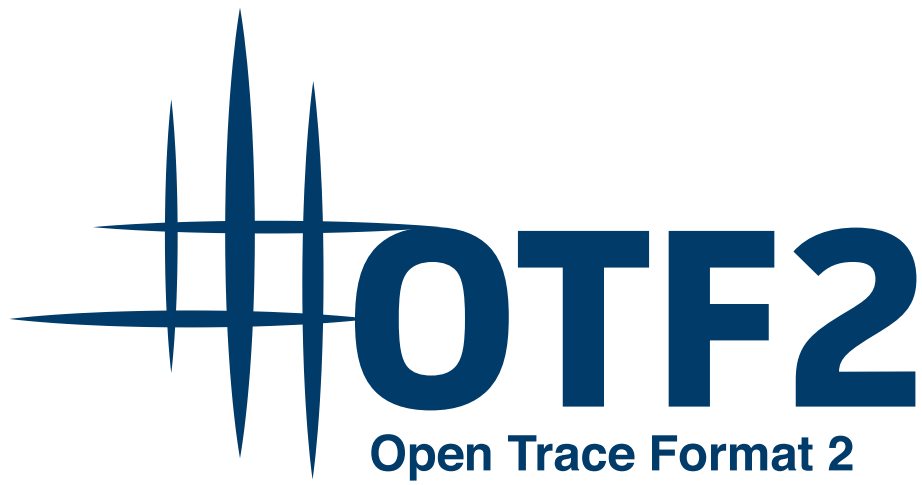


OPEN TRACE FORMAT 2
USER MANUAL
2.2-beta (revision 4718)



OTF2 LICENSE AGREEMENT

COPYRIGHT ©2009-2012,
RWTH Aachen University, Germany
COPYRIGHT ©2009-2012,
Gesellschaft für numerische Simulation mbH, Germany
COPYRIGHT ©2009-2018,
Technische Universität Dresden, Germany
COPYRIGHT ©2009-2012,
University of Oregon, Eugene, USA
COPYRIGHT ©2009-2018,
Forschungszentrum Jülich GmbH, Germany
COPYRIGHT ©2009-2014,
German Research School for Simulation Sciences GmbH, Germany
COPYRIGHT ©2009-2013,
Technische Universität München, Germany

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the names of
RWTH Aachen University,
Gesellschaft für numerische Simulation mbH Braunschweig,
Technische Universität Dresden,
University of Oregon, Eugene,
Forschungszentrum Jülich GmbH,
German Research School for Simulation Sciences GmbH, or the
Technische Universität München,
nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Contents

	Page
Contents	iii
1 Open Trace Format 2	1
1.1 Introduction	1
1.2 Get started	1
Appendix A OTF2 INSTALL	5
Appendix B Deprecated List	15
Appendix C Module Documentation	17
C.1 OTF2 usage examples	17
C.2 OTF2 records	18
C.2.1 Detailed Description	18
C.3 OTF2 callbacks	19
C.4 Usage of OTF2 tools	20
C.4.1 Detailed Description	20
C.5 OTF2 config tool	21
C.6 OTF2 print tool	22
C.7 OTF2 snapshots tool	23
C.8 OTF2 marker tool	24
C.9 OTF2 estimator tool	25
C.10 OTF2 I/O recording	26
C.11 List of all definition records	28
C.12 ClockProperties	28
C.13 Paradigm	29
C.14 ParadigmProperty	29
C.15 IoParadigm	30
C.16 MappingTable	30
C.17 ClockOffset	31
C.18 String	31
C.19 Attribute	32

C.20 SystemTreeNode	32
C.21 LocationGroup	33
C.22 Location	33
C.23 Region	34
C.24 Callsite	34
C.25 Callpath	35
C.26 Group	35
C.27 MetricMember	36
C.28 Metric	37
C.29 MetricClass	37
C.30 MetricInstance	38
C.31 Comm	39
C.32 Parameter	39
C.33 RmaWin	40
C.34 MetricClassRecorder	41
C.35 SystemTreeNodeProperty	41
C.36 SystemTreeNodeDomain	42
C.37 LocationGroupProperty	42
C.38 LocationProperty	43
C.39 CartDimension	43
C.40 CartTopology	44
C.41 CartCoordinate	45
C.42 SourceCodeLocation	46
C.43 CallingContext	46
C.44 CallingContextProperty	47
C.45 InterruptGenerator	48
C.46 IoFileProperty	49
C.47 IoFile	49
C.48 IoRegularFile	49
C.49 IoDirectory	50
C.50 IoHandle	50
C.51 IoPreCreatedHandleState	51
C.52 CallpathParameter	52
C.53 List of all event records	53
C.54 BufferFlush	53
C.55 MeasurementOnOff	53
C.56 Enter	54
C.57 Leave	55
C.58 MpiSend	55
C.59 Mpisend	56

CONTENTS

C.60	MpilsendComplete	57
C.61	MpilrecvRequest	58
C.62	Mpilrecv	58
C.63	Mpilrecv	59
C.64	MpiRequestTest	60
C.65	MpiRequestCancelled	61
C.66	MpiCollectiveBegin	61
C.67	MpiCollectiveEnd	62
C.68	OmpFork	62
C.69	OmpJoin	63
C.70	OmpAcquireLock	63
C.71	OmpReleaseLock	64
C.72	OmpTaskCreate	65
C.73	OmpTaskSwitch	65
C.74	OmpTaskComplete	66
C.75	Metric	66
C.76	ParameterString	67
C.77	ParameterInt	67
C.78	ParameterUnsignedInt	68
C.79	RmaWinCreate	68
C.80	RmaWinDestroy	69
C.81	RmaCollectiveBegin	69
C.82	RmaCollectiveEnd	70
C.83	RmaGroupSync	71
C.84	RmaRequestLock	72
C.85	RmaAcquireLock	73
C.86	RmaTryLock	73
C.87	RmaReleaseLock	74
C.88	RmaSync	74
C.89	RmaWaitChange	75
C.90	RmaPut	75
C.91	RmaGet	76
C.92	RmaAtomic	77
C.93	RmaOpCompleteBlocking	78
C.94	RmaOpCompleteNonBlocking	79
C.95	RmaOpTest	80
C.96	RmaOpCompleteRemote	80
C.97	ThreadFork	81
C.98	ThreadJoin	81
C.99	ThreadTeamBegin	82

C.100ThreadTeamEnd	82
C.101ThreadAcquireLock	83
C.102ThreadReleaseLock	83
C.103ThreadTaskCreate	84
C.104ThreadTaskSwitch	85
C.105ThreadTaskComplete	86
C.106ThreadCreate	86
C.107ThreadBegin	87
C.108ThreadWait	87
C.109ThreadEnd	88
C.110CallingContextEnter	89
C.111CallingContextLeave	89
C.112CallingContextSample	90
C.113IoCreateHandle	90
C.114IoDestroyHandle	91
C.115IoDuplicateHandle	92
C.116IoSeek	92
C.117IoChangeStatusFlags	93
C.118IoDeleteFile	94
C.119IoOperationBegin	95
C.120IoOperationTest	96
C.121IoOperationIssued	96
C.122IoOperationComplete	97
C.123IoOperationCancelled	97
C.124IoAcquireLock	98
C.125IoReleaseLock	98
C.126IoTryLock	99
C.127ProgramBegin	99
C.128ProgramEnd	100
C.129List of all marker records	102
C.130DefMarker	102
C.131Marker	102
C.132List of all snapshot records	103
C.133SnapshotStart	103
C.134SnapshotEnd	103
C.135MeasurementOnOffSnap	104
C.136EnterSnap	105
C.137MpiSendSnap	106
C.138MpisendSnap	106
C.139MpisendCompleteSnap	107

CONTENTS

C.140MpiRecvSnap	107
C.141MpiRecvRequestSnap	108
C.142MpiRecvSnap	109
C.143MpiCollectiveBeginSnap	109
C.144MpiCollectiveEndSnap	110
C.145OmpForkSnap	111
C.146OmpAcquireLockSnap	111
C.147OmpTaskCreateSnap	112
C.148OmpTaskSwitchSnap	112
C.149MetricSnap	113
C.150ParameterStringSnap	114
C.151ParameterIntSnap	114
C.152ParameterUnsignedIntSnap	115
Appendix Index	117

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. OTF2 is available under the BSD open source license that allows free usage for academic and commercial applications.

1.2 Get started

[OTF2 records](#)

[Usage of OTF2 tools](#)

Appendices

Appendix A

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=no]
--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 be 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.

Appendix B

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

Appendix C

Module Documentation

C.1 OTF2 usage examples

Listing of example code.

C.2 OTF2 records

Modules

- [List of all definition records](#)
- [List of all event records](#)
- [List of all marker records](#)
- [List of all snapshot records](#)

C.2.1 Detailed Description

Listings of all OTF2 records.

C.3 OTF2 callbacks

Description of the non-records callbacks available in OTF2.

C.4 Usage of OTF2 tools

Modules

- [OTF2 config tool](#)
- [OTF2 print tool](#)
- [OTF2 snapshots tool](#)
- [OTF2 marker tool](#)
- [OTF2 estimator tool](#)

C.4.1 Detailed Description

Usage instructions of the OTF2 command line tools.

C.5 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 and
--otf2-revision
               prints the revision number of the OTF2 package
--common-revision
               prints the revision number of the common package
--interface-version
               prints the interface version number
--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)
```

C.6 OTF2 print tool

A call to `otf2-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
```

C.7 OTF2 snapshots tool

A call to `otf2-snapshots` has the following syntax:

Usage: `otf2-snapshots [OPTION]... ANCHORFILE`

Append snapshots to existing otf2 traces at given 'break' timestamps.

Options:

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

C.8 OTF2 marker tool

A call to `otf2-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'.
```


C.9 OTF2 estimator tool

A call to `otf2-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:

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

C.10 OTF2 I/O recording

C.10.0.1 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`

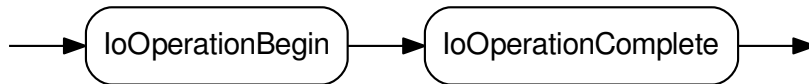
Flags `OTF2_IO_PARADIGM_FLAG_NONE`

C.10 OTF2 I/O recording

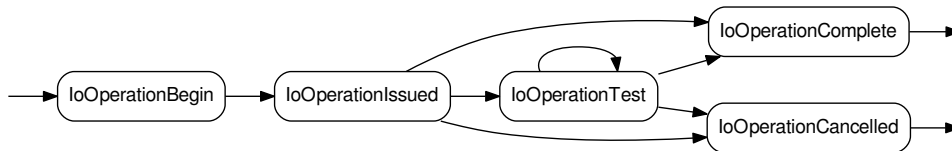
C.10.0.2 Event order for I/O operation records

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:



C.13 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

<i>OTF2</i> _↔ <i>Paradigm</i>	paradigm	The paradigm to attest.
<i>OTF2</i> _↔ <i>StringRef</i>	name	The name of the paradigm. References a String definition.
<i>OTF2</i> _↔ <i>Paradigm</i> _↔ <i>Class</i>	paradigmClass	The class of this paradigm.

See also

`OTF2_GlobalDefWriter_WriteParadigm()`
`OTF2_GlobalDefReaderCallbacks_SetParadigmCallback()`

Since

Version 1.5

C.14 ParadigmProperty

Extensible annotation for the [Paradigm](#) definition.

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

This definition is only valid as a global definition.

Attributes

<i>OTF2</i> _↔ <i>Paradigm</i>	paradigm	The paradigm to annotate.
<i>OTF2</i> _↔ <i>Paradigm</i> _↔ <i>Property</i>	property	The property.
<i>OTF2</i> _↔ <i>Type</i>	type	The type of this property. Must match with the defined type of the <i>property</i> .
<i>OTF2</i> _↔ <i>Attribute</i> _↔ <i>Value</i>	value	The value of this property.

See also

`OTF2_GlobalDefWriter_WriteParadigmProperty()`
`OTF2_GlobalDefReaderCallbacks_SetParadigmPropertyCallback()`

Since

Version 1.5

C.15 *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

<i>OTF2_↔ StringRef</i>	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 <code>name</code> 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.
<i>OTF2_↔ StringRef</i>	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.
<i>OTF2_Io↔ Paradigm↔ Class</i>	ioParadigmClass	The class of this I/O paradigm.
<i>OTF2_Io↔ Paradigm↔ Flag</i>	ioParadigmFlags	Boolean properties of this I/O paradigm.
<i>uint8_t</i>	numberOf↔ Properties	Number of properties.
<i>OTF2_Io↔ Paradigm↔ Property</i>	properties [↔ numberOf↔ Properties]	The property.
<i>OTF2_↔ Type</i>	types [↔ number↔ OfProperties]	The type of this property. Must match with the defined type of the <i>property</i> .
<i>OTF2_↔ Attribute↔ Value</i>	values [↔ numberOf↔ Properties]	The value of this property.

See also

`OTF2_GlobalDefWriter_WriteIoParadigm()`
`OTF2_GlobalDefReaderCallbacks_SetIoParadigmCallback()`

Since

Version 2.1

C.16 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.

C.17 ClockOffset

Attributes

<i>OTF2_↔ Mapping↔ Type</i>	mappingType	Says to what type of ID the mapping table has to be applied.
<i>const OT↔ F2_IdMap*</i>	idMap	Mapping table.

See also

OTF2_DefWriter_WriteMappingTable()
OTF2_DefReaderCallbacks_SetMappingTableCallback()

Since

Version 1.0

C.17 ClockOffset

Clock offsets are used for clock corrections.

This definition is only valid as a local definition.

Attributes

<i>OTF2_↔ Time↔ Stamp</i>	time	Time when this offset was determined.
<i>int64_t</i>	offset	The offset to the global clock which was determined at <i>time</i> .
<i>double</i>	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

C.18 *OTF2_StringRef* String

The string definition.

Attributes

<i>const char*</i>	string	The string, null terminated.
------------------------	--------	------------------------------

See also

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

Since

Version 1.0

C.19 *OTF2_AttributeRef* Attribute

The attribute definition.

Attributes

<i>OTF2_↔ StringRef</i>	name	Name of the attribute. References a String definition.
<i>OTF2_↔ StringRef</i>	description	Description of the attribute. References a String definition. Since version 1.4.
<i>OTF2_↔ Type</i>	type	Type of the attribute value.

See also

[OTF2_GlobalDefWriter_WriteAttribute\(\)](#)
[OTF2_GlobalDefReaderCallbacks_SetAttributeCallback\(\)](#)
[OTF2_DefWriter_WriteAttribute\(\)](#)
[OTF2_DefReaderCallbacks_SetAttributeCallback\(\)](#)

Since

Version 1.0

C.20 *OTF2_SystemTreeNodeRef* SystemTreeNode

The system tree node definition.

Attributes

<i>OTF2_↔ StringRef</i>	name	Free form instance name of this node. References a String definition.
<i>OTF2_↔ StringRef</i>	className	Free form class name of this node References a String definition.
<i>OTF2_↔ System↔ Tree↔ NodeRef</i>	parent	Parent id of this node. May be <i>OTF2_UNDEFINED_SYSTEM_TREE_NODE</i> 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\(\)](#)

C.21 LocationGroup

Since

Version 1.0

C.21 OTF2_LocationGroupRef LocationGroup

The location group definition.

Attributes

<i>OTF2_↔ StringRef</i>	name	Name of the group. References a String definition.
<i>OTF2_↔ Location↔ GroupType</i>	locationGroup↔ Type	Type of this group.
<i>OTF2_↔ System↔ Tree↔ NodeRef</i>	systemTree↔ Parent	Parent of this location group in the system tree. References a SystemTree↔ Node definition.

Supplements

[LocationGroupProperty](#)

See also

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

Since

Version 1.0

C.22 OTF2_LocationRef Location

The location definition.

Attributes

<i>OTF2_↔ StringRef</i>	name	Name of the location References a String definition.
<i>OTF2_↔ Location↔ Type</i>	locationType	Location type.
<i>uint64_t</i>	numberOfEvents	Number of events this location has recorded.
<i>OTF2_↔ Location↔ GroupRef</i>	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

C.23 OTF2_RegionRef Region

The region definition.

Attributes

<i>OTF2_↔ StringRef</i>	name	Name of the region (demangled name if available). References a String definition.
<i>OTF2_↔ StringRef</i>	canonicalName	Alternative name of the region (e.g. mangled name). References a String definition. Since version 1.1.
<i>OTF2_↔ StringRef</i>	description	A more detailed description of this region. References a String definition.
<i>OTF2_↔ Region↔ Role</i>	regionRole	Region role. Since version 1.1.
<i>OTF2_↔ Paradigm</i>	paradigm	Paradigm. Since version 1.1.
<i>OTF2_↔ Region↔ Flag</i>	regionFlags	Region flags. Since version 1.1.
<i>OTF2_↔ StringRef</i>	sourceFile	The source file where this region was declared. References a String definition.
<i>uint32_t</i>	beginLine↔ Number	Starting line number of this region in the source file.
<i>uint32_t</i>	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

C.24 OTF2_CallsiteRef Callsite

The callsite definition.

C.25 Callpath

Attributes

<i>OTF2_↔ StringRef</i>	sourceFile	The source file where this call was made. References a String definition.
<i>uint32_t</i>	lineNumber	Line number in the source file where this call was made.
<i>OTF2_↔ RegionRef</i>	enteredRegion	The region which was called. References a Region definition.
<i>OTF2_↔ RegionRef</i>	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

C.25 *OTF2_CallpathRef* Callpath

The callpath definition.

Attributes

<i>OTF2_↔ Callpath↔ Ref</i>	parent	The parent of this callpath. References a Callpath definition.
<i>OTF2_↔ RegionRef</i>	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

C.26 *OTF2_GroupRef* Group

The group definition.

Attributes

<i>OTF2_↔ StringRef</i>	name	Name of this group References a <i>String</i> definition.
<i>OTF2_↔ GroupType</i>	groupType	The type of this group. Since version 1.2.
<i>OTF2_↔ Paradigm</i>	paradigm	The paradigm of this communication group. Since version 1.2.
<i>OTF2_↔ GroupFlag</i>	groupFlags	Flags for this group. Since version 1.2.
<i>uint32_t</i>	numberOf↔ Members	The number of members in this group.
<i>uint64_t</i>	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

C.27 *OTF2_MetricMemberRef* MetricMember

A metric is defined by a metric member 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.

C.28 Metric

Attributes

<i>OTF2_↔ StringRef</i>	name	Name of the metric. References a <i>String</i> definition.
<i>OTF2_↔ StringRef</i>	description	Description of the metric. References a <i>String</i> definition.
<i>OTF2_↔ MetricType</i>	metricType	Metric type: PAPI, etc.
<i>OTF2_↔ Metric↔ Mode</i>	metricMode	Metric mode: accumulative, fix, relative, etc.
<i>OTF2_↔ Type</i>	valueType	Type of the value. Only <i>OTF2_TYPE_INT64</i> , <i>OTF2_TYPE_UINT64</i> , and <i>OTF2_TYPE_DOUBLE</i> are valid types. If this metric member is recorded in a <i>Metric</i> event, than this type and the type in the event must match.
<i>OTF2_↔ Base</i>	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.
<i>int64_t</i>	exponent	The values inside the Metric events should be scaled by the factor $\text{base}^{\text{exponent}}$, to get the value in its base unit. For example, if the metric values come in as KiBi, than the base should be <i>OTF2_BASE_BINARY</i> 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.
<i>OTF2_↔ StringRef</i>	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 <i>String</i> definition.

See also

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

Since

Version 1.0

C.28 *OTF2_MetricRef* Metric

This is a polymorphic definition class.

Derivations

MetricClass
MetricInstance

C.29 *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

<i>uint8_t</i>	numberOf↔ Metrics	Number of metrics within the set.
<i>OTF2_↔ Metric↔ Member↔ Ref</i>	metricMembers [numberOf↔ Metrics]	List of metric members. References a MetricMember definition.
<i>OTF2_↔ Metric↔ Occurrence</i>	metric↔ Occurrence	Defines occurrence of a metric set.
<i>OTF2_↔ Recorder↔ Kind</i>	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

C.30 *OTF2_MetricRef* MetricInstance

A metric instance 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

<i>OTF2_↔ MetricRef</i>	metricClass	The instanced MetricClass . This metric class must be of kind <i>OTF2_REC↔ ORDER_KIND_ABSTRACT</i> . References a MetricClass , or a MetricInstance definition.
<i>OTF2_↔ Location↔ Ref</i>	recorder	Recorder of the metric: location ID. References a Location definition.
<i>OTF2_↔ Metric↔ Scope</i>	metricScope	Defines type of scope: location, location group, system tree node, or a generic group of locations.
<i>uint64_t</i>	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()

C.31 Comm

Since

Version 1.0

C.31 *OTF2_CommRef* Comm

The communicator definition.

Attributes

<i>OTF2_↔ StringRef</i>	name	The name given by calling <code>MPI_Comm_set_name</code> on this communicator. Or the empty name to indicate that no name was given. References a String definition.
<i>OTF2_↔ GroupRef</i>	group	The describing MPI group of this MPI communicator The group needs to be of type <code>OTF2_GROUP_TYPE_COMM_GROUP</code> or <code>OTF2_GROUP_TYPE_COMM_SELF</code> . References a Group definition.
<i>OTF2_↔ CommRef</i>	parent	The parent MPI communicator from which this communicator was created, if any. Use <code>OTF2_UNDEFINED_COMM</code> to indicate no parent. References a Comm definition.

See also

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

Since

Version 1.0

C.32 *OTF2_ParameterRef* Parameter

The parameter definition.

Attributes

<i>OTF2_↔ StringRef</i>	name	Name of the parameter (variable name etc.) References a String definition.
<i>OTF2_↔ Parameter↔ Type</i>	parameterType	Type of the parameter, <code>OTF2_ParameterType</code> for possible types.

See also

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

Since

Version 1.0

C.33 *OTF2_RmaWinRef* RmaWin

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

C.34 MetricClassRecorder

Attributes

<i>OTF2_↔ StringRef</i>	name	Name, e.g. 'GASPI Queue 1', 'Nvidia Card 2', etc.. References a String definition.
<i>OTF2_↔ CommRef</i>	comm	Communicator object used to create the window. References a Comm definition.

See also

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

Since

Version 1.2

C.34 MetricClassRecorder

The metric class recorder definition.

Attributes

<i>OTF2_↔ MetricRef</i>	metric	Parent MetricClass , or MetricInstance definition to which this one is a supplementary definition. References a MetricClass , or a MetricInstance definition.
<i>OTF2_↔ Location↔ Ref</i>	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

C.35 SystemTreeNodeProperty

An arbitrary key/value property for a [SystemTreeNode](#) definition.

Attributes

<i>OTF2_↔ System↔ Tree↔ NodeRef</i>	systemTreeNode	Parent SystemTreeNode definition to which this one is a supplementary definition. References a SystemTreeNode definition.
---	----------------	---

<i>OTF2_↔ StringRef</i>	name	Name of the property. References a <i>String</i> definition.
<i>OTF2_↔ Type</i>	type	The type of this property. Since version 2.0.
<i>OTF2_↔ Attribute↔ Value</i>	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

C.36 SystemTreeNodeDomain

The system tree node domain definition.

Attributes

<i>OTF2_↔ System↔ Tree↔ NodeRef</i>	systemTreeNode	Parent <i>SystemTreeNode</i> definition to which this one is a supplementary definition. References a <i>SystemTreeNode</i> definition.
<i>OTF2_↔ System↔ Tree↔ Domain</i>	systemTree↔ Domain	The domain in which the referenced <i>SystemTreeNode</i> operates in.

See also

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

Since

Version 1.2

C.37 LocationGroupProperty

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

C.38 LocationProperty

Attributes

<i>OTF2_↔ Location↔ GroupRef</i>	locationGroup	Parent LocationGroup definition to which this one is a supplementary definition. References a LocationGroup definition.
<i>OTF2_↔ StringRef</i>	name	Name of the property. References a String definition.
<i>OTF2_↔ Type</i>	type	The type of this property. Since version 2.0.
<i>OTF2_↔ Attribute↔ Value</i>	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

C.38 LocationProperty

An arbitrary key/value property for a [Location](#) definition.

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	Parent Location definition to which this one is a supplementary definition. References a Location definition.
<i>OTF2_↔ StringRef</i>	name	Name of the property. References a String definition.
<i>OTF2_↔ Type</i>	type	The type of this property. Since version 2.0.
<i>OTF2_↔ Attribute↔ Value</i>	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

C.39 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

<i>OTF2_↔ StringRef</i>	name	The name of the cartesian topology dimension. References a String definition.
<i>uint32_t</i>	size	The size of the cartesian topology dimension.
<i>OTF2_↔ Cart↔ Periodicity</i>	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

C.40 *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

<i>OTF2_↔ StringRef</i>	name	The name of the topology. References a String definition.
<i>OTF2_↔ CommRef</i>	communicator	Communicator object used to create the topology. References a Comm definition.
<i>uint8_t</i>	numberOf↔ Dimensions	Number of dimensions.
<i>OTF2_↔ Cart↔ Dimension↔ Ref</i>	cartDimensions [↔ numberOf↔ Dimensions ↔]	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

C.41 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

<i>OTF2_↔ Cart_↔ Topology_↔ Ref</i>	cartTopology	Parent <i>CartTopology</i> definition to which this one is a supplementary definition. References a <i>CartTopology</i> definition.
<i>uint32_t</i>	rank	The rank w.r.t. the communicator associated to the topology referencing this coordinate.
<i>uint8_t</i>	numberOf↔ Dimensions	Number of dimensions.
<i>uint32_t</i>	coordinates [↔ numberOf↔ Dimensions]	Coordinates, indexed by dimension.

See also

[OTF2_GlobalDefWriter_WriteCartCoordinate\(\)](#)
[OTF2_GlobalDefReaderCallbacks_SetCartCoordinateCallback\(\)](#)
[OTF2_DefWriter_WriteCartCoordinate\(\)](#)
[OTF2_DefReaderCallbacks_SetCartCoordinateCallback\(\)](#)

Since

Version 1.3

C.42 *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 a *Attribute* definition named "SOURCE_CODE_LOCATION" and typed *OTF2_TYPE_SOURCE_CODE_LOCATION*.

Attributes

<i>OTF2_↔ StringRef</i>	file	The name of the file for the source code location. References a <i>String</i> definition.
<i>uint32_t</i>	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

C.43 *OTF2_CallingContextRef* CallingContext

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

C.44 CallingContextProperty

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 [CallingContext↔Leave](#) 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↔Source↔Code↔Location↔Ref	sourceCode↔Location	The absolute source code location of this calling context. References a SourceCodeLocation definition.
OTF2↔Calling↔ContextRef	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

C.44 CallingContextProperty

An arbitrary key/value property for a [CallingContext](#) definition.

Attributes

<i>OTF2_↔ Calling↔ ContextRef</i>	callingContext	Parent <i>CallingContext</i> definition to which this one is a supplementary definition. References a <i>CallingContext</i> definition.
<i>OTF2_↔ StringRef</i>	name	Property name. References a <i>String</i> definition.
<i>OTF2_↔ Type</i>	type	The type of this property. Must match with the defined type of the <i>property</i> .
<i>OTF2_↔ Attribute↔ Value</i>	value	The value of this property.

See also

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

Since

Version 2.0

C.45 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_INTERRUPT_↔
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:

$$\text{base-period} = \text{period} \times \text{base}^{\text{exponent}}$$

Attributes

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

See also

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


C.46 *ioFileProperty*

Since

Version 1.5

C.46 *ioFileProperty*

Extensible annotation for the polymorphic *ioFile* definition class.

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

Attributes

<i>OTF2_</i> ↔ <i>FileRef</i>	<i>ioFile</i>	Parent <i>ioRegularFile</i> definition to which this one is a supplementary definition. References a <i>ioRegularFile</i> definition.
<i>OTF2_</i> ↔ <i>StringRef</i>	<i>name</i>	Property name. References a <i>String</i> definition.
<i>OTF2_</i> ↔ <i>Type</i>	<i>type</i>	The type of this property.
<i>OTF2_</i> ↔ <i>Attribute</i> ↔ <i>Value</i>	<i>value</i>	The value of this property.

See also

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

Since

Version 2.1

C.47 *OTF2_*↔*ioFileRef* *ioFile*

This is a polymorphic definition class.

Derivations

ioRegularFile
ioDirectory

C.48 *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

<i>OTF2_↔ StringRef</i>	name	Name of the file. References a <i>String</i> definition.
<i>OTF2_↔ System↔ Tree↔ NodeRef</i>	scope	Defines the physical scope of this <i>IoRegularFile</i> in the system tree. E.g., two <i>IoRegularFile</i> definitions with the same name but different <code>scope</code> values are physically different, thus I/O operations through <i>IoHandles</i> do not operate on the same file. References a <i>SystemTreeNode</i> definition.

Supplements

IoFileProperty

See also

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

Since

Version 2.1

C.49 *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

<i>OTF2_↔ StringRef</i>	name	Name of the directory. References a <i>String</i> definition.
<i>OTF2_↔ System↔ Tree↔ NodeRef</i>	scope	Defines the physical scope of this <i>IoDirectory</i> in the system tree. E.g., two <i>IoDirectory</i> definitions with the same name but different <code>scope</code> values are physically different, thus I/O operations through <i>IoHandles</i> do not operate on the same directory. References a <i>SystemTreeNode</i> definition.

See also

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

Since

Version 2.1

C.50 *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_CREATE↔ED* flag, after it was referenced in an *IoCreateHandle* event, or it was referenced in the *newHandle* attribute of an

C.51 IoPreCreatedHandleState

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 *Locations* of a *LocationGroup* have access to an *active IoHandle*, regardless which *Location* of the *LocationGroup* activated the *IoHandle*.

Attributes

<i>OTF2_↔StringRef</i>	name	Handle name. References a <i>String</i> definition.
<i>OTF2_↔FileRef</i>	file	File identifier. References a <i>IoRegularFile</i> , or a <i>IoDirectory</i> definition.
<i>OTF2_↔Paradigm↔Ref</i>	ioParadigm	The I/O paradigm. References a <i>IoParadigm</i> definition.
<i>OTF2_↔Handle↔Flag</i>	ioHandleFlags	Special characteristics of this handle.
<i>OTF2_↔CommRef</i>	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 <i>Comm</i> definition.
<i>OTF2_↔HandleRef</i>	parent	Parent, in case this I/O handle was created and operated by an higher- level I/O paradigm. References a <i>IoHandle</i> definition.

Supplements

[IoPreCreatedHandleState](#)

See also

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

Since

Version 2.1

C.51 IoPreCreatedHandleState

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

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

Attributes

<i>OTF2_↔HandleRef</i>	ioHandle	Parent <i>IoHandle</i> definition to which this one is a supplementary definition. References a <i>IoHandle</i> definition.
<i>OTF2_↔Access↔Mode</i>	mode	The access mode of the <i>pre-created IoHandle</i> .

<i>OTF2_</i> ↔ <i>StatusFlag</i>	statusFlags	The status flags of the <i>pre-created loHandle</i> .
-------------------------------------	-------------	---

See also

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

Since

Version 2.1

C.52 CallpathParameter

A parameter for a callpath definition.

Attributes

<i>OTF2_</i> ↔ <i>Callpath</i> ↔ <i>Ref</i>	callpath	Parent <i>Callpath</i> definition to which this one is a supplementary definition. References a <i>Callpath</i> definition.
<i>OTF2_</i> ↔ <i>Parameter</i> ↔ <i>Ref</i>	parameter	The parameter of this callpath. References a <i>Parameter</i> definition.
<i>OTF2_</i> ↔ <i>Type</i>	type	The type of the attribute value. Must match the type of the parameter.
<i>OTF2_</i> ↔ <i>Attribute</i> ↔ <i>Value</i>	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

C.53 List of all event records

C.54 BufferFlush

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

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	stopTime	The time the buffer flush finished.

See also

OTF2_EvtWriter_BufferFlush()
 OTF2_GlobalEvtReaderCallbacks_SetBufferFlushCallback()
 OTF2_EvtReaderCallbacks_SetBufferFlushCallback()

Since

Version 1.0

C.55 MeasurementOnOff

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

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>Measurement↔</i> <i>Mode</i>	measurement↔ Mode	Is the measurement turned on (<i>OTF2_MEASUREMENT_ON</i>) or off (<i>OTF2_↔_MEASUREMENT_OFF</i>)?

See also

OTF2_EvtWriter_MeasurementOnOff()
 OTF2_GlobalEvtReaderCallbacks_SetMeasurementOnOffCallback()
 OTF2_EvtReaderCallbacks_SetMeasurementOnOffCallback()

Since

Version 1.0

C.56 Enter

An enter record indicates that the program enters a code region.

C.57 Leave

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ RegionRef</i>	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 <i>OTF2_MA↔PPING_REGION</i> is available.

See also

OTF2_EvtWriter_Enter()
OTF2_GlobalEvtReaderCallbacks_SetEnterCallback()
OTF2_EvtReaderCallbacks_SetEnterCallback()

Since

Version 1.0

C.57 Leave

A leave record indicates that the program leaves a code region.

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ RegionRef</i>	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 <i>OTF2_MA↔PPING_REGION</i> is available.

See also

OTF2_EvtWriter_Leave()
OTF2_GlobalEvtReaderCallbacks_SetLeaveCallback()
OTF2_EvtReaderCallbacks_SetLeaveCallback()

Since

Version 1.0

C.58 MpiSend

A MpiSend record indicates that a MPI message send process 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

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>uint32_t</i>	receiver	MPI rank of receiver in <code>communicator</code> .
<i>OTF2_↔</i> <i>CommRef</i>	communicator	Communicator ID. References a Comm definition and will be mapped to the global definition if a mapping table of type <code>OTF2_MAPPING_COMM</code> is available.
<i>uint32_t</i>	msgTag	Message tag
<i>uint64_t</i>	msgLength	Message length

See also

[OTF2_EvtWriter_MpiSend\(\)](#)
[OTF2_GlobalEvtReaderCallbacks_SetMpiSendCallback\(\)](#)
[OTF2_EvtReaderCallbacks_SetMpiSendCallback\(\)](#)

Since

Version 1.0

C.59 Mpisend

A `Mpisend` record indicates that a MPI message send process 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

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>uint32_t</i>	receiver	MPI rank of receiver in <code>communicator</code> .
<i>OTF2_↔</i> <i>CommRef</i>	communicator	Communicator ID. References a Comm definition and will be mapped to the global definition if a mapping table of type <code>OTF2_MAPPING_COMM</code> is available.
<i>uint32_t</i>	msgTag	Message tag
<i>uint64_t</i>	msgLength	Message length
<i>uint64_t</i>	requestID	ID of the related request

See also

[OTF2_EvtWriter_Mpisend\(\)](#)
[OTF2_GlobalEvtReaderCallbacks_SetMpisendCallback\(\)](#)
[OTF2_EvtReaderCallbacks_SetMpisendCallback\(\)](#)

Since

Version 1.0

C.60 MpilsendComplete

Signals the completion of non-blocking send request.

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>uint64_t</i>	requestID	ID of the related request

See also

OTF2_EvtWriter_MpilsendComplete()
 OTF2_GlobalEvtReaderCallbacks_SetMpilsendCompleteCallback()
 OTF2_EvtReaderCallbacks_SetMpilsendCompleteCallback()

Since

Version 1.0

C.61 MpilrecvRequest

Signals the request of a receive, which can be completed later.

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>uint64_t</i>	requestID	ID of the requested receive

See also

OTF2_EvtWriter_MpilrecvRequest()
 OTF2_GlobalEvtReaderCallbacks_SetMpilrecvRequestCallback()
 OTF2_EvtReaderCallbacks_SetMpilrecvRequestCallback()

Since

Version 1.0

C.62 MpiRecv

A MpiRecv record indicates that a 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).

C.63 MpiRecv

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>uint32_t</i>	sender	MPI rank of sender in <code>communicator</code> .
<i>OTF2_↔</i> <i>CommRef</i>	communicator	Communicator ID. References a <i>Comm</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_COMM</i> is available.
<i>uint32_t</i>	msgTag	Message tag
<i>uint64_t</i>	msgLength	Message length

See also

`OTF2_EvtWriter_MpiRecv()`
`OTF2_GlobalEvtReaderCallbacks_SetMpiRecvCallback()`
`OTF2_EvtReaderCallbacks_SetMpiRecvCallback()`

Since

Version 1.0

C.63 MpiRecv

A `MpiRecv` record indicates that a MPI message was received (`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

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>uint32_t</i>	sender	MPI rank of sender in <code>communicator</code> .
<i>OTF2_↔</i> <i>CommRef</i>	communicator	Communicator ID. References a <i>Comm</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_COMM</i> is available.
<i>uint32_t</i>	msgTag	Message tag
<i>uint64_t</i>	msgLength	Message length
<i>uint64_t</i>	requestID	ID of the related request

See also

`OTF2_EvtWriter_MpiRecv()`
`OTF2_GlobalEvtReaderCallbacks_SetMpiRecvCallback()`
`OTF2_EvtReaderCallbacks_SetMpiRecvCallback()`

Since

Version 1.0

C.64 `MpiRequestTest`

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

C.65 `MpiRequestCancelled`

Attributes

<code>OTF2_↔ Location↔ Ref</code>	location	The location where this event happened.
<code>OTF2_↔ Time↔ Stamp</code>	timestamp	The time when this event happened.
<code>uint64_t</code>	requestID	ID of the related request

See also

`OTF2_EvtWriter_MpiRequestTest()`
`OTF2_GlobalEvtReaderCallbacks_SetMpiRequestTestCallback()`
`OTF2_EvtReaderCallbacks_SetMpiRequestTestCallback()`

Since

Version 1.0

C.65 `MpiRequestCancelled`

This events appears if the program canceled a request.

Attributes

<code>OTF2_↔ Location↔ Ref</code>	location	The location where this event happened.
<code>OTF2_↔ Time↔ Stamp</code>	timestamp	The time when this event happened.
<code>uint64_t</code>	requestID	ID of the related request

See also

`OTF2_EvtWriter_MpiRequestCancelled()`
`OTF2_GlobalEvtReaderCallbacks_SetMpiRequestCancelledCallback()`
`OTF2_EvtReaderCallbacks_SetMpiRequestCancelledCallback()`

Since

Version 1.0

C.66 `MpiCollectiveBegin`

A `MpiCollectiveBegin` record marks the begin of a MPI collective operation (`MPI_GATHER`, `MPI_SCATTER` etc.).

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.

See also

OTF2_EvtWriter_MpiCollectiveBegin()
 OTF2_GlobalEvtReaderCallbacks_SetMpiCollectiveBeginCallback()
 OTF2_EvtReaderCallbacks_SetMpiCollectiveBeginCallback()

Since

Version 1.0

C.67 MpiCollectiveEnd

A `MpiCollectiveEnd` record marks the end of a 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

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ Collective↔ Op</i>	collectiveOp	Determines which collective operation it is.
<i>OTF2_↔ CommRef</i>	communicator	Communicator References a Comm definition and will be mapped to the global definition if a mapping table of type <code>OTF2_MAPPING_COMM</code> is available.
<i>uint32_t</i>	root	MPI rank of root in <code>communicator</code> or <code>OTF2_UNDEFINED_UINT32</code> if the call has no root rank.
<i>uint64_t</i>	sizeSent	Size of the sent message.
<i>uint64_t</i>	sizeReceived	Size of the received message.

See also

OTF2_EvtWriter_MpiCollectiveEnd()
 OTF2_GlobalEvtReaderCallbacks_SetMpiCollectiveEndCallback()
 OTF2_EvtReaderCallbacks_SetMpiCollectiveEndCallback()

Since

Version 1.0

C.68 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.

C.69 OmpJoin

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>uint32_t</i>	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

C.69 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

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	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

C.70 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 [Thread↔AcquireLock](#) event record is in use.

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>uint32_t</i>	lockID	ID of the lock.
<i>uint32_t</i>	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

C.71 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 [Thread↔ReleaseLock](#) event record is in use.

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>uint32_t</i>	lockID	ID of the lock.
<i>uint32_t</i>	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

C.72 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

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>uint64_t</i>	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

C.73 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

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>uint64_t</i>	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

C.74 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

C.75 Metric

A metric event is always stored at the location that recorded the metric. A metric event can reference a metric class or metric instance. Therefore, metric classes and instances share same ID space. Synchronous metrics are always located right before the according 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 <code>OTF2_MAPPING_METRIC</code> 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.

C.76 ParameterString

<i>OTF2_↔ Metric↔ Value</i>	metricValues [numberOf↔ Metrics]	List of metric values.
-------------------------------------	---	------------------------

See also

OTF2_EvtWriter_Metric()
OTF2_GlobalEvtReaderCallbacks_SetMetricCallback()
OTF2_EvtReaderCallbacks_SetMetricCallback()

Since

Version 1.0

C.76 ParameterString

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

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ Parameter↔ Ref</i>	parameter	Parameter ID. References a Parameter definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_PARAMETER</i> is available.
<i>OTF2_↔ StringRef</i>	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 <i>OTF2_MAPPING_↔ _STRING</i> is available.

See also

OTF2_EvtWriter_ParameterString()
OTF2_GlobalEvtReaderCallbacks_SetParameterStringCallback()
OTF2_EvtReaderCallbacks_SetParameterStringCallback()

Since

Version 1.0

C.77 ParameterInt

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

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>Parameter↔</i> <i>Ref</i>	parameter	Parameter ID. References a Parameter definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_PARAMETER</i> is available.
<i>int64_t</i>	value	Value of the recorded parameter.

See also

OTF2_EvtWriter_ParameterInt()
 OTF2_GlobalEvtReaderCallbacks_SetParameterIntCallback()
 OTF2_EvtReaderCallbacks_SetParameterIntCallback()

Since

Version 1.0

C.78 ParameterUnsignedInt

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

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>Parameter↔</i> <i>Ref</i>	parameter	Parameter ID. References a Parameter definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_PARAMETER</i> is available.
<i>uint64_t</i>	value	Value of the recorded parameter.

See also

OTF2_EvtWriter_ParameterUnsignedInt()
 OTF2_GlobalEvtReaderCallbacks_SetParameterUnsignedIntCallback()
 OTF2_EvtReaderCallbacks_SetParameterUnsignedIntCallback()

Since

Version 1.0

C.79 RmaWinCreate

A `RmaWinCreate` record denotes the creation of a RMA window.

C.80 RmaWinDestroy

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>RmaWin↔</i> <i>Ref</i>	win	ID of the window created. References a <i>RmaWin</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_RMA_WIN</i> is available.

See also

`OTF2_EvtWriter_RmaWinCreate()`
`OTF2_GlobalEvtReaderCallbacks_SetRmaWinCreateCallback()`
`OTF2_EvtReaderCallbacks_SetRmaWinCreateCallback()`

Since

Version 1.2

C.80 RmaWinDestroy

A *RmaWinDestroy* record denotes the destruction of a RMA window.

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>RmaWin↔</i> <i>Ref</i>	win	ID of the window destroyed. References a <i>RmaWin</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_↔_RMA_WIN</i> is available.

See also

`OTF2_EvtWriter_RmaWinDestroy()`
`OTF2_GlobalEvtReaderCallbacks_SetRmaWinDestroyCallback()`
`OTF2_EvtReaderCallbacks_SetRmaWinDestroyCallback()`

Since

Version 1.2

C.81 RmaCollectiveBegin

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

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.

See also

[OTF2_EvtWriter_RmaCollectiveBegin\(\)](#)
[OTF2_GlobalEvtReaderCallbacks_SetRmaCollectiveBeginCallback\(\)](#)
[OTF2_EvtReaderCallbacks_SetRmaCollectiveBeginCallback\(\)](#)

Since

Version 1.2

C.82 RmaCollectiveEnd

A RmaCollectiveEnd record denotes the end of a collective RMA operation.

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ Collective↔ Op</i>	collectiveOp	Determines which collective operation it is.
<i>OTF2_↔ Rma↔ SyncLevel</i>	syncLevel	Synchronization level of this collective operation.
<i>OTF2_↔ RmaWin↔ Ref</i>	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 <i>OTF2_MA↔PPING_RMA_WIN</i> is available.
<i>uint32_t</i>	root	Root process for this operation or <i>OTF2_UNDEFINED_UINT32</i> if the call has no root rank.
<i>uint64_t</i>	bytesSent	Bytes sent in operation.
<i>uint64_t</i>	bytesReceived	Bytes receives in operation.

See also

[OTF2_EvtWriter_RmaCollectiveEnd\(\)](#)
[OTF2_GlobalEvtReaderCallbacks_SetRmaCollectiveEndCallback\(\)](#)
[OTF2_EvtReaderCallbacks_SetRmaCollectiveEndCallback\(\)](#)

Since

Version 1.2

C.83 RmaGroupSync

A RmaGroupSync record denotes the synchronization with a subgroup of processes on a window.

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ Rma↔ SyncLevel</i>	syncLevel	Synchronization level of this collective operation.
<i>OTF2_↔ RmaWin↔ Ref</i>	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 <i>OTF2_MA↔PPING_RMA_WIN</i> is available.
<i>OTF2_↔ GroupRef</i>	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 <i>OTF2_MAPPING_GROUP</i> is available.

See also

[OTF2_EvtWriter_RmaGroupSync\(\)](#)
[OTF2_GlobalEvtReaderCallbacks_SetRmaGroupSyncCallback\(\)](#)
[OTF2_EvtReaderCallbacks_SetRmaGroupSyncCallback\(\)](#)

Since

Version 1.2

C.84 RmaRequestLock

A `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

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ RmaWin↔ Ref</i>	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 <i>OTF2_MA↔PPING_RMA_WIN</i> is available.
<i>uint32_t</i>	remote	Rank of the locked remote process or <i>OTF2_UNDEFINED_UINT32</i> if all processes of the specified window are locked.
<i>uint64_t</i>	lockId	ID of the lock acquired, if multiple locks are defined on a window.
<i>OTF2_↔ LockType</i>	lockType	Type of lock acquired.

See also

[OTF2_EvtWriter_RmaRequestLock\(\)](#)
[OTF2_GlobalEvtReaderCallbacks_SetRmaRequestLockCallback\(\)](#)
[OTF2_EvtReaderCallbacks_SetRmaRequestLockCallback\(\)](#)

C.85 RmaAcquireLock

Since

Version 1.2

C.85 RmaAcquireLock

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

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>RmaWin↔</i> <i>Ref</i>	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 <i>OTF2_MA↔PPING_RMA_WIN</i> is available.
<i>uint32_t</i>	remote	Rank of the locked remote process or <i>OTF2_UNDEFINED_UINT32</i> if all processes of the specified window are locked.
<i>uint64_t</i>	lockId	ID of the lock acquired, if multiple locks are defined on a window.
<i>OTF2_↔</i> <i>LockType</i>	lockType	Type of lock acquired.

See also

`OTF2_EvtWriter_RmaAcquireLock()`
`OTF2_GlobalEvtReaderCallbacks_SetRmaAcquireLockCallback()`
`OTF2_EvtReaderCallbacks_SetRmaAcquireLockCallback()`

Since

Version 1.2

C.86 RmaTryLock

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

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>RmaWin↔</i> <i>Ref</i>	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 <i>OTF2_MA↔PPING_RMA_WIN</i> is available.

<i>uint32_t</i>	remote	Rank of the locked remote process or <i>OTF2_UNDEFINED_UINT32</i> if all processes of the specified window are locked.
<i>uint64_t</i>	lockId	ID of the lock acquired, if multiple locks are defined on a window.
<i>OTF2_↔ LockType</i>	lockType	Type of lock acquired.

See also

OTF2_EvtWriter_RmaTryLock()
 OTF2_GlobalEvtReaderCallbacks_SetRmaTryLockCallback()
 OTF2_EvtReaderCallbacks_SetRmaTryLockCallback()

Since

Version 1.2

C.87 RmaReleaseLock

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

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ RmaWin↔ Ref</i>	win	ID of the window used for this operation. References a <i>RmaWin</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MA↔ PPING_RMA_WIN</i> is available.
<i>uint32_t</i>	remote	Rank of the locked remote process or <i>OTF2_UNDEFINED_UINT32</i> if all processes of the specified window are locked.
<i>uint64_t</i>	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

C.88 RmaSync

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

C.89 RmaWaitChange

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>RmaWin↔</i> <i>Ref</i>	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 <i>OTF2_MA↔PPING_RMA_WIN</i> is available.
<i>uint32_t</i>	remote	Rank of the locked remote process.
<i>OTF2_↔</i> <i>Rma↔</i> <i>SyncType</i>	syncType	Type of synchronization.

See also

`OTF2_EvtWriter_RmaSync()`
`OTF2_GlobalEvtReaderCallbacks_SetRmaSyncCallback()`
`OTF2_EvtReaderCallbacks_SetRmaSyncCallback()`

Since

Version 1.2

C.89 RmaWaitChange

A `RmaWaitChange` record denotes the change of a window that was waited for.

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>RmaWin↔</i> <i>Ref</i>	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 <i>OTF2_MA↔PPING_RMA_WIN</i> is available.

See also

`OTF2_EvtWriter_RmaWaitChange()`
`OTF2_GlobalEvtReaderCallbacks_SetRmaWaitChangeCallback()`
`OTF2_EvtReaderCallbacks_SetRmaWaitChangeCallback()`

Since

Version 1.2

C.90 RmaPut

A `RmaPut` record denotes the time a put operation was issued.

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>RmaWin↔</i> <i>Ref</i>	win	ID of the window used for this operation. References a <i>RmaWin</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MA↔</i> <i>PPING_RMA_WIN</i> is available.
<i>uint32_t</i>	remote	Rank of the target process.
<i>uint64_t</i>	bytes	Bytes sent to target.
<i>uint64_t</i>	matchingId	ID used for matching the corresponding completion record.

See also

OTF2_EvtWriter_RmaPut()
 OTF2_GlobalEvtReaderCallbacks_SetRmaPutCallback()
 OTF2_EvtReaderCallbacks_SetRmaPutCallback()

Since

Version 1.2

C.91 RmaGet

A RmaGet record denotes the time a get operation was issued.

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>RmaWin↔</i> <i>Ref</i>	win	ID of the window used for this operation. References a <i>RmaWin</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MA↔</i> <i>PPING_RMA_WIN</i> is available.
<i>uint32_t</i>	remote	Rank of the target process.
<i>uint64_t</i>	bytes	Bytes received from target.
<i>uint64_t</i>	matchingId	ID used for matching the corresponding completion record.

See also

OTF2_EvtWriter_RmaGet()
 OTF2_GlobalEvtReaderCallbacks_SetRmaGetCallback()
 OTF2_EvtReaderCallbacks_SetRmaGetCallback()

Since

Version 1.2

C.92 RmaAtomic

A RmaAtomic record denotes the time a atomic operation was issued.

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>RmaWin↔</i> <i>Ref</i>	win	ID of the window used for this operation. References a <i>RmaWin</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MA↔PPING_RMA_WIN</i> is available.
<i>uint32_t</i>	remote	Rank of the target process.
<i>OTF2_↔</i> <i>Rma↔</i> <i>Atomic↔</i> <i>Type</i>	type	Type of atomic operation.
<i>uint64_t</i>	bytesSent	Bytes sent to target.
<i>uint64_t</i>	bytesReceived	Bytes received from target.
<i>uint64_t</i>	matchingId	ID used for matching the corresponding completion record.

See also

OTF2_EvtWriter_RmaAtomic()
 OTF2_GlobalEvtReaderCallbacks_SetRmaAtomicCallback()
 OTF2_EvtReaderCallbacks_SetRmaAtomicCallback()

Since

Version 1.2

C.93 RmaOpCompleteBlocking

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

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>RmaWin↔</i> <i>Ref</i>	win	ID of the window used for this operation. References a <i>RmaWin</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MA↔PPING_RMA_WIN</i> is available.
<i>uint64_t</i>	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

C.94 RmaOpCompleteNonBlocking

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

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ RmaWin↔ Ref</i>	win	ID of the window used for this operation. References a <i>RmaWin</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MA↔ PPING_RMA_WIN</i> is available.
<i>uint64_t</i>	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

C.95 RmaOpTest

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

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ RmaWin↔ Ref</i>	win	ID of the window used for this operation. References a <i>RmaWin</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MA↔ PPING_RMA_WIN</i> is available.
<i>uint64_t</i>	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

C.96 RmaOpCompleteRemote

A *RmaOpCompleteRemote* record denotes the remote completion of a RMA operation.

C.97 ThreadFork

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>RmaWin↔</i> <i>Ref</i>	win	ID of the window used for this operation. References a <i>RmaWin</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MA↔</i> <i>PPING_RMA_WIN</i> is available.
<i>uint64_t</i>	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

C.97 ThreadFork

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

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>Paradigm</i>	model	The threading paradigm this event took place.
<i>uint32_t</i>	<i>numberOf↔</i> <i>Requested↔</i> <i>Threads</i>	Requested size of the team.

See also

`OTF2_EvtWriter_ThreadFork()`
`OTF2_GlobalEvtReaderCallbacks_SetThreadForkCallback()`
`OTF2_EvtReaderCallbacks_SetThreadForkCallback()`

Since

Version 1.2

C.98 ThreadJoin

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

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ Paradigm</i>	model	The threading paradigm this event took place.

See also

OTF2_EvtWriter_ThreadJoin()
 OTF2_GlobalEvtReaderCallbacks_SetThreadJoinCallback()
 OTF2_EvtReaderCallbacks_SetThreadJoinCallback()

Since

Version 1.2

C.99 ThreadTeamBegin

The current location enters the specified thread team.

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ CommRef</i>	threadTeam	Thread team References a <i>Comm</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_COMM</i> is available.

See also

OTF2_EvtWriter_ThreadTeamBegin()
 OTF2_GlobalEvtReaderCallbacks_SetThreadTeamBeginCallback()
 OTF2_EvtReaderCallbacks_SetThreadTeamBeginCallback()

Since

Version 1.2

C.100 ThreadTeamEnd

The current location leaves the specified thread team.

C.101 ThreadAcquireLock

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ CommRef</i>	threadTeam	Thread team References a <i>Comm</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_COMM</i> is available.

See also

OTF2_EvtWriter_ThreadTeamEnd()
OTF2_GlobalEvtReaderCallbacks_SetThreadTeamEndCallback()
OTF2_EvtReaderCallbacks_SetThreadTeamEndCallback()

Since

Version 1.2

C.101 ThreadAcquireLock

A ThreadAcquireLock record marks that a thread acquires a lock.

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ Paradigm</i>	model	The threading paradigm this event took place.
<i>uint32_t</i>	lockID	ID of the lock.
<i>uint32_t</i>	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

C.102 ThreadReleaseLock

A ThreadReleaseLock record marks that a thread releases a lock.

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ Paradigm</i>	model	The threading paradigm this event took place.
<i>uint32_t</i>	lockID	ID of the lock.
<i>uint32_t</i>	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

C.103 ThreadTaskCreate

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

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ CommRef</i>	threadTeam	Thread team References a <i>Comm</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_COMM</i> is available.
<i>uint32_t</i>	creatingThread	Creating thread of this task.
<i>uint32_t</i>	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

C.104 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

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ CommRef</i>	threadTeam	Thread team References a <i>Comm</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_COMM</i> is available.
<i>uint32_t</i>	creatingThread	Creating thread of this task.
<i>uint32_t</i>	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

C.105 ThreadTaskComplete

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

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ CommRef</i>	threadTeam	Thread team References a <i>Comm</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_COMM</i> is available.
<i>uint32_t</i>	creatingThread	Creating thread of this task.
<i>uint32_t</i>	generation↔ Number	Thread-private generation number of task's creating thread.

See also

OTF2_EvtWriter_ThreadTaskComplete()
 OTF2_GlobalEvtReaderCallbacks_SetThreadTaskCompleteCallback()
 OTF2_EvtReaderCallbacks_SetThreadTaskCompleteCallback()

Since

Version 1.2

C.106 ThreadCreate

The location created successfully a new thread.

C.107 ThreadBegin

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>CommRef</i>	thread↔ Contingent	The thread contingent. References a <i>Comm</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_COMM</i> is available.
<i>uint64_t</i>	sequenceCount	A <i>threadContingent</i> unique number. The corresponding <i>ThreadBegin</i> event does have the same number.

See also

`OTF2_EvtWriter_ThreadCreate()`
`OTF2_GlobalEvtReaderCallbacks_SetThreadCreateCallback()`
`OTF2_EvtReaderCallbacks_SetThreadCreateCallback()`

Since

Version 1.3

C.107 ThreadBegin

Marks the begin of a thread created by another thread.

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>CommRef</i>	thread↔ Contingent	The thread contingent. References a <i>Comm</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_COMM</i> is available.
<i>uint64_t</i>	sequenceCount	A <i>threadContingent</i> unique number. The corresponding <i>ThreadCreate</i> event does have the same number.

See also

`OTF2_EvtWriter_ThreadBegin()`
`OTF2_GlobalEvtReaderCallbacks_SetThreadBeginCallback()`
`OTF2_EvtReaderCallbacks_SetThreadBeginCallback()`

Since

Version 1.3

C.108 ThreadWait

The location waits for the completion of another thread.

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ CommRef</i>	thread↔ Contingent	The thread contingent. References a Comm definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_COMM</i> is available.
<i>uint64_t</i>	sequenceCount	A <code>threadContingent</code> 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

C.109 ThreadEnd

Marks the end of a thread.

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ CommRef</i>	thread↔ Contingent	The thread contingent. References a Comm definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_COMM</i> is available.
<i>uint64_t</i>	sequenceCount	A <code>threadContingent</code> unique number. The corresponding ThreadWait event does have the same number. <i>OTF2_UNDEFINED_UINT64</i> in case no corresponding ThreadWait event exists.

See also

[OTF2_EvtWriter_ThreadEnd\(\)](#)
[OTF2_GlobalEvtReaderCallbacks_SetThreadEndCallback\(\)](#)
[OTF2_EvtReaderCallbacks_SetThreadEndCallback\(\)](#)

Since

Version 1.3

C.110 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

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ Calling↔ ContextRef</i>	callingContext	The entered region as referenced by the <i>CallingContext</i> definition. References a <i>CallingContext</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_CALLING_CONTEXT</i> is available.
<i>uint32_t</i>	unwindDistance	The unwindDistance for this <i>callingContext</i> . See the description in <i>CallingContext</i> .

See also

OTF2_EvtWriter_CallingContextEnter()
 OTF2_GlobalEvtReaderCallbacks_SetCallingContextEnterCallback()
 OTF2_EvtReaderCallbacks_SetCallingContextEnterCallback()

Since

Version 2.0

C.111 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

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
-------------------------------------	----------	---

<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ Calling↔ ContextRef</i>	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 <code>OTF2_MAPPING_CALLING_CONTEXT</code> is available.

See also

[OTF2_EvtWriter_CallingContextLeave\(\)](#)
[OTF2_GlobalEvtReaderCallbacks_SetCallingContextLeaveCallback\(\)](#)
[OTF2_EvtReaderCallbacks_SetCallingContextLeaveCallback\(\)](#)

Since

Version 2.0

C.112 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

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ Calling↔ ContextRef</i>	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 <code>OTF2_MAPPING_CALLING_CONTEXT</code> is available.
<i>uint32_t</i>	unwindDistance	The unwindDistance for this <code>callingContext</code> . See the description in CallingContext .
<i>OTF2_↔ Interrupt↔ Generator↔ Ref</i>	interrupt↔ Generator	References a InterruptGenerator definition and will be mapped to the global definition if a mapping table of type <code>OTF2_MAPPING_INTERRUPT_GENE↔ RATOR</code> is available.

See also

[OTF2_EvtWriter_CallingContextSample\(\)](#)
[OTF2_GlobalEvtReaderCallbacks_SetCallingContextSampleCallback\(\)](#)
[OTF2_EvtReaderCallbacks_SetCallingContextSampleCallback\(\)](#)

Since

Version 1.5

C.113 IoCreateHandle

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

C.114 IoDestroyHandle

An *IoHandle* is *active* between a pair of consecutive *IoCreateHandle* and *IoDestroyHandle* events. All *Locations* 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

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>Io↔</i> <i>HandleRef</i>	handle	A previously <i>inactive</i> I/O handle which will be activated by this record. References a <i>IoHandle</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_IO_HANDLE</i> is available.
<i>OTF2_↔</i> <i>Io↔</i> <i>Access↔</i> <i>Mode</i>	mode	Determines which I/O operations can be applied to this I/O handle (e.g., read-only, write-only, read-write).
<i>OTF2_↔</i> <i>Io↔</i> <i>Creation↔</i> <i>Flag</i>	creationFlags	Requested I/O handle creation flags (e.g., create, exclusive, etc.).
<i>OTF2_↔</i> <i>Io↔</i> <i>StatusFlag</i>	statusFlags	I/O handle status flags which will be associated with the <i>handle</i> 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

C.114 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 *Locations* 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

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.

<i>OTF2_Io↔ HandleRef</i>	handle	An <i>active</i> I/O handle which will be inactivated by this records. References a <i>IoHandle</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_IO_HANDLE</i> is available.
-------------------------------	--------	---

See also

OTF2_EvtWriter_IoDestroyHandle()
 OTF2_GlobalEvtReaderCallbacks_SetIoDestroyHandleCallback()
 OTF2_EvtReaderCallbacks_SetIoDestroyHandleCallback()

Since

Version 2.1

C.115 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 *IoHandles* 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

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_Io↔ HandleRef</i>	oldHandle	An <i>active</i> I/O handle. References a <i>IoHandle</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_IO_HANDLE</i> is available.
<i>OTF2_Io↔ HandleRef</i>	newHandle	A previously <i>inactive</i> I/O handle which will be activated by this record. References a <i>IoHandle</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_IO_HANDLE</i> is available.
<i>OTF2_Io↔ StatusFlag</i>	statusFlags	The status flag for the new I/O handle <i>newHandle</i> . No status flags will be inherited from the I/O handle <i>oldHandle</i> .

See also

OTF2_EvtWriter_IoDuplicateHandle()
 OTF2_GlobalEvtReaderCallbacks_SetIoDuplicateHandleCallback()
 OTF2_EvtReaderCallbacks_SetIoDuplicateHandleCallback()

Since

Version 2.1

C.116 IoSeek

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

C.117 IoChangeStatusFlags

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>HandleRef</i>	handle	An <i>active</i> I/O handle. References a <i>IoHandle</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_IO_HANDLE</i> is available.
<i>int64_t</i>	offsetRequest	Requested offset.
<i>OTF2_↔</i> <i>Seek↔</i> <i>Option</i>	whence	Position inside the file from where <i>offsetRequest</i> should be applied (e.g., absolute from the start or end, relative to the current position).
<i>uint64_t</i>	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

C.117 IoChangeStatusFlags

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

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔</i> <i>HandleRef</i>	handle	An <i>active</i> I/O handle. References a <i>IoHandle</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_IO_HANDLE</i> is available.
<i>OTF2_↔</i> <i>StatusFlag</i>	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

C.118 IoDeleteFile

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

C.119 IoOperationBegin

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ Paradigm↔ Ref</i>	ioParadigm	The I/O paradigm which induced the deletion. References a <i>IoParadigm</i> definition.
<i>OTF2_↔ FileRef</i>	file	File identifier. References a <i>IoRegularFile</i> , or a <i>IoDirectory</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPI↔ NG_IO_FILE</i> is available.

See also

`OTF2_EvtWriter_IoDeleteFile()`
`OTF2_GlobalEvtReaderCallbacks_SetIoDeleteFileCallback()`
`OTF2_EvtReaderCallbacks_SetIoDeleteFileCallback()`

Since

Version 2.1

C.119 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

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_↔ HandleRef</i>	handle	An <i>active</i> I/O handle. References a <i>IoHandle</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_IO_HANDLE</i> is available.
<i>OTF2_↔ Operation↔ Mode</i>	mode	Mode of an I/O handle operation (e.g., read or write).
<i>OTF2_↔ Operation↔ Flag</i>	operationFlags	Special semantic of this operation.
<i>uint64_t</i>	bytesRequest	Requested bytes to write/read.
<i>uint64_t</i>	matchingId	Identifier used to correlate associated event records of an I/O operation. This identifier is unique for the referenced <i>IoHandle</i> .

See also

`OTF2_EvtWriter_IoOperationBegin()`
`OTF2_GlobalEvtReaderCallbacks_SetIoOperationBeginCallback()`
`OTF2_EvtReaderCallbacks_SetIoOperationBeginCallback()`

Since

Version 2.1

C.120 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

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_lo↔</i> <i>HandleRef</i>	handle	An <i>active</i> I/O handle. References a <i>IoHandle</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_IO_HANDLE</i> is available.
<i>uint64_t</i>	matchingId	Identifier used to correlate associated event records of an I/O operation. This identifier is unique for the referenced <i>IoHandle</i> .

See also

OTF2_EvtWriter_IoOperationTest()
 OTF2_GlobalEvtReaderCallbacks_SetIoOperationTestCallback()
 OTF2_EvtReaderCallbacks_SetIoOperationTestCallback()

Since

Version 2.1

C.121 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

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location where this event happened.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_lo↔</i> <i>HandleRef</i>	handle	An <i>active</i> I/O handle. References a <i>IoHandle</i> definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_IO_HANDLE</i> is available.

C.122 IoOperationComplete

<i>uint64_t</i>	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

C.122 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

<i>OTF2_↔ Location↔ Ref</i>	location	The location where this event happened.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>OTF2_Io↔ HandleRef</i>	handle	An <i>active</i> I/O handle. References a IoHandle definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_IO_HANDLE</i> is available.
<i>uint64_t</i>	bytesResult	Number of actual transferred bytes.
<i>uint64_t</i>	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

C.123 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

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

See also

OTF2_EvtWriter_IoOperationCancelled()
 OTF2_GlobalEvtReaderCallbacks_SetIoOperationCancelledCallback()
 OTF2_EvtReaderCallbacks_SetIoOperationCancelledCallback()

Since

Version 2.1

C.124 IoAcquireLock

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

Attributes

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

See also

OTF2_EvtWriter_IoAcquireLock()
 OTF2_GlobalEvtReaderCallbacks_SetIoAcquireLockCallback()
 OTF2_EvtReaderCallbacks_SetIoAcquireLockCallback()

Since

Version 2.1

C.125 IoReleaseLock

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

C.126 IoTryLock

Attributes

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

See also

`OTF2_EvtWriter_IoReleaseLock()`
`OTF2_GlobalEvtReaderCallbacks_SetIoReleaseLockCallback()`
`OTF2_EvtReaderCallbacks_SetIoReleaseLockCallback()`

Since

Version 2.1

C.126 IoTryLock

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

Attributes

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

See also

`OTF2_EvtWriter_IoTryLock()`
`OTF2_GlobalEvtReaderCallbacks_SetIoTryLockCallback()`
`OTF2_EvtReaderCallbacks_SetIoTryLockCallback()`

Since

Version 2.1

C.127 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_LOCATIONS_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

<code>OTF2_LocationRef</code>	location	The location where this event happened.
<code>OTF2_TimeStamp</code>	timestamp	The time when this event happened.
<code>OTF2_StringRef</code>	programName	The name of the program. References a String definition and will be mapped to the global definition if a mapping table of type <code>OTF2_MAPPING_STRING</code> is available.
<code>uint32_t</code>	numberOfArguments	Number of additional arguments to the program.
<code>OTF2_StringRef</code>	programArguments [numberOfArguments]	List of additional arguments to the program.

See also

- `OTF2_EvtWriter_ProgramBegin()`
- `OTF2_GlobalEvtReaderCallbacks_SetProgramBeginCallback()`
- `OTF2_EvtReaderCallbacks_SetProgramBeginCallback()`

Since

Version 2.1

C.128 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_LOCATIONS_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

<code>OTF2_LocationRef</code>	location	The location where this event happened.
-------------------------------	----------	---

C.128 ProgramEnd

<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The time when this event happened.
<i>int64_t</i>	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 <i>OTF2_UNDEFINED_INT64</i> , if the exit status was not available.

See also

`OTF2_EvtWriter_ProgramEnd()`
`OTF2_GlobalEvtReaderCallbacks_SetProgramEndCallback()`
`OTF2_EvtReaderCallbacks_SetProgramEndCallback()`

Since

Version 2.1

C.129 List of all marker records

C.130 *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", ...
<i>OTF2_MarkerSeverity</i>	severity	The severity for these markers.

See also

[OTF2_MarkerWriter_WriteDefMarker\(\)](#)
[OTF2_MarkerReaderCallbacks_SetDefMarkerCallback\(\)](#)

Since

Version 1.2

C.131 Marker

A user marker instance, with implied time stamp.

Attributes

<i>OTF2_TimeStamp</i>	timestamp	The time when this marker happened.
<i>OTF2_TimeStamp</i>	duration	A possible duration of this marker. May be 0.
<i>OTF2_MarkerRef</i>	marker	Groups this marker by name and severity. References a DefMarker definition.
<i>OTF2_MarkerScope</i>	scope	The type of scope of this marker instance.
uint64_t	scopeRef	The scope instance of this marker. Depends on <code>scope</code> .
const char*	text	A textual description for this marker.

See also

[OTF2_MarkerWriter_WriteMarker\(\)](#)
[OTF2_MarkerReaderCallbacks_SetMarkerCallback\(\)](#)

Since

Version 1.2

C.132 List of all snapshot records

C.133 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	numberOf↔ Records	Number of snapshot event records in this snapshot. Excluding the Snapshot↔ End record.

See also

`OTF2_SnapWriter_SnapshotStart()`
`OTF2_GlobalSnapReaderCallbacks_SetSnapshotStartCallback()`
`OTF2_SnapReaderCallbacks_SetSnapshotStartCallback()`

Since

Version 1.2

C.134 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	<code>contReadPos</code>	Position to continue reading in the event trace.

See also

`OTF2_SnapWriter_SnapshotEnd()`
`OTF2_GlobalSnapReaderCallbacks_SetSnapshotEndCallback()`
`OTF2_SnapReaderCallbacks_SetSnapshotEndCallback()`

Since

Version 1.2

C.135 MeasurementOnOffSnap

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

C.136 EnterSnap

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location of the snapshot.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The snapshot time of this record.
<i>OTF2_↔ Time↔ Stamp</i>	origEventTime	The original time this event happened.
<i>OTF2_↔ Measurement↔ Mode</i>	measurement↔ Mode	Is the measurement turned on (<i>OTF2_MEASUREMENT_ON</i>) or off (<i>OTF2↔ _MEASUREMENT_OFF</i>)?

See also

[MeasurementOnOff](#) event
OTF2_SnapWriter_MeasurementOnOff()
OTF2_GlobalSnapReaderCallbacks_SetMeasurementOnOffCallback()
OTF2_SnapReaderCallbacks_SetMeasurementOnOffCallback()

Since

Version 1.2

C.136 EnterSnap

This record exists for each [Enter](#) event where the corresponding [Leave](#) event did not occur before the snapshot.

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location of the snapshot.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The snapshot time of this record.
<i>OTF2_↔ Time↔ Stamp</i>	origEventTime	The original time this event happened.
<i>OTF2_↔ RegionRef</i>	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 <i>OTF2_MA↔ PPING_REGION</i> is available.

See also

[Enter](#) event
OTF2_SnapWriter_Enter()
OTF2_GlobalSnapReaderCallbacks_SetEnterCallback()
OTF2_SnapReaderCallbacks_SetEnterCallback()

Since

Version 1.2

C.137 `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 `Mpilrecv` event. Note that it may so, that a previous `Mpilsend` 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

<code>OTF2_↔ Location↔ Ref</code>	location	The location of the snapshot.
<code>OTF2_↔ Time↔ Stamp</code>	timestamp	The snapshot time of this record.
<code>OTF2_↔ Time↔ Stamp</code>	origEventTime	The original time this event happened.
<code>uint32_t</code>	receiver	MPI rank of receiver in <code>communicator</code> .
<code>OTF2_↔ CommRef</code>	communicator	Communicator ID. References a <code>Comm</code> definition and will be mapped to the global definition if a mapping table of type <code>OTF2_MAPPING_COMM</code> is available.
<code>uint32_t</code>	msgTag	Message tag
<code>uint64_t</code>	msgLength	Message length

See also

`MpiSend` event
`OTF2_SnapWriter_MpiSend()`
`OTF2_GlobalSnapReaderCallbacks_SetMpiSendCallback()`
`OTF2_SnapReaderCallbacks_SetMpiSendCallback()`

Since

Version 1.2

C.138 `MpilsendSnap`

This record exists for each `Mpilsend` event where a corresponding `MpilsendComplete` or `MpiRequestCancelled` event did not occur on this location before the snapshot. Or the corresponding `MpilsendComplete` did occurred (the `MpilsendCompleteSnap` 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 `Mpilrecv` event.)

Attributes

<code>OTF2_↔ Location↔ Ref</code>	location	The location of the snapshot.
<code>OTF2_↔ Time↔ Stamp</code>	timestamp	The snapshot time of this record.

C.139 MpisendCompleteSnap

OTF2_↔ Time↔ Stamp	origEventTime	The original time this event happened.
uint32_t	receiver	MPI rank of receiver in <code>communicator</code> .
OTF2_↔ CommRef	communicator	Communicator ID. References a Comm definition and will be mapped to the global definition if a mapping table of type <code>OTF2_MAPPING_COMM</code> is available.
uint32_t	msgTag	Message tag
uint64_t	msgLength	Message length
uint64_t	requestID	ID of the related request

See also

[Mpisend](#) event
`OTF2_SnapWriter_Mpisend()`
`OTF2_GlobalSnapReaderCallbacks_SetMpisendCallback()`
`OTF2_SnapReaderCallbacks_SetMpisendCallback()`

Since

Version 1.2

C.139 MpisendCompleteSnap

This record exists for each [Mpisend](#) event where the corresponding [MpisendComplete](#) 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 [Mpilrecv](#) 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

[MpisendComplete](#) event
`OTF2_SnapWriter_MpisendComplete()`
`OTF2_GlobalSnapReaderCallbacks_SetMpisendCompleteCallback()`
`OTF2_SnapReaderCallbacks_SetMpisendCompleteCallback()`

Since

Version 1.2

C.140 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 [MpisendComplete](#) event. Or a [MpilrecvRequest](#)

occurred before this event but the corresponding [MpiRecv](#) event did not occurred before this snapshot. In this case the message matching couldn't performed yet, because the envelope of the ongoing [MpiRecvRequest](#) 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 <code>communicator</code> .
OTF2_↔ CommRef	communicator	Communicator ID. References a Comm definition and will be mapped to the global definition if a mapping table of type <code>OTF2_MAPPING_COMM</code> 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

C.141 MpiRecvRequestSnap

This record exists for each [MpiRecvRequest](#) event where an corresponding [MpiRecv](#) or [MpiRequestCancelled](#) event did not occur on this location before the snapshot. Or the corresponding [MpiRecv](#) did occurred (the [MpiRecvSnap](#) 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 [MpiRecv](#) 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.

C.142 MpilrecvSnap

<i>uint64_t</i>	requestID	ID of the requested receive
-----------------	-----------	-----------------------------

See also

[MpilrecvRequest](#) event
OTF2_SnapWriter_MpilrecvRequest()
OTF2_GlobalSnapReaderCallbacks_SetMpilrecvRequestCallback()
OTF2_SnapReaderCallbacks_SetMpilrecvRequestCallback()

Since

Version 1.2

C.142 MpilrecvSnap

This record exists for each [Mpilrecv](#) 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 [MpilsendComplete](#) event. Or a [MpilrecvRequest](#) occurred before this event but the corresponding [Mpilrecv](#) event did not occur before this snapshot. In this case the message matching couldn't be performed yet, because the envelope of the ongoing [MpilrecvRequest](#) is not yet known.

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location of the snapshot.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The snapshot time of this record.
<i>OTF2_↔ Time↔ Stamp</i>	origEventTime	The original time this event happened.
<i>uint32_t</i>	sender	MPI rank of sender in <code>communicator</code> .
<i>OTF2_↔ CommRef</i>	communicator	Communicator ID. References a Comm definition and will be mapped to the global definition if a mapping table of type <code>OTF2_MAPPING_COMM</code> is available.
<i>uint32_t</i>	msgTag	Message tag
<i>uint64_t</i>	msgLength	Message length
<i>uint64_t</i>	requestID	ID of the related request

See also

[Mpilrecv](#) event
OTF2_SnapWriter_Mpilrecv()
OTF2_GlobalSnapReaderCallbacks_SetMpilrecvCallback()
OTF2_SnapReaderCallbacks_SetMpilrecvCallback()

Since

Version 1.2

C.143 MpiCollectiveBeginSnap

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

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location of the snapshot.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The snapshot time of this record.
<i>OTF2_↔ Time↔ Stamp</i>	origEventTime	The original time this event happened.

See also

[MpiCollectiveBegin](#) event
 OTF2_SnapWriter_MpiCollectiveBegin()
 OTF2_GlobalSnapReaderCallbacks_SetMpiCollectiveBeginCallback()
 OTF2_SnapReaderCallbacks_SetMpiCollectiveBeginCallback()

Since

Version 1.2

C.144 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

<i>OTF2_↔ Location↔ Ref</i>	location	The location of the snapshot.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The snapshot time of this record.
<i>OTF2_↔ Time↔ Stamp</i>	origEventTime	The original time this event happened.
<i>OTF2_↔ Collective↔ Op</i>	collectiveOp	Determines which collective operation it is.
<i>OTF2_↔ CommRef</i>	communicator	Communicator References a Comm definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_COMM</i> is available.
<i>uint32_t</i>	root	MPI rank of root in <i>communicator</i> or <i>OTF2_UNDEFINED_UINT32</i> if the call has no root rank.
<i>uint64_t</i>	sizeSent	Size of the sent message.
<i>uint64_t</i>	sizeReceived	Size of the received message.

See also

[MpiCollectiveEnd](#) event
 OTF2_SnapWriter_MpiCollectiveEnd()
 OTF2_GlobalSnapReaderCallbacks_SetMpiCollectiveEndCallback()
 OTF2_SnapReaderCallbacks_SetMpiCollectiveEndCallback()

C.145 OmpForkSnap

Since

Version 1.2

C.145 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

C.146 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.

<i>uint32_t</i>	lockID	ID of the lock.
<i>uint32_t</i>	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

C.147 OmpTaskCreateSnap

This record exists for each [OmpTaskCreate](#) event where the corresponding [OmpTaskComplete](#) event did not occur before this snapshot. Neither on this location nor on any other location in the current thread team.

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location of the snapshot.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The snapshot time of this record.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	origEventTime	The original time this event happened.
<i>uint64_t</i>	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

C.148 OmpTaskSwitchSnap

This record exists for each [OmpTaskSwitch](#) event where the corresponding [OmpTaskComplete](#) event did not occur before this snapshot. Neither on this location nor on any other location in the current thread team.

C.149 MetricSnap

Attributes

<i>OTF2_↔ Location↔ Ref</i>	location	The location of the snapshot.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The snapshot time of this record.
<i>OTF2_↔ Time↔ Stamp</i>	origEventTime	The original time this event happened.
<i>uint64_t</i>	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

C.149 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

<i>OTF2_↔ Location↔ Ref</i>	location	The location of the snapshot.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The snapshot time of this record.
<i>OTF2_↔ Time↔ Stamp</i>	origEventTime	The original time this event happened.
<i>OTF2_↔ MetricRef</i>	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 <i>OTF2_MAPPING_METRIC</i> is available.
<i>uint8_t</i>	numberOf↔ Metrics	Number of metrics with in the set.
<i>OTF2_↔ Type</i>	typeIDs [numberOf↔ Metrics]	List of metric types. These types must match that of the corresponding MetricMember definitions.

<i>OTF2_↔ Metric↔ Value</i>	metricValues [numberOf↔ Metrics]	List of metric values.
-------------------------------------	---	------------------------

See also

[Metric](#) event
 OTF2_SnapWriter_Metric()
 OTF2_GlobalSnapReaderCallbacks_SetMetricCallback()
 OTF2_SnapReaderCallbacks_SetMetricCallback()

Since

Version 1.2

C.150 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

<i>OTF2_↔ Location↔ Ref</i>	location	The location of the snapshot.
<i>OTF2_↔ Time↔ Stamp</i>	timestamp	The snapshot time of this record.
<i>OTF2_↔ Time↔ Stamp</i>	origEventTime	The original time this event happened.
<i>OTF2_↔ Parameter↔ Ref</i>	parameter	Parameter ID. References a Parameter definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_PARAMETER</i> is available.
<i>OTF2_↔ StringRef</i>	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 <i>OTF2_MAPPING_↔ _STRING</i> is available.

See also

[ParameterString](#) event
 OTF2_SnapWriter_ParameterString()
 OTF2_GlobalSnapReaderCallbacks_SetParameterStringCallback()
 OTF2_SnapReaderCallbacks_SetParameterStringCallback()

Since

Version 1.2

C.151 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.

C.152 ParameterUnsignedIntSnap

Attributes

<i>OTF2_↔</i> <i>Location↔</i> <i>Ref</i>	location	The location of the snapshot.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	timestamp	The snapshot time of this record.
<i>OTF2_↔</i> <i>Time↔</i> <i>Stamp</i>	origEventTime	The original time this event happened.
<i>OTF2_↔</i> <i>Parameter↔</i> <i>Ref</i>	parameter	Parameter ID. References a Parameter definition and will be mapped to the global definition if a mapping table of type <i>OTF2_MAPPING_PARAMETER</i> is available.
<i>int64_t</i>	value	Value of the recorded parameter.

See also

[ParameterInt](#) event
OTF2_SnapWriter_ParameterInt()
OTF2_GlobalSnapReaderCallbacks_SetParameterIntCallback()
OTF2_SnapReaderCallbacks_SetParameterIntCallback()

Since

Version 1.2

C.152 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

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

See also

[ParameterUnsignedInt](#) event
OTF2_SnapWriter_ParameterUnsignedInt()
OTF2_GlobalSnapReaderCallbacks_SetParameterUnsignedIntCallback()
OTF2_SnapReaderCallbacks_SetParameterUnsignedIntCallback()

Since

Version 1.2

Index

- List of all definition records, [28](#)
- List of all event records, [53](#)
- List of all marker records, [102](#)
- List of all snapshot records, [103](#)

- OTF2 callbacks, [19](#)
- OTF2 config tool, [21](#)
- OTF2 estimator tool, [25](#)
- OTF2 I/O recording, [26](#)
- OTF2 marker tool, [24](#)
- OTF2 print tool, [22](#)
- OTF2 records, [18](#)
- OTF2 snapshots tool, [23](#)
- OTF2 usage examples, [17](#)

- Usage of OTF2 tools, [20](#)