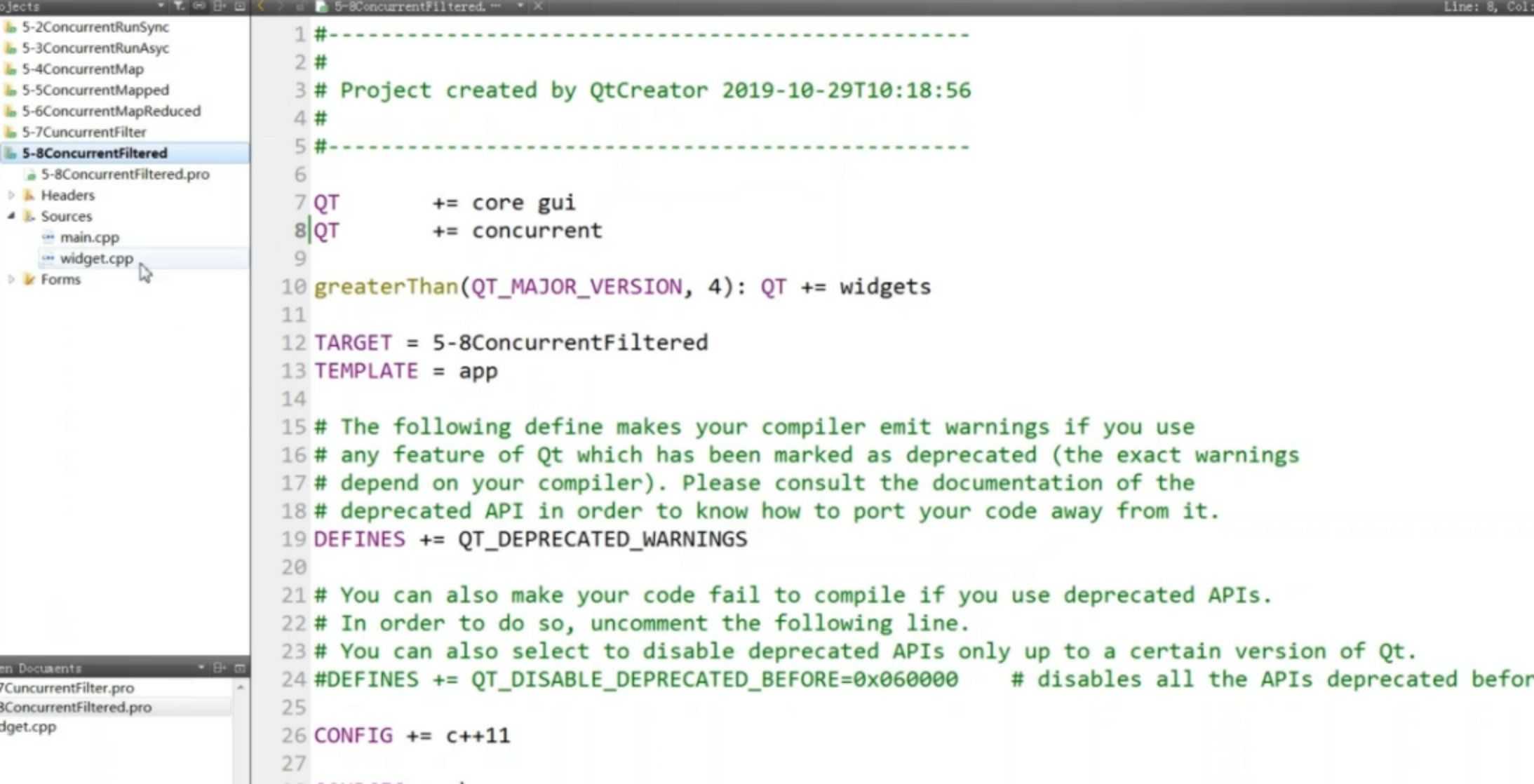
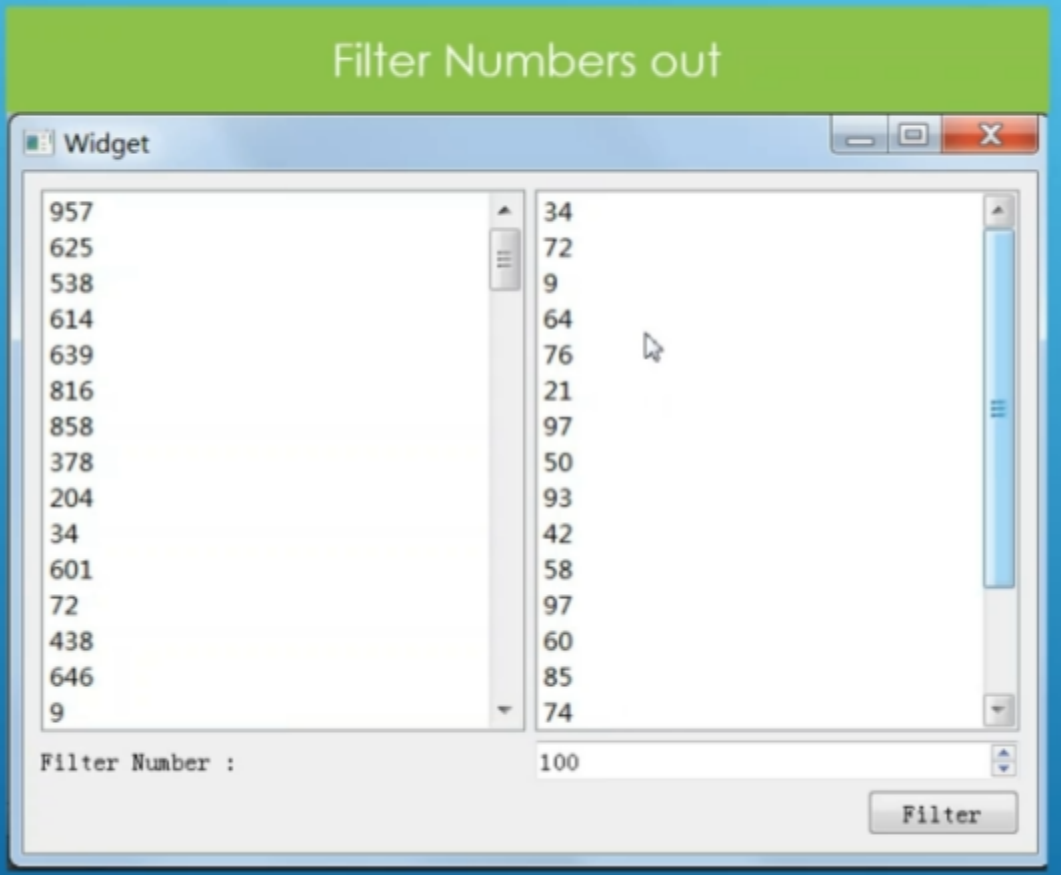

5-2ConcurrentRunSync
5-3ConcurrentRunAsyc
5-4ConcurrentMap
5-5ConcurrentMapped
5-6ConcurrentMapReduced
5-7CuncurrentFilter
5-8ConcurrentFiltered
5-8ConcurrentFiltered.pro
Headers
Sources
main.cpp
widget.cpp
Forms
5-8ConcurrentFiltered.
Line: 8, Col
#---
#
Project created by QtCreator 2019-10-29T10:18:56
#
#---
QT   += core gui
QT   += concurrent
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = 5-8ConcurrentFiltered
TEMPLATE = app
# The following define makes your compiler emit warnings if you use
# any feature of Qt which has been marked as deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS
You can also make your code fail to compile if you use deprecated APIs.
In order to do so, uncomment the following line.
You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000   # disables all the APIs deprecated befor
CONFIG += c++11
7CuncurrentFilter.pro
8ConcurrentFiltered.pro
dget.cpp

Qt Concurrent::filterReduce

Filtering [Parallelized across cores]

Combining results of the filters

Filter Numbers out



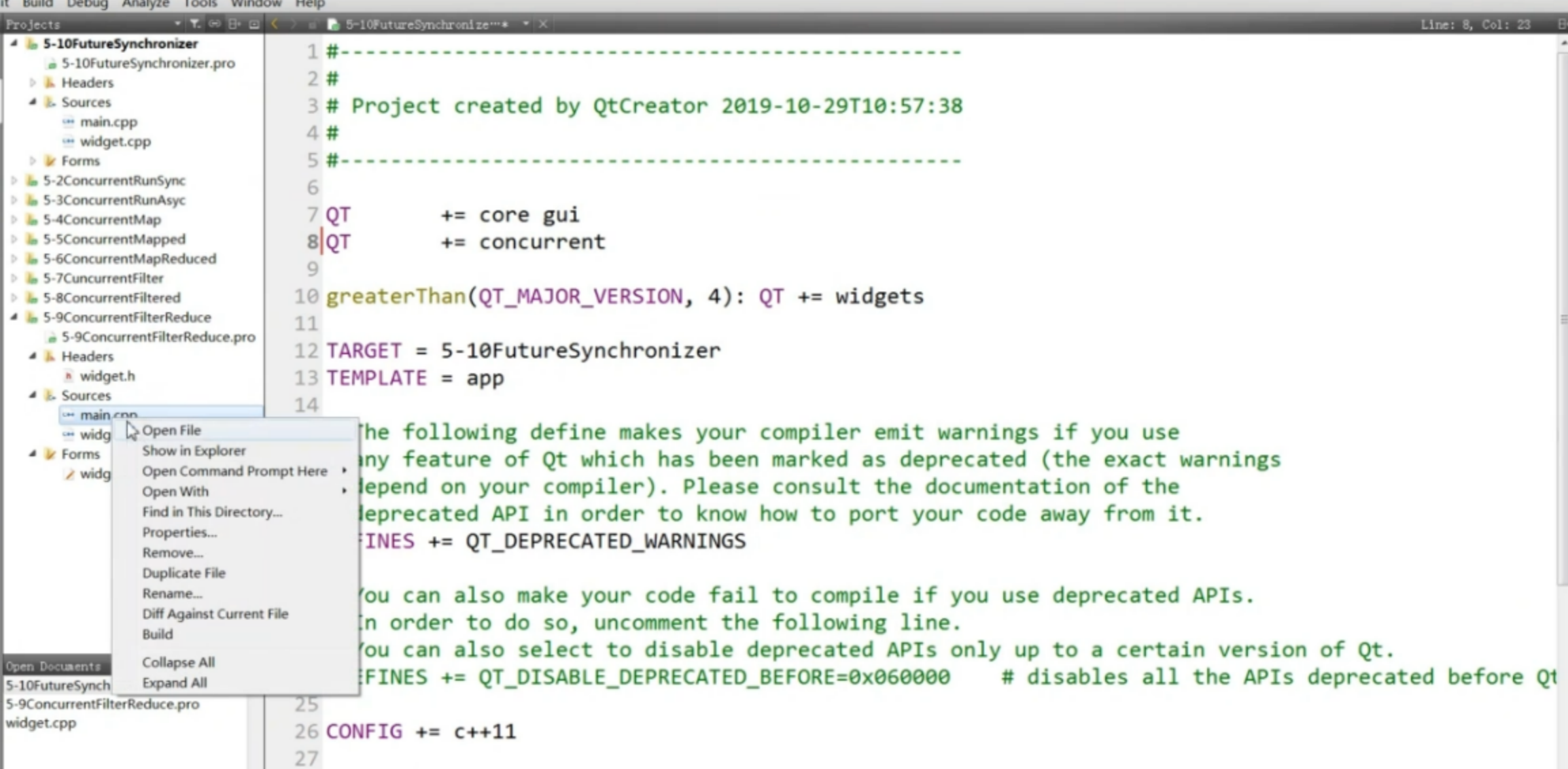
Reduce

```
void Widget::reduce(int &sum, int value){
    sum += value;
}
```

```
//Sum out the filtered elements
QFuture<int> futureReduce = QtConcurrent::filteredReduced(intList,filter,reduce);
futureReduce.waitForFinished();
qDebug() << "The sum is " << futureReduce.result();
```

QFutureSynchonizer

```
void Widget::doWork()
{
    QFutureSynchronizer<void> synchronizer;

    mSum = 0;

    auto filter = [=](const int value){

        if(value >= filterValue){
            return false;
        }
        return true;
    };

    future = QtConcurrent::filtered(intList,filter);
    futureReduce = QtConcurrent::filteredReduced(intList,filter,reduce);
    synchronizer.addFuture(future);
    synchronizer.addFuture(futureReduce);

}
```


Projects
5-10FutureSynchronizer
5-10FutureSynchronizer.pro
Headers
Sources
main.cpp
widget.cpp
Forms
5-2ConcurrentRunSync
5-3ConcurrentRunAsyc
5-4ConcurrentMap
5-5ConcurrentMapped
5-6ConcurrentMapReduced
5-7CuncurrentFilter
5-8ConcurrentFiltered
5-9ConcurrentFilterReduce
5-9ConcurrentFilterReduce.pro
Headers
widget.h
Sources
main.cpp
Forms
Open File
Show in Explorer
Open Command Prompt Here
Open With
Find in This Directory...
Properties...
Remove...
Duplicate File
Rename...
Diff Against Current File
Build
Collapse All
Expand All
Open Documents
5-10FutureSynch
5-9ConcurrentFilterReduce.pro
widget.cpp
5-10FutureSynchronize...*
Line: 8, Col: 23
#---
#
Project created by QtCreator 2019-10-29T10:57:38
#
#---
QT += core gui
QT += concurrent
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = 5-10FutureSynchronizer
TEMPLATE = app
he following define makes your compiler emit warnings if you use
ny feature of Qt which has been marked as deprecated (the exact warnings
epend on your compiler). Please consult the documentation of the
eprecated API in order to know how to port your code away from it.
INES += QT_DEPRECATED_WARNINGS
ou can also make your code fail to compile if you use deprecated APIs.
n order to do so, uncomment the following line.
ou can also select to disable deprecated APIs only up to a certain version of Qt.
FINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000 # disables all the APIs deprecated before Qt
CONFIG += c++11

Course content

32. Qt Concurrent-map
13min Resources

33. Qt Concurrent-maped
15min Resources

34. Qt Concurrent-mapReduced
33min Resources

35. Qt Concurrent-Filter
17min Resources

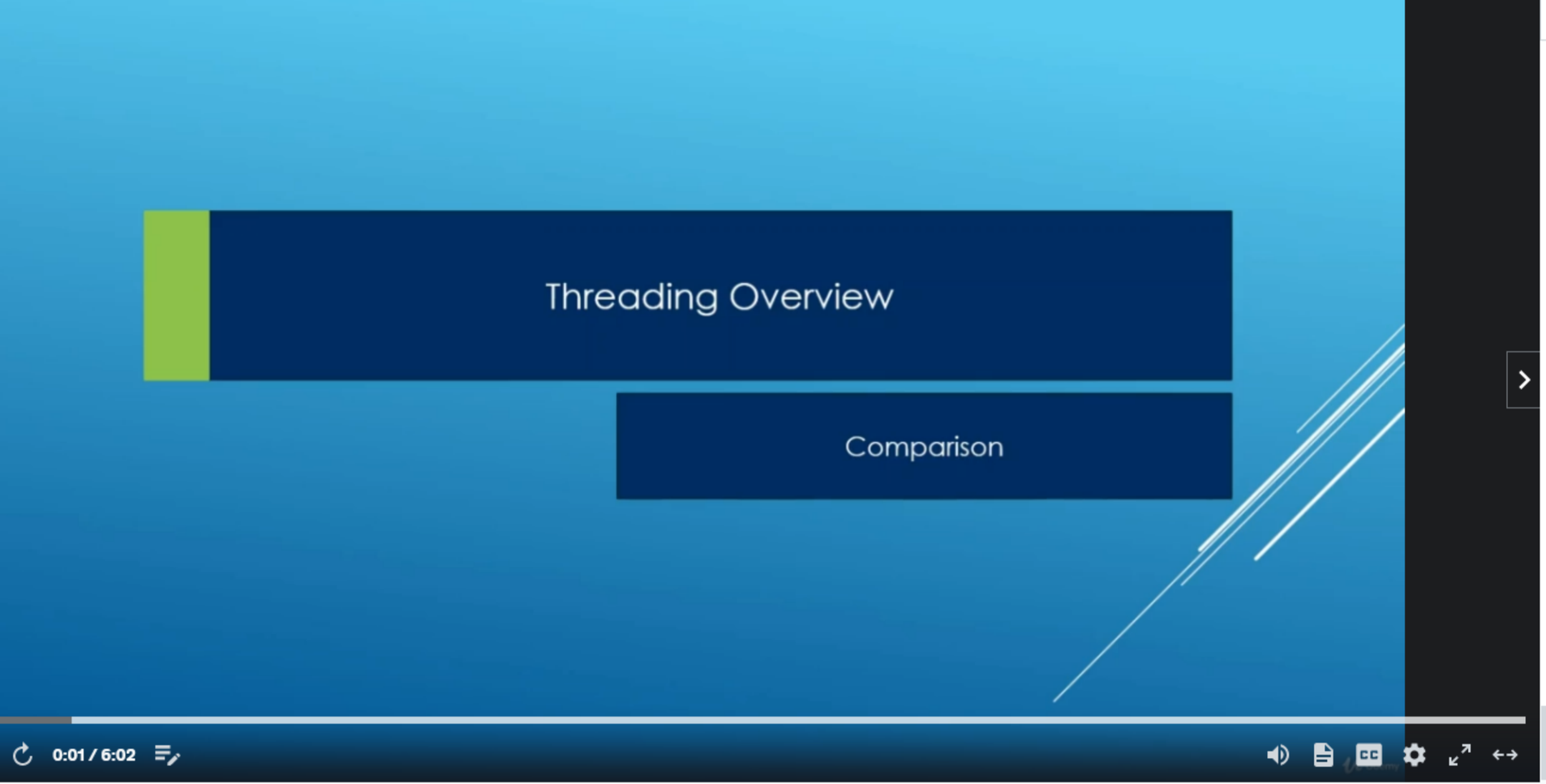
36. Qt Concurrent-Filtered
9min Resources

37. Qt Concurrent-FilterReduce
10min Resources

38. Qt Concurrent-QFutureSynchronizer
7min Resources

39. Qt Concurrent : Feedback
4min

```
void Widget::heavyWork()
{

    qDebug() << "Heavy work running in thread : " << QThread::currentThread();
    for(int i{0} ; i < 1000000001 ; i++){

        if((i%100000) == 0){

            double percentage = ((i/1000000000.0)) * 100;
                        qDebug() << "Percentage : " << QVariant::fromValue(percentage).toInt()
                                 << " | Thread : " << QThread::currentThread();
        }
    }
}

void Widget::on_startButton_clicked()
{
   future = QtConcurrent::run(heavyWork);

   future.waitForFinished();

   qDebug() << "Computation done!";
}
```



w Q&A Notes Announcements Reviews Learning tools

Course content

- 32. Qt Concurrent-map — 13min — Resources
- 33. Qt Concurrent-maped — 15min — Resources
- 34. Qt Concurrent-mapReduced — 33min — Resources
- 35. Qt Concurrent-Filter — 17min — Resources
- 36. Qt Concurrent-Filtered — 9min — Resources
- 37. Qt Concurrent-FilterReduce — 10min — Resources
- 38. Qt Concurrent-QFutureSynchronizer — 7min — Resources
- 39. Qt Concurrent : Feedback — 4min
- 40. Threading Overview-Comparison — 6min

Section 6: Processes

run()

map()
mapped()
mapReduced

filter()
filtered()
filterReduced()

Synchronous
waitForFinished()

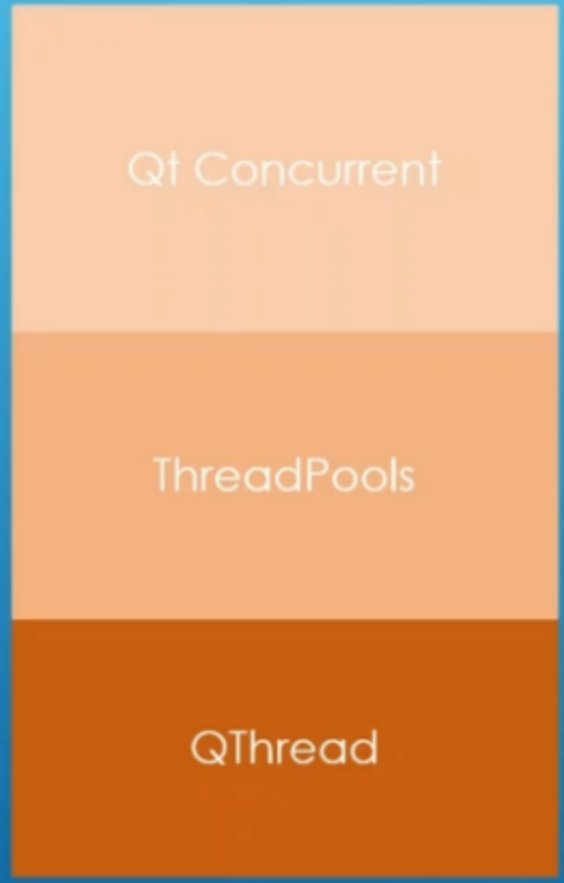
Asynchronous
QFutureWatcher

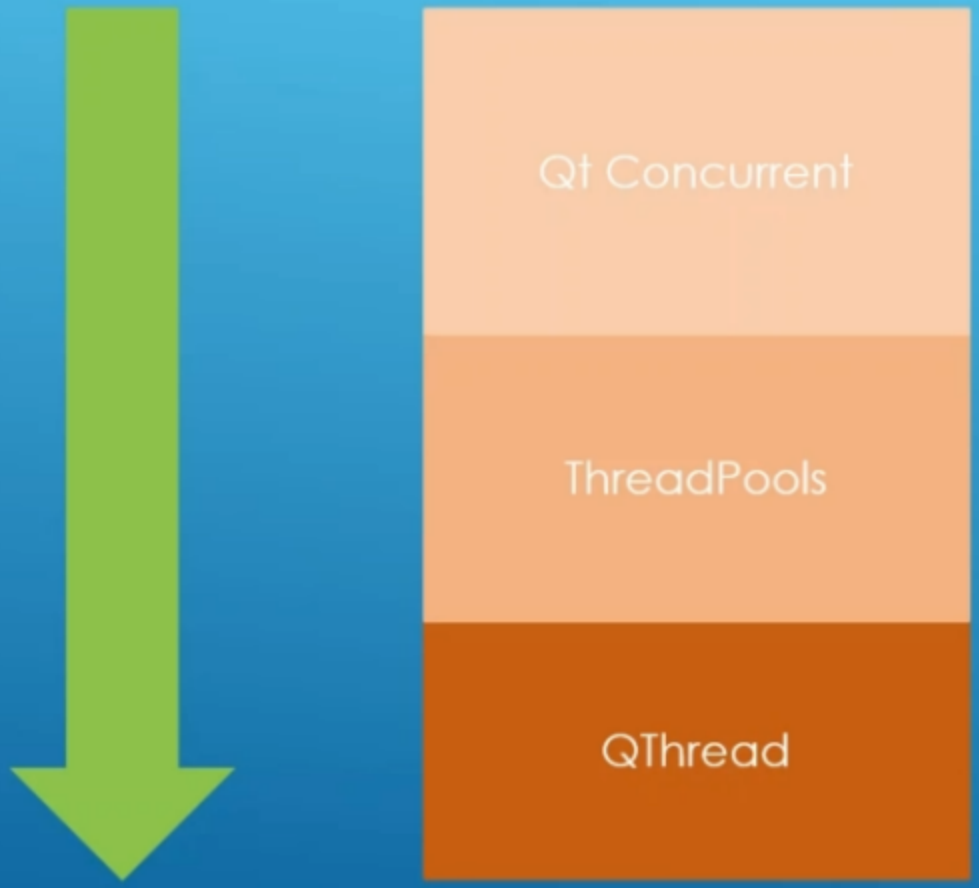
QFutureSynchronizer

High Level API

Uses ThreadPools
under the hood

No operation
specific feedback


Qt Concurrent
ThreadPools
QThread


Qt Concurrent
ThreadPools
QThread

ew Q&A Notes Announcements Reviews Learning tools

Course content

33. Qt Concurrent-maped
15min Resources

34. Qt Concurrent-mapReduced
33min Resources

35. Qt Concurrent-Filter
17min Resources

36. Qt Concurrent-Filtered
9min Resources

37. Qt Concurrent-FilterReduce
10min Resources

38. Qt Concurrent-QFutureSynchronizer
7min Resources

39. Qt Concurrent : Feedback
4min

40. Threading Overview-Comparison
6min

Section 6: Processes
0 / 1 | 15min

41. Processes and QProcess
15min Resources

```
mProcess = new QProcess(this);

connect(mProcess,&QProcess::started,[](){
    qDebug() << "Process started";
});
connect(mProcess,SIGNAL(finished(int)),this,SLOT(finished(int)));
```

```
void Widget::on_chooseProcessButton_clicked()
{
    processPath = QFileDialog::getOpenFileName(this, tr("Open File"),
                                                "/home",
                                                tr("Programs (*.exe )"));
    if(!processPath.isNull()){
        ui->processPathLineEdit->setText(processPath);
    }
}

void Widget::on_startProcessButton_clicked()
{
    if(!processPath.isNull()){
        mProcess->start(processPath);
    }
}
```

- 1 property inherited from QObject

Detailed Description

The QProcess class is used to start external programs and to communicate with them.

Running a Process

To start a process, pass the name and command line arguments of the program you want to run as arguments to start(). Arguments are supplied as individual strings in a QStringList.

Alternatively, you can set the program to run with setProgram() and setArguments(), and then call start() or open().

For example, the following code snippet runs the analog clock example in the Fusion style on X11 platforms by passing strings containing "-style" and "fusion" as two items in the list of arguments:

```
QObject *parent;
...
QString program = "./path/to/Qt/examples/widgets/analogclock";
QStringList arguments;
arguments << "-style" << "fusion";

QProcess *myProcess = new QProcess(parent);
myProcess->start(program, arguments);
```

QProcess then enters the Starting state, and when the program has started, QProcess enters the Running state and emits started().

QProcess allows you to treat a process as a sequential I/O device. You can write to and read from the process just as you would access a network connection using QTcpSocket. You can then write to the

QProcess
QProcess::Crashed
QProcess::CrashExit
QProcess::CreateProcessArgumentModifier
QProcess::ExitStatus
QProcess::FailedToStart
QProcess::ForwardedChannels
QProcess::ForwardedErrorChannel
QProcess::ForwardedInputChannel
QProcess::ForwardedOutputChannel
QProcess::InputChannelMode
QProcess::ManagedInputChannel
QProcess::MergedChannels
QProcess::NormalExit
QProcess::NotRunning
QProcess::ProcessChannel
QProcess::ProcessChannelMode
QProcess::ProcessError
QProcess::ProcessState
QProcess::ReadError
QProcess::Running
QProcess::SeparateChannels
QProcess::StandardError
QProcess::StandardOutput
QProcess::Starting
QProcess::Timedout
QProcess::UnknownError
QProcess::WriteError
QProcessEnvironment
QProcessSignalReceiver
~QProcess
~QProcessEnvironment

QProcess's *write* channels. This is because what we read using QProcess is the process's output, and what we write becomes the process's input.

QProcess can merge the two output channels, so that standard output and standard error data from the running process both use the standard output channel. Call setProcessChannelMode() with MergedChannels before starting the process to activate this feature. You also have the option of forwarding the output of the running process to the calling, main process, by passing ForwardedChannels as the argument. It is also possible to forward only one of the output channels - typically one would use ForwardedErrorChannel, but ForwardedOutputChannel also exists. Note that using channel forwarding is typically a bad idea in GUI applications - you should present errors graphically instead.

Certain processes need special environment settings in order to operate. You can set environment variables for your process by calling setProcessEnvironment(). To set a working directory, call setWorkingDirectory(). By default, processes are run in the current working directory of the calling process.

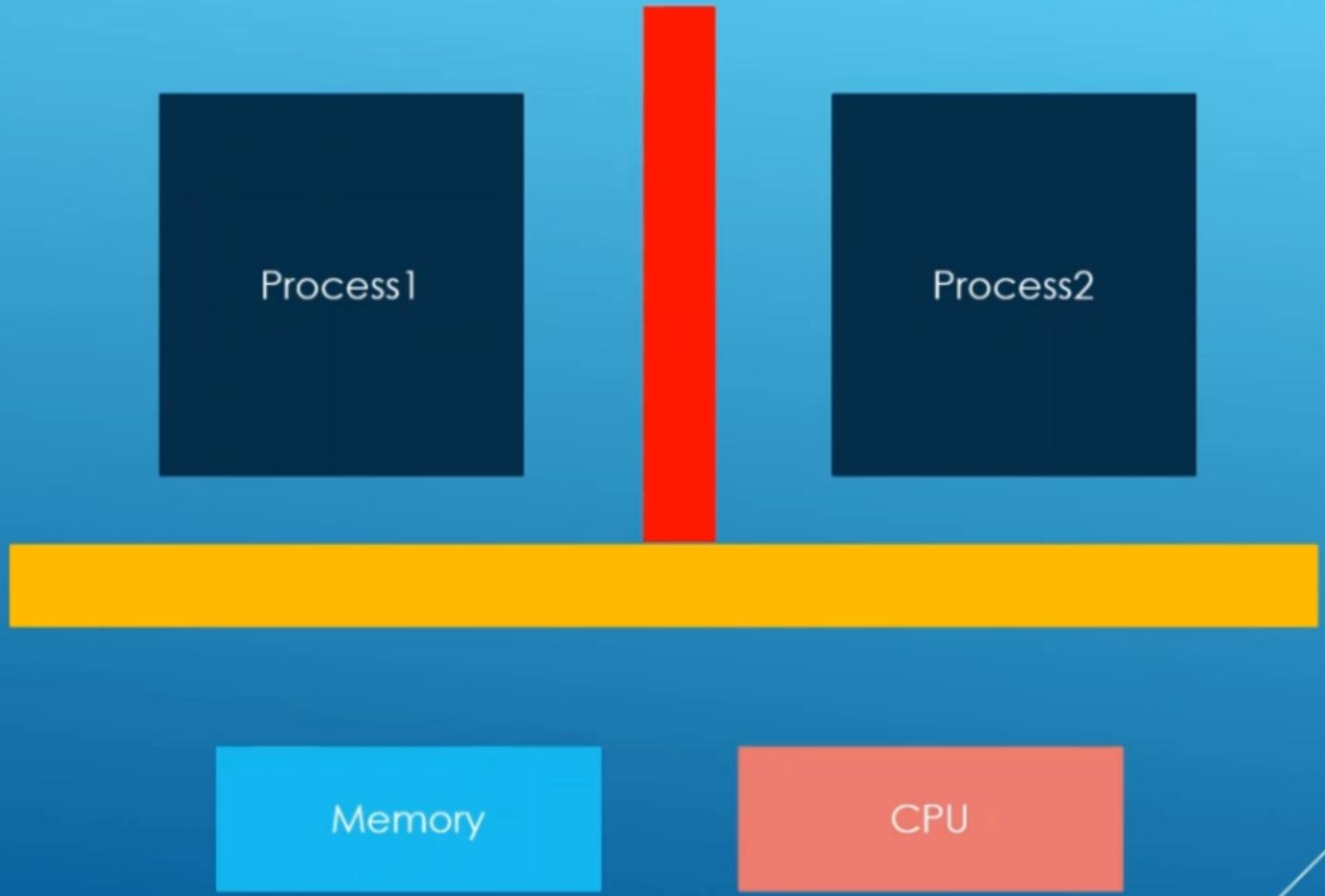
The positioning and the screen Z-order of windows belonging to GUI applications started with QProcess are controlled by the underlying windowing system. For Qt 5 applications, the positioning can be specified using the `-qwindowgeometry` command line option; X11 applications generally accept a `-geometry` command line option.

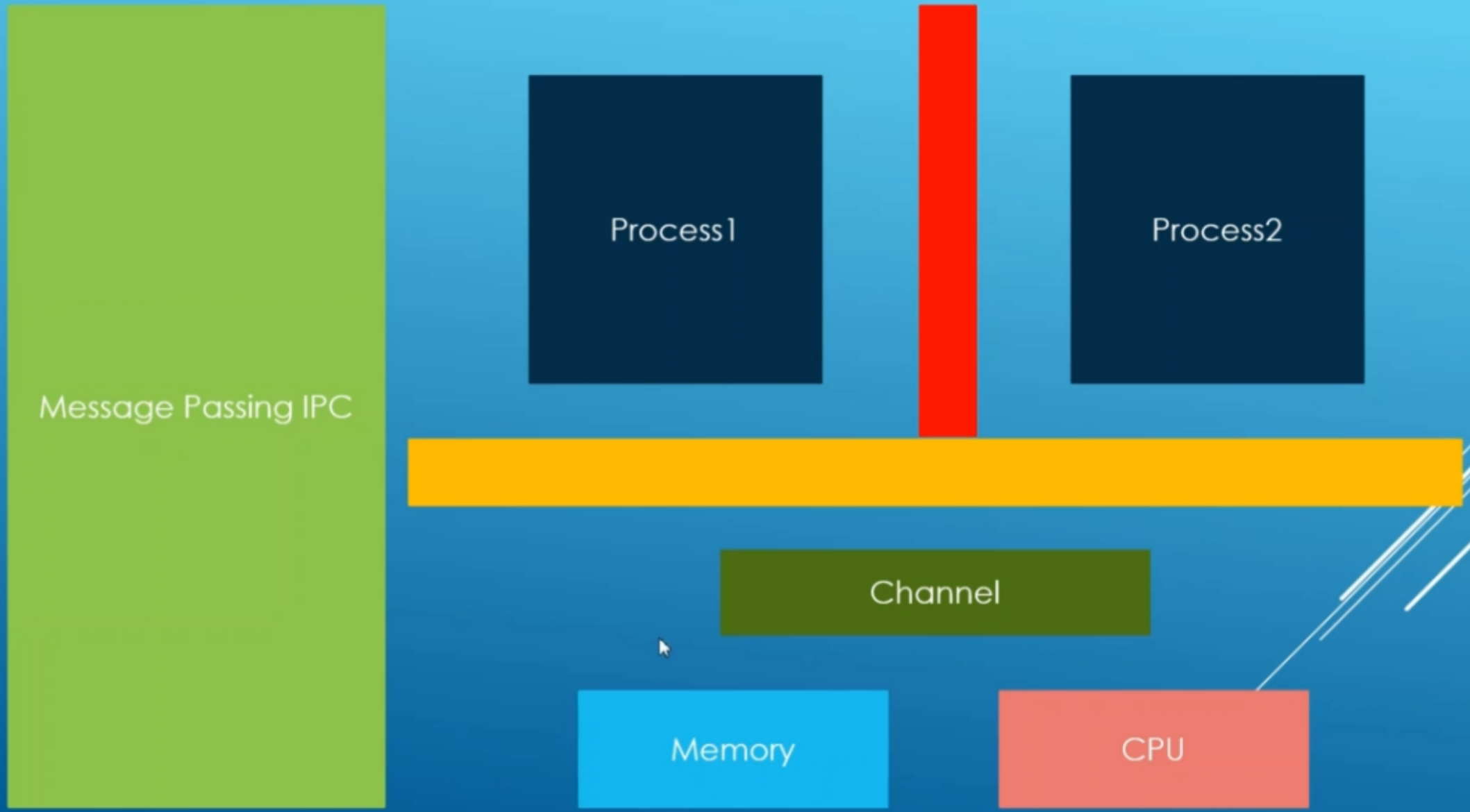
Note: On QNX, setting the working directory may cause all application threads, with the exception of the QProcess caller thread, to temporarily freeze during the spawning process, owing to a limitation in the operating system.

Synchronous Process API

Inter Process Communication


Process1
Communication channel
Process2


Process1
Process2
Memory
CPU


Message Passing IPC
Process1
Process2
Channel
Memory
CPU

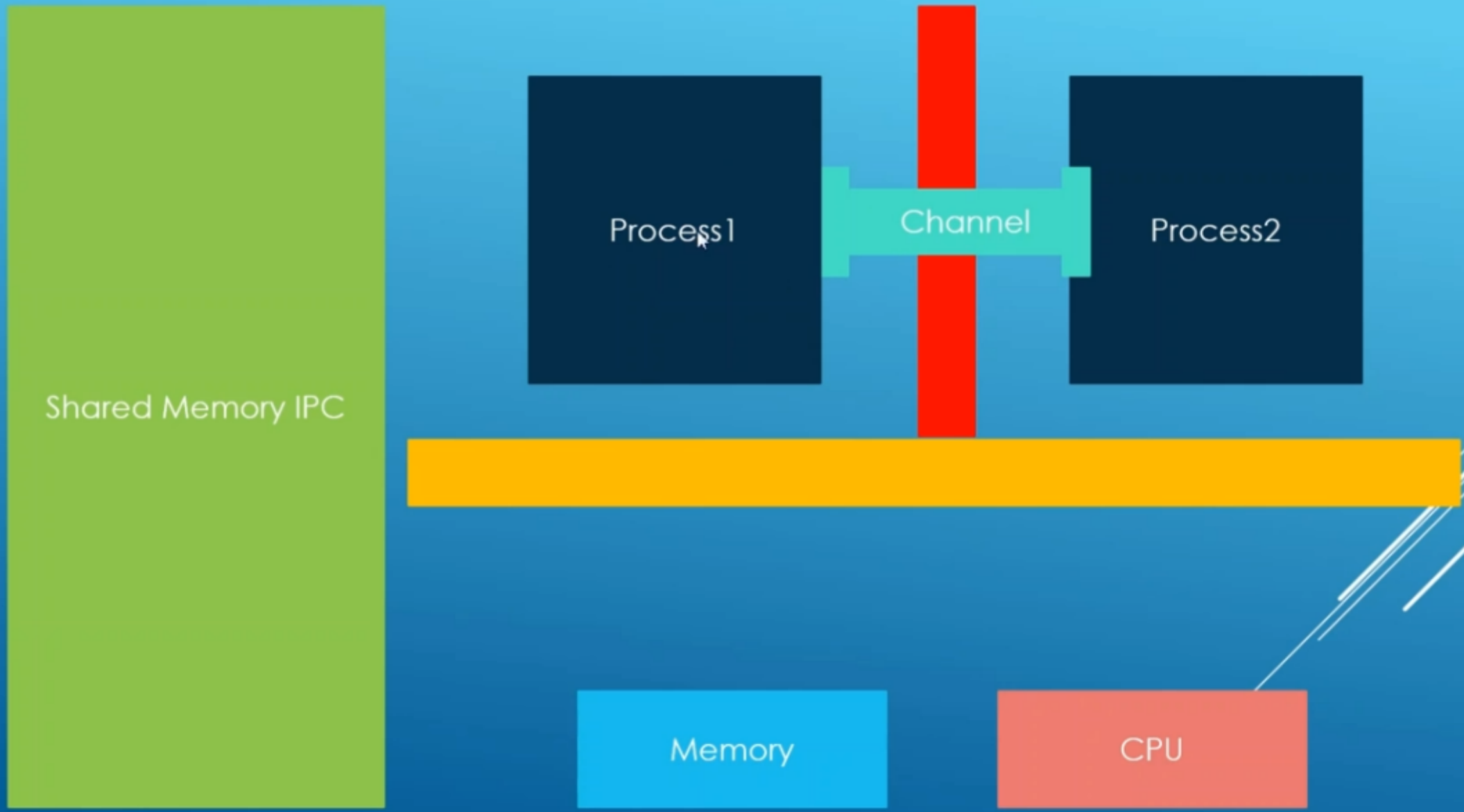
Message Passing IPC

Communication channel like shared buffers,… is provided by the OS itself

Processes write and read to/from the communication channel

\+ Communication managed by OS

\- System call overhead


Shared Memory IPC
Process1
Channel
Process2
Memory
CPU

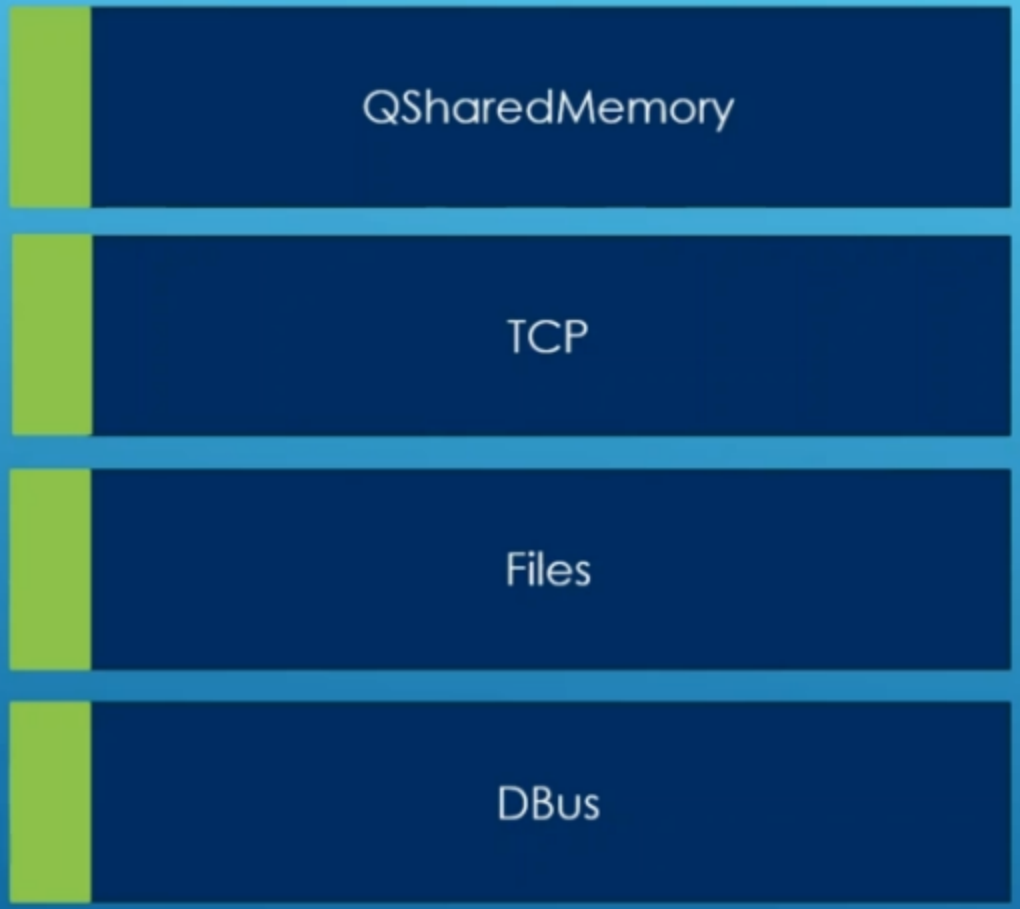
Shared Memory IPC

\+

OS not involved. No OS overhead

\-

Error Prone. No clearly defined API


QSharedMemory
TCP
Files
DBus

ew Q&A Notes Announcements Reviews Learning tools

Course content

Section 3: Thread Synchronization
8 / 8 | 2hr 9min

Section 4: Thread Safety and Reentrancy
6 / 6 | 1hr 10min

Section 5: Qt Concurrent
12 / 12 | 2hr 37min

Section 6: Processes
1 / 1 | 15min

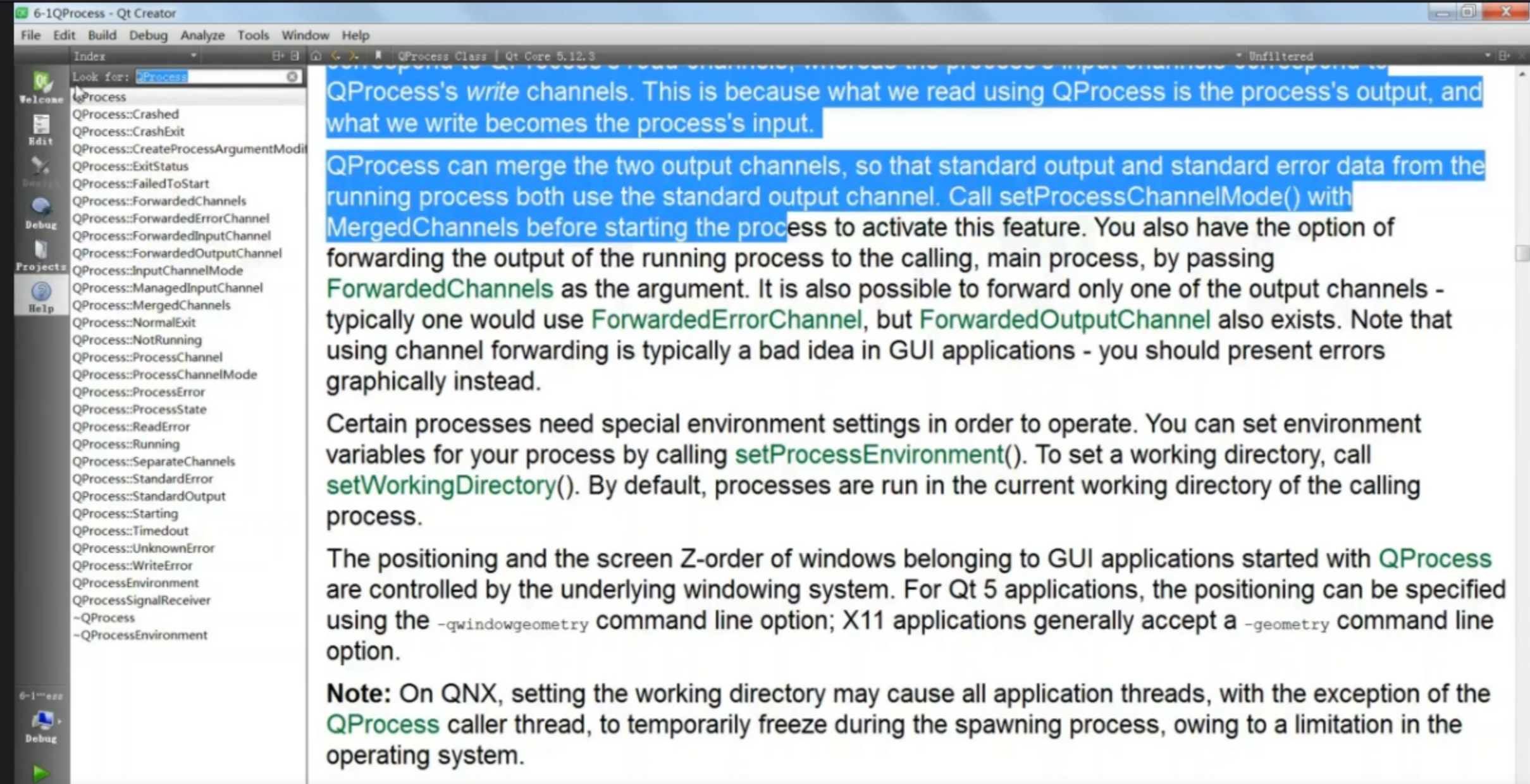
- [x] 41. Processes and QProcess
  15min Resources

Section 7: Inter Process Communication
1 / 5 | 1hr 14min

- [x] 42. IPC Overview
  6min
- [ ] 43. IPC- SharedMemory
  24min Resources
- [ ] 44. IPC -TCP(QTcpSocket)
  25min Resources
- [ ] 45. IPC - Files on FileSystem
  14min Resources
- [ ] 46. IPC Overview-Comparison


Process1
Shared Memory
Process2


Shared Memory
Process1
Process2


6-1QProcess - Qt Creator
File Edit Build Debug Analyze Tools Window Help
Index
Look for: QProcess
QProcess
QProcess::Crashed
QProcess::CrashExit
QProcess::CreateProcessArgumentModif
QProcess::ExitStatus
QProcess::FailedToStart
QProcess::ForwardedChannels
QProcess::ForwardedErrorChannel
QProcess::ForwardedInputChannel
QProcess::ForwardedOutputChannel
QProcess::InputChannelMode
QProcess::ManagedInputChannel
QProcess::MergedChannels
QProcess::NormalExit
QProcess::NotRunning
QProcess::ProcessChannel
QProcess::ProcessChannelMode
QProcess::ProcessError
QProcess::ProcessState
QProcess::ReadError
QProcess::Running
QProcess::SeparateChannels
QProcess::StandardError
QProcess::StandardOutput
QProcess::Starting
QProcess::Timedout
QProcess::UnknownError
QProcess::WriteError
QProcessEnvironment
QProcessSignalReceiver
~QProcess
~QProcessEnvironment
QProcess Class | Qt Core 5.12.3
Unfiltered
Welcome
Edit
Debug
Projects
Help
Debug
QProcess's write channels. This is because what we read using QProcess is the process's output, and what we write becomes the process's input.
QProcess can merge the two output channels, so that standard output and standard error data from the running process both use the standard output channel. Call setProcessChannelMode() with MergedChannels before starting the process to activate this feature. You also have the option of forwarding the output of the running process to the calling, main process, by passing ForwardedChannels as the argument. It is also possible to forward only one of the output channels - typically one would use ForwardedErrorChannel, but ForwardedOutputChannel also exists. Note that using channel forwarding is typically a bad idea in GUI applications - you should present errors graphically instead.
Certain processes need special environment settings in order to operate. You can set environment variables for your process by calling setProcessEnvironment(). To set a working directory, call setWorkingDirectory(). By default, processes are run in the current working directory of the calling process.
The positioning and the screen Z-order of windows belonging to GUI applications started with QProcess are controlled by the underlying windowing system. For Qt 5 applications, the positioning can be specified using the -qwindowgeometry command line option; X11 applications generally accept a -geometry command line option.
Note: On QNX, setting the working directory may cause all application threads, with the exception of the QProcess caller thread, to temporarily freeze during the spawning process, owing to a limitation in the operating system.

- 9 protected functions inherited from QObject

Detailed Description

The QSharedMemory class provides access to a shared memory segment.

QSharedMemory provides access to a shared memory segment by multiple threads and processes. It also provides a way for a single thread or process to lock the memory for exclusive access.

When using this class, be aware of the following platform differences:

- Windows: QSharedMemory does not "own" the shared memory segment. When all threads or processes that have an instance of QSharedMemory attached to a particular shared memory segment have either destroyed their instance of QSharedMemory or exited, the Windows kernel releases the shared memory segment automatically.
- Unix: QSharedMemory "owns" the shared memory segment. When the last thread or process that has an instance of QSharedMemory attached to a particular shared memory segment detaches from the segment by destroying its instance of QSharedMemory, the Unix kernel release the shared memory segment. But if that last thread or process crashes without running the QSharedMemory destructor, the shared memory segment survives the crash.
- HP-UX: Only one attach to a shared memory segment is allowed per process. This means that QSharedMemory should not be used across multiple threads in the same process in HP-UX.

Remember to lock the shared memory with lock() before reading from or writing to the shared memory, and remember to release the lock with unlock() after you are done.

Write data in shared memory

```
// load into shared memory
QBuffer buffer;
buffer.open(QBuffer::ReadWrite);
QDataStream out(&buffer);
out << image;
int size =  buffer.size();      ▵ implicit conversion loses integer precisior

qDebug() << "Size : " << size;
if (!sharedMemory.create(size)) {
    ui->imageLabel->setText(tr("Unable to create shared memory segment."));
    qDebug() << sharedMemory.errorString();
    qDebug()  << "Is attached : " << sharedMemory.isAttached();
    return;
}
sharedMemory.lock();
char *to = (char*)sharedMemory.data();                                  ▵ u
const char *from = buffer.data().data();
memcpy(to, from, qMin(sharedMemory.size(), size));      ▵ implicit conversior
sharedMemory.unlock();
```

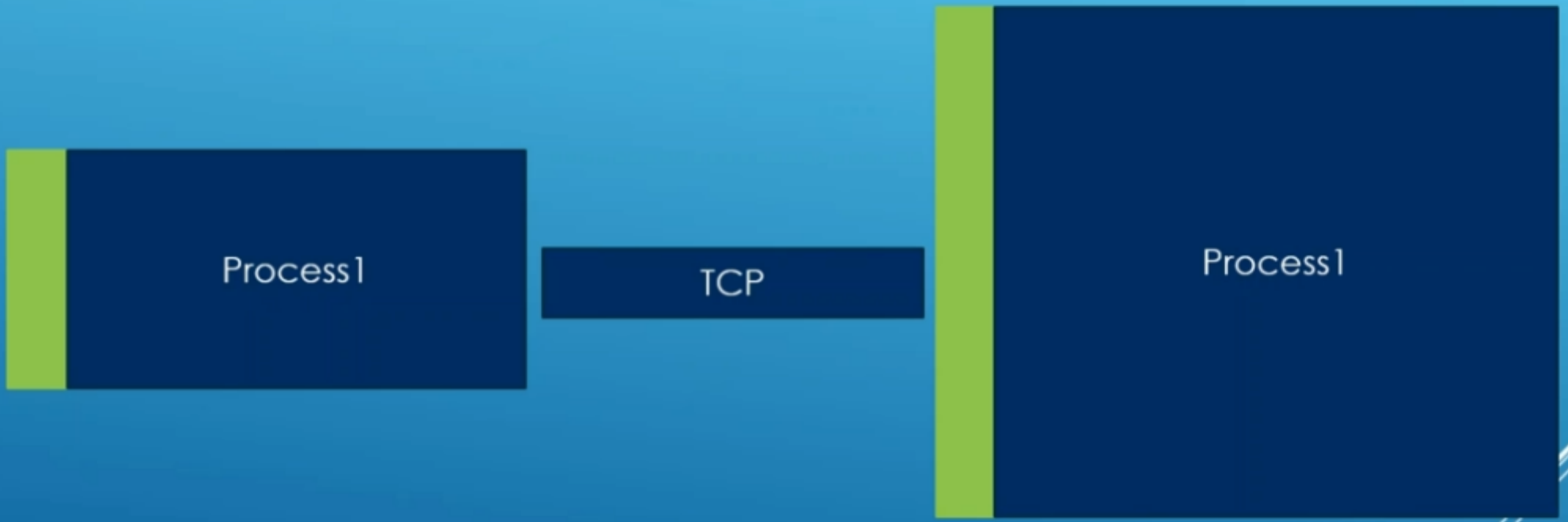
Read data from shared memory

```
if (!sharedMemory.attach()) {
    ui->imageLabel->setText(tr("Unable to attach to shared memory segment.\n" \
                           "Load in the data char const[67] ;
    return;
}

QBuffer buffer;
QDataStream in(&buffer);
QImage image;

sharedMemory.lock();
buffer.setData((char*)sharedMemory.constData(), sharedMemory.size());     use c
buffer.open(QBuffer::ReadOnly);
in >> image;
sharedMemory.unlock();

sharedMemory.detach();
```

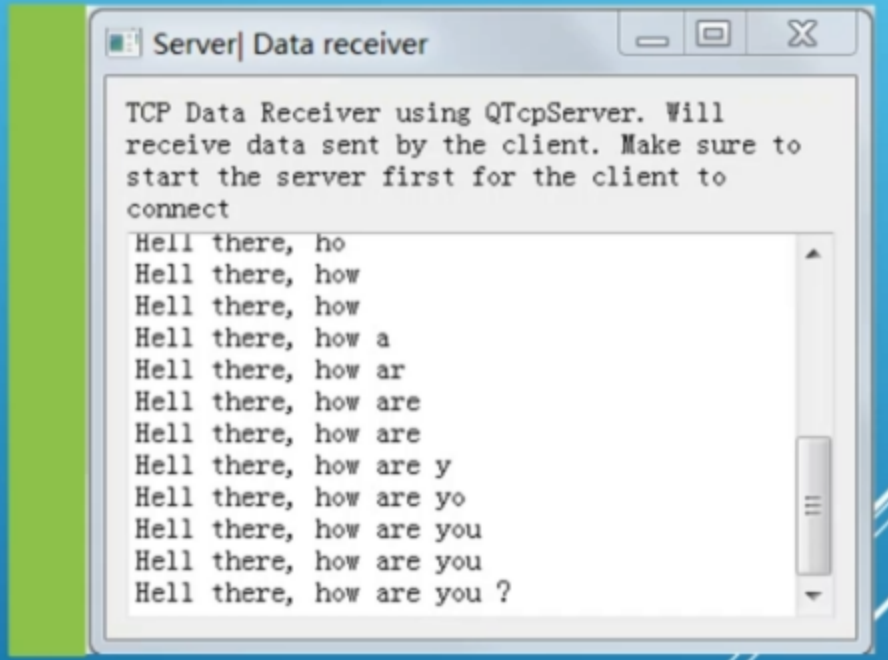
Load File, Write to SharedMemory

Read data from SharedMemory and display in QLabel

IPC : QTcpSocket


Process1
TCP
Process1


Client | Data Sender
This is the client side for IPC using TCP. Will send data to the server usint a QTcpSocket instance
Hell there, how are you ?


Server| Data receiver
TCP Data Receiver using QTcpServer. Will receive data sent by the client. Make sure to start the server first for the client to connect
Hell there, ho
Hell there, how
Hell there, how
Hell there, how a
Hell there, how ar
Hell there, how are
Hell there, how are
Hell there, how are y
Hell there, how are yo
Hell there, how are you
Hell there, how are you
Hell there, how are you ?

Sender setup

```
m_socket->connectToHost("localhost", 4040);
connect(m_socket, &QTcpSocket::connected, this, &Widget::socketReady);
connect(m_socket,&QTcpSocket::stateChanged,this,&Widget::stateChanged);
```

Send Actual Data

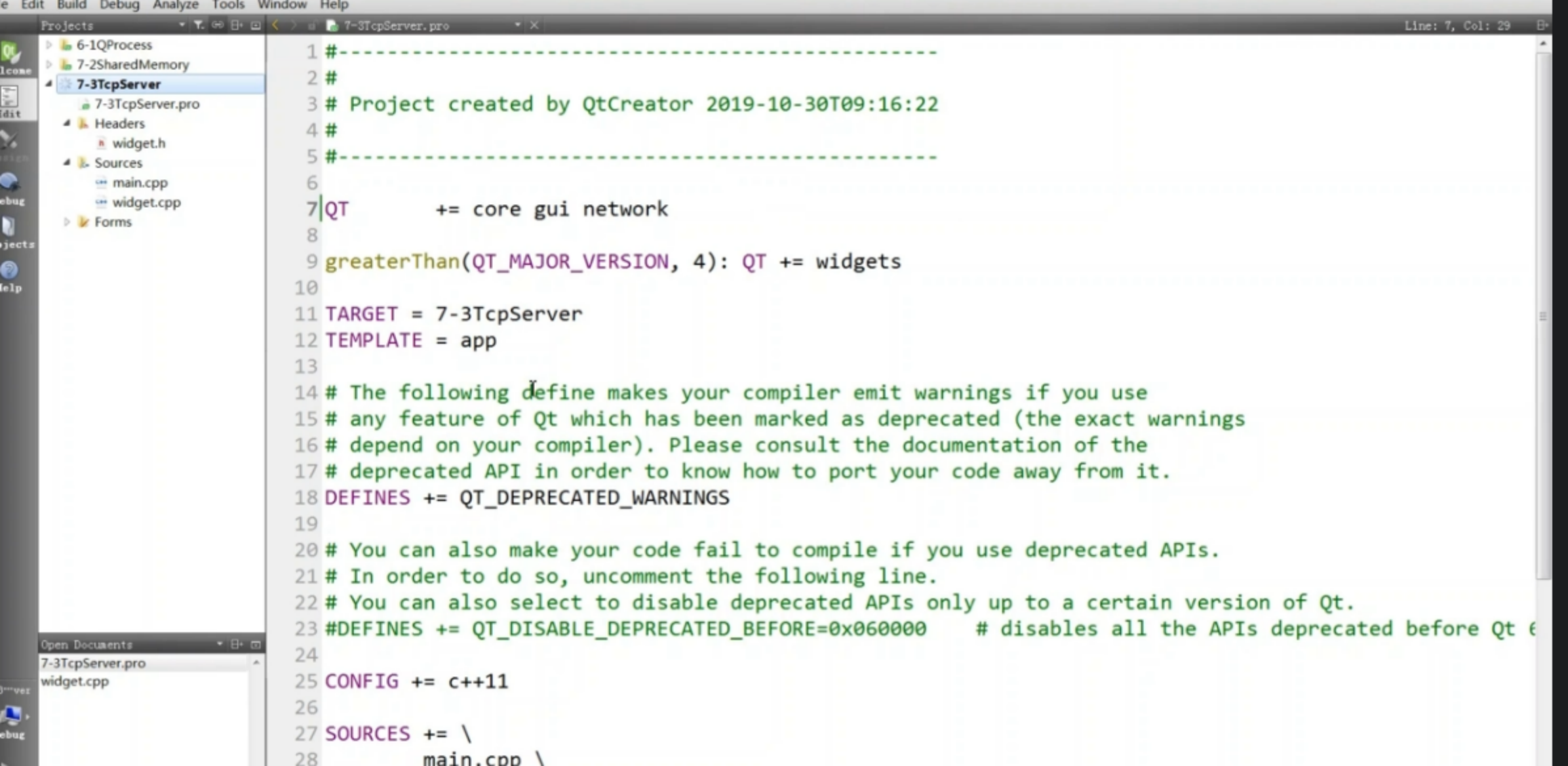
```
void Widget::on_lineEdit_textChanged(const QString &newText)
{
    if (m_socketReady) {
        QDataStream out(m_socket);
        out << newText;
    }
}
```

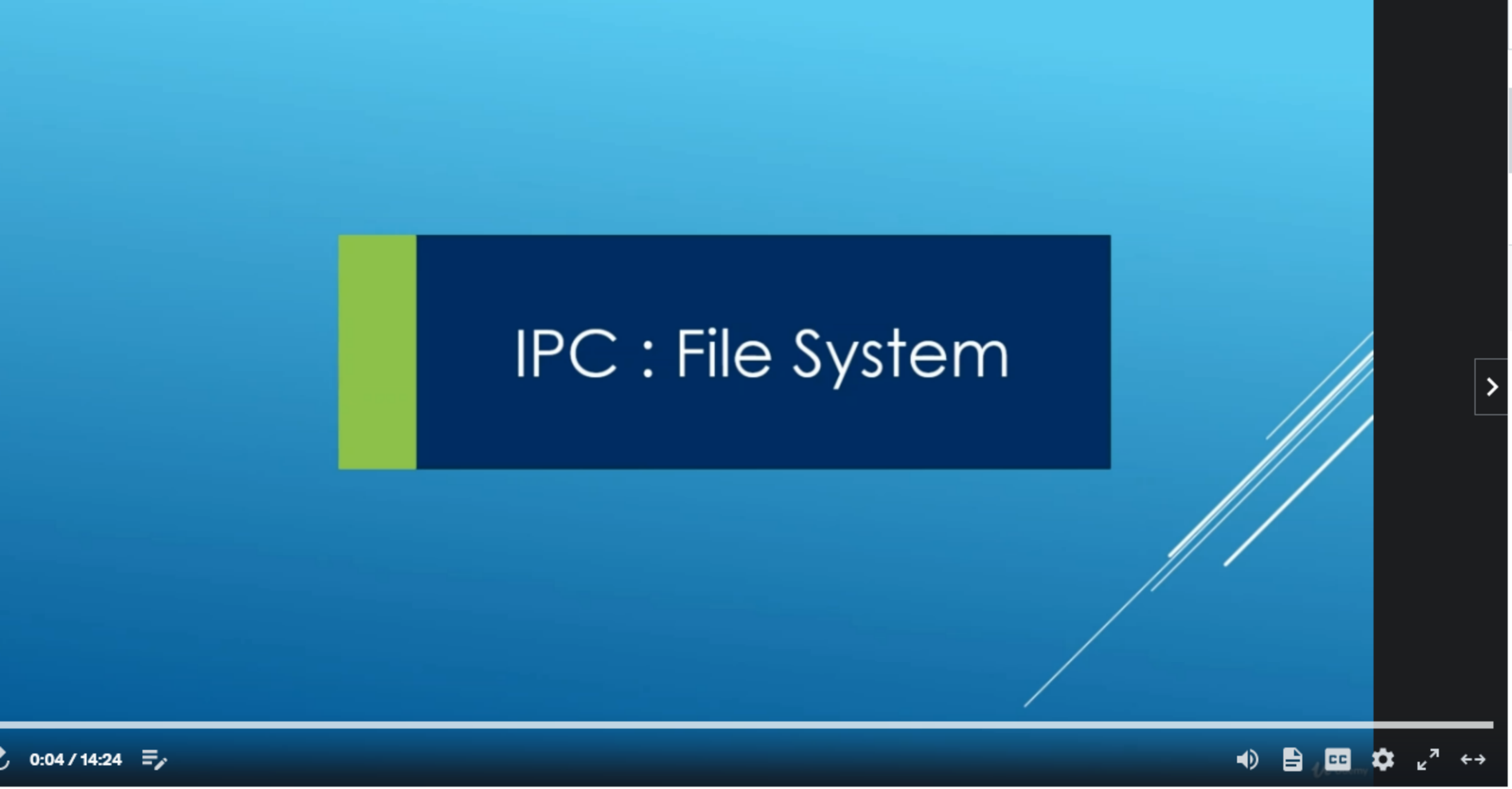
Receiver Setup

```
m_socket = nullptr;

dataSize = 0;

m_server.listen(QHostAddress::LocalHost, 4040);
connect(&m_server, &QTcpServer::newConnection, this, &Widget::gotConnection);
```

Receive Actual Data

```
void Widget::gotConnection()
{
    qDebug() << "Got connection";
    m_socket = m_server.nextPendingConnection();
    in.setDevice(m_socket);
    connect(m_socket, &QTcpSocket::readyRead, this, &Widget::readData);

}

void Widget::readData()
{
    QDataStream in(m_socket);
    in.startTransaction();

    QString rcvString;
    in >> rcvString;

    if (!in.commitTransaction())
        return;     // wait for more data

    ui->textEdit->append(rcvString);
}
```


Edit Build Debug Analyze Tools Window Help
Projects
6-1QProcess
7-2SharedMemory
7-3TcpServer
7-3TcpServer.pro
Headers
widget.h
Sources
main.cpp
widget.cpp
Forms
Open Documents
7-3TcpServer.pro
widget.cpp
7-3TcpServer.pro
Line: 7, Col: 29
#---
#
Project created by QtCreator 2019-10-30T09:16:22
#
#---
QT   += core gui network
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = 7-3TcpServer
TEMPLATE = app
# The following define makes your compiler emit warnings if you use
# any feature of Qt which has been marked as deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS
You can also make your code fail to compile if you use deprecated APIs.
In order to do so, uncomment the following line.
You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000   # disables all the APIs deprecated before Qt
CONFIG += c++11
SOURCES += \
main.cpp \



Course content

25min Resources

45. IPC - Files on FileSystem
14min Resources

46. IPC Overview-Comparison
3min

Section 8: DBus
0 / 8 | 2hr 17min

47. DBus Overview
15min

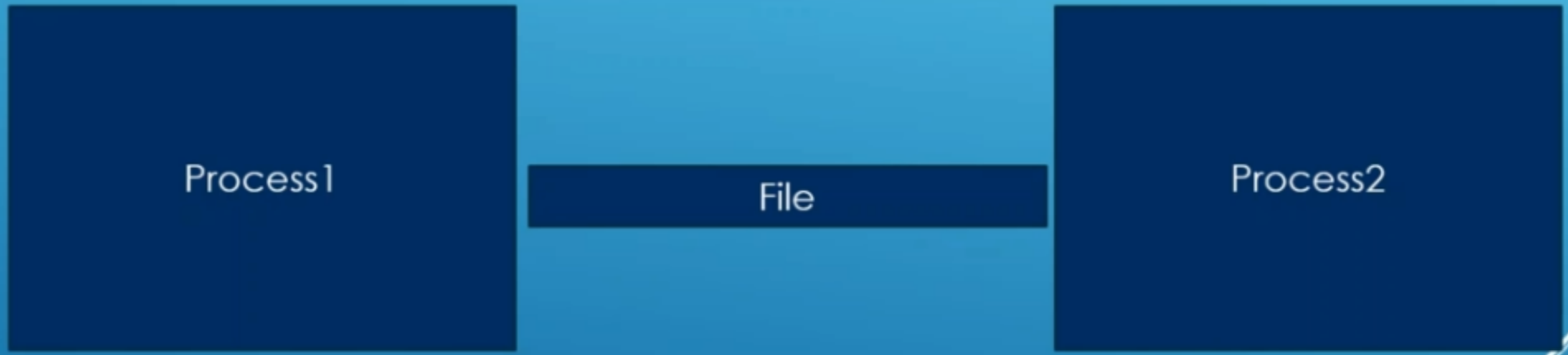
48. DBus-Client-Server : Server
26min Resources

49. DBus Client-Server : Client
14min Resources

50. DBus-SignalsSlots
25min Resources

51. QDBusMessage
26min Resources

52. QDBusInterface


Process1
File
Process2

Producer

```
void Widget::on_lineEdit_textChanged(const QString &newText)
{
    QFile outFile(QString(QDir::tempPath() + "/sharedFile"));
    qDebug() << "File path : " << outFile.fileName();
    outFile.open(QFile::WriteOnly| QIODevice::Append);
    QDataStream out(&outFile);
    out << newText;
}
```

Consumer Setup

```
Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget),
    m_watcher(new  QFileSystemWatcher(this))
{
    ui->setupUi(this);
    createFileIfNotExist();
    setWindowTitle("File Consumer");
    m_watcher->addPath(QString(QDir::tempPath() + "/sharedFile"));
    connect(m_watcher, &QFileSystemWatcher::fileChanged, this, &Widget::loadFile);
}
```

Consume Actual Data

```
void Widget::loadFile()
{
    QFile inFile(QString(QDir::tempPath() + "/sharedFile"));
    QString transferedStr;
    if (!inFile.open(QFile::ReadOnly)){
        qDebug() << "Consumer , can't find file :" << inFile.fileName();
        return;

    }
    qDebug() <<"Found file : " << inFile.fileName();
    QDataStream in(&inFile);

    ui->textEdit->clear();

    while(!in.atEnd()){
        in >> transferedStr;
        ui->textEdit->append(transferedStr);
    }

    qDebug() << "Consumer done appending data";
}
```

Consumer Setup

```
Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget),
    m_watcher(new  QFileSystemWatcher(this))
{
    ui->setupUi(this);
    createFileIfNotExist();
    setWindowTitle("File Consumer");
    m_watcher->addPath(QString(QDir::tempPath() + "/sharedFile"));
    connect(m_watcher, &QFileSystemWatcher::fileChanged, this, &Widget::loadFile);
}
```

- 9 protected functions inherited from QObject

Detailed Description

The QTcpSocket class provides a TCP socket.

TCP (Transmission Control Protocol) is a reliable, stream-oriented, connection-oriented transport protocol. It is especially well suited for continuous transmission of data.

QTcpSocket is a convenience subclass of QAbstractSocket that allows you to establish a TCP connection and transfer streams of data. See the QAbstractSocket documentation for details.

Note: TCP sockets cannot be opened in QIODevice::Unbuffered mode.

See also QTcpServer, QUdpSocket, QNetworkAccessManager, Fortune Server Example, Fortune Client Example, Threaded Fortune Server Example, Blocking Fortune Client Example, Loopback Example, and Torrent Example.

Member Function Documentation

QTcpSocket::**QTcpSocket**(**QObject** **parent* = nullptr)

Creates a QTcpSocket object in state `UnconnectedState`.

parent is passed on to the QObject constructor.

See also socketType().

Contents

QFileSystemWatcher Class

The QFileSystemWatcher class provides an interface for monitoring files and directories for modifications. More...

Header: #include <QFileSystemWatcher>

qmake: QT += core

Since: Qt 4.2

Inherits: QObject

- List of all members, including inherited members
- Obsolete members

Note: All functions in this class are reentrant

Course content

25min Resources

45. IPC - Files on FileSystem
14min Resources

46. IPC Overview-Comparison
3min

Section 8: DBus
0 / 8 | 2hr 17min

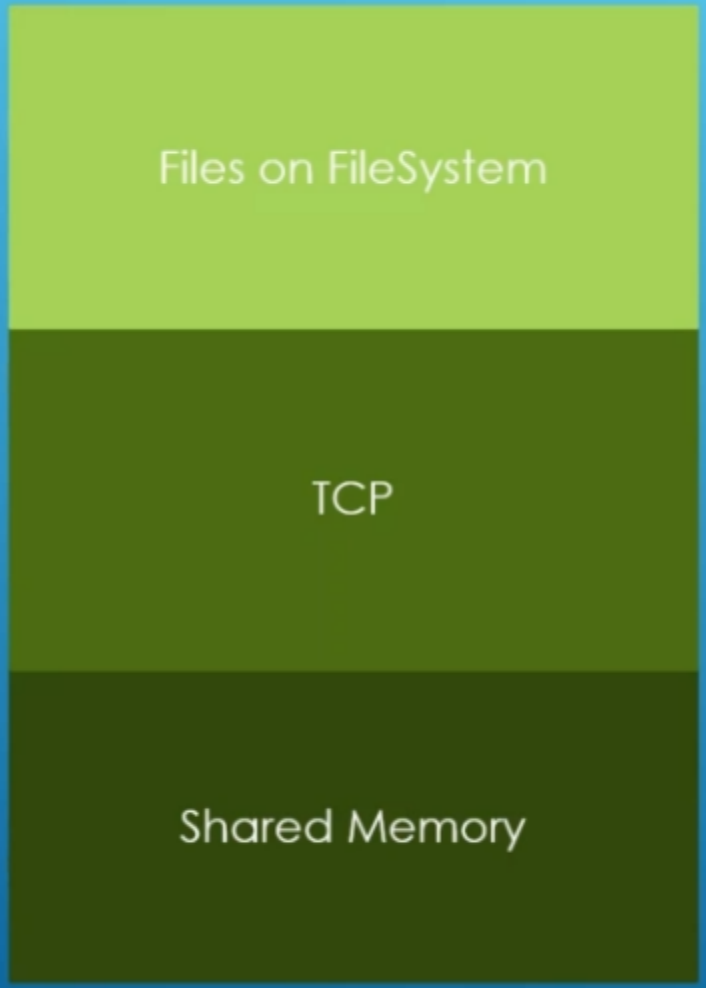
47. DBus Overview
15min

48. DBus-Client-Server : Server
26min Resources

49. DBus Client-Server : Client
14min Resources

50. DBus-SignalsSlots
25min Resources

51. QDBusMessage
26min Resources

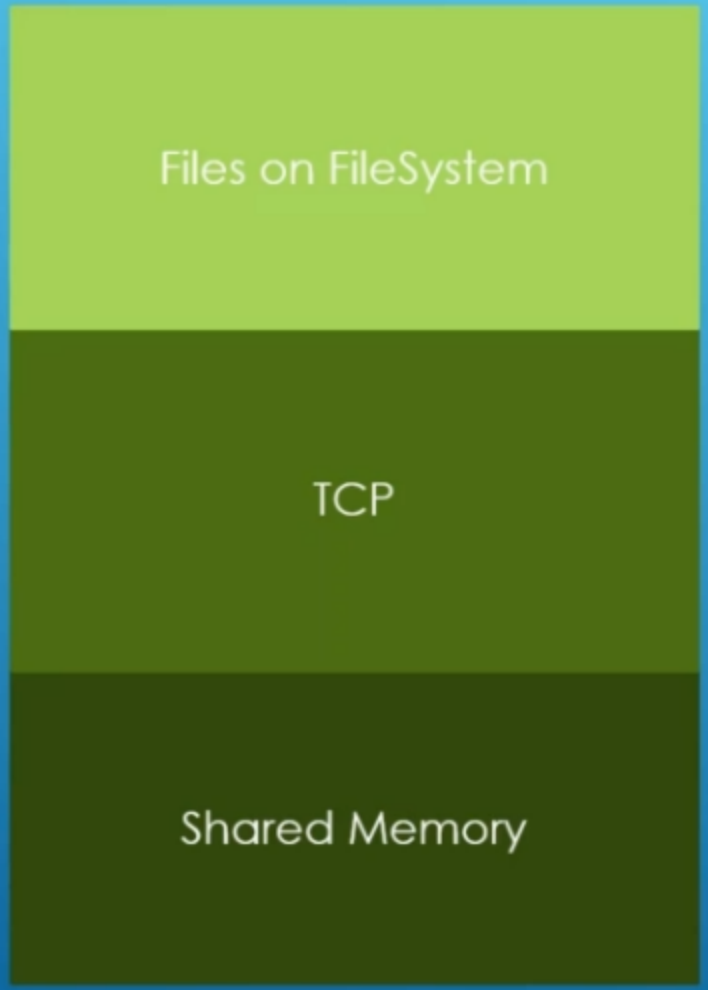

Files on FileSystem
TCP
Shared Memory

DBus

A well defined standard for IPC, supporting signals and slots

*

Well supported ONLY on Linux.


Files on FileSystem
TCP
Shared Memory

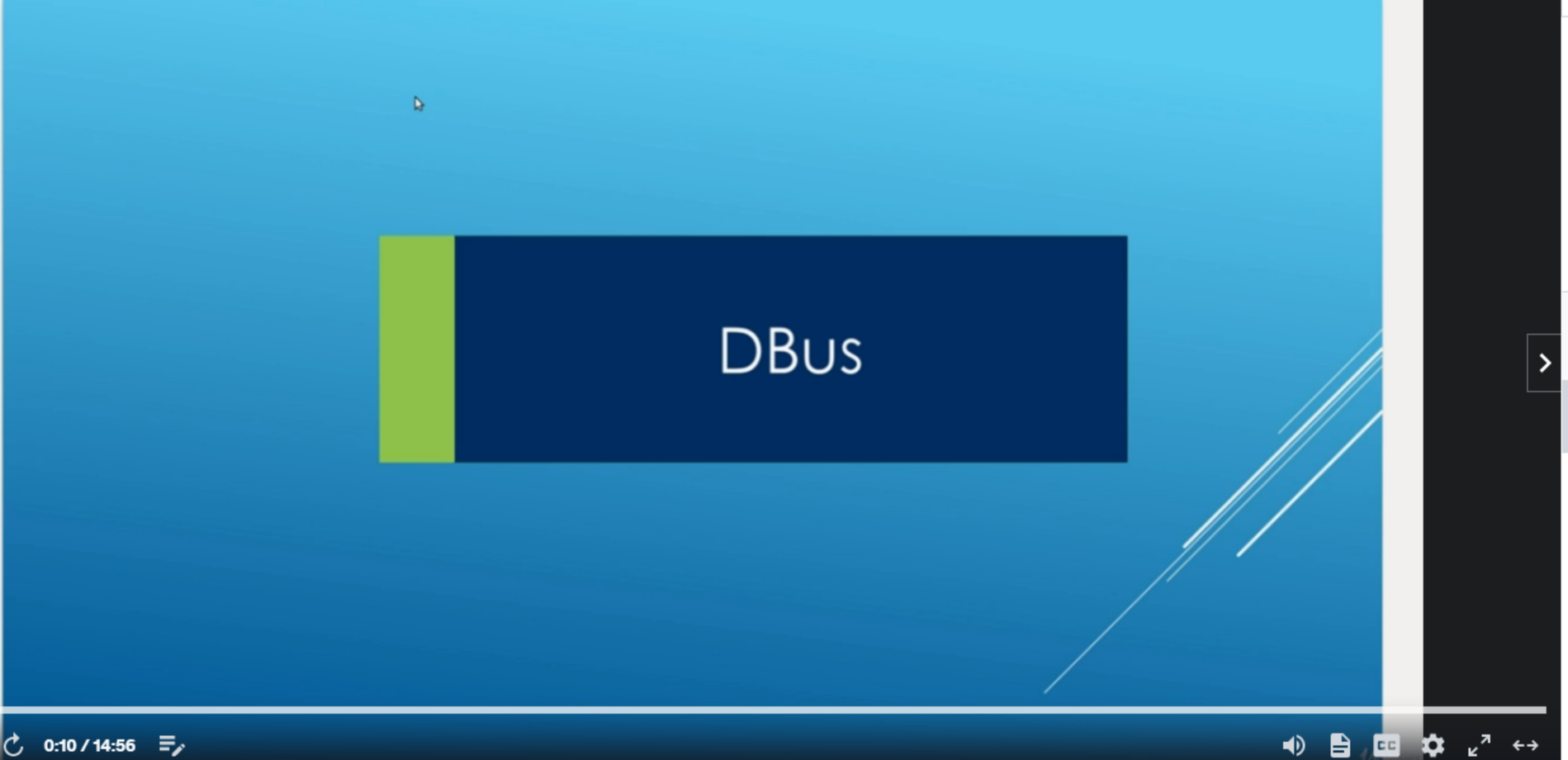
DBus

A well defined standard for IPC, supporting signals and slots

*

Well supported ONLY on Linux.

TCP/IP is recommended if you need your IPC to be cross platform. Use Shared Memory and Files ONLY if you REALLY have to



44. IPC -TCP(QTcpSocket)
25min Resources

45. IPC - Files on FileSystem
14min Resources

46. IPC Overview-Comparison
3min

Section 8: DBus
0 / 8 | 2hr 17min

47. DBus Overview
15min

48. DBus-Client-Server : Server
26min Resources

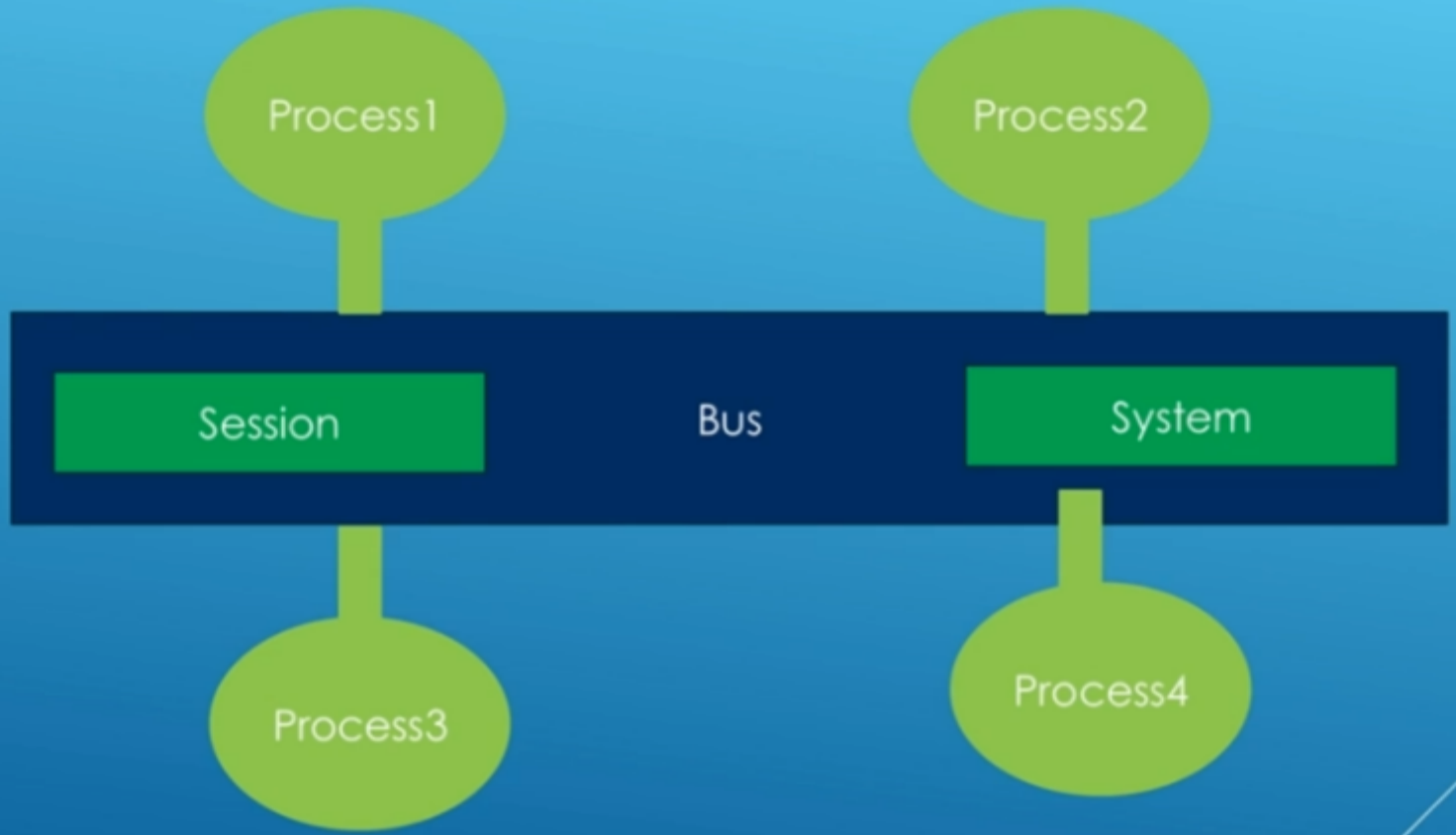
49. DBus Client-Server : Client
14min Resources

50. DBus-SignalsSlots
25min Resources

51. QDBusMessage

An Inter Process Communication mechanism under Linux and other Unix like systems

- \+ Very intuitive, Clearly defined protocol
- \+ Call methods on objects in other processes
- \+ Signals and slots
- \+ Share your services with other processes
- \+ Consume services from other processes

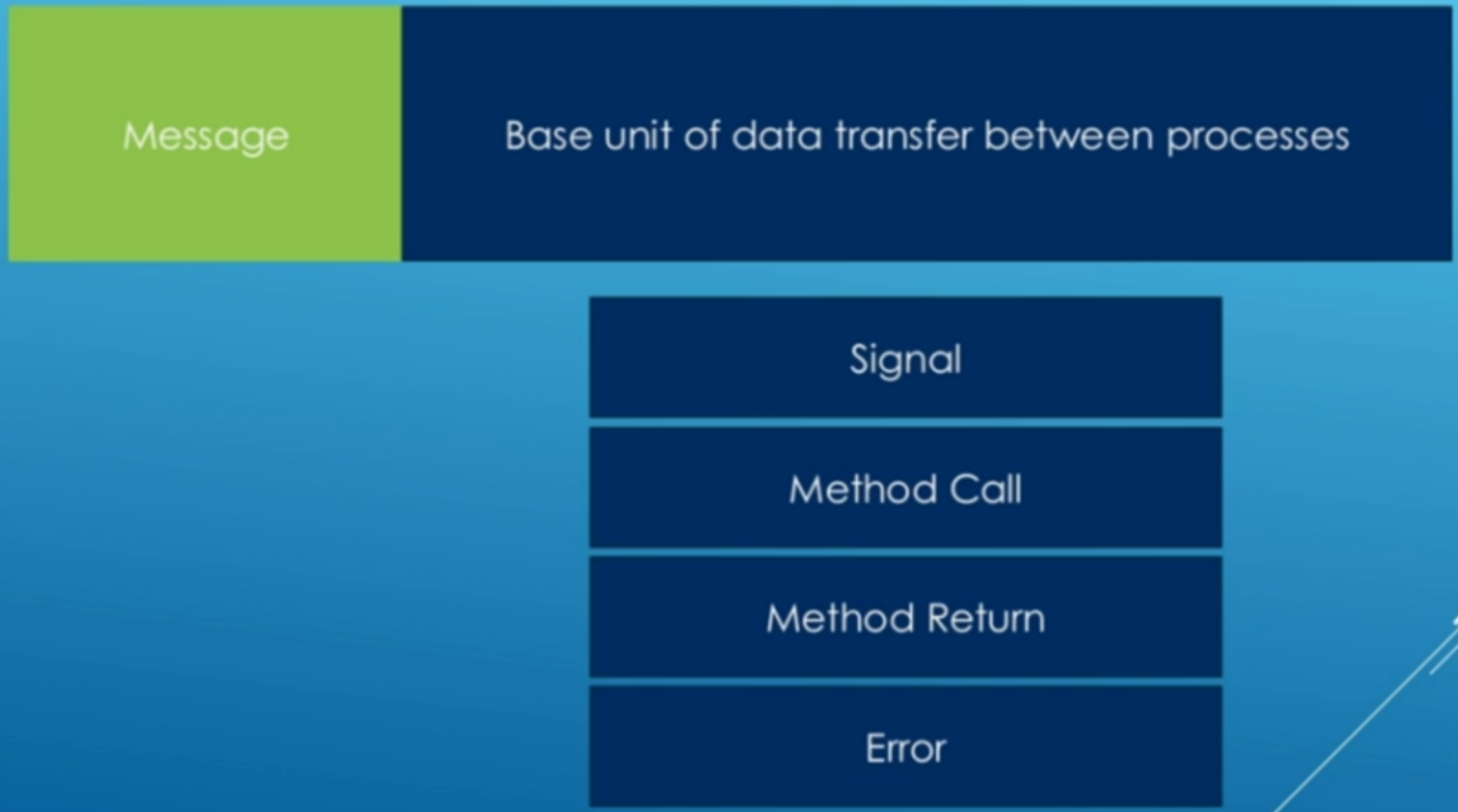

Process1
Process2
Session
Bus
System
Process3
Process4

Clearly defined standard	https://dbus.freedesktop.org/doc/dbus-specification.html

Implementation	libbdbus : C language interface for communication between two processes


Gdbus for GLib
QtDBus for Qt
dbus -java
libdbus

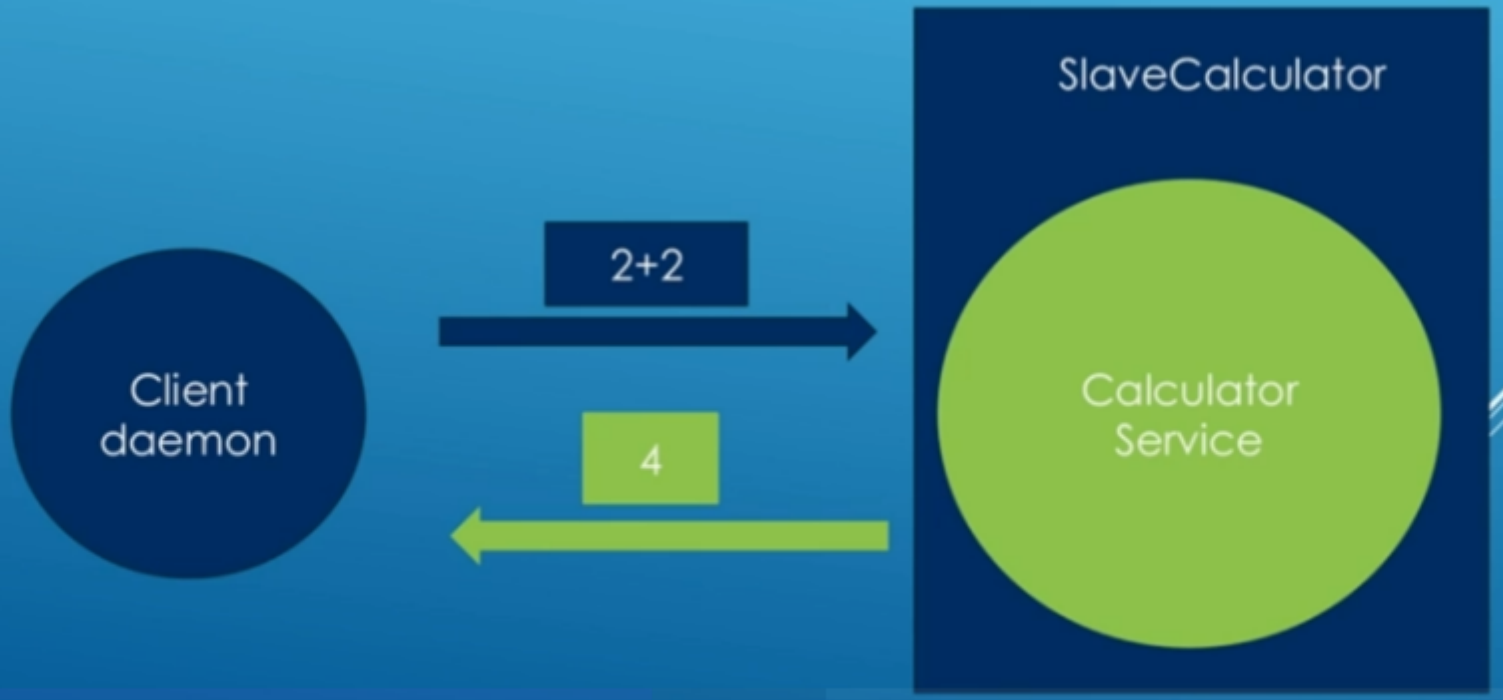
DBus : Basic Concepts


Message
Base unit of data transfer between processes
Signal
Method Call
Method Return
Error

Message Bus	Implemented as a daemon on the system. Routes messages between processes

Service	A daemon on the system that provides some service to other processes

Object	An entity in the process that encapsulates the service provided by the process



Interface	A group of methods exposed by a process

QtDbus

Qt D-Bus

Introduction

D-Bus is an Inter-Process Communication (IPC) and Remote Procedure Calling (RPC) mechanism originally developed for Linux to replace existing and competing IPC solutions with one unified protocol. It has also been designed to allow communication between system-level processes (such as printer and hardware driver services) and normal user processes.

It uses a fast, binary message-passing protocol, which is suitable for same-machine communication due to its low latency and low overhead. Its specification is currently defined by the `freedesktop.org` project, and is available to all parties.

Communication in general happens through a central server application, called the "bus" (hence the name), but direct application-to-application communication is also possible. When communicating on a bus, applications can query which other applications and services are available, as well as activate one on demand.

The Buses

D-Bus buses are used to when many-to-many communication is desired. In order to achieve that, a central server is launched before any applications can connect to the bus: this server is responsible for keeping track of the applications that are connected and for properly routing messages from their source to their destination.

In addition, D-Bus defines two well-known buses, called the system bus and the session bus. These buses are special in the sense that they have well-defined semantics: some services are defined to be found in one or both of these buses.

For example, an application wishing to query the list of hardware devices attached to the computer will probably communicate to a service available on the system bus, while the service providing opening of the user's web browser will be probably found on the session bus.

D-Bus is an Inter-Process Communication (IPC) and Remote Procedure Calling (RPC) mechanism originally developed for Linux to replace existing and competing IPC solutions with one unified protocol. It has also been designed to allow communication between system-level processes (such as printer and hardware driver services) and normal user processes.

It uses a fast, binary message-passing protocol, which is suitable for same-machine communication due to its low latency and low overhead. Its specification is currently defined by the `freedesktop.org` project, and is available to all parties.

Communication in general happens through a central server application, called the "bus" (hence the name), but direct application-to-application communication is also possible. When communicating on a bus, applications can query which other applications and services are available, as well as activate one on demand.

The Buses

D-Bus buses are used to when many-to-many communication is desired. In order to achieve that, a central server is launched before any applications can connect to the bus: this server is responsible for keeping track of the applications that are connected and for properly routing messages from their source to their destination.

In addition, D-Bus defines two well-known buses, called the system bus and the session bus. These buses are special in the sense that they have well-defined semantics: some services are defined to be found in one or both of these buses.

For example, an application wishing to query the list of hardware devices attached to the computer will probably communicate to a service available on the system bus, while the service providing opening of the user's web browser will be probably found on the session bus.

On the system bus, one can also expect to find restrictions on what services each application is allowed to offer. Therefore, one can be reasonably certain that, if a certain service is present, it is being offered by a trusted application.

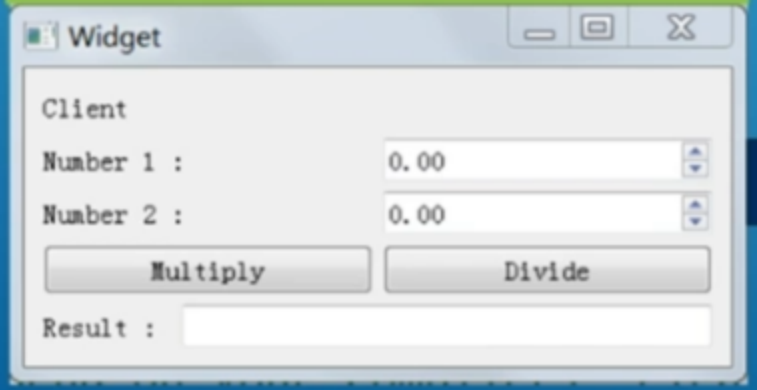
Concepts

Messages

On the low level, applications communicate over D-Bus by sending messages to one another. Messages are used to relay the remote procedure calls as well as the replies and errors associated with them. When used over a bus, messages have

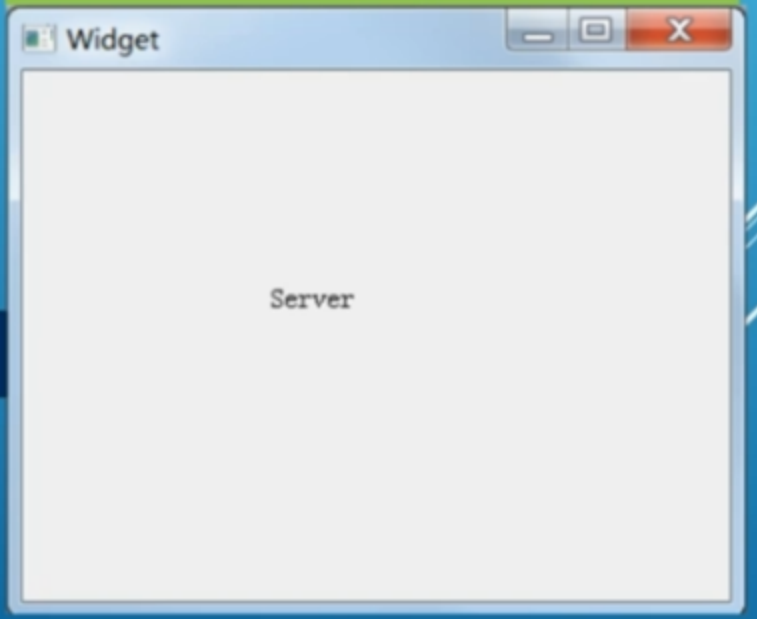
Call methods on objects living in other process using a handle in client

Client

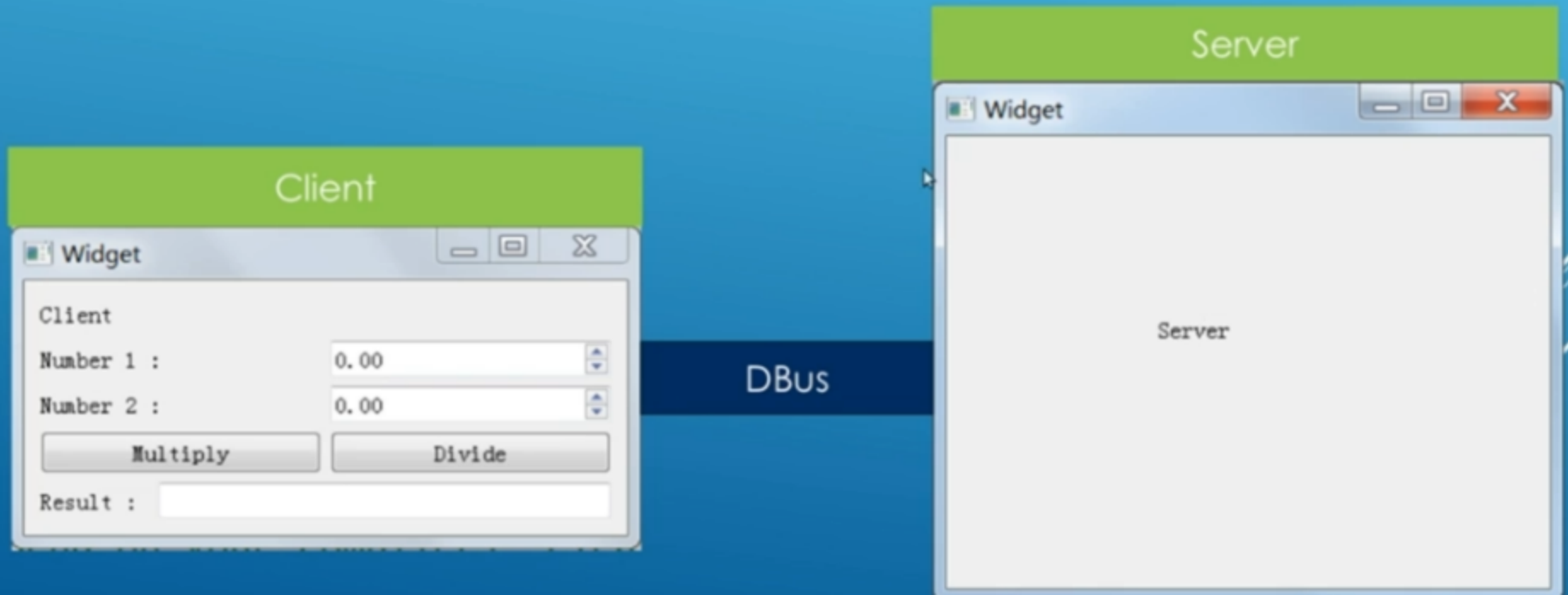


DBus

Server

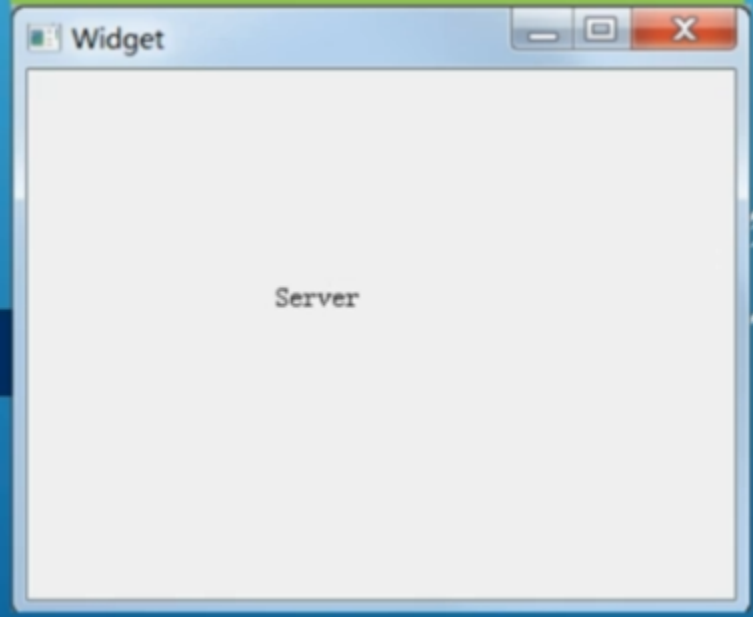


Signals and slots across processes

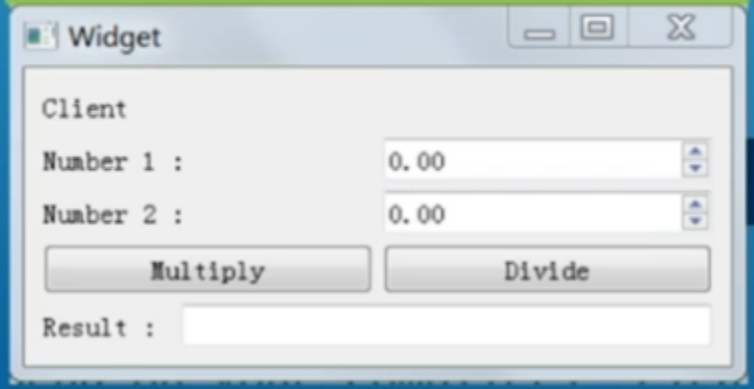


QDBusMessage

Server



Client



DBus

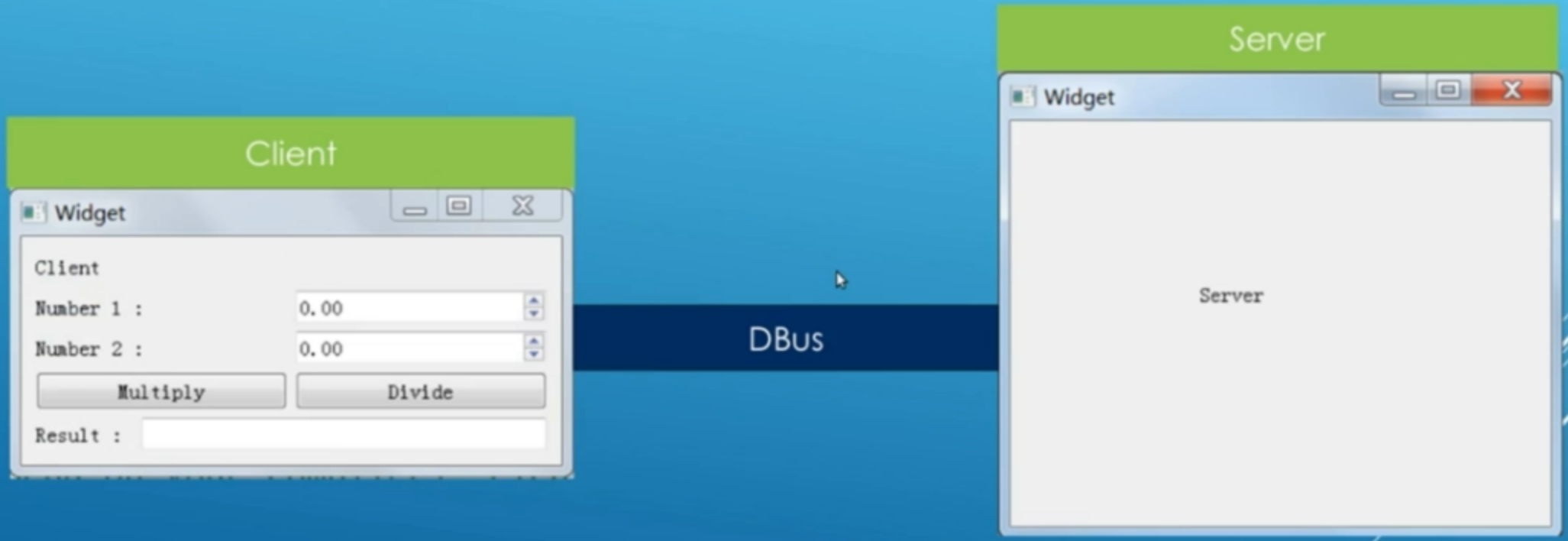
Introspection

Section 8: DBus

1 / 8 | 2hr 17min

Load DBus module in project file

```
QT          += dbus core gui
```


Client
Widget
Client
Number 1 :
0.00
Number 2 :
0.00
Multiply
Divide
Result :
DBus
Server
Widget
Server

Exposed class

```
class SlaveCalculator : public QObject
{
    Q_OBJECT
    Q_CLASSINFO("D-Bus Interface", "com.blikoon.CalculatorInterface")
public:
    explicit SlaveCalculator(QObject *parent = nullptr);

public slots:
    double multiply(double factor1, double factor2);
    double divide(double divident, double divisor);
};
```

Exposed class Implementation

```
SlaveCalculator::SlaveCalculator(QObject *parent) : QObject(parent)
{

}

double SlaveCalculator::multiply(double factor1, double factor2)
{
    double product = factor1 * factor2;
    return product;
}

double SlaveCalculator::divide(double divident, double divisor)
{
    double quotient = divident / divisor;
    return quotient;
}
```

Exposed class

```
class SlaveCalculator : public QObject
{
    Q_OBJECT
    Q_CLASSINFO("D-Bus Interface", "com.blikoon.CalculatorInterfa
public:
    explicit SlaveCalculator(QObject *parent = nullptr);

public slots:
    double multiply(double factor1, double factor2);
    double divide(double divident, double divisor);
};
```

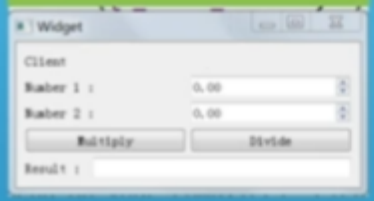
Adaptor

Interface

Interface XML Descriptor file

```
<!DOCTYPE node PUBLIC "-//freedesktop//DTD D-BUS Object Introspection 1.0//EN"
 "http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">
<node name ="com/blikoon/multdivservice">
  <interface name="com.blikoon.CalculatorInterface">
    <method name="multiply">
      <arg type="d" direction="out"/>
      <arg name="factor1" type="d" direction="in"/>
      <arg name="factor2" type="d" direction="in"/>
    </method>
    <method name="divide">
      <arg type="d" direction="out"/>
      <arg name="divident" type="d" direction="in"/>
      <arg name="divisor" type="d" direction="in"/>
    </method>
  </interface>
</node>
```

Client



Server

Other Applications

Interface

XML

Adaptor

XML

DBus

Exposed class

```
class SlaveCalculator : public QObject
{
    Q_OBJECT
    Q_CLASSINFO("D-Bus Interface", "com.blikoon.CalculatorInterface")
public:
    explicit SlaveCalculator(QObject *parent = nullptr);

public slots:
    double multiply(double factor1, double factor2);
    double divide(double divident, double divisor);
};
```

qdbuscpp2xml

```
<!DOCTYPE node PUBLIC "-//freedesktop//DTD D-BUS Object Introspection 1.0//EN"
 "http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">
<node name ="com/blikoon/multdivservice">
  <interface name="com.blikoon.CalculatorInterface">
    <method name="multiply">
      <arg type="d" direction="out"/>
      <arg name="factor1" type="d" direction="in"/>
      <arg name="factor2" type="d" direction="in"/>
    </method>
    <method name="divide">
      <arg type="d" direction="out"/>
      <arg name="divident" type="d" direction="in"/>
      <arg name="divisor" type="d" direction="in"/>
    </method>
  </interface>
</node>
```

```xml
<!DOCTYPE node PUBLIC "-//freedesktop//DTD D-BUS Object Introspection 1.0//EN"
 "http://www.freedesktop.org/standards/dbus/1.0/introspect.dtd">
<node name ="com/blikoon/multdivservice">
  <interface name="com.blikoon.CalculatorInterface">
    <method name="multiply">
      <arg type="d" direction="out"/>
      <arg name="factor1" type="d" direction="in"/>
      <arg name="factor2" type="d" direction="in"/>
    </method>
    <method name="divide">
      <arg type="d" direction="out"/>
      <arg name="divident" type="d" direction="in"/>
      <arg name="divisor" type="d" direction="in"/>
    </method>
  </interface>
</node>
```

qdbusxml2cpp

Interface

Adaptor

Generate XML Descriptor File

```
/path/to/qdbuscpp2xml -M -s slavecalculator.h -o slavecalculator.xml
```

Generate Adaptor File

```
/path/to/qdbusxml2cpp -a calculatoradaptor slavecalculator.xml
```

Generate Interface File

```
/path/to/qdbusxml2cpp -p calculatorInterface slavecalculator.xml
```

Register object and Service [Server]

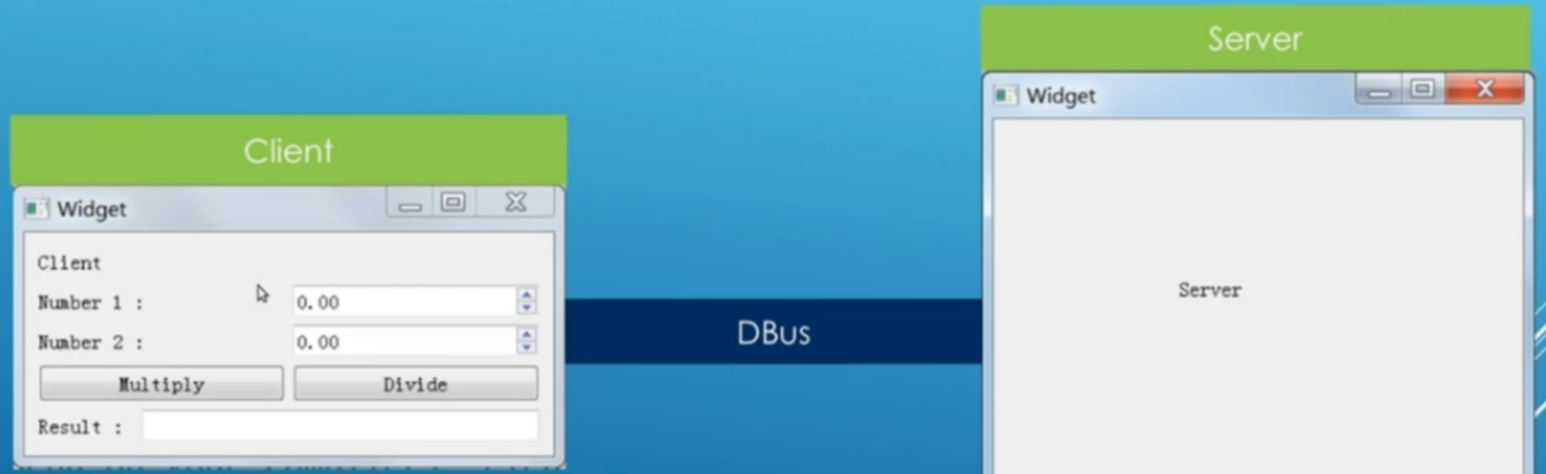
```
//Create actual calculator
slaveCalculator = new SlaveCalculator(this);

/* ...

new CalculatorInterfaceAdaptor(slaveCalculator);

QDBusConnection connection = QDBusConnection::sessionBus();

//Here you pass in the object that you want expose to Dbus.
//Take note of this info, it is used in client.
connection.registerObject("/CalcServicePath", slaveCalculator);

//Here you pass in the service name.
//Take note of this as it is also needed for clients to reach our exposed object.
connection.registerService("com.blikoon.CalculatorService");
```

Client Register Handle

```
calculatorHandle = new com::blikoon::CalculatorInterface("com.blikoon.CalculatorService",
                                                          "/CalcServicePath",
                                                          QDBusConnection::sessionBus());
```

Client using handle to access exposed interface

```
void Widget::on_multiplyButton_clicked()
{
    if(calculatorHandle->isValid()){
        double result = calculatorHandle->multiply(ui->number1SpinBox->value(),ui->number2SpinBox->value());
        ui->resultLineEdit->setText(QString::number(result));
    }else{
        qCritical() << "Calculator Interface is not valid";
    }

}

void Widget::on_divideButton_clicked()
{
    if(calculatorHandle->isValid()){
        double result = calculatorHandle->divide(ui->number1SpinBox->value(),ui->number2SpinBox->value());
        ui->resultLineEdit->setText(QString::number(result));
    }else{
        qCritical() << "Calculator Interface is not valid";
    }
}
```



44. IPC -TCP(QTcpSocket)
25min
Resources

45. IPC - Files on FileSystem
14min
Resources

46. IPC Overview-Comparison
3min

Section 8: DBus
2 / 8 | 2hr 17min

47. DBus Overview
15min

48. DBus-Client-Server : Server
26min
Resources

49. DBus Client-Server : Client
14min
Resources

50. DBus-SignalsSlots

DBus: Signals and Slots

Exposed object emitting signals

```
class SlaveCalculator : public QObject
{
    Q_OBJECT
    Q_CLASSINFO("D-Bus Interface", "com.blikoon.CalculatorInterface")
public:
    explicit SlaveCalculator(QObject *parent = nullptr);

public slots:
    double multiply(double factor1, double factor2);
    double divide(double divident, double divisor);

signals:
    void productResult(double product);
    void divisionResult(double quotient);
};
```

Exposed object emitting signals

```
double SlaveCalculator::multiply(double factor1, double factor2)
{
    double product = factor1 * factor2;
    emit productResult(product);
    return product;
}

double SlaveCalculator::divide(double divident, double divisor)
{
    double quotient = divident / divisor;
    emit divisionResult(quotient);
    return quotient;
}
```

Client Connecting to Server Signals

```
bool connected  = QDBusConnection::sessionBus().connect(QString(),QString(),
                                                        "com.blikoon.CalculatorInterface",
                                                        "productResult","d",
                                                        this,SLOT(gotProduct(double)));
qDebug() << "product signal connected : " << connected;

connected  = QDBusConnection::sessionBus().connect(QString(),QString(),
                                                   "com.blikoon.CalculatorInterface",
                                                   "divisionResult","d",
                                                   this,SLOT(gotQuotient(double)));
qDebug() << "product signal connected : " << connected;
```

Client Slots

```
void Widget::gotProduct(double value)
{
    qDebug() << "Client got server signal for product : " << value;
}

void Widget::gotQuotient(double value)
{
    qDebug() << "Client got server signal for quotient : " << value;
}
```

Generate XML Descriptor File

```
/path/to/qdbuscpp2xml -M -s slavecalculator.h -o slavecalculator.xml
```

Generate Adaptor File

```
/path/to/qdbusxml2cpp -a calculatoradaptor slavecalculator.xml
```

Generate Interface File

```
/path/to/qdbusxml2cpp -p calculatorInterface slavecalculator.xml
```

Generate XML Descriptor File

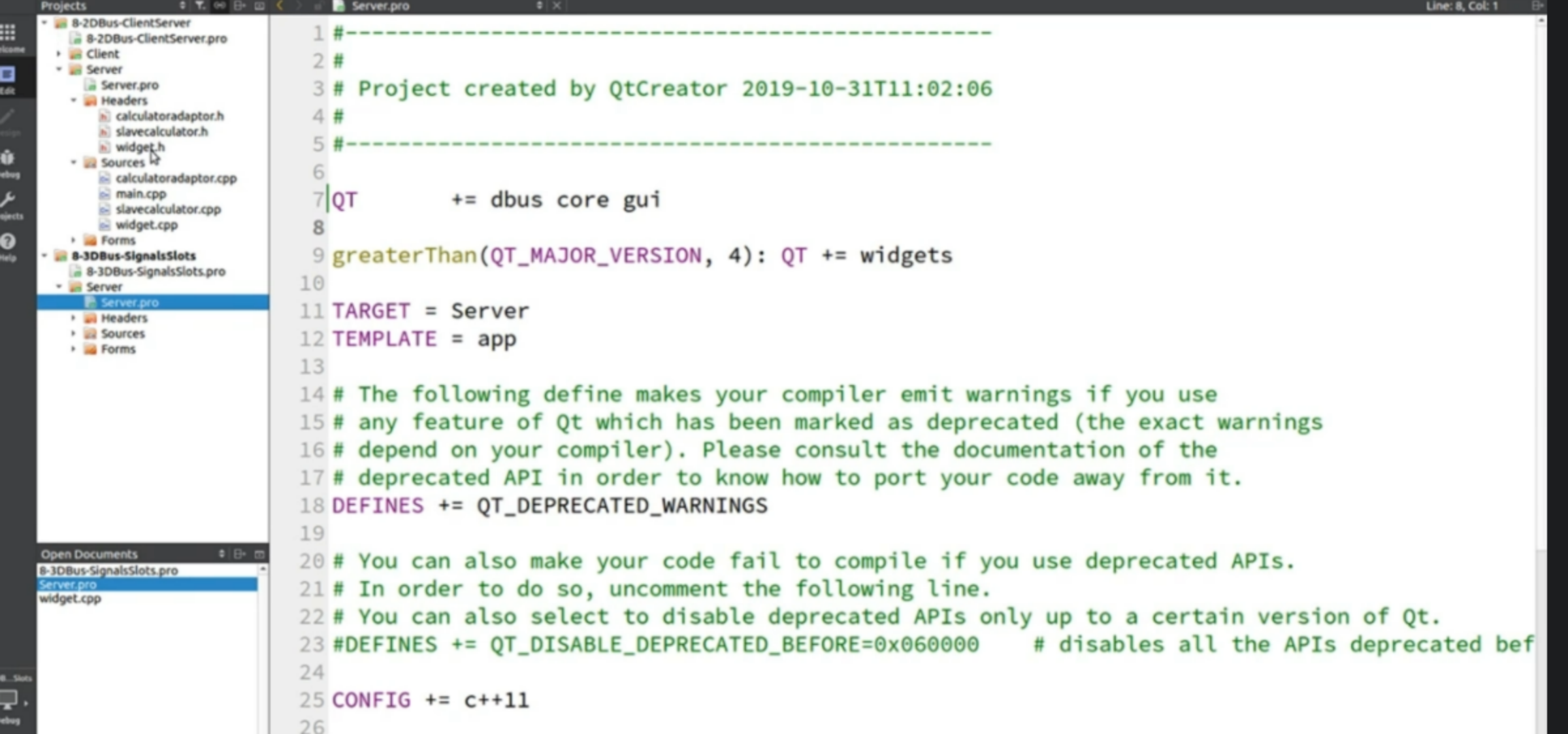
```
/path/to/qdbuscpp2xml -M -s slavecalculator.h -o slavecalculator.xml
```

Generate Adaptor File

```
/path/to/qdbusxml2cpp -a calculatoradaptor slavecalculator.xml
```

Generate Interface File

```
/path/to/qdbusxml2cpp -p calculatorInterface slavecalculator.xml
```


Projects
8-2DBus-ClientServer
8-2DBus-ClientServer.pro
Client
Server
Server.pro
Headers
calculatoradaptor.h
slavecalculator.h
widget.h
Sources
calculatoradaptor.cpp
main.cpp
slavecalculator.cpp
widget.cpp
Forms
8-3DBus-SignalsSlots
8-3DBus-SignalsSlots.pro
Server
Server.pro
Headers
Sources
Forms
Open Documents
8-3DBus-SignalsSlots.pro
Server.pro
widget.cpp
Server.pro
Line: 8, Col: 1
#---
#
Project created by QtCreator 2019-10-31T11:02:06
#
#---
QT += dbus core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = Server
TEMPLATE = app
# The following define makes your compiler emit warnings if you use
# any feature of Qt which has been marked as deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS
You can also make your code fail to compile if you use deprecated APIs.
In order to do so, uncomment the following line.
You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000 # disables all the APIs deprecated bef
CONFIG += c++11

QDBusMessage

Load DBus module in project file

```
QT      += dbus core gui
```

Call methods in other processes without return value : QDBusConnection::send()

```
void Widget::on_multiplyButton_clicked()
{

    QDBusMessage m = QDBusMessage::createMethodCall("com.blikoon.CalculatorService",
                                                    "/CalcServicePath",
                                                    "com.blikoon.CalculatorInterface",
                                                    "multiply");
    //First way to pass arguments
    m << ui->number1SpinBox->value() << ui->number2SpinBox->value();

    //Send doesn't give us response

    bool queued = QDBusConnection::sessionBus().send(m);
    qDebug() << "Message queued : " << queued;
}
```

Call methods in other processes with return values : QDBusConnection::call()

```
void Widget::on_multiplyButton_clicked()
{

    QDBusMessage m = QDBusMessage::createMethodCall("com.blikoon.CalculatorService",
                                                    "/CalcServicePath",
                                                    "com.blikoon.CalculatorInterface",
                                                    "multiply");
    //First way to pass arguments
    m << ui->number1SpinBox->value() << ui->number2SpinBox->value();

    //Call gives a response
    QDBusMessage response = QDBusConnection::sessionBus().call(m);

    if(response.type() == QDBusMessage::ReplyMessage){
        qDebug() << "Got a reply from the server" << response.arguments();
        if(response.arguments().count() ==1){
            ui->resultLineEdit->setText(QString::number(response.arguments()[0].toInt()));
        }
    }
}
```

Send Signals with QDBusMessage

```
void Widget::on_messageToServerButton_clicked()
{
    QDBusMessage msg = QDBusMessage::createSignal("/CalcServicePath",
                                                  "com.blikoon.CalculatorInterface",
                                                  "message");

    msg << "Hello there";
    QDBusConnection::sessionBus().send(msg);
}
```

Connect to Signal in the server process

```
QDBusConnection::sessionBus().connect(QString(), QString(),
                                      "com.blikoon.CalculatorInterface",
                                      "message", this, SLOT(messageSlot(QString)));
```

Signal treated as if it's coming from session bus immediately. We don't care that much where it really comes from

Connect to Signal in the server process

```
QDBusConnection::sessionBus().connect(QString(), QString(),
                                      "com.blikoon.CalculatorInterface",
                                      "message", this, SLOT(messageSlot(QString)));
```

Signal treated as if it's coming from session bus immediately. We don't care that much where it really comes from

QDBusInterface

QDBusInterface can be used directly to call methods on other processes on the system

Initialize interface with correct information

```
interface = new QDBusInterface("com.blikoon.CalculatorService",
                               "/CalcServicePath",
                               "com.blikoon.CalculatorInterface");
interface->setParent(this);//For memory management

//Connect to signal from server process
connect(interface,SIGNAL(productResult(double)),this,SLOT(gotProduct(double)));
```

Call methods using interface object

```
void Widget::on_multiplyButton_clicked()
{
    //Can pas up to 8 parameters this way
    QDBusReply<double> reply = interface->call("multiply",ui->number1SpinBox->value(),
                                               ui->number2SpinBox->value());

    if(reply.isValid()){
        ui->resultLineEdit->setText(QString::number(reply.value()));
        qDebug() << "Got a valid reply for multiplication : " << reply.value();
    }
}
```

QDBusInterface::callWithArgumentList()

```
void Widget::on_multiplyButton_clicked()
{
    QList<QVariant> args;

    args.append(ui->number1SpinBox->value());
    args.append((ui->number2SpinBox->value()));

    //Can use an arg list if we want
    QDBusReply<double> reply = interface->callWithArgumentList(QDBus::Block,"multiply",args);

    if(reply.isValid()){
        ui->resultLineEdit->setText(QString::number(reply.value()));
        qDebug() << "Got a valid reply for multiplication : " << reply.value();
    }
}
```

Call method asynchronously

```
//Can call asynchronouslv. a slot.
interface->callWithCallback("multiply",args,this,SLOT(gotProduct(double)));
```

&A Notes Announcements Reviews Learning tools

25min Resources

45. IPC - Files on FileSystem
14min Resources

46. IPC Overview-Comparison
3min

Section 8: DBus
6 / 8 | 2hr 17min

47. DBus Overview
15min

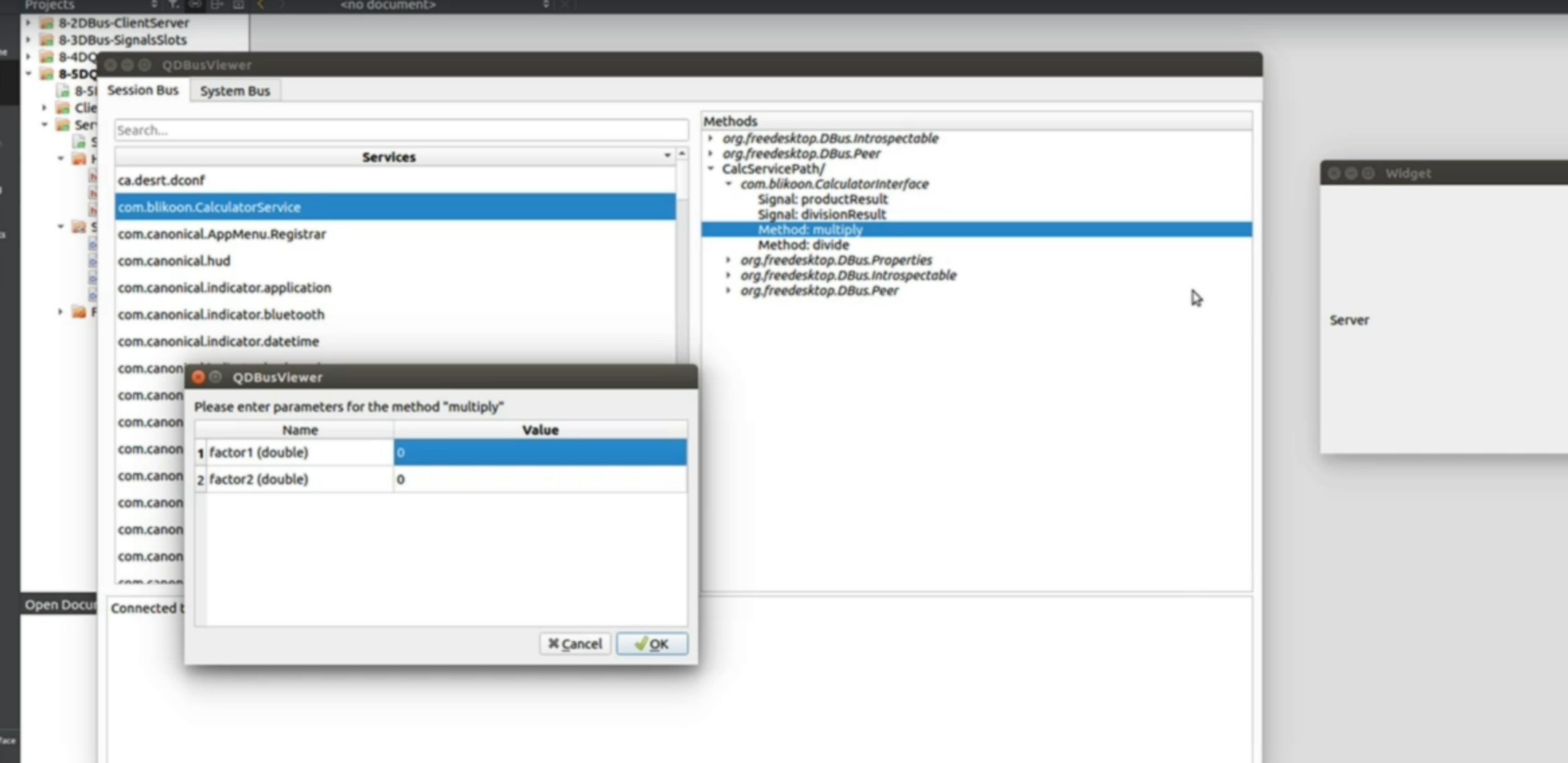
48. DBus-Client-Server : Server
26min Resources

49. DBus Client-Server : Client
14min Resources

50. DBus-SignalsSlots
25min Resources

51. QDBusMessage
26min Resources

52. QDBusInterface
20min Resources


Projects
<no document>
8-2DBus-ClientServer
8-3DBus-SignalsSlots
QDBusViewer
Session Bus
System Bus
Search...
Services
ca.desrt.dconf
com.blikoon.CalculatorService
com.canonical.AppMenu.Registrar
com.canonical.hud
com.canonical.indicator.application
com.canonical.indicator.bluetooth
com.canonical.indicator.datetime
Methods
org.freedesktop.DBus.Introspectable
org.freedesktop.DBus.Peer
CalcServicePath/
com.blikoon.CalculatorInterface
Signal: productResult
Signal: divisionResult
Method: multiply
Method: divide
org.freedesktop.DBus.Properties
org.freedesktop.DBus.Introspectable
org.freedesktop.DBus.Peer
QDBusViewer
Please enter parameters for the method "multiply"
Name
Value
1 factor1 (double)
0
2 factor2 (double)
0
Cancel
OK
Widget
Server
Open Docu
Connected t


qdbuscpp2xml, qdbusxml2cpp
QDbusMessage
QDBusInterface
Introspection
Signals & Slots


Learn
Qt
C++

Creating Threads


Creating Threads
Thread Synchronization
Thread Safety and Reentrancy
Qt Concurrent
Processes
Inter Process Communication
DBus

Build Stuff, Ask Questions, Avoid Tutorial Purgatory.

Course content

Section 1: Introduction
2 / 2 | 17min

Section 2: Creating Threads
12 / 12 | 2hr 50min

Section 3: Thread Synchronization
8 / 8 | 2hr 9min

Section 4: Thread Safety and Reentrancy
6 / 6 | 1hr 10min

Section 5: Qt Concurrent
12 / 12 | 2hr 37min

Section 6: Processes
1 / 1 | 15min

Section 7: Inter Process Communication
5 / 5 | 1hr 14min

Section 8: DBus
8 / 8 | 2hr 17min

Section 9: Wrap up
1 / 1 | 4min

- [x] 55. Course Wrap up
  4min

Section 1: Introduction

2 / 2 | 17min

1. Welcome

5min

2. Threads, Processes and IPC : Context

12min

Section 2: Creating Threads

12 / 12 | 2hr 50min

- [x] 3. Threading Overview
  - 8min
- [x] 4. QThread-Create
  - 27min
  - Resources
- [x] 5. MoveToThread
  - 30min
  - Resources
- [x] 6. Subclass QThread
  - 16min
  - Resources
- [x] 7. QThread with asynchronous code-QThread-Create
  - 8min
  - Resources
- [x] 8. QThread with asynchronous code-MoveToThread
  - 13min
  - Resources
- [x] 9. QThread with asynchronous code-Subclass QThread
  - 10min
  - Resources
- [x] 10. ThreadPool and QRunnable
  - 11min
  - Resources
- [x] 11. ThreadPool and QRunnable - Sending feedback to ui
  - 24min
  - Resources
- [x] 12. ThreadPool and QRunnable - Async Code
  - 5min
  - Resources
- [x] 13. Custom Type Signal Parameters
  - 13min
  - Resources
- [x] 14. Threading Methods Comparison
  - 5min

Section 3: Thread Synchronization

8 / 8 | 2hr 9min

- [x] 15. Thread Synchronization Overview
  - 4min
- [x] 16. Thread Synchronization - Mutex
  - 21min
  - Resources
- [x] 17. Thread Synchronization - Mutex -Shared variable
  - 15min
  - Resources
- [x] 18. Thread Synchronization - ReadWrite Lock
  - 17min
  - Resources
- [x] 19. Thread Synchronization - Semaphores
  - 30min
  - Resources
- [x] 20. Thread Synchronization - WaitConditions
  - 21min
  - Resources
- [x] 21. Wait Conditions - Pause Resume
  - 19min
  - Resources
- [x] 22. Thread Synchronization- Chapter Review
  - 2min

Section 4: Thread Safety and Reentrancy

6 / 6 | 1hr 10min

- [x] 23. Thread Safety and Reentrancy Overview
  16min
- [x] 24. Cross Thread Signals and Slots - Example1
  22min Resources
- [x] 25. Cross Thread Signals and Slots - Example2
  14min Resources
- [x] 26. Cross Thread Signals and Slots - Example3
  7min Resources
- [x] 27. Slots in QThread Subclass
  10min Resources
- [x] 28. Thread Safety and Reentrancy - Chapter Review
  1min

Section 5: Qt Concurrent

12 / 12 | 2hr 37min

- [x] 29. Qt Concurrent Overview
  - 6min
- [x] 30. Qt Concurrent-run-synchronous
  - 14min
  - Resources
- [x] 31. Qt Concurrent Asynchronous - Return values
  - 23min
  - Resources
- [x] 32. Qt Concurrent-map
  - 13min
  - Resources
- [x] 33. Qt Concurrent-maped
  - 15min
  - Resources
- [x] 34. Qt Concurrent-mapReduced
  - 33min
  - Resources
- [x] 35. Qt Concurrent-Filter
  - 17min
  - Resources
- [x] 36. Qt Concurrent-Filtered
  - 9min
  - Resources
- [x] 37. Qt Concurrent-FilterReduce
  - 10min
  - Resources
- [x] 38. Qt Concurrent-QFutureSynchronizer
  - 7min
  - Resources
- [x] 39. Qt Concurrent : Feedback
  - 4min
- [x] 40. Threading Overview-Comparison
  - 6min

Section 6: Processes

1 / 1 | 15min

- [x] 41. Processes and QProcess
  - 15min
  - Resources

Section 7: Inter Process Communication

5 / 5 | 1hr 14min

- [x] 42. IPC Overview
  - 6min
- [x] 43. IPC- SharedMemory
  - 24min
  - Resources
- [x] 44. IPC -TCP(QTcpSocket)
  - 25min
  - Resources
- [x] 45. IPC - Files on FileSystem
  - 14min
  - Resources
- [x] 46. IPC Overview-Comparison
  - 3min

Section 8: DBus

8 / 8 | 2hr 17min

- [x] 47. DBus Overview
  15min
- [x] 48. DBus-Client-Server : Server
  26min Resources
- [x] 49. DBus Client-Server : Client
  14min Resources
- [x] 50. DBus-SignalsSlots
  25min Resources
- [x] 51. QDBusMessage
  26min Resources
- [x] 52. QDBusInterface
  20min Resources
- [x] 53. Introspection
  9min
- [x] 54. DBus-Overview
  2min

Section 9: Wrap up

1 / 1 | 4min

- [x] 55. Course Wrap up
  4min