

EL S v nt Monitorin Syst m f r nl Developer's Guide (++)

Generated by Doxygen 1.3.5

S t Apr 29 10:55:37 2006

2.1	EventMonitorMain.cc	77
2.2	PullSamplerMain.cc	81
2.3	Pu hSamplerMain.cc	85

A TL Map

1.1 ■mon::Conductor on

1.1.2 Constructor & Destructor Documentation

1.1.2.1 `EventManager`

1.1 `mon::Conductor_impl` Class Reference

direction the direction of the adaptation req

1.2.2 Member Data Documentation

1.2.2.1 unsigned long ConductorInfoNamed::activeMonitors

number of active Monitoring Task connected to this conductor

1.2.2.2 unsigned long ConductorInfoNamed::activeSamplers

number of active EventSampler connected to this conductor

1.2.2.3 unsigned long ConductorInfoNamed::available

create a new event wrapper around certain event.

Parameters:

event event to wrap

1.3.3 Member Function Documentation

1.3.3.1 `const unsigned long* umon::Event::data () const` [inline]

Get the data of the event in a single contiguous memory chunk.

This method return the pointer to the event data array.

Examples:

`EventMonitorMain.cc`.

1.3.3.2 `unsigned long umon::Event::size () const` [inline]

Return the overall length of this event.

This method return the number of 4-b

The

Friends

- `class EventSampler_impl`

1.4.1

Parameters:

partition the IP partition of the Event(p.12) Monitoring System

address the sampling address of the **EventSampler**(p. 40)

~~criteria the Evolution Criterion (of the)~~

monitor a ORBA object reference to the root Monitoring Ta

reference to the parent **EventSampler**(p.40),

the EventChannel(p. 14)

sampling object that will be used by the pull am-

er ample event from

```
nnul::~EventChannel () [private,
```

6. **mon: Event Ch**  (

1.4 `mon::EventChannel` Class Ref

is not ready to process it, which may happen if the monitoring task can not
maintain the

1. `mon::EventChannel`

1.6 ■mon::EventCh

1.7 `emon::EventIterator` Class Reference

An iterator object, that may be used to retrieve event .

```
#include <EventIter
```

1.7.2 Constructor & Destructor Documentation

1.7.2.1 `uimon::EventIterator::EventIterator`
(`uimon::EventMonitor_impl * mon = 0`) [inline]

Constructor.

Parameters:

Parameters

1.8 ■mon::EventM

- void push_

1.8.3. `omni::EventMonitor_impl::remove_child` (`EventMonitoring::EventMonitor_ptr child`) [private]

Remove one of the children from the local children list.

By calling this method, the conductor informs Monitoring Task that one of the children has exited, and that it can stop forwarding event to it. This method might be called "from within" `FanThread`(p. 97), so we need to be careful in order to avoid deadlock. This is an IP published method.

Parameters:

child a ORBA reference to the child that shall be removed.

1.11 `mon::EventSampler_`

direction whether to speed up or slow down the sampling rate

See also

1.11.3.8 `void umon::EventSampler_impl::replace_monitor`
(

1.12 `emon::MaskedValue` struct Reference

A class representing a masked value.

```
#include <common.h>
```

Public Member Functions

- **`MaskedValue`** (long value, bool ignore=false)
Constructor.
- **`MaskedValue`** (const EventMonitoring::MaskedValue &t) **`P`**

1.12 ■mon::

1.13 MonitorInfoNamed Class Reference

Contain information about an instance of an active Monitoring Task.

```
#include <MonitorInfoNamed.h>
```


1.14 `emon::MonitorNode` Class Reference

- unsigned int `next_child_`
The index of the child that will next receive an `addChild` invocation if None

1.14 `uimon::MonitorNode` Class Reference

1.14.3.6 `unsigned int uimon::MonitorNode::subTreeSize () const`

Return the size of the subtree of this Monitoring Task.

Returns:

the number of Monitoring Task in the subtree of this Monitoring Task

The documentation for this class was generated from the following file :

- `MonitorNode.h`
- `MonitorNode.cc`

1 a [REDACTED] nc [REDACTED] m in [REDACTED]

1.19.2 Member Function Documentation

- 1.19.2.1 `virtual PullSampling* mon::PullSamplingFactory::startSampling(const SamplingAddress & sa, const SelectionCriteria & sc) throw (BadAddress, BadCriteria, NoResources)` [pure vi

1.20 `emon::Push` `ampln`

This method is invoked, whenever a new sampling thread is started. Each new thread will be associated with a new **PushSampling** object.

Parameters:

- sa* **SamplingAddress**(p. 72) for this sampling thread
- sc* **SelectionCriteria**(p. 74) the event sampled by this thread will satisfy
- channel* an **EventChannel**(p. 14) object that may be used to push event to Monitoring Task

Returns:

- a reference to a new **PushSampling** object responsible for event sampling

Exceptions :

- emon::BadAddress*
- emon::BadCriteria*
- emon::NoResources*

The documentation for this struct was generated from the following file:

- **PushSamplingFactory.h**

Protected Member Functions

- `Sample`

1.22 ■mon::Sampl

1.23 mon::SelectionCriteria Struct Reference



```
int main( int ar c rhar ** ar v )
{
    // initialize IPC
    IPCCEre::init( ar c ar v );
    CmdAr Str partition_name ( 'p' "partition" "partition-name" "partition to wErk in.");
    CmdAr StrList keys ( 'k' "keys" "keys" "address keys" CmdAr ::isREQ | CmdAr ::isLIST);
    CmdAr StrList values ( 'v' "values" "values" "address values" CmdAr ::isREQ | CmdAr ::isLIST);
    CmdAr Str slEt_id ( 's' "slEt" "slEt-id" "id Ef the slEt to read events from (default \"\")");
    CmdAr Int detector_type ( 'D' "detector" "type" "detector type (default -1)");
    CmdAr Int lvl1_tri er_type ( 'T' "lvl1_tri er" "type" "lvl1_tri er type (default -1)");
    CmdAr Int lvl2_tri er_info ( 'I' "lvl2_tri er" "info" "lvl2_tri er info (default -1)");
    CmdAr Int status_wErD ( 'H' "header" "wErD" "status wErD (default 0)");
    CmdAr Int max_events ( 'M' "Max" "event-number" "maximum number Ef events to retrieve (default -1 means wErk fErever" );
    CmdAr Int buffer_
```



```
catch ( emon::N
```

2.2 PullSamplerMain.cc

This class, containing a pure virtual function can be used by the user to implement custom EventSampler following the pull model. Object of the PullSampling inheriting type will be automatically created by the PullSamplingFactory whenever a subscription of a Monitoring Task arrive and a new Event-channel is created.

See also:

PullSamplingFactory

Author:


```
CmdAr Int      event_size ('S' "size" "event-size"
                          "event size in 4-byte words ( 1024 by default )");
CmdAr Int      fra ment_number ('N' "number" "fra ment-number"
                          "number of fra ments in the event ( 1 by default )");
CmdAr Str      event_file ('f' "file" "data-file"
                          "event data to be sent to
```



```
if ( event_file.flags() & CmdA
```

```
        ers::fatal(ex);
    }
    ERS_DEBUG(1 "shutdown complete.");
    timer.stop();
    ERS_DEBUG(1 "total CPU Time : " << timer.userTime() + timer.systemTime());
    ERS_DEBUG(1 "total Time : " << timer.totalTime());
    return 0;
}
```

Index

- ~Event_hannel
 - emon::Event_hannel, 18
- ~EventManager_impl
 - emon::EventManager_impl, 22
- ~MyPullSampling
 - MyPullSampling, 37
- ~MyPushSampler
 - MyPushSampler, 62

```

di connect_sampler
    emon:: onductor_im ,15

emon:: onductor_im 11
    adapt_sampler_rate, 4
    add_monitor, 5
        onductor_im ,14
    connect_sampler, 5
    di connect_CCMpler, 5
    i Exit, 0
    max_children_, 8
    monitor_, 8
    pingLoop, 0
    remove_monitor, 0
    remove_sampler, 7
    CCMpler_, 8
        hutdown, 7
    throwInitializationException,
        7
emon::Event, 12
    data,
thrn:: Event, 740
CCD emon: Dm E

```


