

# Dual State Framework

API Document

Yu Chen - C00151352

2015/04/15 14:10:28



# Contents

|          |                                      |          |
|----------|--------------------------------------|----------|
| <b>1</b> | <b>Get Started</b>                   | <b>1</b> |
| 1.1      | Welcome                              | 1        |
| 1.2      | Short example                        | 1        |
| 1.2.1    | Implementing DualStateFramework      | 1        |
| 1.2.2    | Implementing dsf::SynchronizedObject | 1        |
| 1.2.3    | Creating a manager class             | 2        |
| 1.2.4    | Running DSF                          | 2        |
| <b>2</b> | <b>Installation</b>                  | <b>3</b> |
| 2.1      | Mac OS X                             | 3        |
| 2.1.1    | Install Dependencies                 | 3        |
| 2.1.2    | Install Dual State Framework         | 3        |
| 2.1.3    | Use DSF in Xcode                     | 3        |
| 2.2      | Microsoft Windows                    | 3        |
| 2.2.1    | Install Dependencies                 | 3        |
| 2.2.2    | Install Dual State Framework         | 4        |
| 2.2.3    | Use DSF in Visual Studio             | 4        |
| 2.3      | Linux                                | 4        |
| 2.3.1    | Install Dependencies                 | 4        |
| 2.3.2    | Install Dual State Framework         | 4        |
| 2.4      | Compile source code                  | 4        |
| 2.4.1    | Pre-Build                            | 5        |
| 2.4.2    | Build the project                    | 5        |
| <b>3</b> | <b>Namespace Documentation</b>       | <b>7</b> |
| 3.1      | dsf Namespace Reference              | 7        |
| 3.1.1    | Typedef Documentation                | 8        |
| 3.1.1.1  | function                             | 8        |
| 3.1.1.2  | TaskArgument                         | 8        |
| 3.1.1.3  | TaskArgumentException                | 8        |
| 3.1.1.4  | TaskFunction                         | 8        |
| 3.1.2    | Variable Documentation               | 8        |

|          |   |          |
|----------|---|----------|
| 3.1.2.1  | DualStateFramework                                | 8        |
| 3.1.2.2  | SynchronizedObject                                | 8        |
| 3.1.2.3  | Task  | 8        |
| 3.1.2.4  | TaskBox   | 8        |
| <b>4</b> | <b>Class Documentation</b>                        | <b>9</b> |
| 4.1      | dsf::DualStateFramework Class Reference           | 9        |
| 4.1.1    | Detailed Description                              | 10       |
| 4.1.2    | Example   | 10       |
| 4.1.3    | Constructor & Destructor Documentation            | 10       |
| 4.1.3.1  | DualStateFramework                                | 10       |
| 4.1.3.2  | ~DualStateFramework                               | 10       |
| 4.1.4    | Member Function Documentation                     | 10       |
| 4.1.4.1  | add   | 10       |
| 4.1.4.2  | doOneFrame  | 10       |
| 4.1.4.3  | getState  | 11       |
| 4.1.4.4  | initialize  | 11       |
| 4.1.5    | Example   | 11       |
| 4.1.5.1  | refresh   | 11       |
| 4.1.5.2  | remove  | 12       |
| 4.1.5.3  | run   | 12       |
| 4.1.5.4  | send  | 12       |
| 4.1.5.5  | setNumberOfThreads                                | 12       |
| 4.1.5.6  | start   | 12       |
| 4.2      | dsf::Lock Class Reference                         | 13       |
| 4.2.1    | Detailed Description                              | 13       |
| 4.2.2    | Example   | 13       |
| 4.2.3    | Member Function Documentation                     | 14       |
| 4.2.3.1  | lock  | 14       |
| 4.2.3.2  | unlock  | 14       |
| 4.2.4    | Member Data Documentation                         | 14       |
| 4.2.4.1  | locker  | 14       |
| 4.3      | dsf::Runnable Class Reference                     | 14       |
| 4.3.1    | Detailed Description                              | 15       |
| 4.3.2    | Member Enumeration Documentation                  | 15       |
| 4.3.2.1  | State   | 16       |
| 4.3.3    | Member Function Documentation                     | 16       |
| 4.3.3.1  | getState  | 16       |
| 4.3.3.2  | run   | 16       |
| 4.4      | dsf::Synchronisable< T > Class Template Reference | 16       |

|         |  |    |
|---------|--|----|
| 4.4.1   | Detailed Description                       | 16 |
| 4.4.2   | Example                                    | 17 |
| 4.4.3   | Constructor & Destructor Documentation     | 17 |
| 4.4.3.1 | ~Synchronisable                            | 17 |
| 4.4.4   | Member Function Documentation              | 17 |
| 4.4.4.1 | synchronise                                | 17 |
| 4.4.5   | Member Data Documentation                  | 17 |
| 4.4.5.1 | next                                       | 17 |
| 4.5     | dsf::SynchronizedObject Class Reference    | 17 |
| 4.5.1   | Detailed Description                       | 18 |
| 4.5.2   | Example                                    | 18 |
| 4.5.3   | Constructor & Destructor Documentation     | 19 |
| 4.5.3.1 | SynchronizedObject                         | 19 |
| 4.5.3.2 | ~SynchronizedObject                        | 19 |
| 4.5.4   | Member Function Documentation              | 19 |
| 4.5.4.1 | getState                                   | 19 |
| 4.5.4.2 | receive                                    | 19 |
| 4.5.4.3 | run  | 20 |
| 4.5.5   | Example                                    | 20 |
| 4.5.5.1 | synchronise                                | 20 |
| 4.5.6   | Friends And Related Function Documentation | 20 |
| 4.5.6.1 | DualStateFramework                         | 20 |
| 4.6     | dsf::SynchronizedVar Class Reference       | 20 |
| 4.6.1   | Detailed Description                       | 21 |
| 4.6.2   | Example                                    | 21 |
| 4.6.3   | Constructor & Destructor Documentation     | 21 |
| 4.6.3.1 | SynchronizedVar                            | 22 |
| 4.6.4   | Member Function Documentation              | 22 |
| 4.6.4.1 | operator=                                  | 22 |
| 4.6.4.2 | synchronise                                | 22 |
| 4.7     | dsf::Task Class Reference                  | 22 |
| 4.7.1   | Detailed Description                       | 23 |
| 4.7.2   | Constructor & Destructor Documentation     | 23 |
| 4.7.2.1 | Task                                       | 23 |
| 4.7.2.2 | ~Task                                      | 23 |
| 4.7.3   | Member Data Documentation                  | 23 |
| 4.7.3.1 | from                                       | 23 |
| 4.7.3.2 | taskArgument                               | 23 |
| 4.7.3.3 | taskFunction                               | 23 |
| 4.7.3.4 | to   | 23 |

|         |  |           |
|---------|--|-----------|
| 4.8     | dsf::TaskBox Class Reference . . . . .           | 23        |
| 4.8.1   | Detailed Description . . . . .                   | 24        |
| 4.8.2   | Constructor & Destructor Documentation . . . . . | 24        |
| 4.8.2.1 | TaskBox . . . . .                                | 25        |
| 4.8.2.2 | ~TaskBox . . . . .                               | 25        |
| 4.8.3   | Member Function Documentation . . . . .          | 25        |
| 4.8.3.1 | isEmpty . . . . .                                | 25        |
| 4.8.3.2 | pop . . . . .                                    | 25        |
| 4.8.3.3 | process . . . . .                                | 26        |
| 4.8.3.4 | push . . . . .                                   | 26        |
| 4.8.4   | Member Data Documentation . . . . .              | 26        |
| 4.8.4.1 | tasks . . . . .                                  | 26        |
|         | <b>Index</b>                                     | <b>27</b> |

# 1 | Get Started

## 1.1 Welcome

Welcome to the official DSF documentation. Here you will find a detailed view of all the DSF classes and functions. If you have not installed DSF, you may be interested in how to install DSF on [Mac OS X](#) , [MS Windows](#) , and [Linux](#) . You may also be interested in how to [compile source code](#) .

## 1.2 Short example

Here is a short example, to show you how simple it is to use DSF:

### 1.2.1 Implementing DualStateFramework

MyDSF.h

```
#ifndef __DSFExample__MyDSF__
#define __DSFExample__MyDSF__

#include <dsf/DualStateFramework.h>

class MyDSF : public dsf::DualStateFramework // Extends dsf::DualStateFramework
{
public:
    explicit MyDSF() : DualStateFramework() {} // Uses default super constructor
    void initialize() override {}
};

#endif
```

### 1.2.2 Implementing dsf::SynchronizedObject

SyncObj.h

```
#ifndef DSFExample_SyncObj_h
#define DSFExample_SyncObj_h

#include <dsf/SynchronizedObject.h>

class SyncObj : public dsf::SynchronizedObject // Extends dsf::SynchronizedObject
{
public:
    SyncObj(int v) : SynchronizedObject(), v(v) {} // Uses default super constructor
    int getValue() {
        return this->v;
    }
protected:
    void run() override { // Overrides pure virtual function
        if(this->receive()) // Returns the number of message received
            this->process(); // Executes received messages
    }
private:
    int v;
};

#endif
```

### 1.2.3 Creating a manager class

#### ObjManager.h

```
#ifndef DSFExample_ObjManager_h
#define DSFExample_ObjManager_h

#include "MyDSF.h"
#include "SyncObj.h"
#include <iostream>

class ObjManager
{
public:
    MyDSF* dsf;
    dsf::TaskFunction* print;

    ObjManager(MyDSF* dsf) : dsf(dsf) { // Alias DSF pointer
        // Initialises TaskFunctions
        this->print = new dsf::TaskFunction([this] {
            dsf::SynchronizedObject* to,
                                dsf::SynchronizedObject* from,
                                dsf::TaskArgument* args)
            {
                auto syncObj = args->to<SyncObj*>();
                std::cout << syncObj->getValue();
                this->dsf->remove(to);
            }
        });

        ~ObjManager() {
            delete this->print;
        }
};

#endif
```

### 1.2.4 Running DSF

#### main.cpp

```
#include "MyDSF.h"
#include "SyncObj.h"
#include "ObjManager.h"

int main(int argc, const char * argv[]) {
    const int NUMBER_OF_OBJS = 100;
    auto dsf = new MyDSF();
    auto om = new ObjManager(dsf);
    SyncObj* sos[NUMBER_OF_OBJS];
    for(int i = 0; i < NUMBER_OF_OBJS; i++) { // Creates NUMBER_OF_OBJS SyncObj objects
        sos[i] = new SyncObj(i);
        dsf->add(sos[i]); // Adds objects to DSF object
        dsf->send(sos[i], sos[i], om->print, new dsf::TaskArgument(sos[i])); // Sends
        messages
    }
    dsf->start();
    delete dsf;
    delete om;
    return 0;
}
```



## 2 | Installation

Dual State Framework uses yctools and Intel tbb . Before the installation you need to install them first.

### 2.1 Mac OS X

#### 2.1.1 Install Dependencies

##### yctools

Download: <https://sourceforge.net/projects/yctools/>  
Download the pkg file and install it.

##### Intel tbb

Download: <https://www.threadingbuildingblocks.org/download>  
Download the OS X version and unzip it.  
Inside the directory, copy "libtbb.dylib" in the subdirectory "lib" to "/usr/lib" or "/usr/local/lib".  
Next, copy the directory "include/dsf" to "/usr/include" or "/usr/local/include".

#### 2.1.2 Install Dual State Framework

Download: <https://sourceforge.net/projects/dualstateframework/>  
Download the pkg file and install it.

#### 2.1.3 Use DSF in Xcode

To use this framework in Xcode is very simple. You just need to drag dsf.framework and yctools.framework to your project explorer.

### 2.2 Microsoft Windows

#### 2.2.1 Install Dependencies

##### yctools

Download: <https://sourceforge.net/projects/yctools/>  
Download the exe file and install it.  
The installation will create an environment variable "yctools" which refers to the program installed path.

##### Intel tbb

Download: <https://www.threadingbuildingblocks.org/download>  
Download the Window OS version and unzip it.

Inside the directory, copy "tbb.lib" in the subdirectory "lib/your architecture/your visual studio version" to "where you want to store them/lib".

Copy "tbb.dll" in the subdirectory "bin/your architecture/your visual studio version" to "where you want to store them/bin".

Add an environment variable "tbb", and set its value to "where you want to store them".

Next, copy the directory "include/dsf" to "where you want to store them/include".

### 2.2.2 Install Dual State Framework

Download: <https://sourceforge.net/projects/dualstateframework/>

Download the exe file and install it.

The installation will create an environment variable "dsf" which refers to the program installed path.

### 2.2.3 Use DSF in Visual Studio

Add additional header path

In project properties -> C/C++ -> General -> Additional Include Directories, add \$(yctools)\include, \$(dsf)\include, and \$(tbb)\include.

Add Dependencies

In project properties -> Linker -> General -> Additional Library Directories, add \$(yctools)\lib, \$(dsf)\lib, and \$(tbb)\lib.

In project properties -> Linker -> Input -> Additional Dependencies, add yctools.lib, tbb.lib, and dsf.lib.

## 2.3 Linux

This page is only for Linux with Debian package management tools (Debian, ubuntu, and etc.). Other Linux users please visit [Compile source code](#).

### 2.3.1 Install Dependencies

yctools

Download: <https://sourceforge.net/projects/yctools/>

Download the deb file and install it.

Intel tbb

In terminal or console, type

```
$ sudo apt-get install libtbb2
```

### 2.3.2 Install Dual State Framework

Download: <https://sourceforge.net/projects/dualstateframework/>

Download the deb file and install it.

## 2.4 Compile source code

To compile the code, you need a C++ compiler with c++11 supported, git, and CMake.

### 2.4.1 Pre-Build

#### Get source code

```
$ git clone https://github.com/kuyoonjo/DualStateFramework.git
```

#### Generate project for compiler

GUI version of CMake is recommended to generate the project. For more information about cmake, please visit <http://www.cmake.org>.

### 2.4.2 Build the project

If you generate an Xcode, Visual Studio, or any other GUI IDE project, just open the project and build it.

If you generate a Makefile project, in a terminal or console, type

```
$ cd "your project directory"
$ make
```

Good luck!



## 3 | Namespace Documentation

### 3.1 dsf Namespace Reference

#### Classes

- class [DualStateFramework](#)

*The starting pointer for the framework is the abstract class [dsf::DualStateFramework](#).*

- class [Lock](#)

*Locking variables.*

- class [Runnable](#)

*Executing messages.*

- class [Synchronisable](#)

*Synchronising two states.*

- class [SynchronizedObject](#)

*Dual state object interface.*

- class [SynchronizedVar](#)

*A Class which implements [dsf::Synchronisable](#).*

- class [Task](#)

*Class [Task](#).*

- class [TaskBox](#)

*A [dsf::Task](#) queue.*

#### Typedefs

- typedef `yc::Any` [TaskArgument](#)
- typedef `yc::Exception::AnyException` [TaskArgumentException](#)
- typedef `std::function< void(dsf::SynchronizedObject *, dsf::SynchronizedObject *, TaskArgument *)>` [TaskFunction](#)
- typedef `void` [function](#)

#### Variables

- class DSF\_API [Task](#)
- class DSF\_API [TaskBox](#)
- class DSF\_API [DualStateFramework](#)
- class DSF\_API [SynchronizedObject](#)

### 3.1.1 Typedef Documentation

3.1.1.1 `typedef void dsf::function`

3.1.1.2 `typedef yc::Any dsf::TaskArgument`

3.1.1.3 `typedef yc::Exception::AnyException dsf::TaskArgumentException`

3.1.1.4 `typedef std::function<void (dsf::SynchronizedObject*, dsf::SynchronizedObject*, TaskArgument*)>  
dsf::TaskFunction`

### 3.1.2 Variable Documentation

3.1.2.1 `class DSF_API dsf::DualStateFramework`

3.1.2.2 `class DSF_API dsf::SynchronizedObject`

3.1.2.3 `class DSF_API dsf::Task`

3.1.2.4 `class DSF_API dsf::TaskBox`

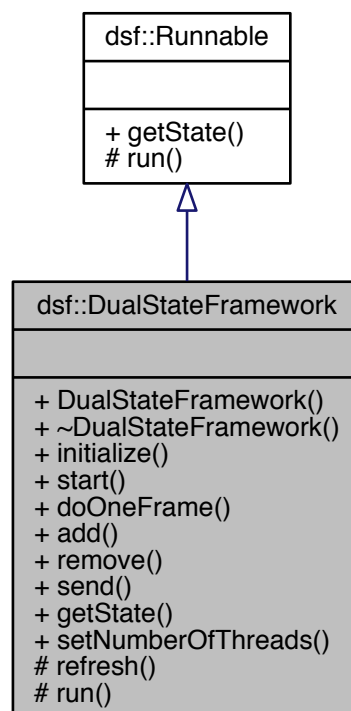
## 4 | Class Documentation

### 4.1 dsf::DualStateFramework Class Reference

The starting pointer for the framework is the abstract class [dsf::DualStateFramework](#).

```
#include <DualStateFramework.h>
```

Inheritance diagram for dsf::DualStateFramework:



#### Public Member Functions

- [DualStateFramework](#) ()
- [~DualStateFramework](#) ()
- virtual void [initialize](#) ()=0
- void [start](#) ()

- void [doOneFrame](#) ()
- void [add](#) ([SynchronizedObject](#) \*syncObj)
- void [remove](#) ([SynchronizedObject](#) \*syncObj)
- void [send](#) ([SynchronizedObject](#) \*to, [SynchronizedObject](#) \*from, [TaskFunction](#) \*taskFunction, [TaskArgument](#) \*args)
- [State](#) [getState](#) () override
- void [setNumberOfThreads](#) (int NumberOfThreads)

## Protected Member Functions

- virtual void [refresh](#) ()
- virtual void [run](#) () override

## Additional Inherited Members

### 4.1.1 Detailed Description

The starting pointer for the framework is the abstract class [dsf::DualStateFramework](#).

It provides essential functions for associating and managing its components ([SynchronizedObject](#) objects, function points, and etc.).

### 4.1.2 Example

```
#ifndef __DSFExample_MyDSF__
#define __DSFExample_MyDSF__

#include <dsf/DualStateFramework.h>

class MyDSF : public dsf::DualStateFramework // Extends dsf::DualStateFramework
{
public:
    explicit MyDSF() : DualStateFramework() {} // Uses default super constructor
    void initialize() override {}
};

#endif
```

### 4.1.3 Constructor & Destructor Documentation

4.1.3.1 [dsf::DualStateFramework::DualStateFramework \( \)](#)

4.1.3.2 [dsf::DualStateFramework::~~DualStateFramework \( \)](#)

### 4.1.4 Member Function Documentation

4.1.4.1 void [dsf::DualStateFramework::add](#) ( [dsf::SynchronizedObject](#) \* *syncObj* )

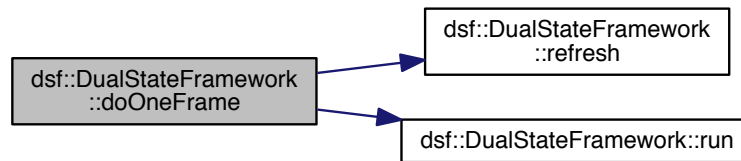
Add a [SynchronizedObject](#).

4.1.4.2 void [dsf::DualStateFramework::doOneFrame](#) ( )

Do one frame of all [SynchronizedObjects](#).



Here is the call graph for this function:



Here is the caller graph for this function:



#### 4.1.4.3 DualStateFramework::State dsf::DualStateFramework::getState ( ) [override],[virtual]

Return the state of the object.

Implements [dsf::Runnable](#).

#### 4.1.4.4 virtual void dsf::DualStateFramework::initialize ( ) [pure virtual]

For Signing TaskFunction Pointers

### 4.1.5 Example

```

this->printHello = new dsf::TaskFunction([this](
    dsf::SynchronizedObject* to,
                                dsf::SynchronizedObject* from,
                                dsf::TaskArgument* args)
{
    std::string str;
    float f;
    std::tie(str, f) = args->to<std::tuple<std::string, float>>();
    std::cout << str << " " << f << std::endl;
    this->remove(from);
});
  
```

#### 4.1.5.1 void dsf::DualStateFramework::refresh ( ) [protected],[virtual]

Clear all SynchronizedObjects which is marked as DELETE

Here is the caller graph for this function:



4.1.5.2 void dsf::DualStateFramework::remove ( dsf::SynchronizedObject \* syncObj )

Remove a [SynchronizedObject](#).

4.1.5.3 void dsf::DualStateFramework::run ( ) [override],[protected],[virtual]

Start all SynchronizedObjects associated.

Implements [dsf::Runnable](#).

Here is the caller graph for this function:



4.1.5.4 void dsf::DualStateFramework::send ( dsf::SynchronizedObject \* to, dsf::SynchronizedObject \* from, TaskFunction \* taskFunction, TaskArgument \* args )

Send messages

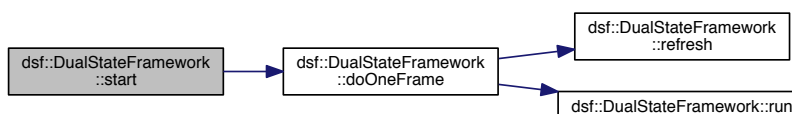
4.1.5.5 void dsf::DualStateFramework::setNumberOfThreads ( int NumberOfThreads )

Set the number of threads. 0 is automatic.

4.1.5.6 void dsf::DualStateFramework::start ( )

Start all SynchronizedObjects associated.

Here is the call graph for this function:



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

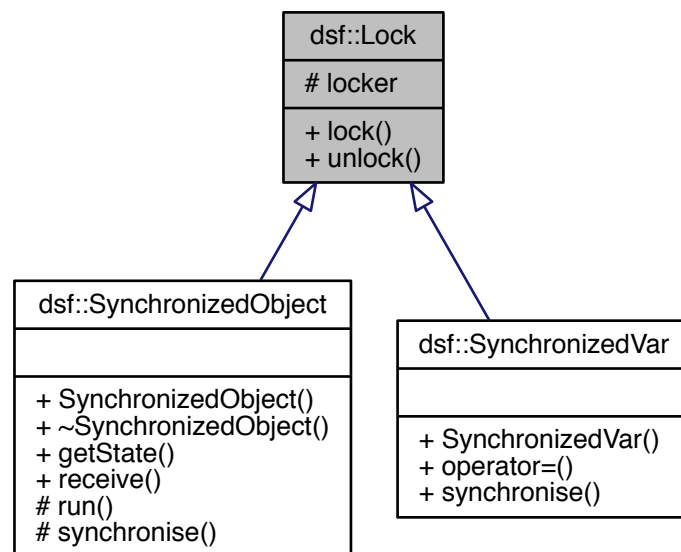
- DualStateFramework.h
- DualStateFramework.cpp

## 4.2 dsf::Lock Class Reference

Locking variables.

```
#include <Lock.h>
```

Inheritance diagram for dsf::Lock:



### Public Member Functions

- void `lock()`
- void `unlock()`

### Protected Attributes

- `std::mutex` `locker`

#### 4.2.1 Detailed Description

Locking variables.

The class can lock the objects using an unspecified sequence of calls to their members `lock` and `unlock` that ensures that all arguments are locked on return (without producing any deadlocks).

#### 4.2.2 Example

```
dsf->lock();
```

```
dsf->drawables->push_back(syncObj); //the object drawables is locked
dsf->unlock();
```

### 4.2.3 Member Function Documentation

#### 4.2.3.1 void dsf::Lock::lock ( )

Locks all the objects passed as arguments, blocking the calling thread if necessary.

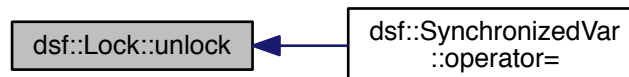
Here is the caller graph for this function:



#### 4.2.3.2 void dsf::Lock::unlock ( )

Unlocks all the objects.

Here is the caller graph for this function:



### 4.2.4 Member Data Documentation

#### 4.2.4.1 std::mutex dsf::Lock::locker [protected]

The locker

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

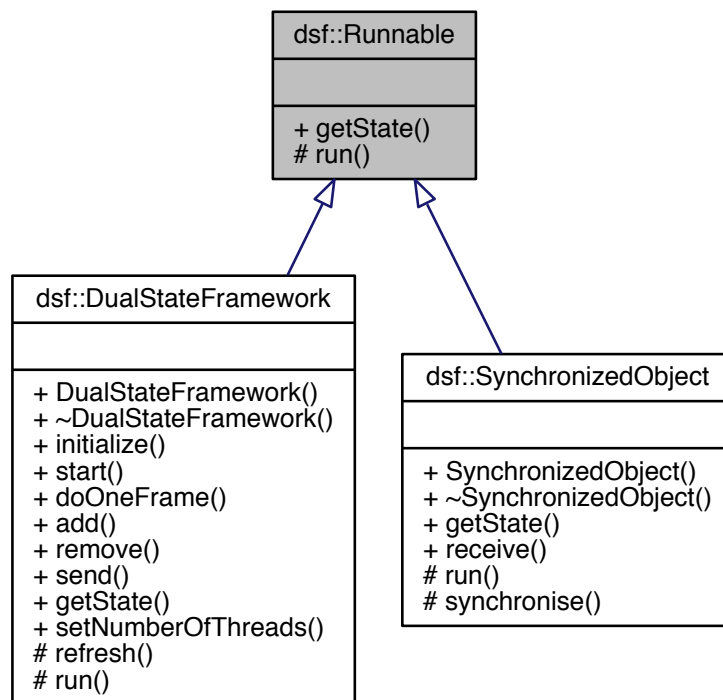
- Lock.h
- Lock.cpp

## 4.3 dsf::Runnable Class Reference

Executing messages.

```
#include <Runnable.h>
```

Inheritance diagram for dsf::Runnable:



## Public Types

- enum `State` { `RUNNING`, `STOPPED`, `READY`, `DELETED` }
- State of the object.*

## Public Member Functions

- virtual `State getState ()`=0

## Protected Member Functions

- virtual void `run ()`=0

### 4.3.1 Detailed Description

Executing messages.

The interface provides a run method which executes messages.

### 4.3.2 Member Enumeration Documentation

#### 4.3.2.1 enum dsf::Runnable::State

State of the object.

RUNNING: The object is running.

STOPPED: The object is stopped.

READY: The object is ready to run.

DELETED: The object is marked as deleted. System will automatically delete it.

Enumerator

***RUNNING***

***STOPPED***

***READY***

***DELETED***

#### 4.3.3 Member Function Documentation

##### 4.3.3.1 virtual State dsf::Runnable::getState ( ) [pure virtual]

Returns the current state.

Implemented in [dsf::DualStateFramework](#), and [dsf::SynchronizedObject](#).

##### 4.3.3.2 virtual void dsf::Runnable::run ( ) [protected],[pure virtual]

Executes messages

Implemented in [dsf::DualStateFramework](#), and [dsf::SynchronizedObject](#).

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

- Runnable.h

### 4.4 dsf::Synchronisable< T > Class Template Reference

Synchronising two states.

```
#include <Synchronisable.h>
```

#### Public Member Functions

- virtual [~Synchronisable](#) ( )
- virtual void [synchronise](#) ()=0

#### Protected Attributes

- T \* [next](#)

#### 4.4.1 Detailed Description

```
template<class T>class dsf::Synchronisable< T >
```

Synchronising two states.

The template interface provides a copy of current object, and a synchronise method which synchronise two copies.

### 4.4.2 Example

```
#include <dsf/Synchronisable.h>

class Vector3D
{
public:
    float x, y, z;
    Vector3D(float x, float y, float z) : x(x), y(y), z(z){}
}

class SyncVector : public dsf::Synchronisable<Vector3D>, public Vector3D
{
public:
    SyncInt(float x, float y, float z) : Vector3D(x, y, z) {
        this->next = new Vector3D(x, y, z);
    }
    void synchronise() override {
        this->x = this->next->x;
        this->y = this->next->y;
        this->z = this->next->z;
    }
}
```

### 4.4.3 Constructor & Destructor Documentation

4.4.3.1 `template<class T> virtual dsf::Synchronisable< T >::~Synchronisable ( )` [inline],[virtual]

### 4.4.4 Member Function Documentation

4.4.4.1 `template<class T> virtual void dsf::Synchronisable< T >::synchronise ( )` [pure virtual]

Signs current value to next value.

Implemented in [dsf::SynchronizedObject](#), and [dsf::SynchronizedVar](#).

### 4.4.5 Member Data Documentation

4.4.5.1 `template<class T> T* dsf::Synchronisable< T >::next` [protected]

A copy of current object.

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

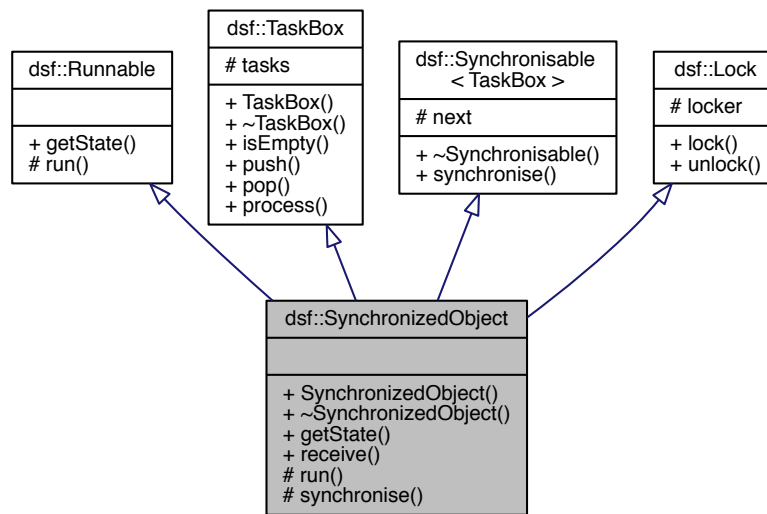
- Synchronisable.h

## 4.5 dsf::SynchronizedObject Class Reference

Dual state object interface.

```
#include <SynchronizedObject.h>
```

Inheritance diagram for dsf::SynchronizedObject:



## Public Member Functions

- [SynchronizedObject \(\)](#)
- virtual [~SynchronizedObject \(\)](#)
- [State getState \(\)](#) override
- int [receive \(\)](#)

## Protected Member Functions

- virtual void [run \(\)](#) override=0
- void [synchronise \(\)](#) override

## Friends

- class [DualStateFramework](#)

## Additional Inherited Members

### 4.5.1 Detailed Description

Dual state object interface.

The [dsf::SynchronizedObject](#) is a subclass of [dsf::TaskBox](#). In this framework, you can regard it as thread. It provides methods for implementing parallelism such as “send”, “receive”, and etc.

### 4.5.2 Example

```
// SyncCircle.h
#include <dsf/SynchronizedObject.h>
```



```

#include <SFML/Graphics.hpp>

class SyncCircle : public dsf::SynchronizedObject, public sf::CircleShape
{
public:
    SyncCircle();
protected:
    void run() override;
};

// SyncCircle.cpp

#include "SyncCircle.h"

SyncCircle::SyncCircle() : dsf::SynchronizedObject::
    SynchronizedObject(), sf::CircleShape::CircleShape()
{
}

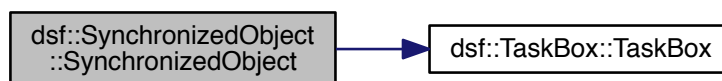
void SyncCircle::run()
{
    if(this->receive())
        this->process();
}

```

### 4.5.3 Constructor & Destructor Documentation

#### 4.5.3.1 dsf::SynchronizedObject::SynchronizedObject ( )

Here is the call graph for this function:



#### 4.5.3.2 dsf::SynchronizedObject::~~SynchronizedObject ( ) [virtual]

### 4.5.4 Member Function Documentation

#### 4.5.4.1 SynchronizedObject::State dsf::SynchronizedObject::getState ( ) [override], [virtual]

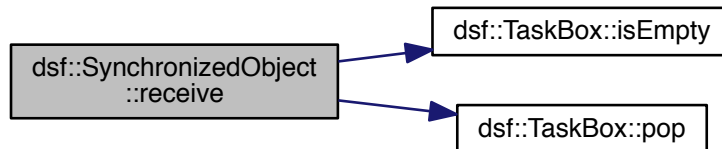
Returns the current state.

Implements [dsf::Runnable](#).

#### 4.5.4.2 int dsf::SynchronizedObject::receive ( )

Returns the number of message received

Here is the call graph for this function:



#### 4.5.4.3 virtual void dsf::SynchronizedObject::run ( ) [override],[protected],[pure virtual]

Executes messages

### 4.5.5 Example

```

void run() override { // Overrides pure virtual function
    if(this->receive()) // Returns the number of message received
        this->process(); // Executes received messages
}
  
```

Implements [dsf::Runnable](#).

#### 4.5.5.1 void dsf::SynchronizedObject::synchronise ( ) [override],[protected],[virtual]

Signs current taskbox to next taskbox.

Implements [dsf::Synchronisable< TaskBox >](#).

### 4.5.6 Friends And Related Function Documentation

#### 4.5.6.1 friend class DualStateFramework [friend]

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

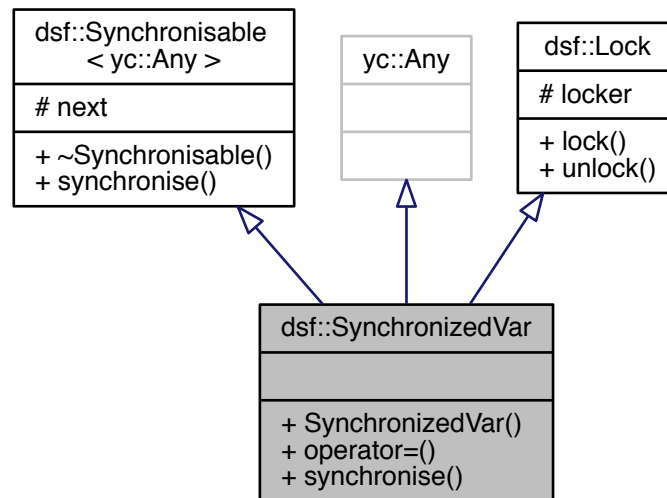
- SynchronizedObject.h
- SynchronizedObject.cpp

## 4.6 dsf::SynchronizedVar Class Reference

A Class which implements [dsf::Synchronisable](#).

```
#include <SynchronizedVar.h>
```

Inheritance diagram for dsf::SynchronizedVar:



## Public Member Functions

- `template<typename T >`  
`SynchronizedVar` (T &&value)
- `template<typename T >`  
`void operator=` (T &&value)
- `void synchronise` () override

## Additional Inherited Members

### 4.6.1 Detailed Description

A Class which implements `dsf::Synchronisable`.

The purpose of this class is to make thread-safe variables for `dsf::SynchronizedObject` objects. A `dsf::SynchronizedVar` object has two states - "current" and "next". The "current" is for read operation, and the "next" is for write operation. The function "synchronise" signs "next" to "current".

### 4.6.2 Example

```

dsf::SynchronizedVar myInt;
myInt = int(8); // value == NULL, next == 8
myInt.synchronize(); // value == 8, next == 8
std::cout << myInt.to<int>() << std::endl; // output 8
myInt = int(9); // value == 8, next == 9
std::cout << myInt.to<int>() << std::endl; // output 8
myInt.synchronize(); // value == 9, next == 9
std::cout << myInt.to<int>() << std::endl; // output 9
  
```

### 4.6.3 Constructor & Destructor Documentation

4.6.3.1 `template<typename T > dsf::SynchronizedVar::SynchronizedVar ( T && value )`

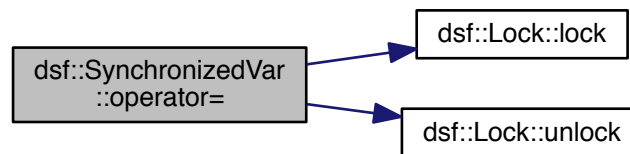
The value of "current" and the value of "next" is initialized as "value".

#### 4.6.4 Member Function Documentation

4.6.4.1 `template<typename T > void dsf::SynchronizedVar::operator= ( T && value )`

Signs a value to "next".

Here is the call graph for this function:



4.6.4.2 `void dsf::SynchronizedVar::synchronise ( ) [override],[virtual]`

Signs current value to next value.

Implements [dsf::Synchronisable< yc::Any >](#).

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

- SynchronizedVar.h
- SynchronizedVar.cpp

## 4.7 dsf::Task Class Reference

Class [Task](#).

```
#include <Task.h>
```

### Public Member Functions

- [Task](#) ([SynchronizedObject](#) \*to, [SynchronizedObject](#) \*from, [TaskFunction](#) \*taskFunction, [TaskArgument](#) \*taskArgument)
- [~Task](#) ()

### Public Attributes

- [SynchronizedObject](#) \* to
- [SynchronizedObject](#) \* from
- [TaskFunction](#) \* taskFunction
- [TaskArgument](#) \* taskArgument

### 4.7.1 Detailed Description

Class [Task](#).

This class have four members: from, to, function, and arguments, where "from" is a [dsf::SynchronizedObject](#) object who sent message to you.

### 4.7.2 Constructor & Destructor Documentation

4.7.2.1 `dsf::Task::Task ( SynchronizedObject * to, SynchronizedObject * from, TaskFunction * taskFunction, TaskArgument * taskArgument )` `[explicit]`

4.7.2.2 `dsf::Task::~~Task ( )`

### 4.7.3 Member Data Documentation

4.7.3.1 `SynchronizedObject* dsf::Task::from`

Where the message is sent from.

4.7.3.2 `TaskArgument* dsf::Task::taskArgument`

The arguments for the function pointer.

4.7.3.3 `TaskFunction* dsf::Task::taskFunction`

The function pointer.

4.7.3.4 `SynchronizedObject* dsf::Task::to`

Where the message is sent to.

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

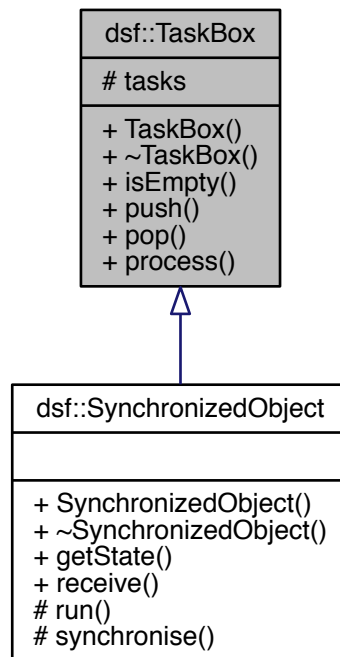
- Task.h
- Task.cpp

## 4.8 dsf::TaskBox Class Reference

A [dsf::Task](#) queue.

```
#include <TaskBox.h>
```

Inheritance diagram for dsf::TaskBox:



## Public Member Functions

- [TaskBox](#) ()
- virtual [~TaskBox](#) ()
- bool [isEmpty](#) ()
- void [push](#) ([Task](#) \*task)
- [Task](#) \* [pop](#) ()
- void [process](#) ()

## Protected Attributes

- `std::vector< Task * > * tasks`

### 4.8.1 Detailed Description

A [dsf::Task](#) queue.

The [dsf::TaskBox](#) contains a list of `def::Task` objects. It provides essential methods to control the list such as “push”, “pop”, and “isEmpty”.

### 4.8.2 Constructor & Destructor Documentation

#### 4.8.2.1 dsf::TaskBox::TaskBox ( )

Here is the caller graph for this function:



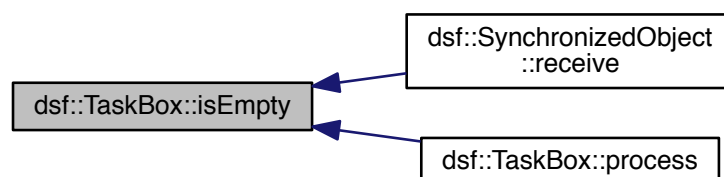
#### 4.8.2.2 dsf::TaskBox::~~TaskBox ( ) [virtual]

### 4.8.3 Member Function Documentation

#### 4.8.3.1 bool dsf::TaskBox::isEmpty ( )

Checks wheather the queue is empty or not.

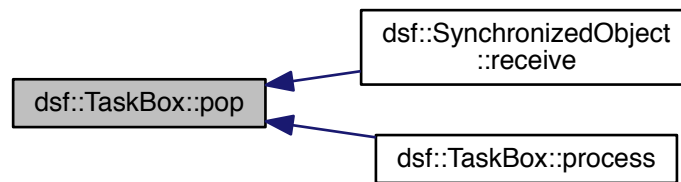
Here is the caller graph for this function:



#### 4.8.3.2 Task \* dsf::TaskBox::pop ( )

Pops out a task and returns it

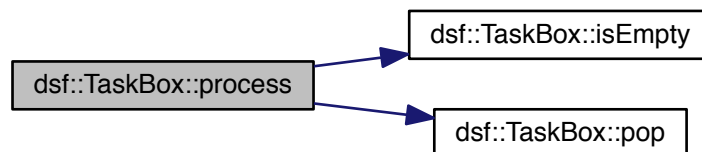
Here is the caller graph for this function:



#### 4.8.3.3 void dsf::TaskBox::process ( )

Pops out all tasks in the queue and executes them.

Here is the call graph for this function:



#### 4.8.3.4 void dsf::TaskBox::push ( dsf::Task \* task )

Pushes a task into the queue.

### 4.8.4 Member Data Documentation

#### 4.8.4.1 std::vector<Task\*>\* dsf::TaskBox::tasks [protected]

The list of [dsf::Task](#).

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

- TaskBox.h
- TaskBox.cpp



# Index

- ~DualStateFramework
  - dsf::DualStateFramework, [10](#)
- ~Synchronisable
  - dsf::Synchronisable, [17](#)
- ~SynchronizedObject
  - dsf::SynchronizedObject, [19](#)
- ~Task
  - dsf::Task, [23](#)
- ~TaskBox
  - dsf::TaskBox, [25](#)
- add
  - dsf::DualStateFramework, [10](#)
- DELETED
  - dsf::Runnable, [16](#)
- doOneFrame
  - dsf::DualStateFramework, [10](#)
- dsf, [7](#)
  - DualStateFramework, [8](#)
  - function, [8](#)
  - SynchronizedObject, [8](#)
  - Task, [8](#)
  - TaskArgument, [8](#)
  - TaskArgumentException, [8](#)
  - TaskBox, [8](#)
  - TaskFunction, [8](#)
- dsf::DualStateFramework, [9](#)
  - ~DualStateFramework, [10](#)
  - add, [10](#)
  - doOneFrame, [10](#)
  - DualStateFramework, [10](#)
  - getState, [11](#)
  - initialize, [11](#)
  - refresh, [11](#)
  - remove, [12](#)
  - run, [12](#)
  - send, [12](#)
  - setNumberOfThreads, [12](#)
  - start, [12](#)
- dsf::Lock, [13](#)
  - lock, [14](#)
  - locker, [14](#)
  - unlock, [14](#)
- dsf::Runnable, [14](#)
  - DELETED, [16](#)
  - getState, [16](#)
  - READY, [16](#)
  - RUNNING, [16](#)
  - run, [16](#)
  - STOPPED, [16](#)
  - State, [15](#)
- dsf::Synchronisable
  - ~Synchronisable, [17](#)
  - next, [17](#)
  - synchronise, [17](#)
- dsf::Synchronisable< T >, [16](#)
- dsf::SynchronizedObject, [17](#)
  - ~SynchronizedObject, [19](#)
  - DualStateFramework, [20](#)
  - getState, [19](#)
  - receive, [19](#)
  - run, [20](#)
  - synchronise, [20](#)
  - SynchronizedObject, [19](#)
- dsf::SynchronizedVar, [20](#)
  - operator=, [22](#)
  - synchronise, [22](#)
  - SynchronizedVar, [21](#)
- dsf::Task, [22](#)
  - ~Task, [23](#)
  - from, [23](#)
  - Task, [23](#)
  - taskArgument, [23](#)
  - taskFunction, [23](#)
  - to, [23](#)
- dsf::TaskBox, [23](#)
  - ~TaskBox, [25](#)
  - isEmpty, [25](#)
  - pop, [25](#)
  - process, [26](#)
  - push, [26](#)
  - TaskBox, [24](#)
  - tasks, [26](#)
- DualStateFramework
  - dsf, [8](#)
  - dsf::DualStateFramework, [10](#)
  - dsf::SynchronizedObject, [20](#)
- from
  - dsf::Task, [23](#)
- function
  - dsf, [8](#)
- getState
  - dsf::DualStateFramework, [11](#)
  - dsf::Runnable, [16](#)
  - dsf::SynchronizedObject, [19](#)

initialize  
     dsf::DualStateFramework, 11  
 isEmpty  
     dsf::TaskBox, 25  
 lock  
     dsf::Lock, 14  
 locker  
     dsf::Lock, 14  
 next  
     dsf::Synchronisable, 17  
 operator=  
     dsf::SynchronizedVar, 22  
 pop  
     dsf::TaskBox, 25  
 process  
     dsf::TaskBox, 26  
 push  
     dsf::TaskBox, 26  
 READY  
     dsf::Runnable, 16  
 RUNNING  
     dsf::Runnable, 16  
 receive  
     dsf::SynchronizedObject, 19  
 refresh  
     dsf::DualStateFramework, 11  
 remove  
     dsf::DualStateFramework, 12  
 run  
     dsf::DualStateFramework, 12  
     dsf::Runnable, 16  
     dsf::SynchronizedObject, 20  
 STOPPED  
     dsf::Runnable, 16  
 send  
     dsf::DualStateFramework, 12  
 setNumberOfThreads  
     dsf::DualStateFramework, 12  
 start  
     dsf::DualStateFramework, 12  
 State  
     dsf::Runnable, 15  
 synchronise  
     dsf::Synchronisable, 17  
     dsf::SynchronizedObject, 20  
     dsf::SynchronizedVar, 22  
 SynchronizedObject  
     dsf, 8  
     dsf::SynchronizedObject, 19  
 SynchronizedVar  
     dsf::SynchronizedVar, 21  
 Task  
     dsf, 8  
     dsf::Task, 23  
 TaskArgument  
     dsf, 8  
 taskArgument  
     dsf::Task, 23  
 TaskArgumentException  
     dsf, 8  
 TaskBox  
     dsf, 8  
     dsf::TaskBox, 24  
 TaskFunction  
     dsf, 8  
 taskFunction  
     dsf::Task, 23  
 tasks  
     dsf::TaskBox, 26  
 to  
     dsf::Task, 23  
 unlock  
     dsf::Lock, 14