

Dual State Framework

Generated by Doxygen 1.8.9.1

Sun Apr 5 2015 00:57:01

Contents

1	Main Page	1
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
2	Mac os X	3
2.1	Install Dependencies	3
2.1.1	yctools	3
2.1.2	Intel tbb	3
2.2	Install Dual State Framework	3
2.3	Use DSF in Xcode	3
3	Microsoft Windows	5
3.1	Install Dependencies	5
3.1.1	yctools	5
3.1.2	Intel tbb	5
3.2	Install Dual State Framework	5
3.3	Use DSF in Visual Studio	5
3.3.1	Add additional header path	5
3.3.2	Add Dependencies	6
4	Linux	7
4.1	Install Dependencies	7
4.1.1	yctools	7
4.1.2	Intel tbb	7
4.2	Install Dual State Framework	7
5	Compile source code	9
5.1	Pre-Build	9
5.1.1	Get source code	9

5.1.2	Generate project for compiler	9
5.2	Build the project	9
6	Namespace Index	11
6.1	Namespace List	11
7	Hierarchical Index	13
7.1	Class Hierarchy	13
8	Class Index	15
8.1	Class List	15
9	Namespace Documentation	17
9.1	dsf Namespace Reference	17
9.1.1	Typedef Documentation	18
9.1.1.1	function	18
9.1.1.2	TaskArgument	18
9.1.1.3	TaskArgumentException	18
9.1.1.4	TaskFunction	18
9.1.2	Variable Documentation	18
9.1.2.1	DualStateFramework	18
9.1.2.2	SynchronizedObject	18
9.1.2.3	Task	18
9.1.2.4	TaskBox	18
10	Class Documentation	19
10.1	dsf::DualStateFramework Class Reference	19
10.1.1	Detailed Description	20
10.1.2	Example	20
10.1.3	Constructor & Destructor Documentation	20
10.1.3.1	DualStateFramework	20
10.1.3.2	~DualStateFramework	20
10.1.4	Member Function Documentation	20
10.1.4.1	add	20
10.1.4.2	doOneFrame	20
10.1.4.3	getState	21
10.1.4.4	initialize	21
10.1.5	Example	21
10.1.5.1	refresh	21
10.1.5.2	remove	21
10.1.5.3	run	21
10.1.5.4	send	21

10.1.5.5	setNumberOfThreads	21
10.1.5.6	start	21
10.2	dsf::Lock Class Reference	22
10.2.1	Detailed Description	22
10.2.2	Example	22
10.2.3	Member Function Documentation	23
10.2.3.1	lock	23
10.2.3.2	unlock	23
10.2.4	Member Data Documentation	23
10.2.4.1	locker	23
10.3	dsf::Runnable Class Reference	23
10.3.1	Detailed Description	24
10.3.2	Member Enumeration Documentation	24
10.3.2.1	State	25
10.3.3	Member Function Documentation	25
10.3.3.1	getState	25
10.3.3.2	run	25
10.4	dsf::Synchronisable< T > Class Template Reference	25
10.4.1	Detailed Description	25
10.4.2	Example	26
10.4.3	Constructor & Destructor Documentation	26
10.4.3.1	~Synchronisable	26
10.4.4	Member Function Documentation	26
10.4.4.1	synchronise	26
10.4.5	Member Data Documentation	26
10.4.5.1	next	26
10.5	dsf::SynchronizedObject Class Reference	26
10.5.1	Detailed Description	27
10.5.2	Example	27
10.5.3	Constructor & Destructor Documentation	28
10.5.3.1	SynchronizedObject	28
10.5.3.2	~SynchronizedObject	28
10.5.4	Member Function Documentation	28
10.5.4.1	getState	28
10.5.4.2	receive	28
10.5.4.3	run	28
10.5.5	Example	28
10.5.5.1	synchronise	28
10.5.6	Friends And Related Function Documentation	29
10.5.6.1	DualStateFramework	29

10.6 dsf::SynchronizedVar Class Reference	29
10.6.1 Detailed Description	29
10.6.2 Example	30
10.6.3 Constructor & Destructor Documentation	30
10.6.3.1 SynchronizedVar	30
10.6.4 Member Function Documentation	30
10.6.4.1 operator=	30
10.6.4.2 synchronise	30
10.7 dsf::Task Class Reference	30
10.7.1 Detailed Description	31
10.7.2 Constructor & Destructor Documentation	31
10.7.2.1 Task	31
10.7.2.2 ~Task	31
10.7.3 Member Data Documentation	31
10.7.3.1 from	31
10.7.3.2 taskArgument	31
10.7.3.3 taskFunction	31
10.7.3.4 to	31
10.8 dsf::TaskBox Class Reference	32
10.8.1 Detailed Description	32
10.8.2 Constructor & Destructor Documentation	33
10.8.2.1 TaskBox	33
10.8.2.2 ~TaskBox	33
10.8.3 Member Function Documentation	33
10.8.3.1 isEmpty	33
10.8.3.2 pop	33
10.8.3.3 process	33
10.8.3.4 push	33
10.8.4 Member Data Documentation	33
10.8.4.1 tasks	33
Index	35

Chapter 1

Main Page

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;
}

```


Chapter 2

Mac os X

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

2.1 Install Dependencies

2.1.1 yctools

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

2.1.2 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".

2.2 Install Dual State Framework

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

2.3 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.

Chapter 3

Microsoft Windows

Dual State Framework uses [yctools](#) and [Intel tbb](#) . Before the installation you need to install them first.

3.1 Install Dependencies

3.1.1 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.

3.1.2 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".

3.2 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.

3.3 Use DSF in Visual Studio

3.3.1 Add additional header path

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

3.3.2 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.

Chapter 4

Linux

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

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

4.1 Install Dependencies

4.1.1 yctools

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

Download the deb file and install it.

4.1.2 Intel tbb

In terminal or console, type

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

4.2 Install Dual State Framework

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

Download the deb file and install it.

Chapter 5

Compile source code

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

5.1 Pre-Build

5.1.1 Get source code

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

5.1.2 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>.

5.2 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!

Chapter 6

Namespace Index

6.1 Namespace List

Here is a list of all namespaces with brief descriptions:

dsf	17
-------------------------------	----

Chapter 7

Hierarchical Index

7.1 Class Hierarchy

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

Any	
dsf::SynchronizedVar	29
dsf::Lock	22
dsf::SynchronizedObject	26
dsf::SynchronizedVar	29
dsf::Runnable	23
dsf::DualStateFramework	19
dsf::SynchronizedObject	26
dsf::Synchronisable< T >	25
dsf::Synchronisable< TaskBox >	25
dsf::SynchronizedObject	26
dsf::Synchronisable< yc::Any >	25
dsf::SynchronizedVar	29
dsf::Task	30
dsf::TaskBox	32
dsf::SynchronizedObject	26

Chapter 8

Class Index

8.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

dsf::DualStateFramework	
The starting pointer for the framework is the abstract class dsf::DualStateFramework	19
dsf::Lock	
Locking variables	22
dsf::Runnable	
Executing messages	23
dsf::Synchronisable< T >	
Synchronising two states	25
dsf::SynchronizedObject	
Dual state object interface	26
dsf::SynchronizedVar	
A Class which implements dsf::Synchronisable	29
dsf::Task	
Class Task	30
dsf::TaskBox	
A dsf::Task queue	32

Chapter 9

Namespace Documentation

9.1 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](#)

9.1.1 Typedef Documentation

9.1.1.1 `typedef void dsf::function`

9.1.1.2 `typedef yc::Any dsf::TaskArgument`

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

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

9.1.2 Variable Documentation

9.1.2.1 `class DSF_API dsf::DualStateFramework`

9.1.2.2 `class DSF_API dsf::SynchronizedObject`

9.1.2.3 `class DSF_API dsf::Task`

9.1.2.4 `class DSF_API dsf::TaskBox`

Chapter 10

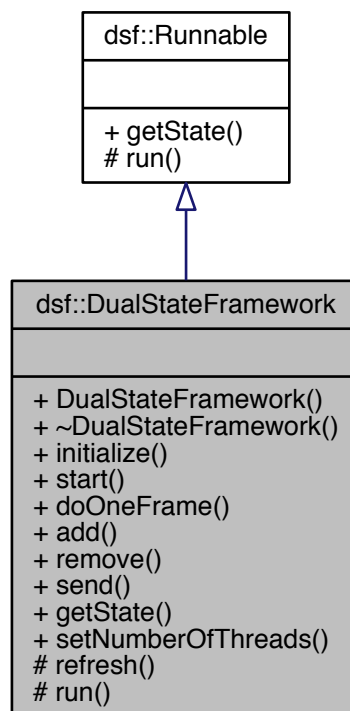
Class Documentation

10.1 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

10.1.1 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.).

10.1.2 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
```

10.1.3 Constructor & Destructor Documentation

10.1.3.1 dsf::DualStateFramework::DualStateFramework ()

10.1.3.2 dsf::DualStateFramework::~~DualStateFramework ()

10.1.4 Member Function Documentation

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

Add a [SynchronizedObject](#).

10.1.4.2 void dsf::DualStateFramework::doOneFrame ()

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

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

Return the state of the object.

Implements [dsf::Runnable](#).

10.1.4.4 virtual void dsf::DualStateFramework::initialize () [pure virtual]

For Signing TaskFunction Pointers

10.1.5 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);
});
```

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

Clear all SynchronizedObjects which is marked as DELETE

10.1.5.2 void dsf::DualStateFramework::remove (SynchronizedObject * syncObj)

Remove a [SynchronizedObject](#).

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

Start all SynchronizedObjects associated.

Implements [dsf::Runnable](#).

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

Send messages

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

Set the number of threads. 0 is automatic.

10.1.5.6 void dsf::DualStateFramework::start ()

Start all SynchronizedObjects associated.

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

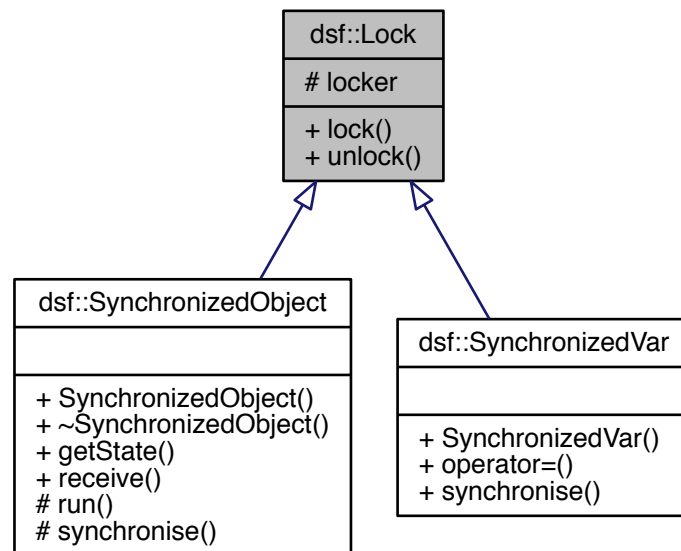
- /Users/yuchen/Repositories/DualStateFramework/dsf/include/dsf/DualStateFramework.h

10.2 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`

10.2.1 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).

10.2.2 Example

```

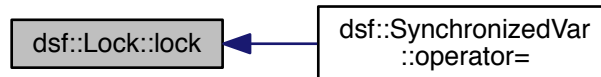
dsf->lock();
dsf->drawables->push_back(syncObj); //the object drawables is locked
dsf->unlock();
  
```

10.2.3 Member Function Documentation

10.2.3.1 void dsf::Lock::lock ()

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

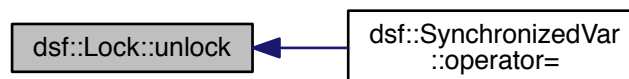
Here is the caller graph for this function:



10.2.3.2 void dsf::Lock::unlock ()

Unlocks all the objects.

Here is the caller graph for this function:



10.2.4 Member Data Documentation

10.2.4.1 std::mutex dsf::Lock::locker [protected]

The locker

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

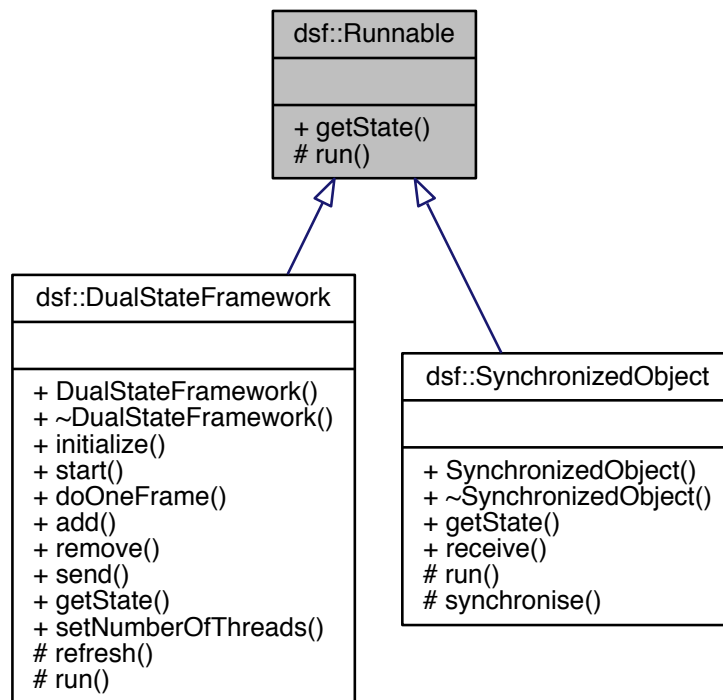
- /Users/yuchen/Repositories/DualStateFramework/dsf/include/dsf/Lock.h

10.3 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

10.3.1 Detailed Description

Executing messages.

The interface provides a run method which executes messages.

10.3.2 Member Enumeration Documentation

10.3.2.1 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

10.3.3 Member Function Documentation

10.3.3.1 virtual State dsf::Runnable::getState () [pure virtual]

Returns the current state.

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

10.3.3.2 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:

- /Users/yuchen/Repositories/DualStateFramework/dsf/include/dsf/Runnable.h

10.4 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](#)

10.4.1 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.

10.4.2 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;
    }
}
```

10.4.3 Constructor & Destructor Documentation

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

10.4.4 Member Function Documentation

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

Signs current value to next value.

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

10.4.5 Member Data Documentation

10.4.5.1 `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:

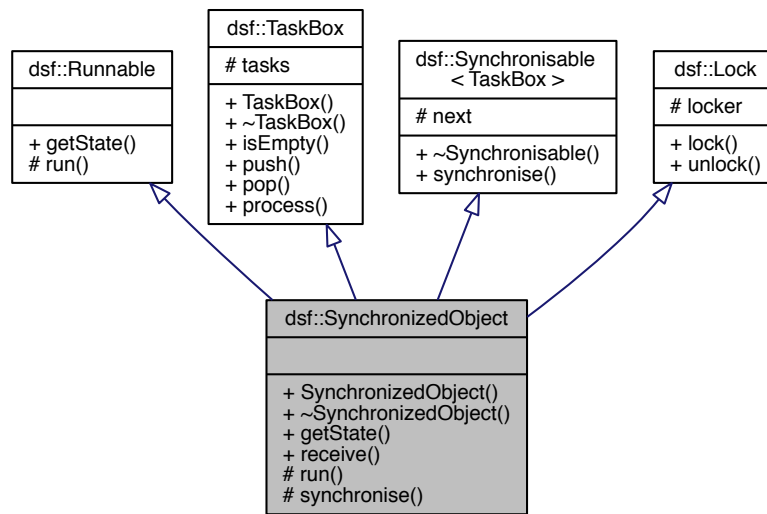
- `/Users/yuchen/Repositories/DualStateFramework/dsf/include/dsf/Synchronisable.h`

10.5 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

10.5.1 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.

10.5.2 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();
}

```

10.5.3 Constructor & Destructor Documentation

10.5.3.1 `dsf::SynchronizedObject::SynchronizedObject ()`

10.5.3.2 `virtual dsf::SynchronizedObject::~~SynchronizedObject ()` [virtual]

10.5.4 Member Function Documentation

10.5.4.1 `State dsf::SynchronizedObject::getState ()` [override],[virtual]

Returns the current state.

Implements [dsf::Runnable](#).

10.5.4.2 `int dsf::SynchronizedObject::receive ()`

Returns the number of message received

10.5.4.3 `virtual void dsf::SynchronizedObject::run ()` [override],[protected],[pure virtual]

Executes messages

10.5.5 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](#).

10.5.5.1 `void dsf::SynchronizedObject::synchronise ()` [override],[protected],[virtual]

Signs current taskbox to next taskbox.

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

10.5.6 Friends And Related Function Documentation

10.5.6.1 friend class DualStateFramework [friend]

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

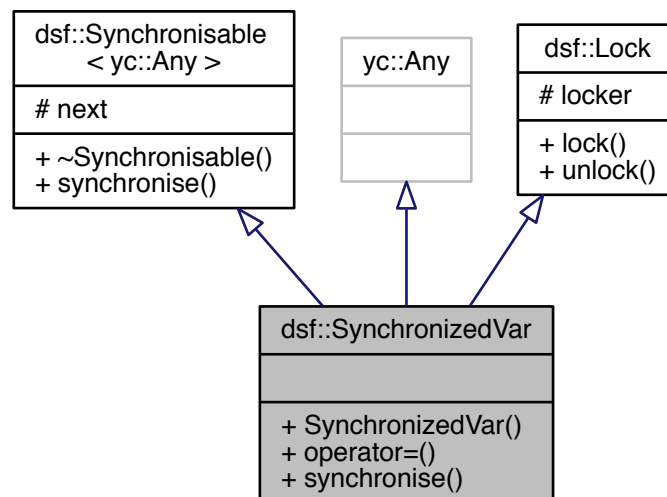
- /Users/yuchen/Repositories/DualStateFramework/dsf/include/dsf/SynchronizedObject.h

10.6 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

10.6.1 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”.

10.6.2 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
```

10.6.3 Constructor & Destructor Documentation

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

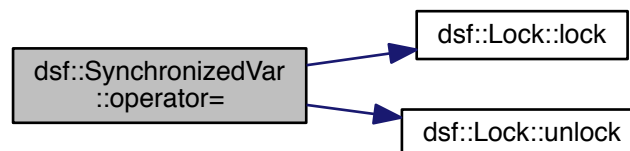
The value of “current” and the value of “next” is initialized as “value”.

10.6.4 Member Function Documentation

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

Signs a value to “next”.

Here is the call graph for this function:



10.6.4.2 `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 file:

- /Users/yuchen/Repositories/DualStateFramework/dsf/include/dsf/SynchronizedVar.h

10.7 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

10.7.1 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.

10.7.2 Constructor & Destructor Documentation

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

10.7.2.2 [dsf::Task::~~Task](#) ()

10.7.3 Member Data Documentation

10.7.3.1 [SynchronizedObject*](#) [dsf::Task::from](#)

Where the message is sent from.

10.7.3.2 [TaskArgument*](#) [dsf::Task::taskArgument](#)

The arguments for the function pointer.

10.7.3.3 [TaskFunction*](#) [dsf::Task::taskFunction](#)

The function pointer.

10.7.3.4 [SynchronizedObject*](#) [dsf::Task::to](#)

Where the message is sent to.

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

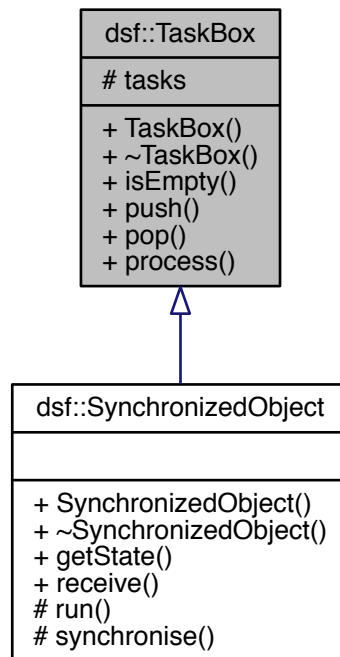
- [/Users/yuchen/Repositories/DualStateFramework/dsf/include/dsf/Task.h](#)

10.8 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`

10.8.1 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”.

10.8.2 Constructor & Destructor Documentation

10.8.2.1 `dsf::TaskBox::TaskBox ()`

10.8.2.2 `virtual dsf::TaskBox::~~TaskBox ()` `[virtual]`

10.8.3 Member Function Documentation

10.8.3.1 `bool dsf::TaskBox::isEmpty ()`

Checks wheather the queue is empty or not.

10.8.3.2 `Task* dsf::TaskBox::pop ()`

Pops out a task and returns it

10.8.3.3 `void dsf::TaskBox::process ()`

Pops out all tasks in the queue and executes them.

10.8.3.4 `void dsf::TaskBox::push (Task * task)`

Pushes a task into the queue.

10.8.4 Member Data Documentation

10.8.4.1 `std::vector<Task*>* dsf::TaskBox::tasks` `[protected]`

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

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

- `/Users/yuchen/Repositories/DualStateFramework/dsf/include/dsf/TaskBox.h`

Index

- ~DualStateFramework
 - dsf::DualStateFramework, [20](#)
- ~Synchronisable
 - dsf::Synchronisable, [26](#)
- ~SynchronizedObject
 - dsf::SynchronizedObject, [28](#)
- ~Task
 - dsf::Task, [31](#)
- ~TaskBox
 - dsf::TaskBox, [33](#)
- add
 - dsf::DualStateFramework, [20](#)
- DELETED
 - dsf::Runnable, [25](#)
- doOneFrame
 - dsf::DualStateFramework, [20](#)
- dsf, [17](#)
 - DualStateFramework, [18](#)
 - function, [18](#)
 - SynchronizedObject, [18](#)
 - Task, [18](#)
 - TaskArgument, [18](#)
 - TaskArgumentException, [18](#)
 - TaskBox, [18](#)
 - TaskFunction, [18](#)
- dsf::DualStateFramework, [19](#)
 - ~DualStateFramework, [20](#)
 - add, [20](#)
 - doOneFrame, [20](#)
 - DualStateFramework, [20](#)
 - getState, [20](#)
 - initialize, [21](#)
 - refresh, [21](#)
 - remove, [21](#)
 - run, [21](#)
 - send, [21](#)
 - setNumberOfThreads, [21](#)
 - start, [21](#)
- dsf::Lock, [22](#)
 - lock, [23](#)
 - locker, [23](#)
 - unlock, [23](#)
- dsf::Runnable, [23](#)
 - DELETED, [25](#)
 - getState, [25](#)
 - READY, [25](#)
 - RUNNING, [25](#)
 - run, [25](#)
- STOPPED, [25](#)
- State, [24](#)
- dsf::Synchronisable
 - ~Synchronisable, [26](#)
 - next, [26](#)
 - synchronise, [26](#)
- dsf::Synchronisable< T >, [25](#)
- dsf::SynchronizedObject, [26](#)
 - ~SynchronizedObject, [28](#)
 - DualStateFramework, [29](#)
 - getState, [28](#)
 - receive, [28](#)
 - run, [28](#)
 - synchronise, [28](#)
 - SynchronizedObject, [28](#)
- dsf::SynchronizedVar, [29](#)
 - operator=, [30](#)
 - synchronise, [30](#)
 - SynchronizedVar, [30](#)
- dsf::Task, [30](#)
 - ~Task, [31](#)
 - from, [31](#)
 - Task, [31](#)
 - taskArgument, [31](#)
 - taskFunction, [31](#)
 - to, [31](#)
- dsf::TaskBox, [32](#)
 - ~TaskBox, [33](#)
 - isEmpty, [33](#)
 - pop, [33](#)
 - process, [33](#)
 - push, [33](#)
 - TaskBox, [33](#)
 - tasks, [33](#)
- DualStateFramework
 - dsf, [18](#)
 - dsf::DualStateFramework, [20](#)
 - dsf::SynchronizedObject, [29](#)
- from
 - dsf::Task, [31](#)
- function
 - dsf, [18](#)
- getState
 - dsf::DualStateFramework, [20](#)
 - dsf::Runnable, [25](#)
 - dsf::SynchronizedObject, [28](#)
- initialize

- dsf::DualStateFramework, 21
- isEmpty
 - dsf::TaskBox, 33
- lock
 - dsf::Lock, 23
- locker
 - dsf::Lock, 23
- next
 - dsf::Synchronisable, 26
- operator=
 - dsf::SynchronizedVar, 30
- pop
 - dsf::TaskBox, 33
- process
 - dsf::TaskBox, 33
- push
 - dsf::TaskBox, 33
- READY
 - dsf::Runnable, 25
- RUNNING
 - dsf::Runnable, 25
- receive
 - dsf::SynchronizedObject, 28
- refresh
 - dsf::DualStateFramework, 21
- remove
 - dsf::DualStateFramework, 21
- run
 - dsf::DualStateFramework, 21
 - dsf::Runnable, 25
 - dsf::SynchronizedObject, 28
- STOPPED
 - dsf::Runnable, 25
- send
 - dsf::DualStateFramework, 21
- setNumberOfThreads
 - dsf::DualStateFramework, 21
- start
 - dsf::DualStateFramework, 21
- State
 - dsf::Runnable, 24
- synchronise
 - dsf::Synchronisable, 26
 - dsf::SynchronizedObject, 28
 - dsf::SynchronizedVar, 30
- SynchronizedObject
 - dsf, 18
 - dsf::SynchronizedObject, 28
- SynchronizedVar
 - dsf::SynchronizedVar, 30
- Task
 - dsf, 18
 - dsf::Task, 31
- TaskArgument
 - dsf, 18
- taskArgument
 - dsf::Task, 31
- TaskArgumentException
 - dsf, 18
- TaskBox
 - dsf, 18
 - dsf::TaskBox, 33
- TaskFunction
 - dsf, 18
- taskFunction
 - dsf::Task, 31
- tasks
 - dsf::TaskBox, 33
- to
 - dsf::Task, 31
- unlock
 - dsf::Lock, 23