionRole	regionRole	Region role. Since version 1.1.
OTF2_Paradigm	paradigm	Paradigm. Since version 1.1.
OTF2_RegionFlag	regionFlags	Region flags. Since version 1.1.
OTF2_StringRef	sourceFile	The source file where this region was declared. References a *String* definition.
uint32_t	beginLineNumber	Starting line number of this region in the source file.
uint32_t	endLineNumber	Ending line number of this region in the source file.

See also

OTF2_GlobalDefWriter_WriteRegion()
OTF2_GlobalDefReaderCallbacks_SetRegionCallback()
OTF2_DefWriter_WriteRegion()
OTF2_DefReaderCallbacks_SetRegionCallback()

Since

Version 1.0

.20 *OTF2_CallsiteRef* Callsite

The callsite definition.

Attributes

OTF2_StringRef	sourceFile	The source file where this call was made. References a *String* definition.
uint32_t	lineNumber	Line number in the source file where this call was made.
OTF2_RegionRef	enteredRegion	The region which was called. References a *Region* definition.
OTF2_RegionRef	leftRegion	The region which made the call. References a *Region* definition.

See also

OTF2_GlobalDefWriter_WriteCallsite()
OTF2_GlobalDefReaderCallbacks_SetCallsiteCallback()
OTF2_DefWriter_WriteCallsite()
OTF2_DefReaderCallbacks_SetCallsiteCallback()

Since

Version 1.0

Deprecated In version 2.0

.21 *OTF2_CallpathRef* Callpath

The callpath definition.

Attributes

OTF2_↩ Callpath↩ Ref	parent	The parent of this callpath. References a *Callpath* definition.
OTF2_↩ RegionRef	region	The region of this callpath. References a *Region* definition.

Supplements

CallpathParameter

See also

OTF2_GlobalDefWriter_WriteCallpath()
OTF2_GlobalDefReaderCallbacks_SetCallpathCallback()
OTF2_DefWriter_WriteCallpath()
OTF2_DefReaderCallbacks_SetCallpathCallback()

Since

Version 1.0

.22 *OTF2_GroupRef* Group

The group definition.

Attributes

OTF2_↩ StringRef	name	Name of this group References a *String* definition.
OTF2_↩ GroupType	groupType	The type of this group. Since version 1.2.
OTF2_↩ Paradigm	paradigm	The paradigm of this communication group. Since version 1.2.
OTF2_↩ GroupFlag	groupFlags	Flags for this group. Since version 1.2.
uint32_t	numberOf↩ Members	The number of members in this group.
uint32_t	members [numberOf↩ Members]	The identifiers of the group members.

See also

OTF2_GlobalDefWriter_WriteGroup()
OTF2_GlobalDefReaderCallbacks_SetGroupCallback()
OTF2_DefWriter_WriteGroup()
OTF2_DefReaderCallbacks_SetGroupCallback()

Since

Version 1.0

.23 *OTF2_MetricMemberRef* MetricMember

A metric is defined by a *MetricMember* definition. A metric member is always a member of a metric class. Therefore, a single metric is a special case of a metric class with only one member. It is not allowed to reference a metric member ID in a *Metric* event, but only metric class IDs.

Attributes

OTF2_StringRef	name	Name of the metric. References a *String* definition.
OTF2_StringRef	description	Description of the metric. References a *String* definition.
OTF2_MetricType	metricType	Metric type: PAPI, etc.
OTF2_MetricMode	metricMode	Metric mode: accumulative, fix, relative, etc.
OTF2_Type	valueType	Type of the value. Only *OTF2_TYPE_INT64*, *OTF2_TYPE_UINT64*, and *OTF2_TYPE_DOUBLE* are valid types. If this metric member is recorded in a *Metric* event, than this type and the type in the event must match.
OTF2_Base	base	The recorded values should be handled in this given base, either binary or decimal. This information can be used if the value needs to be scaled.
int64_t	exponent	The values inside the Metric events should be scaled by the factor base^exponent, to get the value in its base unit. For example, if the metric values come in as KiBi, than the base should be *OTF2_BASE_BINARY* and the exponent 10. Than the writer does not need to scale the values up to bytes, but can directly write the KiBi values into the Metric event. At reading time, the reader can apply the scaling factor to get the value in its base unit, ie. in bytes.
OTF2_StringRef	unit	Unit of the metric. This needs to be the scale free base unit, ie. "bytes", "operations", or "seconds". In particular this unit should not have any scale prefix. References a *String* definition.

See also

OTF2_GlobalDefWriter_WriteMetricMember()
OTF2_GlobalDefReaderCallbacks_SetMetricMemberCallback()
OTF2_DefWriter_WriteMetricMember()
OTF2_DefReaderCallbacks_SetMetricMemberCallback()

Since

Version 1.0

.24 *OTF2_MetricRef* Metric

This is a polymorphic definition class.

Derivations

MetricClass
MetricInstance

.25 *OTF2_MetricRef* MetricClass

For a metric class it is implicitly given that the event stream that records the metric is also the scope. A metric class can contain multiple different metrics.

Attributes

uint8_t	numberOf↩ Metrics	Number of metrics within the set.
OTF2_↩ Metric↩ Member↩ Ref	metricMembers [numberOf↩ Metrics]	List of metric members. References a *MetricMember* definition.
OTF2_↩ Metric↩ Occurrence	metric↩ Occurrence	Defines occurrence of a metric set.
OTF2_↩ Recorder↩ Kind	recorderKind	What kind of locations will record this metric class, or will this metric class only be recorded by metric instances. Since version 1.2.

Supplements

MetricClassRecorder

See also

OTF2_GlobalDefWriter_WriteMetricClass()
OTF2_GlobalDefReaderCallbacks_SetMetricClassCallback()
OTF2_DefWriter_WriteMetricClass()
OTF2_DefReaderCallbacks_SetMetricClassCallback()

Since

Version 1.0

.26 *OTF2_MetricRef* MetricInstance

A *MetricInstance* is used to define metrics that are recorded at one location for multiple locations or for another location. The occurrence of a metric instance is implicitly of type *OTF2_METRIC_ASYNCHRONOUS*.

Attributes

OTF2_↩ MetricRef	metricClass	The instanced *MetricClass*. This metric class must be of kind *OTF2_REC↩ ORDER_KIND_ABSTRACT*. References a *MetricClass*, or a *MetricInstance* definition.
OTF2_↩ Location↩ Ref	recorder	Recorder of the metric: location ID. References a *Location* definition.
OTF2_↩ Metric↩ Scope	metricScope	Defines type of scope: location, location group, system tree node, or a generic group of locations.

uint64_t	scope	Scope of metric: ID of a location, location group, system tree node, or a generic group of locations.

See also

OTF2_GlobalDefWriter_WriteMetricInstance()
OTF2_GlobalDefReaderCallbacks_SetMetricInstanceCallback()
OTF2_DefWriter_WriteMetricInstance()
OTF2_DefReaderCallbacks_SetMetricInstanceCallback()

Since

Version 1.0

.27 *OTF2_CommRef* Comm

This is a polymorphic definition class.

Derivations

Comm
InterComm

.28 *OTF2_CommRef* Comm

The communicator definition.

Attributes

OTF2_StringRef	name	The name given by calling MPI_Comm_set_name on this communicator. Or the empty name to indicate that no name was given. References a *String* definition.
OTF2_GroupRef	group	The describing MPI group of this MPI communicator The group needs to be of type *OTF2_GROUP_TYPE_COMM_GROUP* or *OTF2_GROUP_TYPE_COMM_SELF*. References a *Group* definition.
OTF2_CommRef	parent	The parent MPI communicator from which this communicator was created, if any. Use *OTF2_UNDEFINED_COMM* to indicate no parent. References a *Comm* definition.
OTF2_CommFlag	flags	Special characteristics of this communicator. Since version 3.0.

See also

OTF2_GlobalDefWriter_WriteComm()
OTF2_GlobalDefReaderCallbacks_SetCommCallback()
OTF2_DefWriter_WriteComm()
OTF2_DefReaderCallbacks_SetCommCallback()

Since

Version 1.0

.29 *OTF2_ParameterRef* Parameter

The parameter definition.

Attributes

OTF2_StringRef	name	Name of the parameter (variable name etc.) References a *String* definition.
OTF2_ParameterType	parameterType	Type of the parameter, *OTF2_ParameterType* for possible types.

See also

OTF2_GlobalDefWriter_WriteParameter()
OTF2_GlobalDefReaderCallbacks_SetParameterCallback()
OTF2_DefWriter_WriteParameter()
OTF2_DefReaderCallbacks_SetParameterCallback()

Since

Version 1.0

.30 *OTF2_RmaWinRef* RmaWin

A window defines the communication context for any remote-memory access operation.

Attributes

OTF2_StringRef	name	Name, e.g. 'GASPI Queue 1', 'NVidia Card 2', etc.. References a *String* definition.
OTF2_CommRef	comm	Communicator object used to create the window. References a *Comm* definition.
OTF2_RmaWinFlag	flags	Special characteristics of this RMA window. Since version 3.0.

See also

OTF2_GlobalDefWriter_WriteRmaWin()
OTF2_GlobalDefReaderCallbacks_SetRmaWinCallback()
OTF2_DefWriter_WriteRmaWin()
OTF2_DefReaderCallbacks_SetRmaWinCallback()

Since

Version 1.2

.31 MetricClassRecorder

The metric class recorder definition.

Attributes

OTF2_↩ MetricRef	metric	Parent *MetricClass*, or *MetricInstance* definition to which this one is a supplementary definition. References a *MetricClass*, or a *MetricInstance* definition.
OTF2_↩ Location↩ Ref	recorder	The location which recorded the referenced metric class. References a *Location* definition.

See also

OTF2_GlobalDefWriter_WriteMetricClassRecorder()
OTF2_GlobalDefReaderCallbacks_SetMetricClassRecorderCallback()
OTF2_DefWriter_WriteMetricClassRecorder()
OTF2_DefReaderCallbacks_SetMetricClassRecorderCallback()

Since

Version 1.2

.32 SystemTreeNodeProperty

An arbitrary key/value property for a *SystemTreeNode* definition.

Attributes

OTF2_↩ System↩ Tree↩ NodeRef	systemTreeNode	Parent *SystemTreeNode* definition to which this one is a supplementary definition. References a *SystemTreeNode* definition.
OTF2_↩ StringRef	name	Name of the property. References a *String* definition.
OTF2_↩ Type	type	The type of this property. Since version 2.0.
OTF2_↩ Attribute↩ Value	value	The value of this property. Since version 2.0.

See also

OTF2_GlobalDefWriter_WriteSystemTreeNodeProperty()
OTF2_GlobalDefReaderCallbacks_SetSystemTreeNodePropertyCallback()
OTF2_DefWriter_WriteSystemTreeNodeProperty()
OTF2_DefReaderCallbacks_SetSystemTreeNodePropertyCallback()

Since

Version 1.2

.33 SystemTreeNodeDomain

The system tree node domain definition.

Attributes

OTF2_↩ System↩ Tree↩ NodeRef	systemTreeNode	Parent *SystemTreeNode* definition to which this one is a supplementary definition. References a *SystemTreeNode* definition.
OTF2_↩ System↩ Tree↩ Domain	systemTree↩ Domain	The domain in which the referenced *SystemTreeNode* operates in.

See also

OTF2_GlobalDefWriter_WriteSystemTreeNodeDomain()
OTF2_GlobalDefReaderCallbacks_SetSystemTreeNodeDomainCallback()
OTF2_DefWriter_WriteSystemTreeNodeDomain()
OTF2_DefReaderCallbacks_SetSystemTreeNodeDomainCallback()

Since

Version 1.2

.34 LocationGroupProperty

An arbitrary key/value property for a *LocationGroup* definition.

Attributes

OTF2_↩ Location↩ GroupRef	locationGroup	Parent *LocationGroup* definition to which this one is a supplementary definition. References a *LocationGroup* definition.
OTF2_↩ StringRef	name	Name of the property. References a *String* definition.
OTF2_↩ Type	type	The type of this property. Since version 2.0.
OTF2_↩ Attribute↩ Value	value	The value of this property. Since version 2.0.

See also

OTF2_GlobalDefWriter_WriteLocationGroupProperty()
OTF2_GlobalDefReaderCallbacks_SetLocationGroupPropertyCallback()
OTF2_DefWriter_WriteLocationGroupProperty()
OTF2_DefReaderCallbacks_SetLocationGroupPropertyCallback()

Since

Version 1.3

.35 LocationProperty

An arbitrary key/value property for a *Location* definition.

Attributes

OTF2_Location Ref	location	Parent *Location* definition to which this one is a supplementary definition. References a *Location* definition.
OTF2_StringRef	name	Name of the property. References a *String* definition.
OTF2_Type	type	The type of this property. Since version 2.0.
OTF2_Attribute Value	value	The value of this property. Since version 2.0.

See also

OTF2_GlobalDefWriter_WriteLocationProperty()
OTF2_GlobalDefReaderCallbacks_SetLocationPropertyCallback()
OTF2_DefWriter_WriteLocationProperty()
OTF2_DefReaderCallbacks_SetLocationPropertyCallback()

Since

Version 1.3

.36 *OTF2_CartDimensionRef* CartDimension

Each dimension in a Cartesian topology is composed of a global ID, a name, its size, and whether it is periodic or not.

Attributes

OTF2_StringRef	name	The name of the Cartesian topology dimension. References a *String* definition.
uint32_t	size	The size of the Cartesian topology dimension.
OTF2_Cart Periodicity	cartPeriodicity	Periodicity of the Cartesian topology dimension.

See also

OTF2_GlobalDefWriter_WriteCartDimension()
OTF2_GlobalDefReaderCallbacks_SetCartDimensionCallback()
OTF2_DefWriter_WriteCartDimension()
OTF2_DefReaderCallbacks_SetCartDimensionCallback()

Since

Version 1.3

.37 *OTF2_CartTopologyRef* CartTopology

Each topology is described by a global ID, a reference to its name, a reference to a communicator, the number of dimensions, and references to those dimensions. The topology type is defined by the paradigm of the group referenced by the associated communicator.

Attributes

OTF2_StringRef	name	The name of the topology. References a *String* definition.
OTF2_CommRef	communicator	Communicator object used to create the topology. References a *Comm* definition.
uint8_t	numberOfDimensions	Number of dimensions.
OTF2_CartDimensionRef	cartDimensions [numberOfDimensions]	The dimensions of this topology. References a *CartDimension* definition.

Supplements

CartCoordinate

See also

OTF2_GlobalDefWriter_WriteCartTopology()
OTF2_GlobalDefReaderCallbacks_SetCartTopologyCallback()
OTF2_DefWriter_WriteCartTopology()
OTF2_DefReaderCallbacks_SetCartTopologyCallback()

Since

Version 1.3

.38 CartCoordinate

Defines the coordinate of the location referenced by the given rank (w.r.t. the communicator associated to the topology) in the referenced topology.

Attributes

OTF2_CartTopologyRef	cartTopology	Parent *CartTopology* definition to which this one is a supplementary definition. References a *CartTopology* definition.
uint32_t	rank	The rank w.r.t. the communicator associated to the topology referencing this coordinate.
uint8_t	numberOfDimensions	Number of dimensions.
uint8_t	coordinates [numberOfDimensions]	Coordinates, indexed by dimension.

See also

OTF2_GlobalDefWriter_WriteCartCoordinate()
OTF2_GlobalDefReaderCallbacks_SetCartCoordinateCallback()
OTF2_DefWriter_WriteCartCoordinate()
OTF2_DefReaderCallbacks_SetCartCoordinateCallback()

Since

Version 1.3

.39 *OTF2_SourceCodeLocationRef* SourceCodeLocation

The definition of a source code location as tuple of the corresponding file name and line number.

When used to attach source code annotations to events, use the *OTF2_AttributeList* with an *Attribute* definition named "SOURCE_CODE_LOCATION" and of type *OTF2_TYPE_SOURCE_CODE_LOCATION*.

Attributes

OTF2_StringRef	file	The name of the file for the source code location. References a *String* definition.
uint32_t	lineNumber	The line number for the source code location.

See also

OTF2_GlobalDefWriter_WriteSourceCodeLocation()
OTF2_GlobalDefReaderCallbacks_SetSourceCodeLocationCallback()
OTF2_DefWriter_WriteSourceCodeLocation()
OTF2_DefReaderCallbacks_SetSourceCodeLocationCallback()

Since

Version 1.5

.40 *OTF2_CallingContextRef* CallingContext

Defines a node in the calling context tree. These nodes are referenced in the *CallingContextSample*, *CallingContextEnter*, and *CallingContextLeave* events.

The referenced *CallingContext* node in these events form a path which represents the calling context at this time. This path will be partitioned into at most three sub-paths by the *unwindDistance* attribute. For the *CallingContextLeave* event, the *unwindDistance* is defined to be 1.

Starting from the referenced *CallingContext* node, the first $N \geq 0$ nodes were newly entered regions since the previous calling context event. The next node is a region which was not left but made progress since the previous calling context event. All other nodes did not make progress at all, and thus the regions were neither left nor entered again. The *unwindDistance* is then $N + 1$. In case the unwindDistance is 0, there are neither newly entered regions nor regions which made progress.

It is guaranteed, that the node referenced by the *unwindDistance* exists in the previous and current calling context. All descendants of this node's child in the previous calling context were left since the previous calling context event.

It is valid that this node is the *OTF2_UNDEFINED_CALLING_CONTEXT* node and that this node is already reached after *unwindDistance* -1 steps. In the latter case, there exists no region which made progress, all regions in the previous calling context were left and all regions in the current calling context were newly entered.

Note that for *CallingContextLeave* events, the parent of the referenced *CallingContext* must be used as the previous calling context for the next event.

Regions which were entered with a *CallingContextEnter* event form an upper bound for the unwind distance, i.e., the *unwindDistance* points either to the parent of the last such entered region, or a node which is a descendant to this parent.

To summarize, an *unwindDistance* of 0 means that no regions were left, newly entered, or made any progress. An *unwindDistance* of 1 means that some regions were left regarding the previous calling context, no regions were newly entered, and there was progress in the region of the first node. An *unwindDistance* greater than 1 means that some regions were left regarding the previous calling context, there was progress in one region, and the first *unwindDistance* -1 regions were newly entered.

Attributes

OTF2_RegionRef	region	The region. References a *Region* definition.
OTF2_SourceCodeLocationRef	sourceCodeLocation	The absolute source code location of this calling context. References a *SourceCodeLocation* definition.
OTF2_CallingContextRef	parent	Parent ID of this context. References a *CallingContext* definition.

Supplements

CallingContextProperty

See also

OTF2_GlobalDefWriter_WriteCallingContext()
OTF2_GlobalDefReaderCallbacks_SetCallingContextCallback()
OTF2_DefWriter_WriteCallingContext()
OTF2_DefReaderCallbacks_SetCallingContextCallback()

Since

Version 1.5

.41 CallingContextProperty

An arbitrary key/value property for a *CallingContext* definition.

Attributes

OTF2_CallingContextRef	callingContext	Parent *CallingContext* definition to which this one is a supplementary definition. References a *CallingContext* definition.
OTF2_StringRef	name	Property name. References a *String* definition.
OTF2_Type	type	The type of this property. Must match with the defined type of the *property*.
OTF2_AttributeValue	value	The value of this property.

See also

OTF2_GlobalDefWriter_WriteCallingContextProperty()
OTF2_GlobalDefReaderCallbacks_SetCallingContextPropertyCallback()
OTF2_DefWriter_WriteCallingContextProperty()
OTF2_DefReaderCallbacks_SetCallingContextPropertyCallback()

Since

Version 2.0

.42 *OTF2_InterruptGeneratorRef* InterruptGenerator

Defines an interrupt generator which periodically triggers *CallingContextSample* events. If the mode of the interrupt generator is set to *OTF2_INTERRUPT_GENERATOR_MODE_TIME*, the generator produces interrupts which are uniformly distributed over time, and the unit of the period is implicitly in seconds. If the mode is *OTF2_INTERRUP↩ T_GENERATOR_MODE_COUNT*, the interrupt is triggered if a specific counter threshold is reached in the system. Therefore these samples are unlikely to be uniformly distributed over time. The unit of the period is then implicitly a number (threshold value).

The interrupts period in base unit (which is implicitly seconds or number, based on the `mode`) is derived out of the `base`, the `exponent`, and the `period` attributes by this formula:

base-period = period x base^exponent

Attributes

OTF2_↩ StringRef	name	The name of this interrupt generator. References a *String* definition.
OTF2_↩ Interrupt↩ Generator↩ Mode	interrupt↩ GeneratorMode	Mode of the interrupt generator.
OTF2_↩ Base	base	The base for the period calculation.
int64_t	exponent	The exponent for the period calculation.
uint64_t	period	The period this interrupt generator generates interrupts.

See also

OTF2_GlobalDefWriter_WriteInterruptGenerator()
OTF2_GlobalDefReaderCallbacks_SetInterruptGeneratorCallback()
OTF2_DefWriter_WriteInterruptGenerator()
OTF2_DefReaderCallbacks_SetInterruptGeneratorCallback()

Since

Version 1.5

.43 IoFileProperty

Extensible annotation for the polymorphic *IoFile* definition class.

The tuple (*ioFile*, *name*) must be unique.

Attributes

OTF2_Io↩ FileRef	ioFile	Parent *IoRegularFile* definition to which this one is a supplementary definition. References a *IoRegularFile* definition.

OTF2_StringRef	name	Property name. References a *String* definition.
OTF2_Type	type	The type of this property.
OTF2_AttributeValue	value	The value of this property.

See also

OTF2_GlobalDefWriter_WriteIoFileProperty()
OTF2_GlobalDefReaderCallbacks_SetIoFilePropertyCallback()
OTF2_DefWriter_WriteIoFileProperty()
OTF2_DefReaderCallbacks_SetIoFilePropertyCallback()

Since

Version 2.1

.44 *OTF2_IoFileRef* IoFile

This is a polymorphic definition class.

Derivations

IoRegularFile
IoDirectory

.45 *OTF2_IoFileRef* IoRegularFile

Defines a regular file from which an *IoHandle* can be created.

This definition is member of the polymorphic *IoFile* definition class. All definitions of this polymorphic definition class share the same global identifier namespace.

Attributes

OTF2_StringRef	name	Name of the file. References a *String* definition.
OTF2_SystemTreeNodeRef	scope	Defines the physical scope of this *IoRegularFile* in the system tree. E.g., two *IoRegularFile* definitions with the same name but different `scope` values are physically different, thus I/O operations through *IoHandle*s do not operate on the same file. References a *SystemTreeNode* definition.

Supplements

IoFileProperty

See also

OTF2_GlobalDefWriter_WriteIoRegularFile()
OTF2_GlobalDefReaderCallbacks_SetIoRegularFileCallback()
OTF2_DefWriter_WriteIoRegularFile()
OTF2_DefReaderCallbacks_SetIoRegularFileCallback()

Since

Version 2.1

.46 *OTF2_IoFileRef* IoDirectory

Defines a directory from which an *IoHandle* can be created.

This definition is member of the polymorphic *IoFile* definition class. All definitions of this polymorphic definition class share the same global identifier namespace.

Attributes

OTF2_StringRef	name	Name of the directory. References a *String* definition.
OTF2_SystemTreeNodeRef	scope	Defines the physical scope of this *IoDirectory* in the system tree. E.g., two *IoDirectory* definitions with the same name but different `scope` values are physically different, thus I/O operations through *IoHandle*s do not operate on the same directory. References a *SystemTreeNode* definition.

See also

OTF2_GlobalDefWriter_WriteIoDirectory()
OTF2_GlobalDefReaderCallbacks_SetIoDirectoryCallback()
OTF2_DefWriter_WriteIoDirectory()
OTF2_DefReaderCallbacks_SetIoDirectoryCallback()

Since

Version 2.1

.47 *OTF2_IoHandleRef* IoHandle

Defines an I/O handle which will be used by subsequent I/O operations. I/O operations can only be applied to *active* I/O handles. An I/O handle gets *active* either if it was marked with the *OTF2_IO_HANDLE_FLAG_PRE_CREATED* flag, after it was referenced in an *IoCreateHandle* event, or it was referenced in the *newHandle* attribute of an *IoDuplicateHandle* event. It gets *inactive* if it was referenced in an *IoDestroyHandle* event. This life cycle can be repeated indefinitely. Though the *OTF2_IO_HANDLE_FLAG_PRE_CREATED* flag is unset after a *IoDuplicateHandle* event. All *Location*s of a *LocationGroup* have access to an *active* *IoHandle*, regardless which *Location* of the *LocationGroup* activated the *IoHandle*.

Attributes

OTF2_StringRef	name	Handle name. References a *String* definition.
OTF2_IoFileRef	file	File identifier. References a *IoRegularFile*, or a *IoDirectory* definition.
OTF2_IoParadigmRef	ioParadigm	The I/O paradigm. References a *IoParadigm* definition.
OTF2_IoHandleFlag	ioHandleFlags	Special characteristics of this handle.
OTF2_CommRef	comm	Scope of the file handle. This scope defines which process can access this file via this handle and also defines the collective context for this handle. References a *Comm* definition.
OTF2_IoHandleRef	parent	Parent, in case this I/O handle was created and operated by an higher- level I/O paradigm. References a *IoHandle* definition.

Supplements

IoPreCreatedHandleState

See also

OTF2_GlobalDefWriter_WriteIoHandle()
OTF2_GlobalDefReaderCallbacks_SetIoHandleCallback()
OTF2_DefWriter_WriteIoHandle()
OTF2_DefReaderCallbacks_SetIoHandleCallback()

Since

Version 2.1

.48 IoPreCreatedHandleState

Provide the I/O access mode and status flags for *pre-created IoHandle*s.

Only allowed once for a *IoHandle* definition with the *OTF2_IO_HANDLE_FLAG_PRE_CREATED* flag set.

Attributes

OTF2_IoHandleRef	ioHandle	Parent *IoHandle* definition to which this one is a supplementary definition. References a *IoHandle* definition.
OTF2_IoAccessMode	mode	The access mode of the *pre-created IoHandle*.
OTF2_IoStatusFlag	statusFlags	The status flags of the *pre-created IoHandle*.

See also

OTF2_GlobalDefWriter_WriteIoPreCreatedHandleState()
OTF2_GlobalDefReaderCallbacks_SetIoPreCreatedHandleStateCallback()
OTF2_DefWriter_WriteIoPreCreatedHandleState()
OTF2_DefReaderCallbacks_SetIoPreCreatedHandleStateCallback()

Since

Version 2.1

.49 CallpathParameter

A parameter for a callpath definition.

Attributes

OTF2_Callpath_Ref	callpath	Parent *Callpath* definition to which this one is a supplementary definition. References a *Callpath* definition.
OTF2_Parameter_Ref	parameter	The parameter of this callpath. References a *Parameter* definition.
OTF2_Type	type	The type of the attribute value. Must match the type of the parameter.
OTF2_Attribute_Value	value	The value of the parameter for this callpath.

See also

OTF2_GlobalDefWriter_WriteCallpathParameter()
OTF2_GlobalDefReaderCallbacks_SetCallpathParameterCallback()
OTF2_DefWriter_WriteCallpathParameter()
OTF2_DefReaderCallbacks_SetCallpathParameterCallback()

Since

Version 2.2

.50 *OTF2_CommRef* InterComm

The inter-communicator definition.

Attributes

OTF2_StringRef	name	The name given by calling MPI_Comm_set_name on this communicator. Or the empty name to indicate that no name was given. References a *String* definition.
OTF2_GroupRef	groupA	One of the two MPI process groups in the intercommunicator. The group needs to be of type *OTF2_GROUP_TYPE_COMM_GROUP* or *OTF2_GROUP_TYPE_COMM_SELF*. References a *Group* definition.
OTF2_GroupRef	groupB	The other of the two MPI process groups in the intercommunicator. The group needs to be of type *OTF2_GROUP_TYPE_COMM_GROUP* or *OTF2_GROUP_TYPE_COMM_SELF*. References a *Group* definition.
OTF2_CommRef	commonCommunicator	The common peer MPI communicator used to create this inter- communicator. Use *OTF2_UNDEFINED_COMM* if no such communicator was used. References a *Comm*, or a *InterComm* definition.
OTF2_CommFlag	flags	Special characteristics of this communicator.

See also

OTF2_GlobalDefWriter_WriteInterComm()
OTF2_GlobalDefReaderCallbacks_SetInterCommCallback()
OTF2_DefWriter_WriteInterComm()
OTF2_DefReaderCallbacks_SetInterCommCallback()

Since

Version 3.0

.51 List of all event records

.52 BufferFlush

This event signals that the internal buffer was flushed at the given time.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
OTF2_↩ Time↩ Stamp	stopTime	The time the buffer flush finished.

See also

OTF2_EvtWriter_BufferFlush()
OTF2_GlobalEvtReaderCallbacks_SetBufferFlushCallback()
OTF2_EvtReaderCallbacks_SetBufferFlushCallback()

Since

Version 1.0

.53 MeasurementOnOff

This event signals where the measurement system turned measurement on or off.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
OTF2_↩ Measurement↩ Mode	measurement↩ Mode	Is the measurement turned on (*OTF2_MEASUREMENT_ON*) or off (*OTF2↩ _MEASUREMENT_OFF*)?

See also

OTF2_EvtWriter_MeasurementOnOff()
OTF2_GlobalEvtReaderCallbacks_SetMeasurementOnOffCallback()
OTF2_EvtReaderCallbacks_SetMeasurementOnOffCallback()

Since

Version 1.0

.54 Enter

An *Enter* record indicates that the program enters a code region.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_RegionRef	region	Needs to be defined in a definition record References a *Region* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_REGION* is available.

See also

OTF2_EvtWriter_Enter()
OTF2_GlobalEvtReaderCallbacks_SetEnterCallback()
OTF2_EvtReaderCallbacks_SetEnterCallback()

Since

Version 1.0

.55 Leave

A *Leave* record indicates that the program leaves a code region.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_RegionRef	region	Needs to be defined in a definition record References a *Region* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_REGION* is available.

See also

OTF2_EvtWriter_Leave()
OTF2_GlobalEvtReaderCallbacks_SetLeaveCallback()
OTF2_EvtReaderCallbacks_SetLeaveCallback()

Since

Version 1.0

.56 MpiSend

An *MpiSend* record indicates that an MPI send operation was initiated (MPI_SEND). It keeps the necessary information for this event: receiver of the message, communicator, and the message tag. You can optionally add further information like the message length (size of the send buffer).

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
uint32_t	receiver	MPI rank of receiver in `communicator`.
OTF2_CommRef	communicator	Communicator ID. References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_COMM* is available.
uint32_t	msgTag	Message tag
uint64_t	msgLength	Message length

See also

OTF2_EvtWriter_MpiSend()
OTF2_GlobalEvtReaderCallbacks_SetMpiSendCallback()
OTF2_EvtReaderCallbacks_SetMpiSendCallback()

Since

Version 1.0

.57 MpiIsend

An *MpiIsend* record indicates that a non-blocking MPI send operation was initiated (MPI_ISEND). It keeps the necessary information for this event: receiver of the message, communicator, and the message tag. You can optionally add further information like the message length (size of the send buffer).

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
uint32_t	receiver	MPI rank of receiver in `communicator`.
OTF2_CommRef	communicator	Communicator ID. References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_COMM* is available.
uint32_t	msgTag	Message tag
uint64_t	msgLength	Message length
uint64_t	requestID	ID of the related request

See also

OTF2_EvtWriter_MpiIsend()
OTF2_GlobalEvtReaderCallbacks_SetMpiIsendCallback()
OTF2_EvtReaderCallbacks_SetMpiIsendCallback()

Since

Version 1.0

.58 MpiIsendComplete

An *MpiIsendComplete* record indicates the completion of a non- blocking MPI send operation. In the case where the send request is released before it is completed by MPI, this record will only indicate the release, as it becomes impossible to track the completion of the send operation afterwards. This case may be identified by the surrounding events.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
uint64_t	requestID	ID of the related request

See also

OTF2_EvtWriter_MpiIsendComplete()
OTF2_GlobalEvtReaderCallbacks_SetMpiIsendCompleteCallback()
OTF2_EvtReaderCallbacks_SetMpiIsendCompleteCallback()

Since

Version 1.0

.59 MpiIrecvRequest

An *MpiIrecvRequest* record indicates that a non-blocking MPI receive operation was initiated (MPI_IRECV).

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
uint64_t	requestID	ID of the requested receive

See also

OTF2_EvtWriter_MpiIrecvRequest()
OTF2_GlobalEvtReaderCallbacks_SetMpiIrecvRequestCallback()
OTF2_EvtReaderCallbacks_SetMpiIrecvRequestCallback()

Since

Version 1.0

.60 MpiRecv

An *MpiRecv* record indicates that an MPI message was received (MPI_RECV). It keeps the necessary information for this event: sender of the message, communicator, and the message tag. You can optionally add further information like the message length (size of the receive buffer).

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
uint32_t	sender	MPI rank of sender in `communicator`.
OTF2_CommRef	communicator	Communicator ID. References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_COMM* is available.
uint32_t	msgTag	Message tag
uint64_t	msgLength	Message length

See also

OTF2_EvtWriter_MpiRecv()
OTF2_GlobalEvtReaderCallbacks_SetMpiRecvCallback()
OTF2_EvtReaderCallbacks_SetMpiRecvCallback()

Since

Version 1.0

.61 MpiIrecv

An *MpiIrecv* record indicates the completion of a non-blocking MPI receive operation completed (MPI_IRECV). It keeps the necessary information for this event: sender of the message, communicator, and the message tag. You can optionally add further information like the message length (size of the receive buffer).

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
uint32_t	sender	MPI rank of sender in `communicator`.
OTF2_CommRef	communicator	Communicator ID. References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_COMM* is available.
uint32_t	msgTag	Message tag
uint64_t	msgLength	Message length
uint64_t	requestID	ID of the related request

See also

OTF2_EvtWriter_MpiIrecv()
OTF2_GlobalEvtReaderCallbacks_SetMpiIrecvCallback()
OTF2_EvtReaderCallbacks_SetMpiIrecvCallback()

Since

Version 1.0

.62 MpiRequestTest

This events appears if the program tests if a request has already completed but the test failed.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
uint64_t	requestID	ID of the related request

See also

OTF2_EvtWriter_MpiRequestTest()
OTF2_GlobalEvtReaderCallbacks_SetMpiRequestTestCallback()
OTF2_EvtReaderCallbacks_SetMpiRequestTestCallback()

Since

Version 1.0

.63 MpiRequestCancelled

This events appears if the program canceled a request.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
uint64_t	requestID	ID of the related request

See also

OTF2_EvtWriter_MpiRequestCancelled()
OTF2_GlobalEvtReaderCallbacks_SetMpiRequestCancelledCallback()
OTF2_EvtReaderCallbacks_SetMpiRequestCancelledCallback()

Since

Version 1.0

.64 MpiCollectiveBegin

An *MpiCollectiveBegin* record marks the begin of an MPI collective operation (MPI_GATHER, MPI_SCATTER etc.).

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.

See also

OTF2_EvtWriter_MpiCollectiveBegin()
OTF2_GlobalEvtReaderCallbacks_SetMpiCollectiveBeginCallback()
OTF2_EvtReaderCallbacks_SetMpiCollectiveBeginCallback()

Since

Version 1.0

.65 MpiCollectiveEnd

An *MpiCollectiveEnd* record marks the end of an MPI collective operation (MPI_GATHER, MPI_SCATTER etc.). It keeps the necessary information for this event: type of collective operation, communicator, the root of this collective operation. You can optionally add further information like sent and received bytes.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
OTF2_↩ Collective↩ Op	collectiveOp	Determines which collective operation it is.
OTF2_↩ CommRef	communicator	Communicator References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING↩ _COMM* is available.
uint32_t	root	Rank of root in `communicator` or any predefined constant of *OTF2_↩ CollectiveRoot*.
uint64_t	sizeSent	Size of the sent message.
uint64_t	sizeReceived	Size of the received message.

See also

OTF2_EvtWriter_MpiCollectiveEnd()
OTF2_GlobalEvtReaderCallbacks_SetMpiCollectiveEndCallback()
OTF2_EvtReaderCallbacks_SetMpiCollectiveEndCallback()

Since

Version 1.0

.66 OmpFork

An *OmpFork* record marks that an OpenMP Thread forks a thread team.

This event record is superseded by the *ThreadFork* event record and should not be used when the *ThreadFork* event record is in use.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
uint32_t	numberOf↩ Requested↩ Threads	Requested size of the team.

See also

OTF2_EvtWriter_OmpFork()
OTF2_GlobalEvtReaderCallbacks_SetOmpForkCallback()
OTF2_EvtReaderCallbacks_SetOmpForkCallback()

Since

Version 1.0

Deprecated In version 1.2

.67 OmpJoin

An *OmpJoin* record marks that a team of threads is joint and only the master thread continues execution.

This event record is superseded by the *ThreadJoin* event record and should not be used when the *ThreadJoin* event record is in use.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.

See also

OTF2_EvtWriter_OmpJoin()
OTF2_GlobalEvtReaderCallbacks_SetOmpJoinCallback()
OTF2_EvtReaderCallbacks_SetOmpJoinCallback()

Since

Version 1.0

Deprecated In version 1.2

.68 OmpAcquireLock

An *OmpAcquireLock* record marks that a thread acquires an OpenMP lock.

This event record is superseded by the *ThreadAcquireLock* event record and should not be used when the *ThreadAcquireLock* event record is in use.

Attributes

OTF2_LocationRef	location	The location where this event happened.
OTF2_TimeStamp	timestamp	The time when this event happened.
uint32_t	lockID	ID of the lock.
uint32_t	acquisitionOrder	A monotonically increasing number to determine the order of lock acquisitions (with unsynchronized clocks this is otherwise not possible). Corresponding acquire-release events have same number.

See also

OTF2_EvtWriter_OmpAcquireLock()
OTF2_GlobalEvtReaderCallbacks_SetOmpAcquireLockCallback()
OTF2_EvtReaderCallbacks_SetOmpAcquireLockCallback()

Since

Version 1.0

Deprecated In version 1.2

.69 OmpReleaseLock

An *OmpReleaseLock* record marks that a thread releases an OpenMP lock.

This event record is superseded by the *ThreadReleaseLock* event record and should not be used when the *ThreadReleaseLock* event record is in use.

Attributes

OTF2_LocationRef	location	The location where this event happened.
OTF2_TimeStamp	timestamp	The time when this event happened.

uint32_t	lockID	ID of the lock.
uint32_t	acquisitionOrder	A monotonically increasing number to determine the order of lock acquisitions (with unsynchronized clocks this is otherwise not possible). Corresponding acquire-release events have same number.

See also

OTF2_EvtWriter_OmpReleaseLock()
OTF2_GlobalEvtReaderCallbacks_SetOmpReleaseLockCallback()
OTF2_EvtReaderCallbacks_SetOmpReleaseLockCallback()

Since

Version 1.0

Deprecated In version 1.2

.70 OmpTaskCreate

An *OmpTaskCreate* record marks that an OpenMP Task was/will be created in the current region.

This event record is superseded by the *ThreadTaskCreate* event record and should not be used when the *ThreadTaskCreate* event record is in use.

Attributes

OTF2_Location Ref	location	The location where this event happened.
OTF2_Time Stamp	timestamp	The time when this event happened.
uint64_t	taskID	Identifier of the newly created task instance.

See also

OTF2_EvtWriter_OmpTaskCreate()
OTF2_GlobalEvtReaderCallbacks_SetOmpTaskCreateCallback()
OTF2_EvtReaderCallbacks_SetOmpTaskCreateCallback()

Since

Version 1.0

Deprecated In version 1.2

.71 OmpTaskSwitch

An *OmpTaskSwitch* record indicates that the execution of the current task will be suspended and another task starts/restarts its execution. Please note that this may change the current call stack of the executing location.

This event record is superseded by the *ThreadTaskSwitch* event record and should not be used when the *ThreadTaskSwitch* event record is in use.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
uint64_t	taskID	Identifier of the now active task instance.

See also

OTF2_EvtWriter_OmpTaskSwitch()
OTF2_GlobalEvtReaderCallbacks_SetOmpTaskSwitchCallback()
OTF2_EvtReaderCallbacks_SetOmpTaskSwitchCallback()

Since

Version 1.0

Deprecated In version 1.2

.72 OmpTaskComplete

An *OmpTaskComplete* record indicates that the execution of an OpenMP task has finished.

This event record is superseded by the *ThreadTaskComplete* event record and should not be used when the *ThreadTaskComplete* event record is in use.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
uint64_t	taskID	Identifier of the completed task instance.

See also

OTF2_EvtWriter_OmpTaskComplete()
OTF2_GlobalEvtReaderCallbacks_SetOmpTaskCompleteCallback()
OTF2_EvtReaderCallbacks_SetOmpTaskCompleteCallback()

Since

Version 1.0

Deprecated In version 1.2

.73 Metric

A *Metric* event is always stored at the location that recorded the metric. The event can reference a *MetricClass* or *MetricInstance*. Therefore, metric classes and instances share same ID space. Synchronous metrics are always located right immediately the corresponding *Enter* and *Leave* event.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_MetricRef	metric	Could be a metric class or a metric instance. References a *MetricClass*, or a *MetricInstance* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_METRIC* is available.
uint8_t	numberOfMetrics	Number of metrics with in the set.
OTF2_Type	typeIDs [numberOfMetrics]	List of metric types. These types must match that of the corresponding *MetricMember* definitions.
OTF2_MetricValue	metricValues [numberOfMetrics]	List of metric values.

See also

OTF2_EvtWriter_Metric()
OTF2_GlobalEvtReaderCallbacks_SetMetricCallback()
OTF2_EvtReaderCallbacks_SetMetricCallback()

Since

Version 1.0

.74 ParameterString

A *ParameterString* record marks that in the current region, the specified string parameter has the specified value.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_Parameter_Ref	parameter	Parameter ID. References a *Parameter* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_PARAMETER* is available.

OTF2_StringRef	string	Value: Handle of a string definition References a *String* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_STRING* is available.

See also

OTF2_EvtWriter_ParameterString()
OTF2_GlobalEvtReaderCallbacks_SetParameterStringCallback()
OTF2_EvtReaderCallbacks_SetParameterStringCallback()

Since

Version 1.0

.75 ParameterInt

A *ParameterInt* record marks that in the current region, the specified integer parameter has the specified value.

Attributes

OTF2_LocationRef	location	The location where this event happened.
OTF2_TimeStamp	timestamp	The time when this event happened.
OTF2_ParameterRef	parameter	Parameter ID. References a *Parameter* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_PARAMETER* is available.
int64_t	value	Value of the recorded parameter.

See also

OTF2_EvtWriter_ParameterInt()
OTF2_GlobalEvtReaderCallbacks_SetParameterIntCallback()
OTF2_EvtReaderCallbacks_SetParameterIntCallback()

Since

Version 1.0

.76 ParameterUnsignedInt

A *ParameterUnsignedInt* record marks that in the current region, the specified unsigned integer parameter has the specified value.

Attributes

OTF2_Location↩Ref	location	The location where this event happened.
OTF2_Time↩Stamp	timestamp	The time when this event happened.
OTF2_Parameter↩Ref	parameter	Parameter ID. References a *Parameter* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_PARAMETER* is available.
uint64_t	value	Value of the recorded parameter.

See also

OTF2_EvtWriter_ParameterUnsignedInt()
OTF2_GlobalEvtReaderCallbacks_SetParameterUnsignedIntCallback()
OTF2_EvtReaderCallbacks_SetParameterUnsignedIntCallback()

Since

Version 1.0

.77 RmaWinCreate

An *RmaWinCreate* record denotes the creation of an RMA window. Only valid if the *RmaWin* definition was flagged with *OTF2_RMA_WIN_FLAG_CREATE_DESTROY_EVENTS*. This event can be enclosed by an *RmaCollective↩Begin* and *RmaCollectiveEnd* event pair either with *OTF2_COLLECTIVE_OP_CREATE_HANDLE* or *OTF2_CO↩LLECTIVE_OP_CREATE_HANDLE_AND_ALLOCATE* as the operation type.

Attributes

OTF2_Location↩Ref	location	The location where this event happened.
OTF2_Time↩Stamp	timestamp	The time when this event happened.
OTF2_RmaWin↩Ref	win	ID of the window created. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.

See also

OTF2_EvtWriter_RmaWinCreate()
OTF2_GlobalEvtReaderCallbacks_SetRmaWinCreateCallback()
OTF2_EvtReaderCallbacks_SetRmaWinCreateCallback()

Since

Version 1.2

.78 RmaWinDestroy

An *RmaWinDestroy* record denotes the destruction of an RMA window. Only valid if the *RmaWin* definition was flagged with *OTF2_RMA_WIN_FLAG_CREATE_DESTROY_EVENTS*. This event can be enclosed by an *RmaCollectiveBegin* and *RmaCollectiveEnd* event pair either with *OTF2_COLLECTIVE_OP_DESTROY_HANDLE* or *OTF2_COLLECTIVE_OP_DESTROY_HANDLE_AND_DEALLOCATE* as the operation type. In this case the RMA window is only marked for destruction, which will happen with the *RmaCollectiveEnd* event.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_RmaWin_Ref	win	ID of the window destructed. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.

See also

OTF2_EvtWriter_RmaWinDestroy()
OTF2_GlobalEvtReaderCallbacks_SetRmaWinDestroyCallback()
OTF2_EvtReaderCallbacks_SetRmaWinDestroyCallback()

Since

Version 1.2

.79 RmaCollectiveBegin

An *RmaCollectiveBegin* record denotes the beginning of a collective RMA operation.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.

See also

OTF2_EvtWriter_RmaCollectiveBegin()
OTF2_GlobalEvtReaderCallbacks_SetRmaCollectiveBeginCallback()
OTF2_EvtReaderCallbacks_SetRmaCollectiveBeginCallback()

Since

Version 1.2

.80 RmaCollectiveEnd

An *RmaCollectiveEnd* record denotes the end of a collective RMA operation.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_CollectiveOp	collectiveOp	Determines which collective operation it is.
OTF2_RmaSyncLevel	syncLevel	Synchronization level of this collective operation.
OTF2_RmaWin_Ref	win	ID of the window used for this operation. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.
uint32_t	root	Root process for this operation or any predefined constant of *OTF2_CollectiveRoot*.
uint64_t	bytesSent	Bytes sent in operation.
uint64_t	bytesReceived	Bytes receives in operation.

See also

OTF2_EvtWriter_RmaCollectiveEnd()
OTF2_GlobalEvtReaderCallbacks_SetRmaCollectiveEndCallback()
OTF2_EvtReaderCallbacks_SetRmaCollectiveEndCallback()

Since

Version 1.2

.81 RmaGroupSync

An *RmaGroupSync* record denotes the synchronization with a subgroup of processes on an RMA window.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_RmaSyncLevel	syncLevel	Synchronization level of this collective operation.

OTF2_RmaWinRef	win	ID of the window used for this operation. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.
OTF2_GroupRef	group	Group of remote processes involved in synchronization. References a *Group* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_GROUP* is available.

See also

OTF2_EvtWriter_RmaGroupSync()
OTF2_GlobalEvtReaderCallbacks_SetRmaGroupSyncCallback()
OTF2_EvtReaderCallbacks_SetRmaGroupSyncCallback()

Since

Version 1.2

.82 RmaRequestLock

An *RmaRequestLock* record denotes the time a lock was requested and with it the earliest time it could have been granted. It is used to mark (possibly) non-blocking lock request, as defined by the MPI standard.

Attributes

OTF2_LocationRef	location	The location where this event happened.
OTF2_TimeStamp	timestamp	The time when this event happened.
OTF2_RmaWinRef	win	ID of the window used for this operation. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.
uint32_t	remote	Rank of the locked remote process or *OTF2_UNDEFINED_UINT32* if all processes of the specified window are locked.
uint64_t	lockId	ID of the lock acquired, if multiple locks are defined on a window.
OTF2_LockType	lockType	Type of lock acquired.

See also

OTF2_EvtWriter_RmaRequestLock()
OTF2_GlobalEvtReaderCallbacks_SetRmaRequestLockCallback()
OTF2_EvtReaderCallbacks_SetRmaRequestLockCallback()

Since

Version 1.2

.83 RmaAcquireLock

An *RmaAcquireLock* record denotes the time a lock was acquired by the process.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_RmaWin_Ref	win	ID of the window used for this operation. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.
uint32_t	remote	Rank of the locked remote process or *OTF2_UNDEFINED_UINT32* if all processes of the specified window are locked.
uint64_t	lockId	ID of the lock acquired, if multiple locks are defined on a window.
OTF2_LockType	lockType	Type of lock acquired.

See also

OTF2_EvtWriter_RmaAcquireLock()
OTF2_GlobalEvtReaderCallbacks_SetRmaAcquireLockCallback()
OTF2_EvtReaderCallbacks_SetRmaAcquireLockCallback()

Since

Version 1.2

.84 RmaTryLock

An *RmaTryLock* record denotes the time of an unsuccessful attempt to acquire the lock.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_RmaWin_Ref	win	ID of the window used for this operation. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.
uint32_t	remote	Rank of the locked remote process or *OTF2_UNDEFINED_UINT32* if all processes of the specified window are locked.
uint64_t	lockId	ID of the lock acquired, if multiple locks are defined on a window.
OTF2_LockType	lockType	Type of lock acquired.

See also

OTF2_EvtWriter_RmaTryLock()
OTF2_GlobalEvtReaderCallbacks_SetRmaTryLockCallback()
OTF2_EvtReaderCallbacks_SetRmaTryLockCallback()

Since

Version 1.2

.85 RmaReleaseLock

An *RmaReleaseLock* record denotes the time the lock was released.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_RmaWin_Ref	win	ID of the window used for this operation. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.
uint32_t	remote	Rank of the locked remote process or *OTF2_UNDEFINED_UINT32* if all processes of the specified window are locked.
uint64_t	lockId	ID of the lock released, if multiple locks are defined on a window.

See also

OTF2_EvtWriter_RmaReleaseLock()
OTF2_GlobalEvtReaderCallbacks_SetRmaReleaseLockCallback()
OTF2_EvtReaderCallbacks_SetRmaReleaseLockCallback()

Since

Version 1.2

.86 RmaSync

An *RmaSync* record denotes the direct synchronization with a possibly remote process.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_RmaWin_Ref	win	ID of the window used for this operation. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.
uint32_t	remote	Rank of the locked remote process.
OTF2_Rma_SyncType	syncType	Type of synchronization.

See also

OTF2_EvtWriter_RmaSync()
OTF2_GlobalEvtReaderCallbacks_SetRmaSyncCallback()
OTF2_EvtReaderCallbacks_SetRmaSyncCallback()

Since

Version 1.2

.87 RmaWaitChange

An *RmaWaitChange* record denotes the change of a window that was waited for.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_RmaWin_Ref	win	ID of the window used for this operation. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.

See also

OTF2_EvtWriter_RmaWaitChange()
OTF2_GlobalEvtReaderCallbacks_SetRmaWaitChangeCallback()
OTF2_EvtReaderCallbacks_SetRmaWaitChangeCallback()

Since

Version 1.2

.88 RmaPut

An *RmaPut* record denotes the time a put operation was issued.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_RmaWin_Ref	win	ID of the window used for this operation. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.
uint32_t	remote	Rank of the target process.
uint64_t	bytes	Bytes sent to target.
uint64_t	matchingId	ID used for matching the corresponding completion record.

See also

OTF2_EvtWriter_RmaPut()
OTF2_GlobalEvtReaderCallbacks_SetRmaPutCallback()
OTF2_EvtReaderCallbacks_SetRmaPutCallback()

Since

Version 1.2

.89 RmaGet

An *RmaGet* record denotes the time a get operation was issued.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_RmaWin_Ref	win	ID of the window used for this operation. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.
uint32_t	remote	Rank of the target process.
uint64_t	bytes	Bytes received from target.
uint64_t	matchingId	ID used for matching the corresponding completion record.

See also

OTF2_EvtWriter_RmaGet()
OTF2_GlobalEvtReaderCallbacks_SetRmaGetCallback()
OTF2_EvtReaderCallbacks_SetRmaGetCallback()

Since

Version 1.2

.90 RmaAtomic

An *RmaAtomic* record denotes the time an atomic RMA operation was issued.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_RmaWin_Ref	win	ID of the window used for this operation. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.
uint32_t	remote	Rank of the target process.
OTF2_Rma_Atomic_Type	type	Type of atomic operation.
uint64_t	bytesSent	Bytes sent to target.
uint64_t	bytesReceived	Bytes received from target.
uint64_t	matchingId	ID used for matching the corresponding completion record.

See also

OTF2_EvtWriter_RmaAtomic()
OTF2_GlobalEvtReaderCallbacks_SetRmaAtomicCallback()
OTF2_EvtReaderCallbacks_SetRmaAtomicCallback()

Since

Version 1.2

.91 RmaOpCompleteBlocking

An *RmaOpCompleteBlocking* record denotes the local completion of a blocking RMA operation.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_RmaWin_Ref	win	ID of the window used for this operation. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.
uint64_t	matchingId	ID used for matching the corresponding RMA operation record.

See also

OTF2_EvtWriter_RmaOpCompleteBlocking()
OTF2_GlobalEvtReaderCallbacks_SetRmaOpCompleteBlockingCallback()
OTF2_EvtReaderCallbacks_SetRmaOpCompleteBlockingCallback()

Since

Version 1.2

.92 RmaOpCompleteNonBlocking

An *RmaOpCompleteNonBlocking* record denotes the local completion of a non-blocking RMA operation.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_RmaWin_Ref	win	ID of the window used for this operation. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.
uint64_t	matchingId	ID used for matching the corresponding RMA operation record.

See also

OTF2_EvtWriter_RmaOpCompleteNonBlocking()
OTF2_GlobalEvtReaderCallbacks_SetRmaOpCompleteNonBlockingCallback()
OTF2_EvtReaderCallbacks_SetRmaOpCompleteNonBlockingCallback()

Since

Version 1.2

.93 RmaOpTest

An *RmaOpTest* record denotes that a non-blocking RMA operation has been tested for completion unsuccessfully.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_RmaWin_Ref	win	ID of the window used for this operation. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.
uint64_t	matchingId	ID used for matching the corresponding RMA operation record.

See also

OTF2_EvtWriter_RmaOpTest()
OTF2_GlobalEvtReaderCallbacks_SetRmaOpTestCallback()
OTF2_EvtReaderCallbacks_SetRmaOpTestCallback()

Since

Version 1.2

.94 RmaOpCompleteRemote

An *RmaOpCompleteRemote* record denotes the remote completion of an RMA operation.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_RmaWin_Ref	win	ID of the window used for this operation. References a *RmaWin* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_RMA_WIN* is available.
uint64_t	matchingId	ID used for matching the corresponding RMA operation record.

See also

OTF2_EvtWriter_RmaOpCompleteRemote()
OTF2_GlobalEvtReaderCallbacks_SetRmaOpCompleteRemoteCallback()
OTF2_EvtReaderCallbacks_SetRmaOpCompleteRemoteCallback()

Since

Version 1.2

.95 ThreadFork

A *ThreadFork* record marks that a thread forks a thread team.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_Paradigm	model	The threading paradigm this event took place.
uint32_t	numberOfRequestedThreads	Requested size of the team.

See also

OTF2_EvtWriter_ThreadFork()
OTF2_GlobalEvtReaderCallbacks_SetThreadForkCallback()
OTF2_EvtReaderCallbacks_SetThreadForkCallback()

Since

Version 1.2

.96 ThreadJoin

A *ThreadJoin* record marks that a team of threads is joint and only the master thread continues execution.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_Paradigm	model	The threading paradigm this event took place.

See also

OTF2_EvtWriter_ThreadJoin()
OTF2_GlobalEvtReaderCallbacks_SetThreadJoinCallback()
OTF2_EvtReaderCallbacks_SetThreadJoinCallback()

Since

Version 1.2

.97 ThreadTeamBegin

The current location enters the specified thread team.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
OTF2_↩ CommRef	threadTeam	Thread team References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPIN↩ G_COMM* is available.

See also

OTF2_EvtWriter_ThreadTeamBegin()
OTF2_GlobalEvtReaderCallbacks_SetThreadTeamBeginCallback()
OTF2_EvtReaderCallbacks_SetThreadTeamBeginCallback()

Since

Version 1.2

.98 ThreadTeamEnd

The current location leaves the specified thread team.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
OTF2_↩ CommRef	threadTeam	Thread team References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPIN↩ G_COMM* is available.

See also

OTF2_EvtWriter_ThreadTeamEnd()
OTF2_GlobalEvtReaderCallbacks_SetThreadTeamEndCallback()
OTF2_EvtReaderCallbacks_SetThreadTeamEndCallback()

Since

Version 1.2

.99 ThreadAcquireLock

A *ThreadAcquireLock* record marks that a thread acquires a lock.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
OTF2_↩ Paradigm	model	The threading paradigm this event took place.
uint32_t	lockID	ID of the lock.
uint32_t	acquisitionOrder	A monotonically increasing number to determine the order of lock acquisitions (with unsynchronized clocks this is otherwise not possible). Corresponding acquire-release events have same number.

See also

OTF2_EvtWriter_ThreadAcquireLock()
OTF2_GlobalEvtReaderCallbacks_SetThreadAcquireLockCallback()
OTF2_EvtReaderCallbacks_SetThreadAcquireLockCallback()

Since

Version 1.2

.100 ThreadReleaseLock

A *ThreadReleaseLock* record marks that a thread releases a lock.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
OTF2_↩ Paradigm	model	The threading paradigm this event took place.
uint32_t	lockID	ID of the lock.
uint32_t	acquisitionOrder	A monotonically increasing number to determine the order of lock acquisitions (with unsynchronized clocks this is otherwise not possible). Corresponding acquire-release events have same number.

See also

OTF2_EvtWriter_ThreadReleaseLock()
OTF2_GlobalEvtReaderCallbacks_SetThreadReleaseLockCallback()
OTF2_EvtReaderCallbacks_SetThreadReleaseLockCallback()

Since

Version 1.2

.101 ThreadTaskCreate

A *ThreadTaskCreate* record marks that a task in was/will be created and will be processed by the specified thread team.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
OTF2_↩ CommRef	threadTeam	Thread team References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPIN↩ G_COMM* is available.
uint32_t	creatingThread	Creating thread of this task.
uint32_t	generation↩ Number	Thread-private generation number of task's creating thread.

See also

OTF2_EvtWriter_ThreadTaskCreate()
OTF2_GlobalEvtReaderCallbacks_SetThreadTaskCreateCallback()
OTF2_EvtReaderCallbacks_SetThreadTaskCreateCallback()

Since

Version 1.2

.102 ThreadTaskSwitch

A *ThreadTaskSwitch* record indicates that the execution of the current task will be suspended and another task starts/restarts its execution. Please note that this may change the current call stack of the executing location.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
OTF2_↩ CommRef	threadTeam	Thread team References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPIN↩ G_COMM* is available.
uint32_t	creatingThread	Creating thread of this task.
uint32_t	generation↩ Number	Thread-private generation number of task's creating thread.

See also

OTF2_EvtWriter_ThreadTaskSwitch()
OTF2_GlobalEvtReaderCallbacks_SetThreadTaskSwitchCallback()
OTF2_EvtReaderCallbacks_SetThreadTaskSwitchCallback()

Since

Version 1.2

.103 ThreadTaskComplete

A *ThreadTaskComplete* record indicates that the execution of an OpenMP task has finished.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_CommRef	threadTeam	Thread team References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_COMM* is available.
uint32_t	creatingThread	Creating thread of this task.
uint32_t	generationNumber	Thread-private generation number of task's creating thread.

See also

OTF2_EvtWriter_ThreadTaskComplete()
OTF2_GlobalEvtReaderCallbacks_SetThreadTaskCompleteCallback()
OTF2_EvtReaderCallbacks_SetThreadTaskCompleteCallback()

Since

Version 1.2

.104 ThreadCreate

The location created successfully a new thread.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_CommRef	threadContingent	The thread contingent. References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_COMM* is available.
uint64_t	sequenceCount	A `threadContingent` unique number. The corresponding *ThreadBegin* event does have the same number.

See also

OTF2_EvtWriter_ThreadCreate()
OTF2_GlobalEvtReaderCallbacks_SetThreadCreateCallback()
OTF2_EvtReaderCallbacks_SetThreadCreateCallback()

Since

Version 1.3

.105 ThreadBegin

Marks the begin of a thread created by another thread.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_CommRef	threadContingent	The thread contingent. References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_COMM* is available.
uint64_t	sequenceCount	A `threadContingent` unique number. The corresponding *ThreadCreate* event does have the same number.

See also

OTF2_EvtWriter_ThreadBegin()
OTF2_GlobalEvtReaderCallbacks_SetThreadBeginCallback()
OTF2_EvtReaderCallbacks_SetThreadBeginCallback()

Since

Version 1.3

.106 ThreadWait

The location waits for the completion of another thread.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_CommRef	threadContingent	The thread contingent. References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_COMM* is available.
uint64_t	sequenceCount	A `threadContingent` unique number. The corresponding *ThreadEnd* event does have the same number.

See also

OTF2_EvtWriter_ThreadWait()
OTF2_GlobalEvtReaderCallbacks_SetThreadWaitCallback()
OTF2_EvtReaderCallbacks_SetThreadWaitCallback()

Since

Version 1.3

.107 ThreadEnd

Marks the end of a thread.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_CommRef	threadContingent	The thread contingent. References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_COMM* is available.
uint64_t	sequenceCount	A `threadContingent` unique number. The corresponding *ThreadWait* event does have the same number. *OTF2_UNDEFINED_UINT64* in case no corresponding *ThreadWait* event exists.

See also

OTF2_EvtWriter_ThreadEnd()
OTF2_GlobalEvtReaderCallbacks_SetThreadEndCallback()
OTF2_EvtReaderCallbacks_SetThreadEndCallback()

Since

Version 1.3

.108 CallingContextEnter

The thread entered an instrumented region, represented by the referenced *CallingContext*. In contrast to the *Enter* event, it gives the full calling context through the *CallingContext* tree.

Events based on the *CallingContext* definition are mutually exclusive with the *Enter*/*Leave* events in a trace.

If no callback for this event is set but a callback for *Enter* events is defined, the reader will automatically generate an *Enter* callback call for the *Region* referenced by the *CallingContext* attribute of this event. Note that this emulation does **not** re-create the full calling context! It only re-creates the event order for instrumented regions.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_CallingContextRef	callingContext	The entered region as referenced by the *CallingContext* definition. References a *CallingContext* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_CALLING_CONTEXT* is available.

uint32_t	unwindDistance	The unwindDistance for this `callingContext`. See the description in *CallingContext*.

See also

OTF2_EvtWriter_CallingContextEnter()
OTF2_GlobalEvtReaderCallbacks_SetCallingContextEnterCallback()
OTF2_EvtReaderCallbacks_SetCallingContextEnterCallback()

Since

Version 2.0

.109 CallingContextLeave

The thread left an instrumented region, represented by the referenced *CallingContext*. In contrast to the *Leave* event, it gives the full calling context through the *CallingContext* tree.

The unwind distance for this *CallingContext* is defined to be 1. Because it must be assumed that the instrumented region made progress since the previous *CallingContext* event.

Events based on the *CallingContext* definition are mutually exclusive with the *Enter*/*Leave* events in a trace.

The parent of the *CallingContext* must be used as the previous calling context for the next event.

If no callback for this event is set but a callback for *Leave* events is defined, the reader will automatically generate an *Leave* callback call for the *Region* referenced by the *CallingContext* attribute of this event. Note that this emulation does **not** re-create the full calling context! It only re-creates the event order for instrumented regions.

Attributes

OTF2_Location Ref	location	The location where this event happened.
OTF2_Time Stamp	timestamp	The time when this event happened.
OTF2_Calling ContextRef	callingContext	The left region as referenced by the *CallingContext* definition. References a *CallingContext* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_CALLING_CONTEXT* is available.

See also

OTF2_EvtWriter_CallingContextLeave()
OTF2_GlobalEvtReaderCallbacks_SetCallingContextLeaveCallback()
OTF2_EvtReaderCallbacks_SetCallingContextLeaveCallback()

Since

Version 2.0

.110 CallingContextSample

The thread was interrupted to take a sample of its current state (region and source code location).

Events based on the *CallingContext* definition are mutually exclusive with the *Enter*/*Leave* events in a trace.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_Calling_ContextRef	callingContext	Describes the calling context of the thread when it was interrupted. References a *CallingContext* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_CALLING_CONTEXT* is available.
uint32_t	unwindDistance	The unwindDistance for this `callingContext`. See the description in *CallingContext*.
OTF2_Interrupt_Generator_Ref	interruptGenerator	References a *InterruptGenerator* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_INTERRUPT_GENERATOR* is available.

See also

OTF2_EvtWriter_CallingContextSample()
OTF2_GlobalEvtReaderCallbacks_SetCallingContextSampleCallback()
OTF2_EvtReaderCallbacks_SetCallingContextSampleCallback()

Since

Version 1.5

.111 IoCreateHandle

An *IoCreateHandle* record marks the creation of a new *active* I/O handle that can be used by subsequent I/O operation events.

An *IoHandle* is *active* between a pair of consecutive *IoCreateHandle* and *IoDestroyHandle* events. All *Location*s of a *LocationGroup* have access to an *active* *IoHandle*.

If the *Comm* attribute of the *IoHandle* handle is not *OTF2_UNDEFINED_COMM*, this is a collective operation over *Comm*.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_IoHandleRef	handle	A previously *inactive* I/O handle which will be activated by this record. References a *IoHandle* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_IO_HANDLE* is available.
OTF2_IoAccessMode	mode	Determines which I/O operations can be applied to this I/O handle (e.g., read-only, write-only, read-write).
OTF2_IoCreationFlag	creationFlags	Requested I/O handle creation flags (e.g., create, exclusive, etc.).
OTF2_IoStatusFlag	statusFlags	I/O handle status flags which will be associated with the `handle` attribute (e.g., append, create, close-on-exec, async, etc).

See also

OTF2_EvtWriter_IoCreateHandle()
OTF2_GlobalEvtReaderCallbacks_SetIoCreateHandleCallback()
OTF2_EvtReaderCallbacks_SetIoCreateHandleCallback()

Since

Version 2.1

.112 IoDestroyHandle

An *IoDestroyHandle* record marks the end of an *active* I/O handle's lifetime.

An *IoHandle* is *active* between a pair of consecutive *IoCreateHandle* and *IoDestroyHandle* events. All *Location*s of a *LocationGroup* have access to an *active IoHandle*.

If the *Comm* attribute of the *IoHandle* handle is not *OTF2_UNDEFINED_COMM*, this is a collective operation over *Comm*.

Attributes

OTF2_Location Ref	location	The location where this event happened.
OTF2_Time Stamp	timestamp	The time when this event happened.
OTF2_IoHandleRef	handle	An *active* I/O handle which will be inactivated by this records. References a *IoHandle* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_IO_HANDLE* is available.

See also

OTF2_EvtWriter_IoDestroyHandle()
OTF2_GlobalEvtReaderCallbacks_SetIoDestroyHandleCallback()
OTF2_EvtReaderCallbacks_SetIoDestroyHandleCallback()

Since

Version 2.1

.113 IoDuplicateHandle

An *IoDuplicateHandle* record marks the duplication of an already existing *active* I/O handle.

The new I/O handle `newHandle` is *active* after this event.

Both *IoHandle*s must reference the same *Comm* definition or be *OTF2_UNDEFINED_COMM*. If the *Comm* attribute of the *IoHandle* handles is not *OTF2_UNDEFINED_COMM*, this is a collective operation over *Comm*.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_IoHandleRef	oldHandle	An *active* I/O handle. References a *IoHandle* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_IO_HANDLE* is available.
OTF2_IoHandleRef	newHandle	A previously *inactive* I/O handle which will be activated by this record. References a *IoHandle* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_IO_HANDLE* is available.
OTF2_IoStatusFlag	statusFlags	The status flag for the new I/O handle `newHandle`. No status flags will be inherited from the I/O handle `oldHandle`.

See also

OTF2_EvtWriter_IoDuplicateHandle()
OTF2_GlobalEvtReaderCallbacks_SetIoDuplicateHandleCallback()
OTF2_EvtReaderCallbacks_SetIoDuplicateHandleCallback()

Since

Version 2.1

.114 IoSeek

An *IoSeek* record marks a change of the position, e.g., within a file.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_IoHandleRef	handle	An *active* I/O handle. References a *IoHandle* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_IO_HANDLE* is available.
int64_t	offsetRequest	Requested offset.
OTF2_IoSeekOption	whence	Position inside the file from where `offsetRequest` should be applied (e.g., absolute from the start or end, relative to the current position).
uint64_t	offsetResult	Resulting offset, e.g., within the file relative to the beginning of the file.

See also

OTF2_EvtWriter_IoSeek()
OTF2_GlobalEvtReaderCallbacks_SetIoSeekCallback()
OTF2_EvtReaderCallbacks_SetIoSeekCallback()

Since

Version 2.1

.115 IoChangeStatusFlags

An *IoChangeStatusFlags* record marks a change to the status flags associated with an *active* I/O handle.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_IoHandleRef	handle	An *active* I/O handle. References a *IoHandle* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_IO_HANDLE* is available.
OTF2_IoStatusFlag	statusFlags	Set flags (e.g., close-on-exec, append, etc.).

See also

OTF2_EvtWriter_IoChangeStatusFlags()
OTF2_GlobalEvtReaderCallbacks_SetIoChangeStatusFlagsCallback()
OTF2_EvtReaderCallbacks_SetIoChangeStatusFlagsCallback()

Since

Version 2.1

.116 IoDeleteFile

An *IoDeleteFile* record marks the deletion of an I/O file.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_IoParadigm_Ref	ioParadigm	The I/O paradigm which induced the deletion. References a *IoParadigm* definition.
OTF2_IoFileRef	file	File identifier. References a *IoRegularFile*, or a *IoDirectory* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_IO_FILE* is available.

See also

OTF2_EvtWriter_IoDeleteFile()
OTF2_GlobalEvtReaderCallbacks_SetIoDeleteFileCallback()
OTF2_EvtReaderCallbacks_SetIoDeleteFileCallback()

Since

Version 2.1

.117 IoOperationBegin

An *IoOperationBegin* record marks the begin of a file operation (read, write, etc.).

See Event order for I/O operation records for the possible event orders.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_IoHandleRef	handle	An *active* I/O handle. References a *IoHandle* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_IO_HANDLE* is available.
OTF2_IoOperationMode	mode	Mode of an I/O handle operation (e.g., read or write).
OTF2_IoOperationFlag	operationFlags	Special semantic of this operation.
uint64_t	bytesRequest	Requested bytes to write/read.
uint64_t	matchingId	Identifier used to correlate associated event records of an I/O operation. This identifier is unique for the referenced *IoHandle*.

See also

OTF2_EvtWriter_IoOperationBegin()
OTF2_GlobalEvtReaderCallbacks_SetIoOperationBeginCallback()
OTF2_EvtReaderCallbacks_SetIoOperationBeginCallback()

Since

Version 2.1

.118 IoOperationTest

An *IoOperationTest* record marks an unsuccessful test whether an I/O operation has already finished.

See Event order for I/O operation records for the possible event orders.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_IoHandleRef	handle	An *active* I/O handle. References a *IoHandle* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_IO_HANDLE* is available.
uint64_t	matchingId	Identifier used to correlate associated event records of an I/O operation. This identifier is unique for the referenced *IoHandle*.

See also

OTF2_EvtWriter_IoOperationTest()
OTF2_GlobalEvtReaderCallbacks_SetIoOperationTestCallback()
OTF2_EvtReaderCallbacks_SetIoOperationTestCallback()

Since

Version 2.1

.119 IoOperationIssued

An *IoOperationIssued* record marks the successful initiation of a non-blocking operation (read, write, etc.) on an *active* I/O handle.

See Event order for I/O operation records for the possible event orders.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
OTF2_Io↩ HandleRef	handle	An *active* I/O handle. References a *IoHandle* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_IO_HANDLE* is available.
uint64_t	matchingId	Identifier used to correlate associated event records of an I/O operation. This identifier is unique for the referenced *IoHandle*.

See also

OTF2_EvtWriter_IoOperationIssued()
OTF2_GlobalEvtReaderCallbacks_SetIoOperationIssuedCallback()
OTF2_EvtReaderCallbacks_SetIoOperationIssuedCallback()

Since

Version 2.1

.120 IoOperationComplete

An *IoOperationComplete* record marks the end of a file operation (read, write, etc.) on an *active* I/O handle.

See Event order for I/O operation records for the possible event orders.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_IoHandleRef	handle	An *active* I/O handle. References a *IoHandle* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_IO_HANDLE* is available.
uint64_t	bytesResult	Number of actual transferred bytes.
uint64_t	matchingId	Identifier used to correlate associated event records of an I/O operation. This identifier is unique for the referenced *IoHandle*.

See also

OTF2_EvtWriter_IoOperationComplete()
OTF2_GlobalEvtReaderCallbacks_SetIoOperationCompleteCallback()
OTF2_EvtReaderCallbacks_SetIoOperationCompleteCallback()

Since

Version 2.1

.121 IoOperationCancelled

An *IoOperationCancelled* record marks the successful cancellation of a non-blocking operation (read, write, etc.) on an *active* I/O handle.

See Event order for I/O operation records for the possible event orders.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_IoHandleRef	handle	An *active* I/O handle. References a *IoHandle* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_IO_HANDLE* is available.
uint64_t	matchingId	Identifier used to correlate associated event records of an I/O operation. This identifier is unique for the referenced *IoHandle*.

See also

OTF2_EvtWriter_IoOperationCancelled()
OTF2_GlobalEvtReaderCallbacks_SetIoOperationCancelledCallback()
OTF2_EvtReaderCallbacks_SetIoOperationCancelledCallback()

Since

Version 2.1

.122 IoAcquireLock

An *IoAcquireLock* record marks the acquisition of an I/O lock.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
OTF2_Io↩ HandleRef	handle	An *active* I/O handle. References a *IoHandle* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_IO_HANDLE* is available.
OTF2_↩ LockType	lockType	Type of the lock.

See also

OTF2_EvtWriter_IoAcquireLock()
OTF2_GlobalEvtReaderCallbacks_SetIoAcquireLockCallback()
OTF2_EvtReaderCallbacks_SetIoAcquireLockCallback()

Since

Version 2.1

.123 IoReleaseLock

An *IoReleaseLock* record marks the release of an I/O lock.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
OTF2_Io↩ HandleRef	handle	An *active* I/O handle. References a *IoHandle* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_IO_HANDLE* is available.
OTF2_↩ LockType	lockType	Type of the lock.

See also

OTF2_EvtWriter_IoReleaseLock()
OTF2_GlobalEvtReaderCallbacks_SetIoReleaseLockCallback()
OTF2_EvtReaderCallbacks_SetIoReleaseLockCallback()

Since

Version 2.1

.124 IoTryLock

An *IoTryLock* record marks when an I/O lock was requested but not granted.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_IoHandleRef	handle	An *active* I/O handle. References a *IoHandle* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_IO_HANDLE* is available.
OTF2_LockType	lockType	Type of the lock.

See also

OTF2_EvtWriter_IoTryLock()
OTF2_GlobalEvtReaderCallbacks_SetIoTryLockCallback()
OTF2_EvtReaderCallbacks_SetIoTryLockCallback()

Since

Version 2.1

.125 ProgramBegin

The *ProgramBegin* record marks the begin of the program.

This event is restricted to happen at most once on any *Location* in a *LocationGroup* that is of type *OTF2_LOCATION_GROUP_TYPE_PROCESS*.

If there is a ProgramBegin record, a corresponding *ProgramEnd* record on any *Location* in the same *LocationGroup* is mandatory and vice versa.

None of the timestamps recorded within the same *LocationGroup* must be smaller than ProgramBegin's timestamp.

Attributes

OTF2_Location_Ref	location	The location where this event happened.
OTF2_Time_Stamp	timestamp	The time when this event happened.
OTF2_StringRef	programName	The name of the program. References a *String* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_STRING* is available.

uint32_t	numberOf↩ Arguments	Number of additional arguments to the program.
OTF2_↩ StringRef	program↩ Arguments [numberOf↩ Arguments]	List of additional arguments to the program.

See also

OTF2_EvtWriter_ProgramBegin()
OTF2_GlobalEvtReaderCallbacks_SetProgramBeginCallback()
OTF2_EvtReaderCallbacks_SetProgramBeginCallback()

Since

Version 2.1

.126 ProgramEnd

The *ProgramEnd* record marks the end of the program.

This event is restricted to happen at most once on any *Location* in a *LocationGroup* that is of type *OTF2_LOCAT↩ ION_GROUP_TYPE_PROCESS*.

If there is a ProgramEnd record, a corresponding *ProgramBegin* record on any *Location* in the same *LocationGroup* is mandatory, and vice versa.

None of the timestamps recorded within the same *LocationGroup* must be larger than ProgramEnd's timestamp.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
int64_t	exitStatus	The exit status of the program. Note, that on some systems only the least significant 8 bits may be visible to other processes. Use *OTF2_UNDEFINE↩ D_INT64*, if the exit status was not available.

See also

OTF2_EvtWriter_ProgramEnd()
OTF2_GlobalEvtReaderCallbacks_SetProgramEndCallback()
OTF2_EvtReaderCallbacks_SetProgramEndCallback()

Since

Version 2.1

.127 NonBlockingCollectiveRequest

A *NonBlockingCollectiveRequest* record indicates that a non-blocking collective operation was initiated.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
uint64_t	requestID	ID of the requested operation.

See also

OTF2_EvtWriter_NonBlockingCollectiveRequest()
OTF2_GlobalEvtReaderCallbacks_SetNonBlockingCollectiveRequestCallback()
OTF2_EvtReaderCallbacks_SetNonBlockingCollectiveRequestCallback()

Since

Version 3.0

.128 NonBlockingCollectiveComplete

A *NonBlockingCollectiveComplete* record indicates that a non- blocking collective operation completed.

Attributes

OTF2_↩ Location↩ Ref	location	The location where this event happened.
OTF2_↩ Time↩ Stamp	timestamp	The time when this event happened.
OTF2_↩ Collective↩ Op	collectiveOp	Determines which collective operation it is.
OTF2_↩ CommRef	communicator	Communicator ID. References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING↩ _COMM* is available.
uint32_t	root	Rank of root in `communicator` or any predefined constant of *OTF2_↩ CollectiveRoot*.
uint64_t	sizeSent	Size of the sent data.
uint64_t	sizeReceived	Size of the received data.
uint64_t	requestID	ID of the requested operation.

See also

OTF2_EvtWriter_NonBlockingCollectiveComplete()
OTF2_GlobalEvtReaderCallbacks_SetNonBlockingCollectiveCompleteCallback()
OTF2_EvtReaderCallbacks_SetNonBlockingCollectiveCompleteCallback()

Since

Version 3.0

.129 CommCreate

A *CommCreate* record denotes the creation of a communicator. Only valid if the *Comm* definition was flagged with *OTF2_COMM_FLAG_CREATE_DESTROY_EVENTS*. This event must be enclosed by an *MpiCollectiveBegin* and *MpiCollectiveEnd* or *NonBlockingCollectiveRequest* and *NonBlockingCollectiveComplete* event pair with *OTF2_COLLECTIVE_OP_CREATE_HANDLE* as the operation type.

Attributes

OTF2_LocationRef	location	The location where this event happened.
OTF2_TimeStamp	timestamp	The time when this event happened.
OTF2_CommRef	communicator	Communicator ID. References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_COMM* is available.

See also

OTF2_EvtWriter_CommCreate()
OTF2_GlobalEvtReaderCallbacks_SetCommCreateCallback()
OTF2_EvtReaderCallbacks_SetCommCreateCallback()

Since

Version 3.0

.130 CommDestroy

A *CommDestroy* record marks the communicator for destruction at the end of the enclosing *MpiCollectiveBegin* and *MpiCollectiveEnd* event pair. Only valid if the *Comm* definition was flagged with *OTF2_COMM_FLAG_CREATE_DESTROY_EVENTS*. This event must be enclosed by an *MpiCollectiveBegin* and *MpiCollectiveEnd* event pair with *OTF2_COLLECTIVE_OP_DESTROY_HANDLE* as the operation type.

Attributes

OTF2_LocationRef	location	The location where this event happened.
OTF2_TimeStamp	timestamp	The time when this event happened.
OTF2_CommRef	communicator	Communicator ID. References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_COMM* is available.

See also

OTF2_EvtWriter_CommDestroy()
OTF2_GlobalEvtReaderCallbacks_SetCommDestroyCallback()
OTF2_EvtReaderCallbacks_SetCommDestroyCallback()

Since

Version 3.0

.131 List of all marker records

.132 *OTF2_MarkerRef* DefMarker

Group markers by name and severity.

Attributes

const char*	markerGroup	Group name, e.g., "MUST", ...
const char*	markerCategory	Marker category, e.g., "Argument type error", ...
OTF2_Marker_Severity	severity	The severity for these markers.

See also

OTF2_MarkerWriter_WriteDefMarker()
OTF2_MarkerReaderCallbacks_SetDefMarkerCallback()

Since

Version 1.2

.133 Marker

A user marker instance, with implied time stamp.

Attributes

OTF2_Time_Stamp	timestamp	The time when this marker happened.
OTF2_Time_Stamp	duration	A possible duration of this marker. May be 0.
OTF2_MarkerRef	marker	Groups this marker by name and severity. References a *DefMarker* definition.
OTF2_Marker_Scope	scope	The type of scope of this marker instance.
uint64_t	scopeRef	The scope instance of this marker. Depends on `scope`.
const char*	text	A textual description for this marker.

See also

OTF2_MarkerWriter_WriteMarker()
OTF2_MarkerReaderCallbacks_SetMarkerCallback()

Since

Version 1.2

.134 List of all snapshot records

.135 SnapshotStart

This record marks the start of a snapshot.

A snapshot consists of a timestamp and a set of snapshot records. All these snapshot records have the same snapshot time. A snapshot starts with one *SnapshotStart* record and closes with one *SnapshotEnd* record. All snapshot records inbetween are ordered by the `origEventTime`, which are also less than the snapshot timestamp. Ie. The timestamp of the next event read from the event stream is greater or equal to the snapshot time.

Attributes

OTF2_Location Ref	location	The location of the snapshot.
OTF2_Time Stamp	timestamp	The snapshot time of this record.
uint64_t	numberOfRecords	Number of snapshot event records in this snapshot. Excluding the *SnapshotEnd* record.

See also

OTF2_SnapWriter_SnapshotStart()
OTF2_GlobalSnapReaderCallbacks_SetSnapshotStartCallback()
OTF2_SnapReaderCallbacks_SetSnapshotStartCallback()

Since

Version 1.2

.136 SnapshotEnd

This record marks the end of a snapshot. It contains the position to continue reading in the event trace for this location. Use *OTF2_EvtReader_Seek* with `contReadPos` as the position.

Attributes

OTF2_Location Ref	location	The location of the snapshot.
OTF2_Time Stamp	timestamp	The snapshot time of this record.

uint64_t	contReadPos	Position to continue reading in the event trace.

See also

OTF2_SnapWriter_SnapshotEnd()
OTF2_GlobalSnapReaderCallbacks_SetSnapshotEndCallback()
OTF2_SnapReaderCallbacks_SetSnapshotEndCallback()

Since

Version 1.2

.137 MeasurementOnOffSnap

The last occurrence of a *MeasurementOnOff* event of this location, if any.

Attributes

OTF2_Location Ref	location	The location of the snapshot.
OTF2_Time Stamp	timestamp	The snapshot time of this record.
OTF2_Time Stamp	origEventTime	The original time this event happened.
OTF2_Measurement Mode	measurementMode	Is the measurement turned on (*OTF2_MEASUREMENT_ON*) or off (*OTF2_MEASUREMENT_OFF*)?

See also

MeasurementOnOff event
OTF2_SnapWriter_MeasurementOnOff()
OTF2_GlobalSnapReaderCallbacks_SetMeasurementOnOffCallback()
OTF2_SnapReaderCallbacks_SetMeasurementOnOffCallback()

Since

Version 1.2

.138 EnterSnap

This record exists for each *Enter* event where the corresponding *Leave* event did not occur before the snapshot.

Attributes

OTF2_Location_Ref	location	The location of the snapshot.
OTF2_Time_Stamp	timestamp	The snapshot time of this record.
OTF2_Time_Stamp	origEventTime	The original time this event happened.
OTF2_RegionRef	region	Needs to be defined in a definition record References a *Region* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_REGION* is available.

See also

Enter event
OTF2_SnapWriter_Enter()
OTF2_GlobalSnapReaderCallbacks_SetEnterCallback()
OTF2_SnapReaderCallbacks_SetEnterCallback()

Since

Version 1.2

.139 MpiSendSnap

This record exists for each *MpiSend* event where the matching receive message event did not occur on the remote location before the snapshot. This could either be a *MpiRecv* or a *MpiIrecv* event. Note that it may so, that a previous *MpiIsend* with the same envelope than this one is neither completed not canceled yet, thus the matching receive may already occurred, but the matching couldn't be done yet.

Attributes

OTF2_Location_Ref	location	The location of the snapshot.
OTF2_Time_Stamp	timestamp	The snapshot time of this record.
OTF2_Time_Stamp	origEventTime	The original time this event happened.
uint32_t	receiver	MPI rank of receiver in `communicator`.
OTF2_CommRef	communicator	Communicator ID. References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_COMM* is available.

uint32_t	msgTag	Message tag
uint64_t	msgLength	Message length

See also

MpiSend event
OTF2_SnapWriter_MpiSend()
OTF2_GlobalSnapReaderCallbacks_SetMpiSendCallback()
OTF2_SnapReaderCallbacks_SetMpiSendCallback()

Since

Version 1.2

.140 MpiIsendSnap

This record exists for each *MpiIsend* event where a corresponding *MpiIsendComplete* or *MpiRequestCancelled* event did not occur on this location before the snapshot. Or the corresponding *MpiIsendComplete* did occurred (the *MpiIsendCompleteSnap* record exists in the snapshot) but the matching receive message event did not occur on the remote location before the snapshot. (This could either be an*MpiRecv* or a *MpiIrecv* event.)

Attributes

OTF2_↩ Location↩ Ref	location	The location of the snapshot.
OTF2_↩ Time↩ Stamp	timestamp	The snapshot time of this record.
OTF2_↩ Time↩ Stamp	origEventTime	The original time this event happened.
uint32_t	receiver	MPI rank of receiver in `communicator`.
OTF2_↩ CommRef	communicator	Communicator ID. References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING↩ _COMM* is available.
uint32_t	msgTag	Message tag
uint64_t	msgLength	Message length
uint64_t	requestID	ID of the related request

See also

MpiIsend event
OTF2_SnapWriter_MpiIsend()
OTF2_GlobalSnapReaderCallbacks_SetMpiIsendCallback()
OTF2_SnapReaderCallbacks_SetMpiIsendCallback()

Since

Version 1.2

.141 MpiIsendCompleteSnap

This record exists for each *MpiIsend* event where the corresponding *MpiIsendComplete* event occurred, but where the matching receive message event did not occur on the remote location before the snapshot. (This could either be a *MpiRecv* or a *MpiIrecv* event.) .

Attributes

OTF2_Location_Ref	location	The location of the snapshot.
OTF2_Time_Stamp	timestamp	The snapshot time of this record.
OTF2_Time_Stamp	origEventTime	The original time this event happened.
uint64_t	requestID	ID of the related request

See also

MpiIsendComplete event
OTF2_SnapWriter_MpiIsendComplete()
OTF2_GlobalSnapReaderCallbacks_SetMpiIsendCompleteCallback()
OTF2_SnapReaderCallbacks_SetMpiIsendCompleteCallback()

Since

Version 1.2

.142 MpiRecvSnap

This record exists for each *MpiRecv* event where the matching send message event did not occur on the remote location before the snapshot. This could either be a *MpiSend* or a *MpiIsendComplete* event. Or a *MpiIrecvRequest* occurred before this event but the corresponding *MpiIrecv* event did not occurred before this snapshot. In this case the message matching couldn't performed yet, because the envelope of the ongoing *MpiIrecvRequest* is not yet known.

Attributes

OTF2_Location_Ref	location	The location of the snapshot.
OTF2_Time_Stamp	timestamp	The snapshot time of this record.
OTF2_Time_Stamp	origEventTime	The original time this event happened.
uint32_t	sender	MPI rank of sender in `communicator`.
OTF2_CommRef	communicator	Communicator ID. References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_COMM* is available.

uint32_t	msgTag	Message tag
uint64_t	msgLength	Message length

See also

MpiRecv event
OTF2_SnapWriter_MpiRecv()
OTF2_GlobalSnapReaderCallbacks_SetMpiRecvCallback()
OTF2_SnapReaderCallbacks_SetMpiRecvCallback()

Since

Version 1.2

.143 MpiIrecvRequestSnap

This record exists for each *MpiIrecvRequest* event where an corresponding *MpiIrecv* or *MpiRequestCancelled* event did not occur on this location before the snapshot. Or the corresponding *MpiIrecv* did occurred (the *MpiIrecvSnap* record exists in the snapshot) but the matching receive message event did not occur on the remote location before the snapshot. This could either be an *MpiRecv* or a *MpiIrecv* event.

Attributes

OTF2_↩ Location↩ Ref	location	The location of the snapshot.
OTF2_↩ Time↩ Stamp	timestamp	The snapshot time of this record.
OTF2_↩ Time↩ Stamp	origEventTime	The original time this event happened.
uint64_t	requestID	ID of the requested receive

See also

MpiIrecvRequest event
OTF2_SnapWriter_MpiIrecvRequest()
OTF2_GlobalSnapReaderCallbacks_SetMpiIrecvRequestCallback()
OTF2_SnapReaderCallbacks_SetMpiIrecvRequestCallback()

Since

Version 1.2

.144 MpiIrecvSnap

This record exists for each *MpiIrecv* event where the matching send message event did not occur on the remote location before the snapshot. This could either be a *MpiSend* or a *MpiIsendComplete* event. Or a *MpiIrecvRequest* occurred before this event but the corresponding *MpiIrecv* event did not occurred before this snapshot. In this case the message matching couldn't performed yet, because the envelope of the ongoing *MpiIrecvRequest* is not yet known.

Attributes

OTF2_Location_Ref	location	The location of the snapshot.
OTF2_Time_Stamp	timestamp	The snapshot time of this record.
OTF2_Time_Stamp	origEventTime	The original time this event happened.
uint32_t	sender	MPI rank of sender in `communicator`.
OTF2_CommRef	communicator	Communicator ID. References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_COMM* is available.
uint32_t	msgTag	Message tag
uint64_t	msgLength	Message length
uint64_t	requestID	ID of the related request

See also

MpiIrecv event
OTF2_SnapWriter_MpiIrecv()
OTF2_GlobalSnapReaderCallbacks_SetMpiIrecvCallback()
OTF2_SnapReaderCallbacks_SetMpiIrecvCallback()

Since

Version 1.2

.145 MpiCollectiveBeginSnap

Indicates that this location started a collective operation but not all of the participating locations completed the operation yet, including this location.

Attributes

OTF2_Location_Ref	location	The location of the snapshot.
OTF2_Time_Stamp	timestamp	The snapshot time of this record.
OTF2_Time_Stamp	origEventTime	The original time this event happened.

See also

MpiCollectiveBegin event
OTF2_SnapWriter_MpiCollectiveBegin()
OTF2_GlobalSnapReaderCallbacks_SetMpiCollectiveBeginCallback()
OTF2_SnapReaderCallbacks_SetMpiCollectiveBeginCallback()

Since

Version 1.2

.146 MpiCollectiveEndSnap

Indicates that this location completed a collective operation locally but not all of the participating locations completed the operation yet. The corresponding *MpiCollectiveBeginSnap* record is still in the snapshot though.

Attributes

OTF2_Location_Ref	location	The location of the snapshot.
OTF2_Time_Stamp	timestamp	The snapshot time of this record.
OTF2_Time_Stamp	origEventTime	The original time this event happened.
OTF2_CollectiveOp	collectiveOp	Determines which collective operation it is.
OTF2_CommRef	communicator	Communicator References a *Comm*, or a *InterComm* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_COMM* is available.
uint32_t	root	Rank of root in `communicator` or any predefined constant of *OTF2_CollectiveRoot*.
uint64_t	sizeSent	Size of the sent message.
uint64_t	sizeReceived	Size of the received message.

See also

MpiCollectiveEnd event
OTF2_SnapWriter_MpiCollectiveEnd()
OTF2_GlobalSnapReaderCallbacks_SetMpiCollectiveEndCallback()
OTF2_SnapReaderCallbacks_SetMpiCollectiveEndCallback()

Since

Version 1.2

.147 OmpForkSnap

This record exists for each *OmpFork* event where the corresponding *OmpJoin* did not occurred before this snapshot.

Attributes

OTF2_Location_Ref	location	The location of the snapshot.
OTF2_Time_Stamp	timestamp	The snapshot time of this record.
OTF2_Time_Stamp	origEventTime	The original time this event happened.

uint32_t	numberOf↩ Requested↩ Threads	Requested size of the team.

See also

OmpFork event
OTF2_SnapWriter_OmpFork()
OTF2_GlobalSnapReaderCallbacks_SetOmpForkCallback()
OTF2_SnapReaderCallbacks_SetOmpForkCallback()

Since

Version 1.2

.148 OmpAcquireLockSnap

This record exists for each *OmpAcquireLock* event where the corresponding *OmpReleaseLock* did not occurred before this snapshot yet.

Attributes

OTF2_↩ Location↩ Ref	location	The location of the snapshot.
OTF2_↩ Time↩ Stamp	timestamp	The snapshot time of this record.
OTF2_↩ Time↩ Stamp	origEventTime	The original time this event happened.
uint32_t	lockID	ID of the lock.
uint32_t	acquisitionOrder	A monotonically increasing number to determine the order of lock acquisitions (with unsynchronized clocks this is otherwise not possible). Corresponding acquire-release events have same number.

See also

OmpAcquireLock event
OTF2_SnapWriter_OmpAcquireLock()
OTF2_GlobalSnapReaderCallbacks_SetOmpAcquireLockCallback()
OTF2_SnapReaderCallbacks_SetOmpAcquireLockCallback()

Since

Version 1.2

.149 OmpTaskCreateSnap

This record exists for each *OmpTaskCreate* event where the corresponding *OmpTaskComplete* event did not occurred before this snapshot. Neither on this location nor on any other location in the current thread team.

Attributes

OTF2_Location_Ref	location	The location of the snapshot.
OTF2_Time_Stamp	timestamp	The snapshot time of this record.
OTF2_Time_Stamp	origEventTime	The original time this event happened.
uint64_t	taskID	Identifier of the newly created task instance.

See also

OmpTaskCreate event
OTF2_SnapWriter_OmpTaskCreate()
OTF2_GlobalSnapReaderCallbacks_SetOmpTaskCreateCallback()
OTF2_SnapReaderCallbacks_SetOmpTaskCreateCallback()

Since

Version 1.2

.150 OmpTaskSwitchSnap

This record exists for each *OmpTaskSwitch* event where the corresponding *OmpTaskComplete* event did not occurred before this snapshot. Neither on this location nor on any other location in the current thread team.

Attributes

OTF2_Location_Ref	location	The location of the snapshot.
OTF2_Time_Stamp	timestamp	The snapshot time of this record.
OTF2_Time_Stamp	origEventTime	The original time this event happened.
uint64_t	taskID	Identifier of the now active task instance.

See also

OmpTaskSwitch event
OTF2_SnapWriter_OmpTaskSwitch()
OTF2_GlobalSnapReaderCallbacks_SetOmpTaskSwitchCallback()
OTF2_SnapReaderCallbacks_SetOmpTaskSwitchCallback()

Since

Version 1.2

.151 MetricSnap

This record exists for each referenced metric class or metric instance event this location recorded metrics before and provides the last known recorded metric values.

As an exception for metric classes where the metric mode denotes an *OTF2_METRIC_VALUE_RELATIVE* mode the value indicates the accumulation of all previous metric values recorded.

Attributes

OTF2_↩ Location↩ Ref	location	The location of the snapshot.
OTF2_↩ Time↩ Stamp	timestamp	The snapshot time of this record.
OTF2_↩ Time↩ Stamp	origEventTime	The original time this event happened.
OTF2_↩ MetricRef	metric	Could be a metric class or a metric instance. References a *MetricClass*, or a *MetricInstance* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_METRIC* is available.
uint8_t	numberOf↩ Metrics	Number of metrics with in the set.
OTF2_↩ Type	typeIDs [numberOf↩ Metrics]	List of metric types. These types must match that of the corresponding *MetricMember* definitions.
OTF2_↩ Metric↩ Value	metricValues [numberOf↩ Metrics]	List of metric values.

See also

Metric event
OTF2_SnapWriter_Metric()
OTF2_GlobalSnapReaderCallbacks_SetMetricCallback()
OTF2_SnapReaderCallbacks_SetMetricCallback()

Since

Version 1.2

.152 ParameterStringSnap

This record must be included in the snapshot until the leave event for the enter event occurs which has the greatest timestamp less or equal the timestamp of this record.

Attributes

OTF2_Location_Ref	location	The location of the snapshot.
OTF2_Time_Stamp	timestamp	The snapshot time of this record.
OTF2_Time_Stamp	origEventTime	The original time this event happened.
OTF2_Parameter_Ref	parameter	Parameter ID. References a *Parameter* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_PARAMETER* is available.
OTF2_StringRef	string	Value: Handle of a string definition References a *String* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_STRING* is available.

See also

ParameterString event
OTF2_SnapWriter_ParameterString()
OTF2_GlobalSnapReaderCallbacks_SetParameterStringCallback()
OTF2_SnapReaderCallbacks_SetParameterStringCallback()

Since

Version 1.2

.153 ParameterIntSnap

This record must be included in the snapshot until the leave event for the enter event occurs which has the greatest timestamp less or equal the timestamp of this record.

Attributes

OTF2_Location_Ref	location	The location of the snapshot.
OTF2_Time_Stamp	timestamp	The snapshot time of this record.
OTF2_Time_Stamp	origEventTime	The original time this event happened.
OTF2_Parameter_Ref	parameter	Parameter ID. References a *Parameter* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_PARAMETER* is available.

int64_t	value	Value of the recorded parameter.

See also

ParameterInt event
OTF2_SnapWriter_ParameterInt()
OTF2_GlobalSnapReaderCallbacks_SetParameterIntCallback()
OTF2_SnapReaderCallbacks_SetParameterIntCallback()

Since

Version 1.2

.154 ParameterUnsignedIntSnap

This record must be included in the snapshot until the leave event for the enter event occurs which has the greatest timestamp less or equal the timestamp of this record.

Attributes

OTF2_↩ Location↩ Ref	location	The location of the snapshot.
OTF2_↩ Time↩ Stamp	timestamp	The snapshot time of this record.
OTF2_↩ Time↩ Stamp	origEventTime	The original time this event happened.
OTF2_↩ Parameter↩ Ref	parameter	Parameter ID. References a *Parameter* definition and will be mapped to the global definition if a mapping table of type *OTF2_MAPPING_PARAMETER* is available.
uint64_t	value	Value of the recorded parameter.

See also

ParameterUnsignedInt event
OTF2_SnapWriter_ParameterUnsignedInt()
OTF2_GlobalSnapReaderCallbacks_SetParameterUnsignedIntCallback()
OTF2_SnapReaderCallbacks_SetParameterUnsignedIntCallback()

Since

Version 1.2

Appendix A

Example Documentation

A.1 otf2_high_level_reader_example.py

Python high-level reading example

```
1 #!/usr/bin/env python
2 #
3 # This file is part of the Score-P software (http://www.score-p.org)
4 #
5 # Copyright (c) 2015-2017,
6 # Technische Universitaet Dresden, Germany
7 #
8 # This software may be modified and distributed under the terms of
9 # a BSD-style license.  See the COPYING file in the package base
10 # directory for details.
11 #
12
13 import otf2
14
15
16 with otf2.reader.open('TestArchive/traces.otf2') as trace:
17     print("Read {} string definitions".format(len(trace.definitions.strings)))
18
19     for location, event in trace.events:
20         print("Encountered {} event {} on {}".format(type(event).__name__,
21                                                      event, location))
```

A.2 otf2_high_level_writer_example.py

Python high-level writing example

```
1 #!/usr/bin/env python
2 #
3 # This file is part of the Score-P software (http://www.score-p.org)
4 #
5 # Copyright (c) 2015-2016, 2021,
6 # Technische Universitaet Dresden, Germany
7 #
8 # This software may be modified and distributed under the terms of
9 # a BSD-style license.  See the COPYING file in the package base
10 # directory for details.
11 #
12
13 import otf2
14 from otf2.enums import Type
15 import time
16
17
18 TIMER_GRANULARITY = 1000000
19
```

```
20
21 def t():
22     return int(round(time.time() * TIMER_GRANULARITY))
23
24
25 with otf2.writer.open("TestArchive", timer_resolution=TIMER_GRANULARITY) as trace:
26
27     function = trace.definitions.region("My Function")
28
29     parent_node = trace.definitions.system_tree_node("node")
30     system_tree_node = trace.definitions.system_tree_node("myHost", parent=parent_node)
31
32     trace.definitions.system_tree_node_property(system_tree_node, "color", value="black")
33     trace.definitions.system_tree_node_property(system_tree_node, "rack #", value=42)
34
35     location_group = trace.definitions.location_group("Initial Process",
36                                                        system_tree_parent=system_tree_node)
37
38     attr = trace.definitions.attribute("StringTest", "A test attribute", Type.STRING)
39     float_attr = trace.definitions.attribute("FloatTest", "Another test attribute",
40                                              Type.DOUBLE)
41
42     writer = trace.event_writer("Main Thread", group=location_group)
43
44     # Write enter and leave event
45     writer.enter(t(), function, {attr: "Hello World"})
46     writer.leave(t(), function, attributes={float_attr: 42.0, attr: "Wurst?"})
47
48     # Get convenience metric object and write one metric event
49     temperature = trace.definitions.metric("Time since last coffee", unit="min")
50     writer.metric(t(), temperature, 72.0)
51
52     # Get metric members
53     temp_member = trace.definitions.metric_member("Temperature", "C", otf2.MetricType.OTHER,
54                                                   otf2.MetricMode.ABSOLUTE_POINT)
55     power_member = trace.definitions.metric_member("Power", "W")
56     # Add metric members to the metric class object
57     mclass = trace.definitions.metric_class([temp_member, power_member])
58     # Add metric object to the location object
59     writer.metric(t(), mclass, [42.0, 12345.6])
```

A.3 otf2_mpi_reader_example.c

MPI reading example

```
/*
 * This file is part of the Score-P software (http://www.score-p.org)
 *
 * Copyright (c) 2009-2013,
 * RWTH Aachen University, Germany
 *
 * Copyright (c) 2009-2013,
 * Gesellschaft fuer numerische Simulation mbH Braunschweig, Germany
 *
 * Copyright (c) 2009-2014,
 * Technische Universitaet Dresden, Germany
 *
 * Copyright (c) 2009-2013,
 * University of Oregon, Eugene, USA
 *
 * Copyright (c) 2009-2014,
 * Forschungszentrum Juelich GmbH, Germany
 *
 * Copyright (c) 2009-2013,
 * German Research School for Simulation Sciences GmbH, Juelich/Aachen, Germany
 *
 * Copyright (c) 2009-2013,
 * Technische Universitaet Muenchen, Germany
 *
 * This software may be modified and distributed under the terms of
 * a BSD-style license.  See the COPYING file in the package base
 * directory for details.
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <inttypes.h>

#include <mpi.h>
```

```
#include <otf2/otf2.h>

#if MPI_VERSION < 3
#define OTF2_MPI_UINT64_T MPI_UNSIGNED_LONG
#define OTF2_MPI_INT64_T  MPI_LONG
#endif
#include <otf2/OTF2_MPI_Collectives.h>

static OTF2_CallbackCode
Enter_print( OTF2_LocationRef    location,
             OTF2_TimeStamp      time,
             void*               userData,
             OTF2_AttributeList* attributes,
             OTF2_RegionRef      region )
{
    printf( "Entering region %u at location %" PRIu64 " at time %" PRIu64 ".\n",
            region, location, time );

    return OTF2_CALLBACK_SUCCESS;
}

static OTF2_CallbackCode
Leave_print( OTF2_LocationRef    location,
             OTF2_TimeStamp      time,
             void*               userData,
             OTF2_AttributeList* attributes,
             OTF2_RegionRef      region )
{
    printf( "Leaving region %u at location %" PRIu64 " at time %" PRIu64 ".\n",
            region, location, time );

    return OTF2_CALLBACK_SUCCESS;
}

struct vector
{
    size_t   capacity;
    size_t   size;
    uint64_t members[];
};

static OTF2_CallbackCode
GlobDefLocation_Register( void*                 userData,
                          OTF2_LocationRef      location,
                          OTF2_StringRef        name,
                          OTF2_LocationType     locationType,
                          uint64_t              numberOfEvents,
                          OTF2_LocationGroupRef locationGroup )
{
    struct vector* locations = userData;

    if ( locations->size == locations->capacity )
    {
        return OTF2_CALLBACK_INTERRUPT;
    }

    locations->members[ locations->size++ ] = location;

    return OTF2_CALLBACK_SUCCESS;
}

int
main( int    argc,
      char** argv )
{
    MPI_Init( &argc, &argv );
    int size;
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    int rank;
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );

    OTF2_Reader* reader = OTF2_Reader_Open( "ArchivePath/ArchiveName.otf2" );

    OTF2_MPI_Reader_SetCollectiveCallbacks( reader, MPI_COMM_WORLD );

    uint64_t number_of_locations;
    OTF2_Reader_GetNumberOfLocations( reader,
                                      &number_of_locations );
    struct vector* locations = malloc( sizeof( *locations )
                                       + number_of_locations
                                       * sizeof( *locations->members ) );
    locations->capacity = number_of_locations;
    locations->size     = 0;

    OTF2_GlobalDefReader* global_def_reader = OTF2_Reader_GetGlobalDefReader( reader );
```

```
OTF2_GlobalDefReaderCallbacks* global_def_callbacks = OTF2_GlobalDefReaderCallbacks_New();
OTF2_GlobalDefReaderCallbacks_SetLocationCallback( global_def_callbacks,
                                                   &GlobDefLocation_Register );
OTF2_Reader_RegisterGlobalDefCallbacks( reader,
                                        global_def_reader,
                                        global_def_callbacks,
                                        locations );
OTF2_GlobalDefReaderCallbacks_Delete( global_def_callbacks );

uint64_t definitions_read = 0;
OTF2_Reader_ReadAllGlobalDefinitions( reader,
                                      global_def_reader,
                                      &definitions_read );

uint64_t number_of_locations_to_read = 0;
for ( size_t i = 0; i < locations->size; i++ )
{
    if ( locations->members[ i ] % size != rank )
    {
        continue;
    }
    number_of_locations_to_read++;
    OTF2_Reader_SelectLocation( reader, locations->members[ i ] );
}

bool successful_open_def_files =
    OTF2_Reader_OpenDefFiles( reader ) == OTF2_SUCCESS;
OTF2_Reader_OpenEvtFiles( reader );

for ( size_t i = 0; i < locations->size; i++ )
{
    if ( locations->members[ i ] % size != rank )
    {
        continue;
    }

    if ( successful_open_def_files )
    {
        OTF2_DefReader* def_reader =
            OTF2_Reader_GetDefReader( reader, locations->members[ i ] );
        if ( def_reader )
        {
            uint64_t def_reads = 0;
            OTF2_Reader_ReadAllLocalDefinitions( reader,
                                                 def_reader,
                                                 &def_reads );
            OTF2_Reader_CloseDefReader( reader, def_reader );
        }
    }
    OTF2_EvtReader* evt_reader =
        OTF2_Reader_GetEvtReader( reader, locations->members[ i ] );
}

if ( successful_open_def_files )
{
    OTF2_Reader_CloseDefFiles( reader );
}

if ( number_of_locations_to_read > 0 )
{
    OTF2_GlobalEvtReader* global_evt_reader = OTF2_Reader_GetGlobalEvtReader( reader );

    OTF2_GlobalEvtReaderCallbacks* event_callbacks = OTF2_GlobalEvtReaderCallbacks_New();
    OTF2_GlobalEvtReaderCallbacks_SetEnterCallback( event_callbacks,
                                                    &Enter_print );
    OTF2_GlobalEvtReaderCallbacks_SetLeaveCallback( event_callbacks,
                                                    &Leave_print );
    OTF2_Reader_RegisterGlobalEvtCallbacks( reader,
                                            global_evt_reader,
                                            event_callbacks,
                                            NULL );
    OTF2_GlobalEvtReaderCallbacks_Delete( event_callbacks );

    uint64_t events_read = 0;
    OTF2_Reader_ReadAllGlobalEvents( reader,
                                     global_evt_reader,
                                     &events_read );

    OTF2_Reader_CloseGlobalEvtReader( reader, global_evt_reader );
}
OTF2_Reader_CloseEvtFiles( reader );

OTF2_Reader_Close( reader );

free( locations );

MPI_Finalize();
```

```
    return EXIT_SUCCESS;
}
```

A.4 otf2_mpi_reader_example.cc

MPI reading example in C++

```
/*
 * This file is part of the Score-P software (http://www.score-p.org)
 *
 * Copyright (c) 2009-2013,
 * RWTH Aachen University, Germany
 *
 * Copyright (c) 2009-2013,
 * Gesellschaft fuer numerische Simulation mbH Braunschweig, Germany
 *
 * Copyright (c) 2009-2014,
 * Technische Universitaet Dresden, Germany
 *
 * Copyright (c) 2009-2013,
 * University of Oregon, Eugene, USA
 *
 * Copyright (c) 2009-2014,
 * Forschungszentrum Juelich GmbH, Germany
 *
 * Copyright (c) 2009-2013,
 * German Research School for Simulation Sciences GmbH, Juelich/Aachen, Germany
 *
 * Copyright (c) 2009-2013,
 * Technische Universitaet Muenchen, Germany
 *
 * This software may be modified and distributed under the terms of
 * a BSD-style license.  See the COPYING file in the package base
 * directory for details.
 *
 */

#include <stdlib.h>
#include <iostream>
#include <vector>

#include <mpi.h>

#include <otf2/otf2.h>

#if MPI_VERSION < 3
#define OTF2_MPI_UINT64_T MPI_UNSIGNED_LONG
#define OTF2_MPI_INT64_T  MPI_LONG
#endif
#include <otf2/OTF2_MPI_Collectives.h>

static OTF2_CallbackCode
Enter_print( OTF2_LocationRef    location,
             OTF2_TimeStamp      time,
             void*               userData,
             OTF2_AttributeList* attributes,
             OTF2_RegionRef      region )
{
    std::cout << "Entering region " << region << " at location " << location << " at time " << time <<
      std::endl;

    return OTF2_CALLBACK_SUCCESS;
}

static OTF2_CallbackCode
Leave_print( OTF2_LocationRef    location,
             OTF2_TimeStamp      time,
             void*               userData,
             OTF2_AttributeList* attributes,
             OTF2_RegionRef      region )
{
    std::cout << "Leaving region " << region << " at location " << location << " at time " << time <<
      std::endl;

    return OTF2_CALLBACK_SUCCESS;
}

static OTF2_CallbackCode
GlobDefLocation_Register( void*                 userData,
```

```
                          OTF2_LocationRef      location,
                          OTF2_StringRef        name,
                          OTF2_LocationType     locationType,
                          uint64_t              numberOfEvents,
                          OTF2_LocationGroupRef locationGroup )
{
    std::vector<OTF2_LocationRef>* locations =
        ( std::vector<OTF2_LocationRef>* )userData;

    locations->push_back( location );

    return OTF2_CALLBACK_SUCCESS;
}

int
main( int    argc,
      char** argv )
{
    MPI_Init( &argc, &argv );
    int size;
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    int rank;
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );

    OTF2_Reader* reader = OTF2_Reader_Open( "ArchivePath/ArchiveName.otf2" );

    OTF2_MPI_Reader_SetCollectiveCallbacks( reader, MPI_COMM_WORLD );

    uint64_t number_of_locations;
    OTF2_Reader_GetNumberOfLocations( reader,
                                      &number_of_locations );
    std::vector<OTF2_LocationRef> locations;
    locations.reserve( number_of_locations );

    OTF2_GlobalDefReader* global_def_reader = OTF2_Reader_GetGlobalDefReader( reader );

    OTF2_GlobalDefReaderCallbacks* global_def_callbacks = OTF2_GlobalDefReaderCallbacks_New();
    OTF2_GlobalDefReaderCallbacks_SetLocationCallback( global_def_callbacks,
                                                       &GlobDefLocation_Register );
    OTF2_Reader_RegisterGlobalDefCallbacks( reader,
                                            global_def_reader,
                                            global_def_callbacks,
                                            &locations );
    OTF2_GlobalDefReaderCallbacks_Delete( global_def_callbacks );

    uint64_t definitions_read = 0;
    OTF2_Reader_ReadAllGlobalDefinitions( reader,
                                          global_def_reader,
                                          &definitions_read );

    uint64_t number_of_locations_to_read = 0;
    for ( std::size_t i = 0; i < locations.size(); i++ )
    {
        if ( locations[ i ] % size != rank )
        {
            continue;
        }
        number_of_locations_to_read++;
        OTF2_Reader_SelectLocation( reader, locations[ i ] );
    }

    bool successful_open_def_files =
        OTF2_Reader_OpenDefFiles( reader ) == OTF2_SUCCESS;
    OTF2_Reader_OpenEvtFiles( reader );

    for ( std::size_t i = 0; i < locations.size(); i++ )
    {
        if ( locations[ i ] % size != rank )
        {
            continue;
        }

        if ( successful_open_def_files )
        {
            OTF2_DefReader* def_reader =
                OTF2_Reader_GetDefReader( reader, locations[ i ] );
            if ( def_reader )
            {
                uint64_t def_reads = 0;
                OTF2_Reader_ReadAllLocalDefinitions( reader,
                                                     def_reader,
                                                     &def_reads );
                OTF2_Reader_CloseDefReader( reader, def_reader );
            }
        }
        OTF2_EvtReader* evt_reader =
            OTF2_Reader_GetEvtReader( reader, locations[ i ] );
```

```
    }

    if ( successful_open_def_files )
    {
        OTF2_Reader_CloseDefFiles( reader );
    }

    if ( number_of_locations_to_read > 0 )
    {
        OTF2_GlobalEvtReader* global_evt_reader = OTF2_Reader_GetGlobalEvtReader( reader );

        OTF2_GlobalEvtReaderCallbacks* event_callbacks = OTF2_GlobalEvtReaderCallbacks_New();
        OTF2_GlobalEvtReaderCallbacks_SetEnterCallback( event_callbacks,
                                                        &Enter_print );
        OTF2_GlobalEvtReaderCallbacks_SetLeaveCallback( event_callbacks,
                                                        &Leave_print );
        OTF2_Reader_RegisterGlobalEvtCallbacks( reader,
                                                global_evt_reader,
                                                event_callbacks,
                                                NULL );
        OTF2_GlobalEvtReaderCallbacks_Delete( event_callbacks );

        uint64_t events_read = 0;
        OTF2_Reader_ReadAllGlobalEvents( reader,
                                         global_evt_reader,
                                         &events_read );

        OTF2_Reader_CloseGlobalEvtReader( reader, global_evt_reader );
    }
    OTF2_Reader_CloseEvtFiles( reader );

    OTF2_Reader_Close( reader );

    MPI_Finalize();

    return EXIT_SUCCESS;
}
```

A.5 otf2_mpi_writer_example.c

MPI writing example

```
/*
 * This file is part of the Score-P software (http://www.score-p.org)
 *
 * Copyright (c) 2009-2013,
 * RWTH Aachen University, Germany
 *
 * Copyright (c) 2009-2013,
 * Gesellschaft fuer numerische Simulation mbH Braunschweig, Germany
 *
 * Copyright (c) 2009-2014, 2021,
 * Technische Universitaet Dresden, Germany
 *
 * Copyright (c) 2009-2013,
 * University of Oregon, Eugene, USA
 *
 * Copyright (c) 2009-2013,
 * Forschungszentrum Juelich GmbH, Germany
 *
 * Copyright (c) 2009-2013,
 * German Research School for Simulation Sciences GmbH, Juelich/Aachen, Germany
 *
 * Copyright (c) 2009-2013,
 * Technische Universitaet Muenchen, Germany
 *
 * This software may be modified and distributed under the terms of
 * a BSD-style license.  See the COPYING file in the package base
 * directory for details.
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <inttypes.h>
#include <time.h>

#include <mpi.h>

#include <otf2/otf2.h>
```

```
#if MPI_VERSION < 3
#define OTF2_MPI_UINT64_T MPI_UNSIGNED_LONG
#define OTF2_MPI_INT64_T  MPI_LONG
#endif

#include <otf2/OTF2_MPI_Collectives.h>

static OTF2_TimeStamp
get_time( void )
{
    double t = MPI_Wtime() * 1e9;
    return ( uint64_t )t;
}

static OTF2_FlushType
pre_flush( void*            userData,
           OTF2_FileType    fileType,
           OTF2_LocationRef location,
           void*            callerData,
           bool             final )
{
    return OTF2_FLUSH;
}

static OTF2_TimeStamp
post_flush( void*            userData,
            OTF2_FileType    fileType,
            OTF2_LocationRef location )
{
    return get_time();
}

static OTF2_FlushCallbacks flush_callbacks =
{
    .otf2_pre_flush  = pre_flush,
    .otf2_post_flush = post_flush
};

enum
{
    REGION_MPI_INIT,
    REGION_MPI_FINALIZE,
    REGION_MPI_COMM_SPLIT,
    REGION_MPI_INTERCOMM_CREATE,
    REGION_MPI_COMM_FREE,
    REGION_MPI_BCAST,
    REGION_MPI_IBARRIER,
    REGION_MPI_TEST,
    REGION_MPI_WAIT
};

enum
{
    COMM_WORLD,
    COMM_SPLIT_0,
    COMM_SPLIT_1,
    COMM_INTERCOMM
};

int
main( int    argc,
      char** argv )
{
    MPI_Init( &argc, &argv );
    int size;
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    int rank;
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );

    OTF2_Archive* archive = OTF2_Archive_Open( "ArchivePath",
                                               "ArchiveName",
                                               OTF2_FILEMODE_WRITE,
                                               1024 * 1024 /* event chunk size */,
                                               4 * 1024 * 1024 /* def chunk size */,
                                               OTF2_SUBSTRATE_POSIX,
                                               OTF2_COMPRESSION_NONE );

    OTF2_Archive_SetFlushCallbacks( archive, &flush_callbacks, NULL );

    OTF2_MPI_Archive_SetCollectiveCallbacks( archive,
                                             MPI_COMM_WORLD,
                                             MPI_COMM_NULL );

    OTF2_Archive_OpenEvtFiles( archive );

    OTF2_EvtWriter* evt_writer = OTF2_Archive_GetEvtWriter( archive,
```

```
                                                        rank );

    struct timespec epoch_timestamp_spec;
    clock_gettime( CLOCK_REALTIME, &epoch_timestamp_spec );
    uint64_t epoch_start = get_time();

    OTF2_EvtWriter_Enter( evt_writer,
                          NULL,
                          get_time(),
                          REGION_MPI_INIT );

    OTF2_EvtWriter_MpiCollectiveBegin( evt_writer,
                                       NULL,
                                       get_time() );

    // fake MPI_Init
    MPI_Barrier( MPI_COMM_WORLD );

    OTF2_EvtWriter_CommCreate( evt_writer,
                               NULL,
                               get_time(),
                               COMM_WORLD );

    OTF2_EvtWriter_MpiCollectiveEnd( evt_writer,
                                     NULL,
                                     get_time(),
                                     OTF2_COLLECTIVE_OP_CREATE_HANDLE,
                                     COMM_WORLD,
                                     OTF2_COLLECTIVE_ROOT_NONE,
                                     0 /* bytes provided */,
                                     0 /* bytes obtained */ );

    OTF2_EvtWriter_Leave( evt_writer,
                          NULL,
                          get_time(),
                          REGION_MPI_INIT );

    OTF2_EvtWriter_Enter( evt_writer,
                          NULL,
                          get_time(),
                          REGION_MPI_IBARRIER );

    uint64_t barrier_request_id = 1; /* freely chosen request ID */
    OTF2_EvtWriter_NonBlockingCollectiveRequest( evt_writer,
                                                 NULL,
                                                 get_time(),
                                                 barrier_request_id );

    MPI_Request barrier_request;
    MPI_Ibarrier( MPI_COMM_WORLD, &barrier_request );

    OTF2_EvtWriter_Leave( evt_writer,
                          NULL,
                          get_time(),
                          REGION_MPI_IBARRIER );

    MPI_Comm split_comm;
    OTF2_EvtWriter_Enter( evt_writer,
                          NULL,
                          get_time(),
                          REGION_MPI_COMM_SPLIT );

    OTF2_EvtWriter_MpiCollectiveBegin( evt_writer,
                                       NULL,
                                       get_time() );

    MPI_Comm_split( MPI_COMM_WORLD, rank % 2, rank, &split_comm );

    if ( 0 == rank % 2 )
    {
        OTF2_EvtWriter_CommCreate( evt_writer,
                                   NULL,
                                   get_time(),
                                   COMM_SPLIT_0 );
    }
    else
    {
        OTF2_EvtWriter_CommCreate( evt_writer,
                                   NULL,
                                   get_time(),
                                   COMM_SPLIT_1 );
    }

    OTF2_EvtWriter_MpiCollectiveEnd( evt_writer,
                                     NULL,
                                     get_time(),
                                     OTF2_COLLECTIVE_OP_CREATE_HANDLE,
```

```
                                COMM_WORLD,
                                OTF2_COLLECTIVE_ROOT_NONE,
                                0 /* bytes provided */,
                                0 /* bytes obtained */ );

OTF2_EvtWriter_Leave( evt_writer,
                      NULL,
                      get_time(),
                      REGION_MPI_COMM_SPLIT );

MPI_Comm inter_comm;
OTF2_EvtWriter_Enter( evt_writer,
                      NULL,
                      get_time(),
                      REGION_MPI_INTERCOMM_CREATE );

OTF2_EvtWriter_MpiCollectiveBegin( evt_writer,
                                   NULL,
                                   get_time() );

if ( 0 == rank % 2 )
{
    MPI_Intercomm_create( split_comm, 0, MPI_COMM_WORLD, 1, 1, &inter_comm );
}
else
{
    MPI_Intercomm_create( split_comm, 0, MPI_COMM_WORLD, 0, 1, &inter_comm );
}

OTF2_EvtWriter_CommCreate( evt_writer,
                           NULL,
                           get_time(),
                           COMM_INTERCOMM );

OTF2_EvtWriter_MpiCollectiveEnd( evt_writer,
                                 NULL,
                                 get_time(),
                                 OTF2_COLLECTIVE_OP_CREATE_HANDLE,
                                 COMM_WORLD,
                                 OTF2_COLLECTIVE_ROOT_NONE,
                                 0 /* bytes provided */,
                                 0 /* bytes obtained */ );

OTF2_EvtWriter_Leave( evt_writer,
                      NULL,
                      get_time(),
                      REGION_MPI_INTERCOMM_CREATE );

OTF2_EvtWriter_Enter( evt_writer,
                      NULL,
                      get_time(),
                      REGION_MPI_COMM_FREE );

OTF2_EvtWriter_MpiCollectiveBegin( evt_writer,
                                   NULL,
                                   get_time() );

MPI_Comm_free( &split_comm );

if ( 0 == rank % 2 )
{
    OTF2_EvtWriter_CommDestroy( evt_writer,
                                NULL,
                                get_time(),
                                COMM_SPLIT_0 );

    OTF2_EvtWriter_MpiCollectiveEnd( evt_writer,
                                     NULL,
                                     get_time(),
                                     OTF2_COLLECTIVE_OP_DESTROY_HANDLE,
                                     COMM_SPLIT_0,
                                     OTF2_COLLECTIVE_ROOT_NONE,
                                     0 /* bytes provided */,
                                     0 /* bytes obtained */ );
}
else
{
    OTF2_EvtWriter_CommDestroy( evt_writer,
                                NULL,
                                get_time(),
                                COMM_SPLIT_1 );

    OTF2_EvtWriter_MpiCollectiveEnd( evt_writer,
                                     NULL,
                                     get_time(),
                                     OTF2_COLLECTIVE_OP_DESTROY_HANDLE,
                                     COMM_SPLIT_1,
```

```
                                    OTF2_COLLECTIVE_ROOT_NONE,
                                    0 /* bytes provided */,
                                    0 /* bytes obtained */ );
    }

    OTF2_EvtWriter_Leave( evt_writer,
                          NULL,
                          get_time(),
                          REGION_MPI_COMM_FREE );

    OTF2_EvtWriter_Enter( evt_writer,
                          NULL,
                          get_time(),
                          REGION_MPI_TEST );

    /* fake failing MPI_Test( &barrier_request, &flag, &status ); */
    OTF2_EvtWriter_MpiRequestTest( evt_writer,
                                   NULL,
                                   get_time(),
                                   barrier_request_id );

    OTF2_EvtWriter_Leave( evt_writer,
                          NULL,
                          get_time(),
                          REGION_MPI_TEST );

    OTF2_EvtWriter_Enter( evt_writer,
                          NULL,
                          get_time(),
                          REGION_MPI_BCAST );

    OTF2_EvtWriter_MpiCollectiveBegin( evt_writer,
                                       NULL,
                                       get_time() );

    int res = -1;
    if ( rank % 2 == 0 )
    {
        if ( rank == 0 )
        {
            res = 1;
            MPI_Bcast( &res, 1, MPI_INT, MPI_ROOT, inter_comm );

            OTF2_EvtWriter_MpiCollectiveEnd( evt_writer,
                                             NULL,
                                             get_time(),
                                             OTF2_COLLECTIVE_OP_BCAST,
                                             COMM_INTERCOMM,
                                             OTF2_COLLECTIVE_ROOT_SELF,
                                             0 /* bytes provided */,
                                             0 /* bytes obtained */ );
        }
        else
        {
            MPI_Bcast( &res, 1, MPI_INT, MPI_PROC_NULL, inter_comm );

            OTF2_EvtWriter_MpiCollectiveEnd( evt_writer,
                                             NULL,
                                             get_time(),
                                             OTF2_COLLECTIVE_OP_BCAST,
                                             COMM_INTERCOMM,
                                             OTF2_COLLECTIVE_ROOT_THIS_GROUP,
                                             0 /* bytes provided */,
                                             0 /* bytes obtained */ );
        }
    }
    else if ( rank % 2 == 1 )
    {
        MPI_Bcast( &res, 1, MPI_INT, 0, inter_comm );

        OTF2_EvtWriter_MpiCollectiveEnd( evt_writer,
                                         NULL,
                                         get_time(),
                                         OTF2_COLLECTIVE_OP_BCAST,
                                         COMM_INTERCOMM,
                                         0 /* root */,
                                         0 /* bytes provided */,
                                         0 /* bytes obtained */ );
    }

    OTF2_EvtWriter_Leave( evt_writer,
                          NULL,
                          get_time(),
                          REGION_MPI_BCAST );

    OTF2_EvtWriter_Enter( evt_writer,
                          NULL,
```

```
                      get_time(),
                      REGION_MPI_COMM_FREE );

    OTF2_EvtWriter_MpiCollectiveBegin( evt_writer,
                                       NULL,
                                       get_time() );

    MPI_Comm_free( &inter_comm );

    OTF2_EvtWriter_CommDestroy( evt_writer,
                                NULL,
                                get_time(),
                                COMM_INTERCOMM );

    OTF2_EvtWriter_MpiCollectiveEnd( evt_writer,
                                     NULL,
                                     get_time(),
                                     OTF2_COLLECTIVE_OP_DESTROY_HANDLE,
                                     COMM_INTERCOMM,
                                     OTF2_COLLECTIVE_ROOT_NONE,
                                     0 /* bytes provided */,
                                     0 /* bytes obtained */ );

    OTF2_EvtWriter_Leave( evt_writer,
                          NULL,
                          get_time(),
                          REGION_MPI_COMM_FREE );

    OTF2_EvtWriter_Enter( evt_writer,
                          NULL,
                          get_time(),
                          REGION_MPI_WAIT );

    MPI_Status barrier_status;
    MPI_Wait( &barrier_request, &barrier_status );

    OTF2_EvtWriter_NonBlockingCollectiveComplete( evt_writer,
                                                  NULL,
                                                  get_time(),
                                                  OTF2_COLLECTIVE_OP_BARRIER,
                                                  COMM_WORLD,
                                                  OTF2_COLLECTIVE_ROOT_NONE,
                                                  0 /* bytes provided */,
                                                  0 /* bytes obtained */,
                                                  barrier_request_id );

    OTF2_EvtWriter_Leave( evt_writer,
                          NULL,
                          get_time(),
                          REGION_MPI_WAIT );

    OTF2_EvtWriter_Enter( evt_writer,
                          NULL,
                          get_time(),
                          REGION_MPI_FINALIZE );

    OTF2_EvtWriter_MpiCollectiveBegin( evt_writer,
                                       NULL,
                                       get_time() );

    // fake MPI_Finalize
    MPI_Barrier( MPI_COMM_WORLD );

    OTF2_EvtWriter_CommDestroy( evt_writer,
                                NULL,
                                get_time(),
                                COMM_WORLD );

    OTF2_EvtWriter_MpiCollectiveEnd( evt_writer,
                                     NULL,
                                     get_time(),
                                     OTF2_COLLECTIVE_OP_DESTROY_HANDLE,
                                     COMM_WORLD,
                                     OTF2_COLLECTIVE_ROOT_NONE,
                                     0 /* bytes provided */,
                                     0 /* bytes obtained */ );

    OTF2_EvtWriter_Leave( evt_writer,
                          NULL,
                          get_time(),
                          REGION_MPI_FINALIZE );

    uint64_t epoch_end = get_time();

    OTF2_Archive_CloseEvtWriter( archive, evt_writer );

    OTF2_Archive_CloseEvtFiles( archive );
```

```
    OTF2_Archive_OpenDefFiles( archive );
    OTF2_DefWriter* def_writer = OTF2_Archive_GetDefWriter( archive,
                                                            rank );
    OTF2_Archive_CloseDefWriter( archive, def_writer );
    OTF2_Archive_CloseDefFiles( archive );

    uint64_t epoch_timestamp = epoch_timestamp_spec.tv_sec * 1000000000 + epoch_timestamp_spec.tv_nsec;
    struct
    {
        uint64_t timestamp;
        int      index;
    } epoch_start_pair, global_epoch_start_pair;
    epoch_start_pair.timestamp = epoch_start;
    epoch_start_pair.index     = rank;
    MPI_Allreduce( &epoch_start_pair,
                   &global_epoch_start_pair,
                   1, MPI_LONG_INT, MPI_MINLOC,
                   MPI_COMM_WORLD );
    if ( epoch_start_pair.index != 0 )
    {
        if ( rank == 0 )
        {
            MPI_Recv( &epoch_timestamp, 1, OTF2_MPI_UINT64_T,
                      epoch_start_pair.index, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE );
        }
        else
        {
            MPI_Send( &epoch_timestamp, 1, OTF2_MPI_UINT64_T,
                      0, 0, MPI_COMM_WORLD );
        }
    }

    uint64_t global_epoch_end;
    MPI_Reduce( &epoch_end,
                &global_epoch_end,
                1, OTF2_MPI_UINT64_T, MPI_MAX,
                0, MPI_COMM_WORLD );

    if ( 0 == rank )
    {
        OTF2_GlobalDefWriter* global_def_writer = OTF2_Archive_GetGlobalDefWriter( archive );

        OTF2_GlobalDefWriter_WriteClockProperties( global_def_writer,
                                                   1000000000,
                                                   global_epoch_start_pair.timestamp,
                                                   global_epoch_end - global_epoch_start_pair.timestamp + 1
      ,
                                                   epoch_timestamp );

        OTF2_GlobalDefWriter_WriteString( global_def_writer,  0, "" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer,  1, "Initial Thread" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer,  2, "MPI_Init" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer,  3, "PMPI_Init" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer,  4, "MPI_Finalize" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer,  5, "PMPI_Finalize" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer,  6, "MPI_Comm_split" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer,  7, "PMPI_Comm_split" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer,  8, "MPI_Intercomm_create" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer,  9, "PMPI_Intercomm_create" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 10, "MPI_Comm_free" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 11, "PMPI_Comm_free" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 12, "MPI_Bcast" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 13, "PMPI_Bcast" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 14, "MPI_Ibarrier" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 15, "PMPI_Ibarrier" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 16, "MPI_Test" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 17, "PMPI_Test" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 18, "MPI_Wait" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 19, "PMPI_Wait" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 20, "MyHost" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 21, "node" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 22, "MPI" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 23, "MPI_COMM_WORLD" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 24, "SPLIT 0" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 25, "SPLIT 1" );
        OTF2_GlobalDefWriter_WriteString( global_def_writer, 26, "INTERCOMM" );

        OTF2_GlobalDefWriter_WriteRegion( global_def_writer,
                                          REGION_MPI_INIT,
                                          2 /* region name  */,
                                          3 /* alternative name */,
                                          0 /* description */,
                                          OTF2_REGION_ROLE_FUNCTION,
                                          OTF2_PARADIGM_MPI,
                                          OTF2_REGION_FLAG_NONE,
                                          22 /* source file */,
```

```
                                    0 /* begin lno */,
                                    0 /* end lno */ );

    OTF2_GlobalDefWriter_WriteRegion( global_def_writer,
                                    REGION_MPI_FINALIZE,
                                    4 /* region name  */,
                                    5 /* alternative name */,
                                    0 /* description */,
                                    OTF2_REGION_ROLE_FUNCTION,
                                    OTF2_PARADIGM_MPI,
                                    OTF2_REGION_FLAG_NONE,
                                    22 /* source file */,
                                    0 /* begin lno */,
                                    0 /* end lno */ );

    OTF2_GlobalDefWriter_WriteRegion( global_def_writer,
                                    REGION_MPI_COMM_SPLIT,
                                    6 /* region name  */,
                                    7 /* alternative name */,
                                    0 /* description */,
                                    OTF2_REGION_ROLE_FUNCTION,
                                    OTF2_PARADIGM_MPI,
                                    OTF2_REGION_FLAG_NONE,
                                    22 /* source file */,
                                    0 /* begin lno */,
                                    0 /* end lno */ );

    OTF2_GlobalDefWriter_WriteRegion( global_def_writer,
                                    REGION_MPI_INTERCOMM_CREATE,
                                    8 /* region name  */,
                                    9 /* alternative name */,
                                    0 /* description */,
                                    OTF2_REGION_ROLE_FUNCTION,
                                    OTF2_PARADIGM_MPI,
                                    OTF2_REGION_FLAG_NONE,
                                    22 /* source file */,
                                    0 /* begin lno */,
                                    0 /* end lno */ );

    OTF2_GlobalDefWriter_WriteRegion( global_def_writer,
                                    REGION_MPI_COMM_FREE,
                                    10 /* region name  */,
                                    11 /* alternative name */,
                                    0 /* description */,
                                    OTF2_REGION_ROLE_FUNCTION,
                                    OTF2_PARADIGM_MPI,
                                    OTF2_REGION_FLAG_NONE,
                                    22 /* source file */,
                                    0 /* begin lno */,
                                    0 /* end lno */ );

    OTF2_GlobalDefWriter_WriteRegion( global_def_writer,
                                    REGION_MPI_BCAST,
                                    12 /* region name  */,
                                    13 /* alternative name */,
                                    0 /* description */,
                                    OTF2_REGION_ROLE_COLL_ONE2ALL,
                                    OTF2_PARADIGM_MPI,
                                    OTF2_REGION_FLAG_NONE,
                                    22 /* source file */,
                                    0 /* begin lno */,
                                    0 /* end lno */ );

    OTF2_GlobalDefWriter_WriteRegion( global_def_writer,
                                    REGION_MPI_IBARRIER,
                                    14 /* region name  */,
                                    15 /* alternative name */,
                                    0 /* description */,
                                    OTF2_REGION_ROLE_BARRIER,
                                    OTF2_PARADIGM_MPI,
                                    OTF2_REGION_FLAG_NONE,
                                    22 /* source file */,
                                    0 /* begin lno */,
                                    0 /* end lno */ );

    OTF2_GlobalDefWriter_WriteRegion( global_def_writer,
                                    REGION_MPI_TEST,
                                    16 /* region name  */,
                                    17 /* alternative name */,
                                    0 /* description */,
                                    OTF2_REGION_ROLE_BARRIER,
                                    OTF2_PARADIGM_MPI,
                                    OTF2_REGION_FLAG_NONE,
                                    22 /* source file */,
                                    0 /* begin lno */,
                                    0 /* end lno */ );
```

```
    OTF2_GlobalDefWriter_WriteRegion( global_def_writer,
                                      REGION_MPI_WAIT,
                                      18 /* region name  */,
                                      19 /* alternative name */,
                                      0 /* description */,
                                      OTF2_REGION_ROLE_FUNCTION,
                                      OTF2_PARADIGM_MPI,
                                      OTF2_REGION_FLAG_NONE,
                                      22 /* source file */,
                                      0 /* begin lno */,
                                      0 /* end lno */ );

    OTF2_GlobalDefWriter_WriteSystemTreeNode( global_def_writer,
                                              0 /* id */,
                                              20 /* name */,
                                              21 /* class */,
                                              OTF2_UNDEFINED_SYSTEM_TREE_NODE /* parent */ );

    for ( int r = 0; r < size; r++ )
    {
        char process_name[ 32 ];
        snprintf( process_name, sizeof( process_name ), "MPI Rank %d", r );
        OTF2_GlobalDefWriter_WriteString( global_def_writer,
                                          27 + r,
                                          process_name );

        OTF2_GlobalDefWriter_WriteLocationGroup( global_def_writer,
                                                 r /* id */,
                                                 27 + r /* name */,
                                                 OTF2_LOCATION_GROUP_TYPE_PROCESS,
                                                 0 /* system tree */,
                                                 OTF2_UNDEFINED_LOCATION_GROUP /* creating process */ )
;

        OTF2_GlobalDefWriter_WriteLocation( global_def_writer,
                                            r /* id */,
                                            1 /* name */,
                                            OTF2_LOCATION_TYPE_CPU_THREAD,
                                            43 /* # events */,
                                            r /* location group */ );
    }

    uint64_t comm_locations[ size ];
    for ( int r = 0; r < size; r++ )
    {
        comm_locations[ r ] = r;
    }
    OTF2_GlobalDefWriter_WriteGroup( global_def_writer,
                                     0 /* id */,
                                     24 /* name */,
                                     OTF2_GROUP_TYPE_COMM_LOCATIONS,
                                     OTF2_PARADIGM_MPI,
                                     OTF2_GROUP_FLAG_NONE,
                                     size,
                                     comm_locations );

    OTF2_GlobalDefWriter_WriteGroup( global_def_writer,
                                     1 /* id */,
                                     0 /* name */,
                                     OTF2_GROUP_TYPE_COMM_GROUP,
                                     OTF2_PARADIGM_MPI,
                                     OTF2_GROUP_FLAG_NONE,
                                     size,
                                     comm_locations );

    OTF2_GlobalDefWriter_WriteComm( global_def_writer,
                                    COMM_WORLD,
                                    23 /* name */,
                                    1 /* group */,
                                    OTF2_UNDEFINED_COMM /* parent */,
                                    OTF2_COMM_FLAG_CREATE_DESTROY_EVENTS /* flags */ );

    for ( int r = 0; r < size; r += 2 )
    {
        comm_locations[ r / 2 ] = r;
    }
    OTF2_GlobalDefWriter_WriteGroup( global_def_writer,
                                     2 /* id */,
                                     0 /* name */,
                                     OTF2_GROUP_TYPE_COMM_GROUP,
                                     OTF2_PARADIGM_MPI,
                                     OTF2_GROUP_FLAG_NONE,
                                     ( size + 1 ) / 2,
                                     comm_locations );

    OTF2_GlobalDefWriter_WriteComm( global_def_writer,
                                    COMM_SPLIT_0,
```

```
                                        24 /* name */,
                                        2 /* group */,
                                        COMM_WORLD,
                                        OTF2_COMM_FLAG_CREATE_DESTROY_EVENTS /* flags */ );

        for ( int r = 1; r < size; r += 2 )
        {
            comm_locations[ r / 2 ] = r;
        }
        OTF2_GlobalDefWriter_WriteGroup( global_def_writer,
                                         3 /* id */,
                                         0 /* name */,
                                         OTF2_GROUP_TYPE_COMM_GROUP,
                                         OTF2_PARADIGM_MPI,
                                         OTF2_GROUP_FLAG_NONE,
                                         size / 2,
                                         comm_locations );

        OTF2_GlobalDefWriter_WriteComm( global_def_writer,
                                        COMM_SPLIT_1,
                                        25 /* name */,
                                        3 /* group */,
                                        COMM_WORLD,
                                        OTF2_COMM_FLAG_CREATE_DESTROY_EVENTS /* flags */ );

        OTF2_GlobalDefWriter_WriteInterComm( global_def_writer,
                                             COMM_INTERCOMM,
                                             26 /* name */,
                                             2 /* groupA */,
                                             3 /* groupB */,
                                             COMM_WORLD,
                                             OTF2_COMM_FLAG_CREATE_DESTROY_EVENTS /* flags */ );
        OTF2_Archive_CloseGlobalDefWriter( archive,
                                           global_def_writer );
    }
    MPI_Barrier( MPI_COMM_WORLD );

    OTF2_Archive_Close( archive );

    MPI_Finalize();

    return EXIT_SUCCESS;
}
```

A.6 otf2_openmp_reader_example.c

OpenMP reader example which reads one location per thread at a time

```
/*
 * This file is part of the Score-P software (http://www.score-p.org)
 *
 * Copyright (c) 2009-2013,
 * RWTH Aachen University, Germany
 *
 * Copyright (c) 2009-2013,
 * Gesellschaft fuer numerische Simulation mbH Braunschweig, Germany
 *
 * Copyright (c) 2009-2014, 2023,
 * Technische Universitaet Dresden, Germany
 *
 * Copyright (c) 2009-2013,
 * University of Oregon, Eugene, USA
 *
 * Copyright (c) 2009-2014,
 * Forschungszentrum Juelich GmbH, Germany
 *
 * Copyright (c) 2009-2013,
 * German Research School for Simulation Sciences GmbH, Juelich/Aachen, Germany
 *
 * Copyright (c) 2009-2013,
 * Technische Universitaet Muenchen, Germany
 *
 * This software may be modified and distributed under the terms of
 * a BSD-style license.  See the COPYING file in the package base
 * directory for details.
 *
 */

#include <stdlib.h>
```

```c
#include <otf2/otf2.h>

#include <otf2/OTF2_OpenMP_Locks.h>

#include <stdlib.h>
#include <stdio.h>
#include <inttypes.h>

static OTF2_CallbackCode
Enter_print( uint64_t            locationID,
             uint64_t            time,
             uint64_t            eventPosition,
             void*               userData,
             OTF2_AttributeList* attributeList,
             OTF2_RegionRef      region )
{
    printf( "Entering region %u at location %" PRIu64 " at time %" PRIu64 ".\n",
            region, location, time );

    return OTF2_CALLBACK_SUCCESS;
}

static OTF2_CallbackCode
Leave_print( uint64_t            locationID,
             uint64_t            time,
             uint64_t            eventPosition,
             void*               userData,
             OTF2_AttributeList* attributeList,
             OTF2_RegionRef      region )
{
    printf( "Leaving region %u at location %" PRIu64 " at time %" PRIu64 ".\n",
            region, location, time );

    return OTF2_CALLBACK_SUCCESS;
}

struct vector
{
    size_t   capacity;
    size_t   size;
    uint64_t members[];
};

static OTF2_CallbackCode
register_location( void*                userData,
                   OTF2_LocationRef     location,
                   OTF2_StringRef       name,
                   OTF2_LocationType    locationType,
                   uint64_t             numberOfEvents,
                   OTF2_LocationGroupRef locationGroup )
{
    struct vector* locations = userData;

    if ( locations->size == locations->capacity )
    {
        return OTF2_CALLBACK_INTERRUPT;
    }

    locations->members[ locations->size++ ] = location;

    return OTF2_CALLBACK_SUCCESS;
}

int
main( int    argc,
      char** argv )
{
    OTF2_Reader* reader = OTF2_Reader_Open( argv[ 1 ] );

    OTF2_OpenMP_Reader_SetLockingCallbacks( reader );

    OTF2_Reader_SetSerialCollectiveCallbacks( reader );

    uint64_t number_of_locations;
    OTF2_Reader_GetNumberOfLocations( reader,
                                      &number_of_locations );
    struct vector* locations = malloc( sizeof( *locations )
                                       + number_of_locations
                                       * sizeof( *locations->members ) );
    locations->capacity = number_of_locations;
    locations->size     = 0;

    OTF2_GlobalDefReader* global_def_reader = OTF2_Reader_GetGlobalDefReader( reader );

    OTF2_GlobalDefReaderCallbacks* global_def_callbacks = OTF2_GlobalDefReaderCallbacks_New();
    OTF2_GlobalDefReaderCallbacks_SetLocationCallback( global_def_callbacks,
                                                       register_location );
```

```
    OTF2_Reader_RegisterGlobalDefCallbacks( reader,
                                            global_def_reader,
                                            global_def_callbacks,
                                            locations );
    OTF2_GlobalDefReaderCallbacks_Delete( global_def_callbacks );

    uint64_t definitions_read = 0;
    OTF2_Reader_ReadAllGlobalDefinitions( reader,
                                          global_def_reader,
                                          &definitions_read );

    for ( size_t i = 0; i < locations->size; i++ )
    {
        OTF2_Reader_SelectLocation( reader, locations->members[ i ] );
    }

    bool successful_open_def_files =
        OTF2_Reader_OpenDefFiles( reader ) == OTF2_SUCCESS;
    OTF2_Reader_OpenEvtFiles( reader );

    for ( size_t i = 0; i < locations->size; i++ )
    {
        if ( successful_open_def_files )
        {
            OTF2_DefReader* def_reader =
                OTF2_Reader_GetDefReader( reader, locations->members[ i ] );
            if ( def_reader )
            {
                uint64_t def_reads = 0;
                OTF2_Reader_ReadAllLocalDefinitions( reader,
                                                     def_reader,
                                                     &def_reads );
                OTF2_Reader_CloseDefReader( reader, def_reader );
            }
        }
    }

    if ( successful_open_def_files )
    {
        OTF2_Reader_CloseDefFiles( reader );
    }

    OTF2_EvtReaderCallbacks* event_callbacks = OTF2_EvtReaderCallbacks_New();
    OTF2_EvtReaderCallbacks_SetEnterCallback( event_callbacks,
                                              &Enter_print );
    OTF2_EvtReaderCallbacks_SetLeaveCallback( event_callbacks,
                                              &Leave_print );

    #pragma omp parallel shared(reader)
    {
        #pragma omp for
        for ( size_t i = 0; i < locations->size; i++ )
        {
            OTF2_EvtReader* evt_reader =
                OTF2_Reader_GetEvtReader( reader, locations->members[ i ] );

            OTF2_Reader_RegisterEvtCallbacks( reader,
                                              evt_reader,
                                              event_callbacks,
                                              NULL );

            uint64_t events_read = 0;
            OTF2_Reader_ReadAllLocalEvents( reader,
                                            evt_reader,
                                            &events_read );

            OTF2_Reader_CloseEvtReader( reader,
                                        evt_reader );
        }
    }

    OTF2_EvtReaderCallbacks_Delete( event_callbacks );

    OTF2_Reader_CloseEvtFiles( reader );

    OTF2_Reader_Close( reader );

    free( locations );

    return EXIT_SUCCESS;
}
```

A.7 otf2_openmp_writer_example.c

OpenMP writing example

```
/*
 * This file is part of the Score-P software (http://www.score-p.org)
 *
 * Copyright (c) 2009-2013,
 * RWTH Aachen University, Germany
 *
 * Copyright (c) 2009-2013,
 * Gesellschaft fuer numerische Simulation mbH Braunschweig, Germany
 *
 * Copyright (c) 2009-2014, 2021,
 * Technische Universitaet Dresden, Germany
 *
 * Copyright (c) 2009-2013,
 * University of Oregon, Eugene, USA
 *
 * Copyright (c) 2009-2013,
 * Forschungszentrum Juelich GmbH, Germany
 *
 * Copyright (c) 2009-2013,
 * German Research School for Simulation Sciences GmbH, Juelich/Aachen, Germany
 *
 * Copyright (c) 2009-2013,
 * Technische Universitaet Muenchen, Germany
 *
 * This software may be modified and distributed under the terms of
 * a BSD-style license.  See the COPYING file in the package base
 * directory for details.
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <inttypes.h>
#include <time.h>

#include <otf2/otf2.h>

#include <otf2/OTF2_OpenMP_Locks.h>

static OTF2_TimeStamp
get_time( void )
{
    static uint64_t sequence;
    #pragma omp threadprivate(sequence)

    return sequence++;
}

static OTF2_FlushType
pre_flush( void*            userData,
           OTF2_FileType    fileType,
           OTF2_LocationRef location,
           void*            callerData,
           bool             final )
{
    return OTF2_FLUSH;
}

static OTF2_TimeStamp
post_flush( void*            userData,
            OTF2_FileType    fileType,
            OTF2_LocationRef location )
{
    return get_time();
}

static OTF2_FlushCallbacks flush_callbacks =
{
    .otf2_pre_flush  = pre_flush,
    .otf2_post_flush = post_flush
};

int
main( int    argc,
      char** argv )
{
    OTF2_Archive* archive = OTF2_Archive_Open( "ArchivePath",
                                               "ArchiveName",
                                               OTF2_FILEMODE_WRITE,
```

```
                                        1024 * 1024 /* event chunk size */,
                                        4 * 1024 * 1024 /* def chunk size */,
                                        OTF2_SUBSTRATE_POSIX,
                                        OTF2_COMPRESSION_NONE );

OTF2_Archive_SetFlushCallbacks( archive, &flush_callbacks, NULL );

OTF2_Archive_SetSerialCollectiveCallbacks( archive );

OTF2_OpenMP_Archive_SetLockingCallbacks( archive );

OTF2_Archive_OpenEvtFiles( archive );

int number_of_threads;
#pragma omp parallel shared(archive)
{
    #pragma omp master
    number_of_threads = omp_get_num_threads();

    OTF2_EvtWriter* evt_writer;
    evt_writer = OTF2_Archive_GetEvtWriter( archive,
                                            omp_get_thread_num() );
    printf( "%p\n", evt_writer );

    OTF2_EvtWriter_Enter( evt_writer,
                          NULL,
                          get_time(),
                          0 /* region */ );
    OTF2_EvtWriter_Leave( evt_writer,
                          NULL,
                          get_time(),
                          0 /* region */ );

    OTF2_Archive_CloseEvtWriter( archive, evt_writer );
}

OTF2_Archive_CloseEvtFiles( archive );

OTF2_Archive_OpenDefFiles( archive );
for ( int thread = 0; thread < number_of_threads; thread++ )
{
    OTF2_DefWriter* def_writer = OTF2_Archive_GetDefWriter( archive,
                                                            thread );
    OTF2_Archive_CloseDefWriter( archive, def_writer );
}
OTF2_Archive_CloseDefFiles( archive );

OTF2_GlobalDefWriter* global_def_writer = OTF2_Archive_GetGlobalDefWriter( archive );

OTF2_GlobalDefWriter_WriteClockProperties( global_def_writer,
                                           1 /* 1 tick per second */,
                                           0 /* epoch */,
                                           2 /* length */,
                                           OTF2_UNDEFINED_TIMESTAMP );

OTF2_GlobalDefWriter_WriteString( global_def_writer, 0, "" );
OTF2_GlobalDefWriter_WriteString( global_def_writer, 1, "Initial Process" );
OTF2_GlobalDefWriter_WriteString( global_def_writer, 2, "Main Thread" );
OTF2_GlobalDefWriter_WriteString( global_def_writer, 3, "MyFunction" );
OTF2_GlobalDefWriter_WriteString( global_def_writer, 4, "Alternative function name (e.g. mangled one)"
  );
OTF2_GlobalDefWriter_WriteString( global_def_writer, 5, "Computes something" );
OTF2_GlobalDefWriter_WriteString( global_def_writer, 6, "MyHost" );
OTF2_GlobalDefWriter_WriteString( global_def_writer, 7, "node" );

OTF2_GlobalDefWriter_WriteRegion( global_def_writer,
                                  0 /* id */,
                                  3 /* region name  */,
                                  4 /* alternative name */,
                                  5 /* description */,
                                  OTF2_REGION_ROLE_FUNCTION,
                                  OTF2_PARADIGM_USER,
                                  OTF2_REGION_FLAG_NONE,
                                  0 /* source file */,
                                  0 /* begin lno */,
                                  0 /* end lno */ );

OTF2_GlobalDefWriter_WriteSystemTreeNode( global_def_writer,
                                          0 /* id */,
                                          6 /* name */,
                                          7 /* class */,
                                          OTF2_UNDEFINED_SYSTEM_TREE_NODE /* parent */ );
OTF2_GlobalDefWriter_WriteLocationGroup( global_def_writer,
                                         0 /* id */,
                                         1 /* name */,
                                         OTF2_LOCATION_GROUP_TYPE_PROCESS,
                                         0 /* system tree */,
```

```
                                        OTF2_UNDEFINED_LOCATION_GROUP /* creating process */ );

    for ( int i = 0; i < number_of_threads; i++ )
    {
        OTF2_StringRef name = 2;
        if ( i > 0 )
        {
            name = 7 + i;
            char name_buf[ 32 ];
            snprintf( name_buf, sizeof( name_buf ), "OpenMP Thread %d", i );
            OTF2_GlobalDefWriter_WriteString( global_def_writer, name, name_buf );
        }

        OTF2_GlobalDefWriter_WriteLocation( global_def_writer,
                                            i /* id */,
                                            name,
                                            OTF2_LOCATION_TYPE_CPU_THREAD,
                                            2 /* # events */,
                                            0 /* location group */ );
    }

    OTF2_Archive_Close( archive );

    return EXIT_SUCCESS;
}
```

A.8 otf2_pthread_writer_example.c

Pthread writing example

```
/*
 * This file is part of the Score-P software (http://www.score-p.org)
 *
 * Copyright (c) 2009-2013,
 * RWTH Aachen University, Germany
 *
 * Copyright (c) 2009-2013,
 * Gesellschaft fuer numerische Simulation mbH Braunschweig, Germany
 *
 * Copyright (c) 2009-2014, 2021,
 * Technische Universitaet Dresden, Germany
 *
 * Copyright (c) 2009-2013,
 * University of Oregon, Eugene, USA
 *
 * Copyright (c) 2009-2013,
 * Forschungszentrum Juelich GmbH, Germany
 *
 * Copyright (c) 2009-2013,
 * German Research School for Simulation Sciences GmbH, Juelich/Aachen, Germany
 *
 * Copyright (c) 2009-2013,
 * Technische Universitaet Muenchen, Germany
 *
 * This software may be modified and distributed under the terms of
 * a BSD-style license.  See the COPYING file in the package base
 * directory for details.
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <inttypes.h>

#include <otf2/otf2.h>

#include <otf2/OTF2_Pthread_Locks.h>

struct thread_data
{
    OTF2_Archive* archive;
    uint64_t      sequence;
    pthread_t     tid;
    uint64_t      lid;
};

static pthread_key_t tpd;

static OTF2_TimeStamp
get_time( void )
```

```
{
    struct thread_data* data = pthread_getspecific( tpd );
    return data->sequence++;
}

static OTF2_FlushType
pre_flush( void*            userData,
           OTF2_FileType    fileType,
           OTF2_LocationRef location,
           void*            callerData,
           bool             final )
{
    return OTF2_FLUSH;
}

static OTF2_TimeStamp
post_flush( void*            userData,
            OTF2_FileType    fileType,
            OTF2_LocationRef location )
{
    return get_time();
}

static OTF2_FlushCallbacks flush_callbacks =
{
    .otf2_pre_flush  = pre_flush,
    .otf2_post_flush = post_flush
};

void*
event_writer( void* arg )
{
    struct thread_data* data = arg;
    pthread_setspecific( tpd, data );

    OTF2_EvtWriter* evt_writer = OTF2_Archive_GetEvtWriter( data->archive,
                                                            data->lid );

    OTF2_EvtWriter_Enter( evt_writer,
                          NULL,
                          get_time(),
                          0 /* region */ );
    OTF2_EvtWriter_Leave( evt_writer,
                          NULL,
                          get_time(),
                          0 /* region */ );

    OTF2_Archive_CloseEvtWriter( data->archive, evt_writer );

    return NULL;
}


int
main( int    argc,
      char** argv )
{
    int number_of_threads = 1;
    if ( argc > 1 )
    {
        number_of_threads = atoi( argv[ 1 ] );
    }
    pthread_key_create( &tpd, NULL );

    OTF2_Archive* archive = OTF2_Archive_Open( "ArchivePath",
                                               "ArchiveName",
                                               OTF2_FILEMODE_WRITE,
                                               1024 * 1024 /* event chunk size */,
                                               4 * 1024 * 1024 /* def chunk size */,
                                               OTF2_SUBSTRATE_POSIX,
                                               OTF2_COMPRESSION_NONE );

    OTF2_Archive_SetFlushCallbacks( archive, &flush_callbacks, NULL );

    OTF2_Archive_SetSerialCollectiveCallbacks( archive );

    OTF2_Pthread_Archive_SetLockingCallbacks( archive, NULL );

    OTF2_Archive_OpenEvtFiles( archive );

    struct thread_data* threads = calloc( number_of_threads, sizeof( *threads ) );
    for ( int i = 0; i < number_of_threads; i++ )
    {
        threads[ i ].archive = archive;
        threads[ i ].lid     = i;
        pthread_create( &threads[ i ].tid, NULL, event_writer, &threads[ i ] );
    }
```

```
    for ( int i = 0; i < number_of_threads; i++ )
    {
        pthread_join( threads[ i ].tid, NULL );
    }

    OTF2_Archive_CloseEvtFiles( archive );

    OTF2_Archive_OpenDefFiles( archive );
    for ( int thread = 0; thread < number_of_threads; thread++ )
    {
        OTF2_DefWriter* def_writer = OTF2_Archive_GetDefWriter( archive,
                                                                thread );
        OTF2_Archive_CloseDefWriter( archive, def_writer );
    }
    OTF2_Archive_CloseDefFiles( archive );

    OTF2_GlobalDefWriter* global_def_writer = OTF2_Archive_GetGlobalDefWriter( archive );

    OTF2_GlobalDefWriter_WriteClockProperties( global_def_writer,
                                               1 /* 1 tick per second */,
                                               0 /* epoch */,
                                               2 /* length */,
                                               OTF2_UNDEFINED_TIMESTAMP );

    OTF2_GlobalDefWriter_WriteString( global_def_writer, 0, "" );
    OTF2_GlobalDefWriter_WriteString( global_def_writer, 1, "Initial Process" );
    OTF2_GlobalDefWriter_WriteString( global_def_writer, 2, "Main Thread" );
    OTF2_GlobalDefWriter_WriteString( global_def_writer, 3, "MyFunction" );
    OTF2_GlobalDefWriter_WriteString( global_def_writer, 4, "Alternative function name (e.g. mangled one)"
      );
    OTF2_GlobalDefWriter_WriteString( global_def_writer, 5, "Computes something" );
    OTF2_GlobalDefWriter_WriteString( global_def_writer, 6, "MyHost" );
    OTF2_GlobalDefWriter_WriteString( global_def_writer, 7, "node" );

    OTF2_GlobalDefWriter_WriteRegion( global_def_writer,
                                      0 /* id */,
                                      3 /* region name  */,
                                      4 /* alternative name */,
                                      5 /* description */,
                                      OTF2_REGION_ROLE_FUNCTION,
                                      OTF2_PARADIGM_USER,
                                      OTF2_REGION_FLAG_NONE,
                                      0 /* source file */,
                                      0 /* begin lno */,
                                      0 /* end lno */ );

    OTF2_GlobalDefWriter_WriteSystemTreeNode( global_def_writer,
                                              0 /* id */,
                                              6 /* name */,
                                              7 /* class */,
                                              OTF2_UNDEFINED_SYSTEM_TREE_NODE /* parent */ );
    OTF2_GlobalDefWriter_WriteLocationGroup( global_def_writer,
                                             0 /* id */,
                                             1 /* name */,
                                             OTF2_LOCATION_GROUP_TYPE_PROCESS,
                                             0 /* system tree */,
                                             OTF2_UNDEFINED_LOCATION_GROUP /* creating process */ );

    for ( int i = 0; i < number_of_threads; i++ )
    {
        OTF2_StringRef name = 2;
        if ( i > 0 )
        {
            name = 7 + i;
            char name_buf[ 32 ];
            snprintf( name_buf, sizeof( name_buf ), "Pthread %d", i );
            OTF2_GlobalDefWriter_WriteString( global_def_writer, name, name_buf );
        }

        OTF2_GlobalDefWriter_WriteLocation( global_def_writer,
                                            i /* id */,
                                            name,
                                            OTF2_LOCATION_TYPE_CPU_THREAD,
                                            2 /* # events */,
                                            0 /* location group */ );
    }

    OTF2_Archive_Close( archive );

    pthread_key_delete( tpd );

    return EXIT_SUCCESS;
}
```

A.9 otf2_reader_example.c

Simple reading example

```
/*
 * This file is part of the Score-P software (http://www.score-p.org)
 *
 * Copyright (c) 2009-2013,
 * RWTH Aachen University, Germany
 *
 * Copyright (c) 2009-2013,
 * Gesellschaft fuer numerische Simulation mbH Braunschweig, Germany
 *
 * Copyright (c) 2009-2014,
 * Technische Universitaet Dresden, Germany
 *
 * Copyright (c) 2009-2013,
 * University of Oregon, Eugene, USA
 *
 * Copyright (c) 2009-2014,
 * Forschungszentrum Juelich GmbH, Germany
 *
 * Copyright (c) 2009-2013,
 * German Research School for Simulation Sciences GmbH, Juelich/Aachen, Germany
 *
 * Copyright (c) 2009-2013,
 * Technische Universitaet Muenchen, Germany
 *
 * This software may be modified and distributed under the terms of
 * a BSD-style license.  See the COPYING file in the package base
 * directory for details.
 *
 */

#include <otf2/otf2.h>

#include <stdlib.h>
#include <stdio.h>
#include <inttypes.h>

static OTF2_CallbackCode
Enter_print( OTF2_LocationRef    location,
             OTF2_TimeStamp      time,
             void*               userData,
             OTF2_AttributeList* attributes,
             OTF2_RegionRef      region )
{
    printf( "Entering region %u at location %" PRIu64 " at time %" PRIu64 ".\n",
            region, location, time );

    return OTF2_CALLBACK_SUCCESS;
}

static OTF2_CallbackCode
Leave_print( OTF2_LocationRef    location,
             OTF2_TimeStamp      time,
             void*               userData,
             OTF2_AttributeList* attributes,
             OTF2_RegionRef      region )
{
    printf( "Leaving region %u at location %" PRIu64 " at time %" PRIu64 ".\n",
            region, location, time );

    return OTF2_CALLBACK_SUCCESS;
}

struct vector
{
    size_t   capacity;
    size_t   size;
    uint64_t members[];
};

static OTF2_CallbackCode
GlobDefLocation_Register( void*                userData,
                          OTF2_LocationRef     location,
                          OTF2_StringRef       name,
                          OTF2_LocationType    locationType,
                          uint64_t             numberOfEvents,
                          OTF2_LocationGroupRef locationGroup )
{
    struct vector* locations = userData;
```

```
    if ( locations->size == locations->capacity )
    {
        return OTF2_CALLBACK_INTERRUPT;
    }

    locations->members[ locations->size++ ] = location;

    return OTF2_CALLBACK_SUCCESS;
}

int
main( int    argc,
      char** argv )
{
    OTF2_Reader* reader = OTF2_Reader_Open( "ArchivePath/ArchiveName.otf2" );

    OTF2_Reader_SetSerialCollectiveCallbacks( reader );

    uint64_t number_of_locations;
    OTF2_Reader_GetNumberOfLocations( reader,
                                      &number_of_locations );
    struct vector* locations = malloc( sizeof( *locations )
                                       + number_of_locations
                                       * sizeof( *locations->members ) );
    locations->capacity = number_of_locations;
    locations->size     = 0;

    OTF2_GlobalDefReader* global_def_reader = OTF2_Reader_GetGlobalDefReader( reader );

    OTF2_GlobalDefReaderCallbacks* global_def_callbacks = OTF2_GlobalDefReaderCallbacks_New();
    OTF2_GlobalDefReaderCallbacks_SetLocationCallback( global_def_callbacks,
                                                       &GlobDefLocation_Register );
    OTF2_Reader_RegisterGlobalDefCallbacks( reader,
                                            global_def_reader,
                                            global_def_callbacks,
                                            locations );
    OTF2_GlobalDefReaderCallbacks_Delete( global_def_callbacks );

    uint64_t definitions_read = 0;
    OTF2_Reader_ReadAllGlobalDefinitions( reader,
                                          global_def_reader,
                                          &definitions_read );

    for ( size_t i = 0; i < locations->size; i++ )
    {
        OTF2_Reader_SelectLocation( reader, locations->members[ i ] );
    }

    bool successful_open_def_files =
        OTF2_Reader_OpenDefFiles( reader ) == OTF2_SUCCESS;
    OTF2_Reader_OpenEvtFiles( reader );

    for ( size_t i = 0; i < locations->size; i++ )
    {
        if ( successful_open_def_files )
        {
            OTF2_DefReader* def_reader =
                OTF2_Reader_GetDefReader( reader, locations->members[ i ] );
            if ( def_reader )
            {
                uint64_t def_reads = 0;
                OTF2_Reader_ReadAllLocalDefinitions( reader,
                                                     def_reader,
                                                     &def_reads );
                OTF2_Reader_CloseDefReader( reader, def_reader );
            }
        }
        OTF2_EvtReader* evt_reader =
            OTF2_Reader_GetEvtReader( reader, locations->members[ i ] );
    }

    if ( successful_open_def_files )
    {
        OTF2_Reader_CloseDefFiles( reader );
    }

    OTF2_GlobalEvtReader* global_evt_reader = OTF2_Reader_GetGlobalEvtReader( reader );

    OTF2_GlobalEvtReaderCallbacks* event_callbacks = OTF2_GlobalEvtReaderCallbacks_New();
    OTF2_GlobalEvtReaderCallbacks_SetEnterCallback( event_callbacks,
                                                    &Enter_print );
    OTF2_GlobalEvtReaderCallbacks_SetLeaveCallback( event_callbacks,
                                                    &Leave_print );
    OTF2_Reader_RegisterGlobalEvtCallbacks( reader,
                                            global_evt_reader,
                                            event_callbacks,
                                            NULL );
```

```
    OTF2_GlobalEvtReaderCallbacks_Delete( event_callbacks );

    uint64_t events_read = 0;
    OTF2_Reader_ReadAllGlobalEvents( reader,
                                     global_evt_reader,
                                     &events_read );

    OTF2_Reader_CloseGlobalEvtReader( reader, global_evt_reader );
    OTF2_Reader_CloseEvtFiles( reader );

    OTF2_Reader_Close( reader );

    free( locations );

    return EXIT_SUCCESS;
}
```

A.10 otf2_writer_example.c

Simple writing example

```
/*
 * This file is part of the Score-P software (http://www.score-p.org)
 *
 * Copyright (c) 2009-2013,
 * RWTH Aachen University, Germany
 *
 * Copyright (c) 2009-2013,
 * Gesellschaft fuer numerische Simulation mbH Braunschweig, Germany
 *
 * Copyright (c) 2009-2014, 2021,
 * Technische Universitaet Dresden, Germany
 *
 * Copyright (c) 2009-2013,
 * University of Oregon, Eugene, USA
 *
 * Copyright (c) 2009-2013,
 * Forschungszentrum Juelich GmbH, Germany
 *
 * Copyright (c) 2009-2013,
 * German Research School for Simulation Sciences GmbH, Juelich/Aachen, Germany
 *
 * Copyright (c) 2009-2013,
 * Technische Universitaet Muenchen, Germany
 *
 * This software may be modified and distributed under the terms of
 * a BSD-style license.  See the COPYING file in the package base
 * directory for details.
 *
 */

#include <otf2/otf2.h>

#include <stdlib.h>

static OTF2_TimeStamp
get_time( void )
{
    static uint64_t sequence;
    return sequence++;
}

static OTF2_FlushType
pre_flush( void*            userData,
           OTF2_FileType    fileType,
           OTF2_LocationRef location,
           void*            callerData,
           bool             final )
{
    return OTF2_FLUSH;
}

static OTF2_TimeStamp
post_flush( void*            userData,
            OTF2_FileType    fileType,
            OTF2_LocationRef location )
{
    return get_time();
}
```

```
static OTF2_FlushCallbacks flush_callbacks =
{
    .otf2_pre_flush  = pre_flush,
    .otf2_post_flush = post_flush
};

int
main( int    argc,
      char** argv )
{
    OTF2_Archive* archive = OTF2_Archive_Open( "ArchivePath",
                                               "ArchiveName",
                                               OTF2_FILEMODE_WRITE,
                                               1024 * 1024 /* event chunk size */,
                                               4 * 1024 * 1024 /* def chunk size */,
                                               OTF2_SUBSTRATE_POSIX,
                                               OTF2_COMPRESSION_NONE );

    OTF2_Archive_SetFlushCallbacks( archive, &flush_callbacks, NULL );

    OTF2_Archive_SetSerialCollectiveCallbacks( archive );

    OTF2_Archive_OpenEvtFiles( archive );

    OTF2_EvtWriter* evt_writer = OTF2_Archive_GetEvtWriter( archive, 0 );

    OTF2_EvtWriter_Enter( evt_writer,
                          NULL,
                          get_time(),
                          0 /* region */ );
    OTF2_EvtWriter_Leave( evt_writer,
                          NULL,
                          get_time(),
                          0 /* region */ );

    OTF2_Archive_CloseEvtWriter( archive, evt_writer );

    OTF2_Archive_CloseEvtFiles( archive );

    OTF2_Archive_OpenDefFiles( archive );
    OTF2_DefWriter* def_writer = OTF2_Archive_GetDefWriter( archive, 0 );
    OTF2_Archive_CloseDefWriter( archive, def_writer );
    OTF2_Archive_CloseDefFiles( archive );

    OTF2_GlobalDefWriter* global_def_writer = OTF2_Archive_GetGlobalDefWriter( archive );

    OTF2_GlobalDefWriter_WriteClockProperties( global_def_writer,
                                               1 /* 1 tick per second */,
                                               0 /* epoch */,
                                               2 /* length */,
                                               OTF2_UNDEFINED_TIMESTAMP );

    OTF2_GlobalDefWriter_WriteString( global_def_writer, 0, "" );
    OTF2_GlobalDefWriter_WriteString( global_def_writer, 1, "Initial Process" );
    OTF2_GlobalDefWriter_WriteString( global_def_writer, 2, "Main Thread" );
    OTF2_GlobalDefWriter_WriteString( global_def_writer, 3, "MyFunction" );
    OTF2_GlobalDefWriter_WriteString( global_def_writer, 4, "Alternative function name (e.g. mangled one)"
      );
    OTF2_GlobalDefWriter_WriteString( global_def_writer, 5, "Computes something" );
    OTF2_GlobalDefWriter_WriteString( global_def_writer, 6, "MyHost" );
    OTF2_GlobalDefWriter_WriteString( global_def_writer, 7, "node" );

    OTF2_GlobalDefWriter_WriteRegion( global_def_writer,
                                      0 /* id */,
                                      3 /* region name  */,
                                      4 /* alternative name */,
                                      5 /* description */,
                                      OTF2_REGION_ROLE_FUNCTION,
                                      OTF2_PARADIGM_USER,
                                      OTF2_REGION_FLAG_NONE,
                                      0 /* source file */,
                                      0 /* begin lno */,
                                      0 /* end lno */ );

    OTF2_GlobalDefWriter_WriteSystemTreeNode( global_def_writer,
                                              0 /* id */,
                                              6 /* name */,
                                              7 /* class */,
                                              OTF2_UNDEFINED_SYSTEM_TREE_NODE /* parent */ );
    OTF2_GlobalDefWriter_WriteLocationGroup( global_def_writer,
                                             0 /* id */,
                                             1 /* name */,
                                             OTF2_LOCATION_GROUP_TYPE_PROCESS,
                                             0 /* system tree */,
                                             OTF2_UNDEFINED_LOCATION_GROUP /* creating process */ );

    OTF2_GlobalDefWriter_WriteLocation( global_def_writer,
```

```
                                        0 /* id */,
                                        2 /* name */,
                                        OTF2_LOCATION_TYPE_CPU_THREAD,
                                        2 /* # events */,
                                        0 /* location group */ );

    OTF2_Archive_Close( archive );

    return EXIT_SUCCESS;
}
```

Index