THE BIOROBOTICS INSTITUTE

Sant'Anna
School of Advanced Studies – Pisa

YARP - BufferedPort

Egidio Falotico

Buffered Port

A mini-server for performing network communication in the background. It is an asynchronous communication method.

By default a BufferedPort attempts to reduce latency between senders and receivers. To do so messages may be dropped by the writer if BufferedPort::write is called too quickly. The reader may also drop old messages if BufferedPort::read is not called fast enough, so that new messages can travel with high priority. This policy is sometimes called Oldest Packet Drop (ODP).

Buffered Port II

You can change the buffering policy.

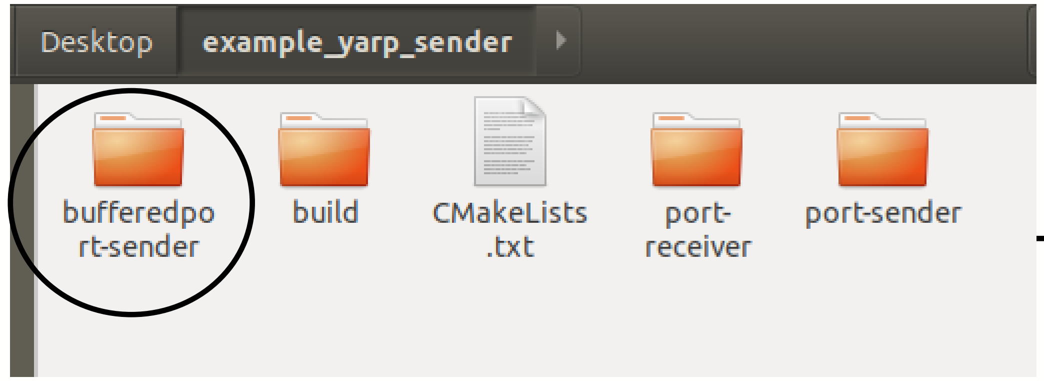
Use **BufferedPort::writeStrict()** when writing to a port, this waits for pending transmissions to be finished before writing new data.

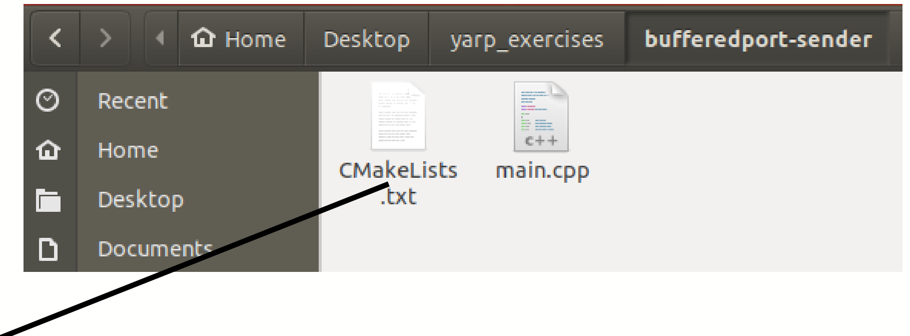
Call **BufferedPort::setStrict()** to change the buffering policy to FIFO at the receiver side. In this way all messages will be stored inside the **BufferedPort** and delivered to the reader. Pay attention that in this case a slow reader may cause increasing latency and memory use.

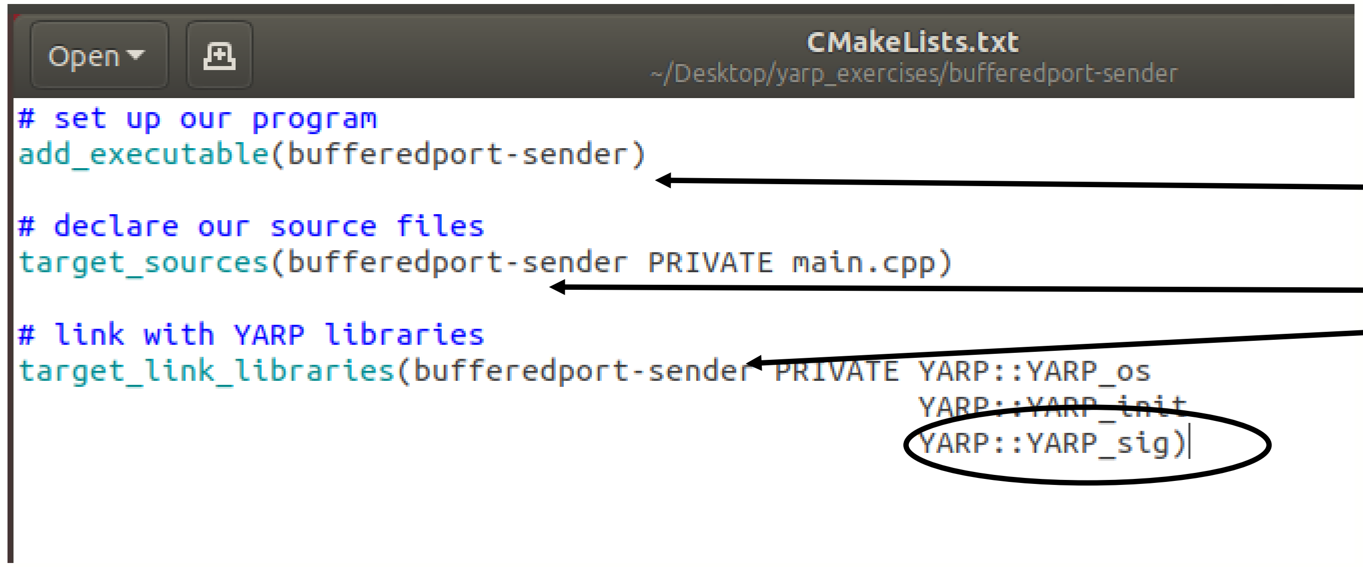
https://www.yarp.it/git-master/classyarp_1_1os_1_1BufferedPort.html#details

Example: Yarp sender using bufferedPorts

Make a copy of your folder port-sender and name it bufferedport-sender







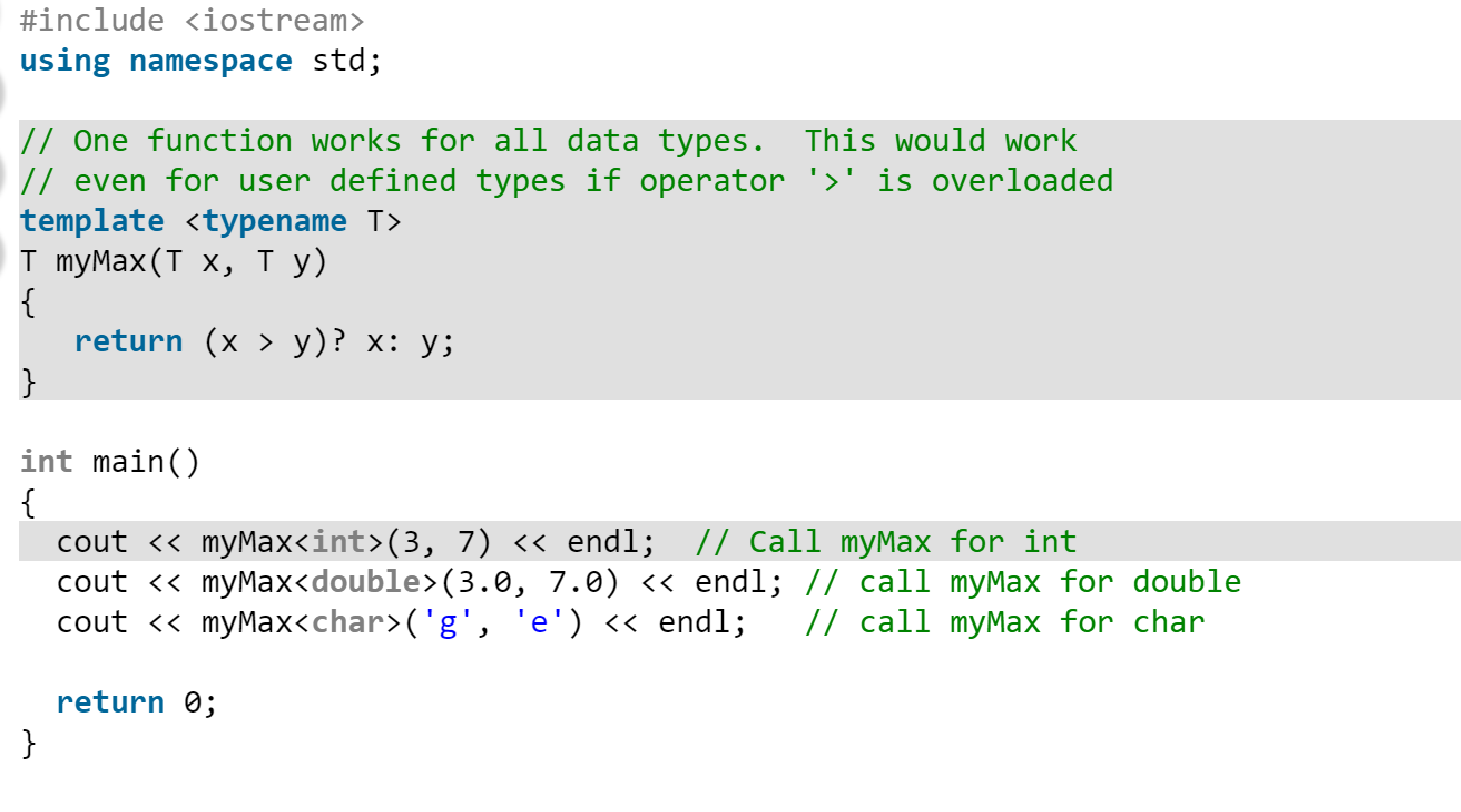
```
# set up our program
add_executable(bufferedport-sender)

# declare our source files
target_sources(bufferedport-sender PRIVATE main.cpp)

# link with YARP libraries
target_link_libraries(bufferedport-sender PRIVATE YARP::YARP_os
                                                  YARP::YARP_init
                                                  YARP::YARP_sig)
```

Change to bufferedport-sender

Add sig library since we will use the class Vector

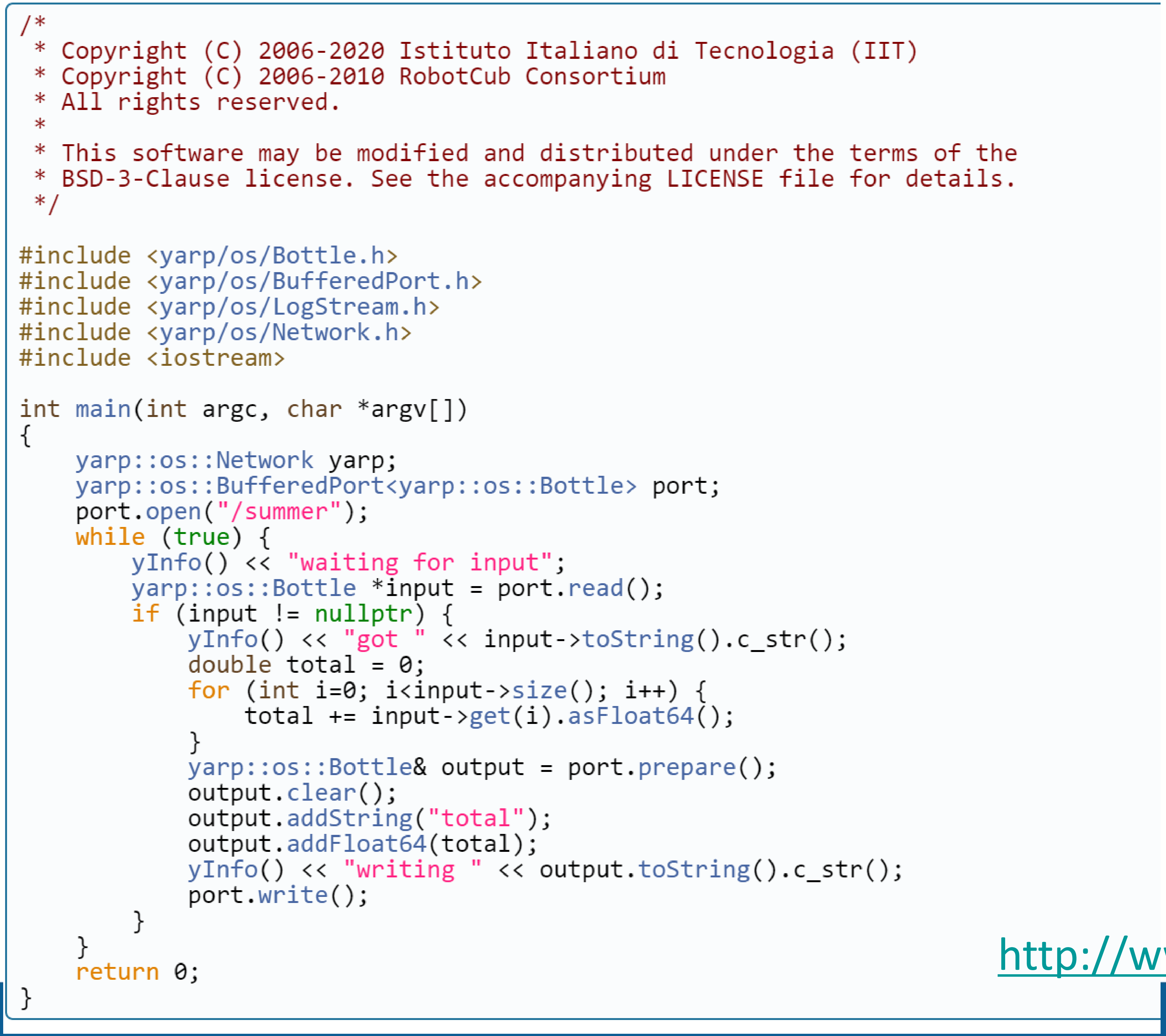
BufferedPort is a Class Template

A template is a powerful tool in C++. The idea is to pass data type as a parameter so that we do not need to write the same code for different data types.



```
#include <iostream>
using namespace std;

// One function works for all data types.  This would work
// even for user defined types if operator '>' is overloaded
template <typename T>
T myMax(T x, T y)
{
   return (x > y)? x: y;
}

int main()
{
  cout << myMax<int>(3, 7) << endl;  // Call myMax for int
  cout << myMax<double>(3.0, 7.0) << endl; // call myMax for double
  cout << myMax<char>('g', 'e') << endl;   // call myMax for char

  return 0;
}
```

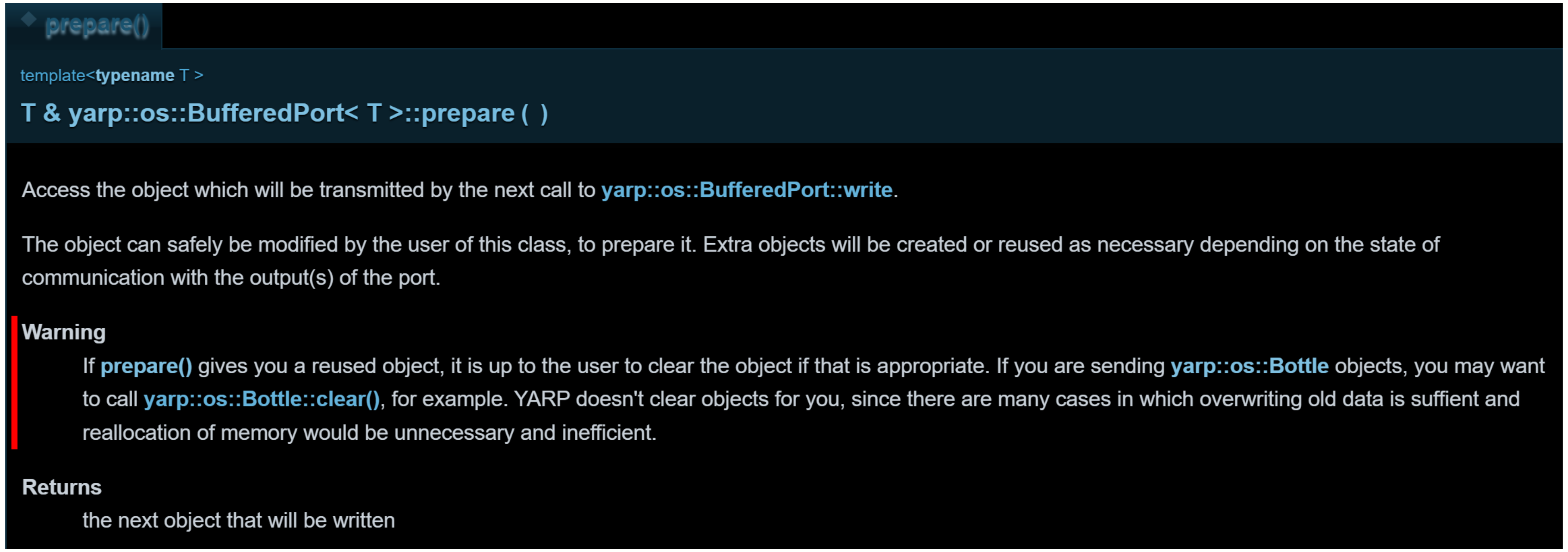
Buffered Ports (example)



```
/*
 * Copyright (C) 2006-2020 Istituto Italiano di Tecnologia (IIT)
 * Copyright (C) 2006-2010 RobotCub Consortium
 * All rights reserved.
 *
 * This software may be modified and distributed under the terms of the
 * BSD-3-Clause license. See the accompanying LICENSE file for details.
 */

#include <yarp/os/Bottle.h>
#include <yarp/os/BufferedPort.h>
#include <yarp/os/LogStream.h>
#include <yarp/os/Network.h>
#include <iostream>

int main(int argc, char *argv[])
{
    yarp::os::Network yarp;
    yarp::os::BufferedPort<yarp::os::Bottle> port;
    port.open("/summer");
    while (true) {
        yInfo() << "waiting for input";
        yarp::os::Bottle *input = port.read();
        if (input != nullptr) {
            yInfo() << "got " << input->toString().c_str();
            double total = 0;
            for (int i=0; i<input->size(); i++) {
                total += input->get(i).asFloat64();
            }
            yarp::os::Bottle& output = port.prepare();
            output.clear();
            output.addString("total");
            output.addFloat64(total);
            yInfo() << "writing " << output.toString().c_str();
            port.write();
        }
    }
    return 0;
}
```

http://www.yarp.it/latest/classyarp_1_1os_1_1BufferedPort.html

Writing with BufferedPort



prepare()

template<typename T >

T & yarp::os::BufferedPort< T >::prepare ( )

Access the object which will be transmitted by the next call to yarp::os::BufferedPort::write.

The object can safely be modified by the user of this class, to prepare it. Extra objects will be created or reused as necessary depending on the state of communication with the output(s) of the port.

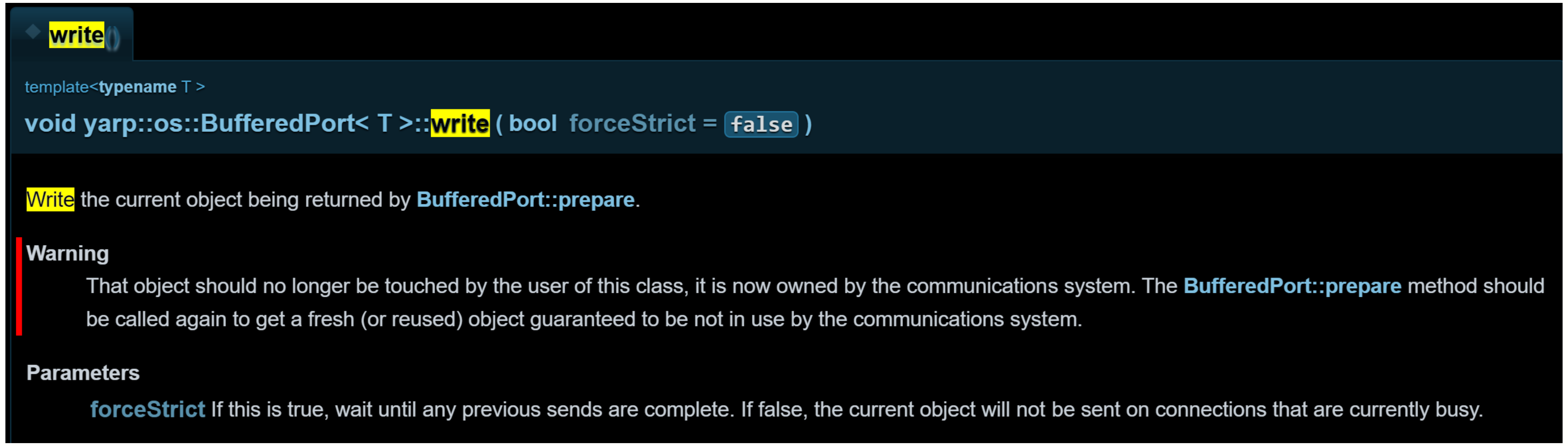
Warning

If prepare() gives you a reused object, it is up to the user to clear the object if that is appropriate. If you are sending yarp::os::Bottle objects, you may want to call yarp::os::Bottle::clear(), for example. YARP doesn't clear objects for you, since there are many cases in which overwriting old data is suffient and reallocation of memory would be unnecessary and inefficient.

Returns

the next object that will be written

Return a reference to the object to be sent. It is needed to call it before the write.



write()

template<typename T >

void yarp::os::BufferedPort< T >::write (bool forceStrict = false)

Write the current object being returned by BufferedPort::prepare.

Warning

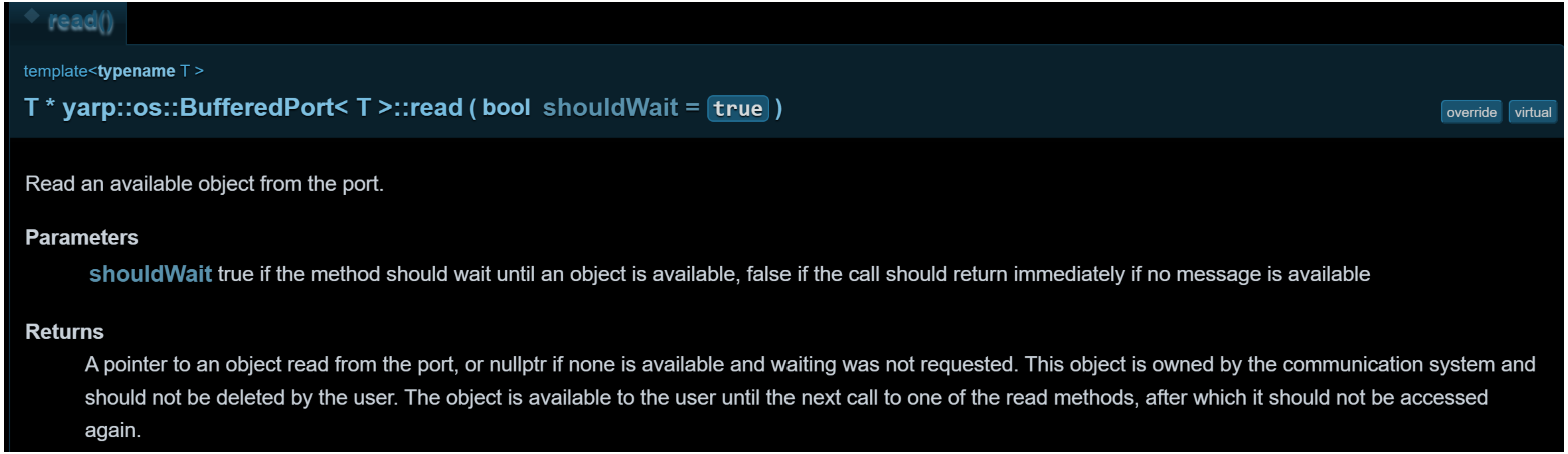
That object should no longer be touched by the user of this class, it is now owned by the communications system. The BufferedPort::prepare method should be called again to get a fresh (or reused) object guaranteed to be not in use by the communications system.

Parameters

forceStrict If this is true, wait until any previous sends are complete. If false, the current object will not be sent on connections that are currently busy.

The only parameter is to set the *forceStrict* to true

Read with BufferedPort



◆ read()

template<**typename** T >

T * yarp::os::BufferedPort< T >::read (bool shouldWait = `true`)

Read an available object from the port.

Parameters

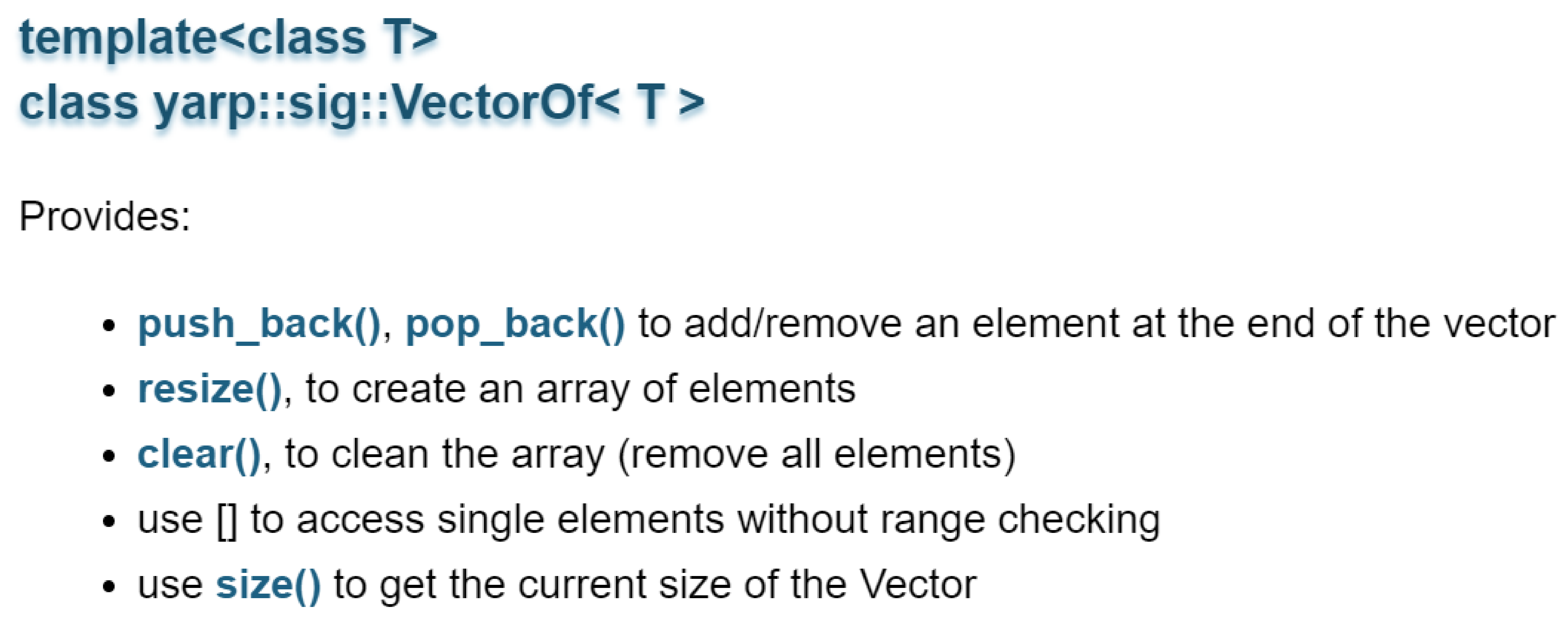
shouldWait true if the method should wait until an object is available, false if the call should return immediately if no message is available

Returns

A pointer to an object read from the port, or nullptr if none is available and waiting was not requested. This object is owned by the communication system and should not be deleted by the user. The object is available to the user until the next call to one of the read methods, after which it should not be accessed again.

Returns a pointer read from the port

Vector



template<class T>
class yarp::sig::VectorOf< T >

Provides:

- **push_back()**, **pop_back()** to add/remove an element at the end of the vector
- **resize()**, to create an array of elements
- **clear()**, to clean the array (remove all elements)
- use [] to access single elements without range checking
- use **size()** to get the current size of the Vector

http://www.yarp.it/latest/classyarp_1_1sig_1_1VectorOf.html

How to use Vector with a (Buffered)Port in the sender

```
 #include <yarp/sig/Vector.h>
  using namespace yarp::sig;
// the vector is given by the port itself
      Vector& v = port.prepare();

      // fill the vector
      v.resize(1);
      v[0] = count;

 // send message
      port.write();
```

How to use Vector with a (Buffered)Port in the receiver

```
Vector* v = port.read(false);

// check if there is actually something
      if (v)

// size of the vector
 (*v).size()
```

Exercise

1) Implement the sender that sends messages in a BufferedPort containing:
   - **vector**. The values are random generated in a range (1-30).
     The size of the vector is 20.

2) Create a receiver that connects to the sender port, receive the messages and print the median.

3) What happens to the receiver when you set the parameter of the method *read(true)* to *true* or *false for the buffered port*?

Generation random number

```
        #include <time.h>

int value;

 /* initialize random seed: */
 srand (time(NULL));

 /* generate number between 1 and 10: */
value= rand() % 10 + 1;
```