

ATLAS TDAQ/DCS ROS

DFThreads package

Document Version: 1.1
Document ID: ATLAS-TDAQ-2004-XXX
Document Date: 24 March 2004
Document Status: Draft for comments

Abstracs52 4 81 0 0 1a0e0d04

1. UNSTARTED : no physical thread associated to this instance has been started yet
2. RUNNING : the thread is running
3. CANCEL_REQUESTED : the thread is running but its cancellation has been requested and will happen when-
even the thread is running
4. CLEANING_UP : the thread has terminated or has been canceled and is now running the code of `cleanup()` method.
5. TERMINATED : the thread has terminated its cleanup.

All the threads associated to instances of **DFThread** subclasses are created in “detached” mode. That means that every resource possibly hold by a running thread is immediately released upon its termination. This ensure a good usage of resources but prevents associating (“join”) other running associated condition (i.e.) to blocking termination). The **DFThread** package provide an alternative (and more powerful) mechanism through the `waitForCondition(Condition c)` method. Any thread can call the `waitForCondition()` method on an instance of a `waitForCondition(of) do Td[(41r35088(41r51(resources)-388(42T`

1.4 Utilities

To be completed.

For the moment look at documentation of classes **DFCountedPointer** <T> and **DFCreator**


```

*/

#include <unistd.h>
#include <exception>
#include <iostream>

#include "ThreadExample.h"
#include "DFThreads/DFStandardQueue.h"
#include "DFThreads/DFFastQueue.h"
#include "DFThreads/DFFastBlockingQueue.h"
#include "DFThreads/DFFastNonBlockingQueue.h"
#include "DFThreads/DFStandardPrioritizedQueue.h"

/*
   Constructor of the test thread
*/

ThreadExample::ThreadExample(int serialNum, int repeat, bool terminateWithException)
    : serialNum_(serialNum),
      repeat_(repeat),
      terminateWithException_(terminateWithException)
{
    messageQueue_ = DFStandardQueue < DFCountedPointer < MessageExample > >::Create("Messages");
}

/*
   Body of the test thread (simply send messages to the main thread)
*/

void ThreadExample::run() {
    std::cout << "Started thread: " << DFThread::id() << std::endl ;

    for (int i = 0; i < repeat_; i++) {
        sleep(1);

        // //Create message that will be sent with LOW priority
        // DFCountedPointer<MessageExample>
        //     messageLow(new MessageExample(serialNum_,i,"LOW"));
        //Create message that will be sent with NORMAL priority
        DFCountedPointer<MessageExample>
            messageNormal(new MessageExample(serialNum_,i,"NORMAL"));
        // //Create message that will be sent with HIGH priority
        // DFCountedPointer<MessageExample>
        //     messageHigh(new MessageExample(serialNum_,i,"HIGH"));

        // //Send message to the main thread with LOW priority
        // messageQueue_>push(messageLow,
        //     DFPrioritizedOutputQueue < DFCountedPointer < MessageExample > >::LOW);
        //Send message to the main thread with HIGH priority
        messageQueue_>push(messageNormal);
        // //Send message to the main thread with HIGH priority
        // messageQueue_>push(messageHigh,
        //     DFPrioritizedOutputQueue < DFCountedPointer < MessageExample > >::HIGH);// //
s/ ///
s/> 51; i@a1-'i' '
        // T(test)-60600(with)-6hatowith

t -9.465 Td[(C4 -9.464 Td[(.B6hueuler51;)]TJcancellatestputQueue)-600(<)-600 main maer51;

```



```
        //If no message arrived in the last 10 seconds, assume that there
        //are is no message leftover and stop waiting for new ones
        loopFlag = false ;
    }
    catch (...) {
        cout << "unexpected exception" << endl ;
    }
}

//When all the threads have sent their last message wait for them
//to terminate cleanly
for (int i = 0; i < nth; i++) {
    cout << "MAIN: waiting for termination of thread " << i << endl;
    thr[i]->stopExecution();
    try {
        thr[i]->waitForCondition(DFThread::TERMINATED, 100);
        cout << "MAIN: thread " << i << " terminated" << endl;
    }
    catch (DFThread::Timeout) {
        cout << "MAIN: thread " << i << " did not terminate in time" << endl;
    }
}

cout << "MAIN: Iall finished with the threads" << endl;

//Release the queue used to communicate with threads
int i = messageQueue->destroy();

if (i == 0) {
    cout << "MAIN: Input queue cleaned up" << endl;
}
else {
    cout << "MAIN: ERROR: Input queue is still referred by some thread" << endl;
}
}
```

2 DFThreads Hierarchical Index

2.1 DFThreads Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

DFConditional

Ribbons and other books under DEFO condition by Other in the external goods Advisory of the book which own the book and the Demitex.

vold@csail.mit.edu, **Kagimura@math.nagoya-u.ac.jp**, **chellapalli@cs.cmu.edu**, **LEAH@cs.kit.edu**, **TIMOTHY.BOWEN@hp.com**, **redec@ling.wisc.edu**, **WPF@cs.cmu.edu**, **the[ion.13J]@th**

3.4 DFException Class Reference

```
#include <DFException.h>
```

3.4.1 Detailed Description

Base class for exceptions.

Public Methods

1

T pop (long int *timeout*) throw (typename DFInputQueue<T>::Timeout) [virtual]

Get next available instance of class T from the input queue. Concrete implementations of this method shall not return as long as the queue is empty. If the number of seconds specified by timeout parameter are passed the implementations shall throw Timeout exception. The thread calling **pop()** (p. 16) method on an empty queue shall

3.6 DFFastMutex Class Reference

```
#include <DFFastMutex.h>
```

3.6.1 Detailed Description

Provides the mutex (MUTual EXclusion) synchronization mechanism. The lock on a DFFastMutex instance is granted to only one thread at a time. Threads requesting to lock a DFFastMutex which is already locked by another thread are blocked with no CPU consumption until the lock is released. DFFastMutex implementation is based on the standard pthread mutex with no additional checks. For a safer but slower implementation see **DFMutex** (p. 25) class.

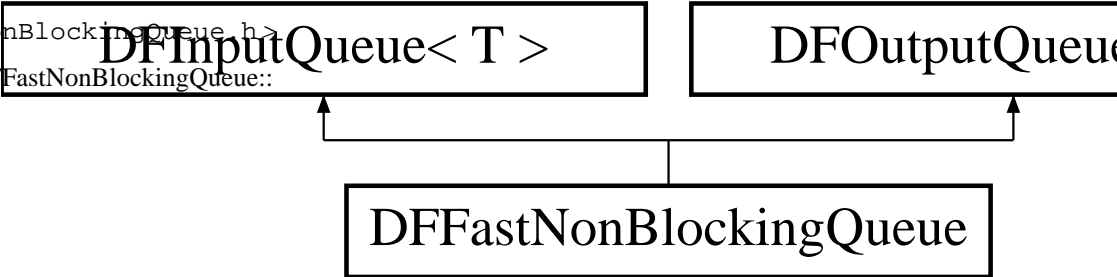
Public Methods

- int **destroy** ()
- void **unlock** ()

3.7 DFFastNonBlockingQueue Class Template Reference

#include <DFFastNonBlockingQueue.h>

Inheritance diagram for DFFastNonBlockingQueue::



T pop (long int *timeout*) throw (typename DFInputQueue<T>::Timeout) [virtual]

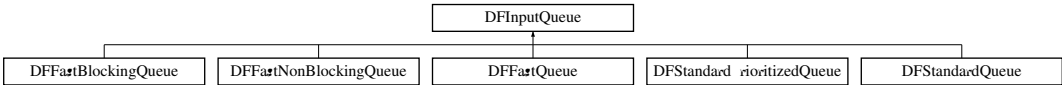
3.8 DFFastQueue Class Template Reference

T pop (long int *timeout*) throw (typename DFInputQueue<T>::Timeout) [virtual]

3.9 DFInputQueue Class Template Reference

```
#include <DFInputQueue.h>
```

Inheritance diagram for DFInputQueue::



3.9.1

Implemented in **DFFastBlockingQueue** (p. 17), **DFFastNonBlockingQueue** (p. 20), **DFFastQueue** (p. 22), **DFStandardPrioritizedQueue** (p. 31), and **DFStandardQueue** (p. 33).

virtual int destroy () [pure virtual]

Release the queue and delete it only if no other thread is referencing it.

Implemented in **DFFastBlockingQueue** (p. 17), **DFFastNonBlockingQueue** (p. 20), **DFFastQueue** (p. 22), **DFStandardPrioritizedQueue** (p. 31), and **DFStandardQueue** (p. 33).

3.10 DFMutex Class Reference

```
#include <DFMutex.h>
```

3.10.1 Detailed Description

Provides the mutex (MUTual EXclusion) synchronization mechanism. The lock on a DFMutex instance is granted to only one thread at a time. Threads requesting to lock a DFMutex which is already locked by another thread are blocked with no CPU consumption until the lock is released. DFMutex implementation is based on the standard pthread mutex but adds to it some checks on possible incorrect operations (see lock(and **unlock()** (p. 25) methods description) which are not mandatory for pthread specifications.

Public Methods

- int **destroy** ()
- void **unlock** () throw (NotLockedByThread)
- void **lock** () throw (Already)

Static Public Methods

- DFMutex const **Create** (char Name)
-

3.12 DFPrioritizedOutputQueue Class Template Reference

#include

3.14 DFStandardPrioritizedQueue Class Template Reference

T pop (long int *timeout*) throw (typename DFInputQueue<T>::Timeout) [virtual]

Get next available instance of class T from the input queue. Concrete implementations of this method shall not return as long as the queue is empty. If the number of seconds specified by timeout parameter are passed the implementations shall throw Timeout exception. The thread calling **pop()** (p. 30) method on an empty queue shall be blocked with no CPU consumption.

Implements **DFInputQueue** (p. 23).

bool empty () [virtual]

Test if the queue is empty.

Implements **DFInputQueue** (p. 23).

unsigned int size () [virtual]

Returns the maximum number of elements that the queue can contain or 0 if the queue is dynamically extensible.

Implements **DFInputQueue** (p. 23).

unsigned int numberOfElements () [virtual]

Returns the number of elements contained in the queue.

Implements **DFInputQueue** (p. 23).

int destroy () [virtual]

Release the queue and delete it only if no other thread is referencing it.

Implements **DFInputQueue** (p. 24).

3.15 DFStandardQueue Class Template Reference

#include

void waitForCondition (Condition *condition*, long int *timeout*) throw (Timeout)

Wait fo the physical thread associated to a DFThread instance to reach a defined running condition (see