Haute École de Bruxelles
École Supérieure d'Informatique

Report for stage done at
Von Karman Institute for Fluid Dynamics

User Interface Development for COOLFluiD

Annexes volume 4 - Source code and code maintenance

GASPER Quentin

Stage master : Professor Herman Deconinck
Advisor : Doctor Tiago Quintino
Reference teacher : Nicolas Vansteenkiste

June 2009

Contents

Part I

Source code

Chapter 1

Client

1.1 *AddNodeDialog* class

1.1.1 AddNodeDialog.h

```
#ifndef COOLFluiD_client_AddNodeDialog_h
#define COOLFluiD_client_AddNodeDialog_h

#include <QDialog>
#include <QObject>

class QComboBox;
class QFormLayout;
class QLabel;
class QLineEdit;
class QMainWindow;
class QDialogButtonBox;

////////////////////////////////////////////////////////////////////////////////

namespace COOLFluiD
{
 namespace client
 {

////////////////////////////////////////////////////////////////////////////////
  /// @brief Dialog used to add a node.

  /// This class inherits from <code>QDialog</code> and is used to show a
  /// dialog allowing the user to create a new node. The dialog is modal,
  /// which means that once it is visible, the calling code execution is
  /// stopped until the dialog is invisible again. The user is invited to
  /// type the name of the new node and select the concrete type of this
  /// node. If the dialog has a parent window, it is centered on this parent.
  /// Otherwise, it is centered on the screen.<br>

  /// After calling the constructor, the dialog is invisible.
  /// <code>show</code> method has to be called to show it. This is a
  /// blocking method: it will not return until is invisible again. This
  /// method returns either the name entered by the user (if he clicked on
  /// "OK" to validate his entry) or an empty string (if he clicked on
  /// "Cancel" or closed the dialog to cancel his entry).<br>

  /// If the user validates his entry, the <code>concreteType</code>
/// parameter is used to store the selected concrete type. The method
/// guarantees that the selected type will be one of the provided list. If
/// the user cancels his entry, the parameter is not modified. If the user
/// clicks on "OK" without typing any name, it is considered as a
/// cancellation.

/// A typical use of this class is (assuming that <code>this</code> is a
/// <code>QMainWindow</code> object and <code>concreteTypes</code> is a
/// <code>QStringList</code> with some concrete types) : <br>
/// \code
/// AddNodeDialog dialog(this);
/// QString type;                // used to store the chosen concrete type
/// QString name = dialog.show(concreteTypes, type);
///
/// if(name != "")
/// {
///  // some treatements
/// }
/// \endcode

/// @author Quentin Gasper.

class AddNodeDialog : QDialog
{
 Q_OBJECT

 private:

  /// @brief Drop-down list that allows the user to select a concrete type.
  QComboBox * cbTypes;

  /// @brief Line edit that allows the user to enter the new object name.
  QLineEdit * editName;

  /// @brief Button box containing "OK" and "Cancel" buttons.
  QDialogButtonBox * buttons;

  /// @brief The parent window.

  /// Can be null.
  QMainWindow * parent;

  /// @brief Label for the line edit
  QLabel * labName;

  /// @brief Label for the drop-down list
  QLabel * labConcreteType;

  /// @brief Layout on which the components will be placed.
  QFormLayout * layout;

  /// @brief Indicates whether the user clicked on "OK" button or not.

  /// If the user clicked on "OK" button, the attribute value is
  /// <code>true</code>, otherwise (if the user closed the window or
  /// clicked on "Cancel" button) it is <code>false</code>.
  bool okClicked;

 public:

  /// @brief Constructor.

  /// @param parent Dialog parent. May be null.
  AddNodeDialog(QMainWindow * parent);
```

```
    /// @brief Destructor.

    /// Frees all allocated before the object is deleted. The parent is not
    /// destroyed.
    ~AddNodeDialog();

    /// @brief Shows the dialog.

    /// This is a blocking method. It will not return until the dialog is
    /// invisible.

    /// @param types List of the available concrete types.
    /// @param concreteType Reference to a <code>QString</code> where the
    /// selected type will be stored if and only if the user clicked on "OK"
    /// and the name is not empty, otherwise the value is unchanged.

    /// @return If the user clicked on "OK", returns the name entered in the
    /// line edit component (may be empty if nothing was entered). Otherwise,
    /// returns an empty string by calling the default <code>QString</code>
    /// constructor. If the provided list is empty, an empty string is
    /// returned.
    QString show(const QStringList & types, QString & concreteType);

   private slots:
    /// @brief Slot called when "OK" button is clicked.
    void btOkClicked();

    /// @brief Slot called when "Cancel" button is clicked.
    void btCancelClicked();
  };

////////////////////////////////////////////////////////////////////////////

 }
}

////////////////////////////////////////////////////////////////////////////

#endif // COOLFluiD_client_AddNodeDialog_h
```

1.1.2 AddNodeDialog.cxx

```
#include <QtGui>

#include "ClientServer/client/AddNodeDialog.h"

using namespace COOLFluiD::client;

AddNodeDialog::AddNodeDialog(QMainWindow * parent)
 : QDialog(parent)
{
 this->setWindowTitle("Add␣a␣new␣child␣node");

 // create the components
 this->labName = new QLabel("Name:");
 this->labConcreteType = new QLabel("Concrete␣type:");
 this->editName = new QLineEdit();
 this->cbTypes = new QComboBox();
 this->layout = new QFormLayout();
 this->buttons = new QDialogButtonBox(QDialogButtonBox::Ok |
   QDialogButtonBox::Cancel);

 // add the components to the layout
 this->layout->addRow(this->labName, this->editName);
 this->layout->addRow(this->labConcreteType, this->cbTypes);
 this->layout->addRow(this->buttons);

 // add the layout to the dialog
 this->setLayout(this->layout);

 // connect useful signals to slots
 connect(this->buttons, SIGNAL(accepted()), this, SLOT(btOkClicked()));
 connect(this->buttons, SIGNAL(rejected()), this, SLOT(btCancelClicked()));

 // the dialog is modal
 this->setModal(true);

 this->okClicked = false;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

AddNodeDialog::~AddNodeDialog()
{
 delete this->labConcreteType;
 delete this->labName;
 delete this->editName;
 delete this->cbTypes;
 delete this->layout;
 delete this->buttons;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


QString AddNodeDialog::show(const QStringList & types, QString & concreteType)
{
 // if the list is empty, there is no need to continue
 if(types.isEmpty())
  return QString();

 // clear the QComboBox and add the new items
 this->cbTypes->clear();
 this->cbTypes->addItems(types);

 // show the dialog (will not return while the dialog is visible)
 this->exec();

 // if the user did not clicked on "OK" or has not entered a name
 // then return an empty string
 if(!this->okClicked || this->editName->text().trimmed() == "")
  return QString();

 // set the selected concrete type and return the name
 concreteType = this->cbTypes->currentText();
 return this->editName->text();
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void AddNodeDialog::btOkClicked()
{
 this->okClicked = true;
 this->setVisible(false);
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void AddNodeDialog::btCancelClicked()
{
 this->okClicked = false;
 this->setVisible(false);
}
```

1.2 *CommClient* class

1.2.1 CommClient.h

```
#ifndef COOLFluid_ClientServer_CommClient_h
#define COOLFluid_ClientServer_CommClient_h

////////////////////////////////////////////////////////////////////////////////

class QDomDocument;
class QDomNode;
class QString;
class QTcpServer;
class QTcpSocket;

#include <QObject>
#include <QAbstractSocket>

#include "ClientServer/network/NetworkException.h"

namespace COOLFluiD
{
 namespace client
 {

////////////////////////////////////////////////////////////////////////////////

  /// @brief This class represents the client network level.

  /// It operates mainly using Qt slots/signals system. Each time a frame
  /// arrives through the socket, the appropriate signal is thrown. Frames
  /// to send are built using functions from
  /// <code>Network::ClientServerXMLParser</code>

  /// @author Quentin Gasper.

  class CommClient : public QObject
  {
   Q_OBJECT

    private:

     /// @brief Socket used to communicate with the server.
     QTcpSocket * socket;

     /// @brief Size of the frame that is being read.

     /// If the value is 0, no frame is currently being recieved.
     quint16 blockSize;

     /// @brief Indicates wether the upper level requested a disconnection.
     bool requestDisc;

     /// @brief Indicates wether the socket is open and connected to the
     /// server.
     bool connectedToServer;

     /// @brief Indicates wether a "Connection refused" error must be skip.

     /// If <code>true</code> when a "Connection refused" error occurs, it is
     /// skipped and this attribute is set to <code>false</code>.
     bool skipRefused;

     /// @brief Sends a frame to the server.
    /// All @e sendXXX methods of this class call this method to
    /// send their frames.

    /// @param frame Frame to send.
    void send(const QString & frame) const;

  public:

    /// @brief Constructor.

    /// The socket <code>client</code> is set to <code>NULL</code>.
    CommClient();

    /// @brief Destructor.

    /// Closes the sockets and free all allocated memory before the object
    /// is deleted.
    ~CommClient();

    /// @brief Attempts to connect the client to the server.

    /// When this method returns, the socket is not open yet. The signal
    /// <code>connected()</code> will be emitted when the first frame
    /// arrives.

    /// @param hostAddress Server address.
    /// @param port Socket port number.
    /// @param skipRefused Value of <code><b>this</b>->skipRefused</code>
    /// during the attempt.
    void connectToServer(const QString & hostAddress = "127.0.0.1",
                         quint16 port = 62784, bool skipRefused = false);

    /// @brief Disconnects from the server, then closes.

    /// After calling this method, <code><b>this</b>->resquetDisc</code>
    /// is <code>true</code>.

    /// @param shutServer If <code>true</code>, a request to shut down the
    /// server is sent.

    void disconnectFromServer(bool shutServer);

    /// @brief Sends an action to the server.

    /// Sends an action to the server. Available actions are defined in
    /// NetworkFrames class. All actions that need an XML tree as data can
    /// be sent through this method.

    /// @param action Type of action. This action must be one of those
    /// defined by NetworkFrames class.
    /// @param data Action data
    void sendAction(int action, const QDomDocument & data);

    /// @brief Sends a request to the server to add a node.

    /// The node parents indicate the path in the tree and all parents must
    /// already exist in the tree on the server, otherwise the server will
    /// send back an error.

    /// @param node Node to add.
    /// @param type Concrete type of the node.
    /// @param absType Abstract type of the node.
    void sendActionAddNode(const QDomNode & node, const QString & type,
                           const QString & absType);
    /// @brief Sends a request to the server to delete a node.

    /// The node parents indicate the path in the tree and all parents and
    /// node to delete must exist in the tree on the server, otherwise the
    /// server will send back an error..

    /// @param node Node to delete.
    void sendActionDeleteNode(const QDomNode & node) const;

    /// @brief Sends a request to the server to get the tree.
    void sendActionGetTree() const;

    /// @brief Sends a request to the server to rename a node.

    /// The node parents indicate the path in the tree and all parents and
    /// node to rename must exist in the tree on the server, otherwise the
    /// server will send back an error. The server will also send back an
    /// error if the another node with the same name as the new already
    /// exists. If the node name and the new name are the same, there is no
    /// error.

    /// @param node Node to rename.
    /// @param newName Node new name.
    void sendActionRenameNode(const QDomNode & node,
                              const QString & newName);

    /// @brief Sends a request to the server to get the abstract types for a
    /// specified type.

    /// @param typeName Type name
    void sendGetAbstractTypes(const QString & typeName);

    /// @brief Sends a request to the server to get the concrete types for a
    /// specified abstract type.

    /// @param typeName Name of the abstract type one want to get the
    /// concrete types list.
    void sendGetConcreteTypes(const QString & typeName);

    /// @brief Sends a request to the server to get the available files list.
    void sendGetFilesList() const;

    /// @brief Sends a request to the server to open a case file.

    /// @param filename File to open
    void sendOpenFile(const QString & filename);

    /// @brief Sends a request to the server to run a simulation.
    void sendRunSimulation();

    /// @brief Sends a request to open a directory and read its content.

    /// @param dirname Directory name to open.
    void sendOpenDir(const QString & dirname);

   public slots :

    /// @brief Slot called when there is an error on the socket.
    void newData();

    /// @brief Slot called when the connection has been broken.
    void disconnected();

    /// @brief Slot called when there is an error on the socket.
    /// @param err Error that occured.
    void socketError(QAbstractSocket::SocketError err);

  signals:

    /// @brief Signal emitted when there is an error in the XML protocol
    /// or if the connection has been broken or refused.

    /// This error can either come from the server or from one of this class
    /// methods.

    /// @param error Error message
    /// @param fromServer <code>true</code> if the error message comes
    /// from the server, otherwise <code>false</code>.
    void error(const QString & error, bool fromServer);

    /// @brief Signal emitted when a message arrives from the server.

    /// @param message Message
    void message(const QString & message);

    /// @brief Signal emitted when the server sends the tree.

    /// @param document The tree.
    void newTree(const QDomDocument & document);

    /// @brief Signal emitted when the socket has been closed due to a
    /// network error.

    /// The signal is not emitted if the user resquested a disconnection (if
    /// <code>this->resquestDisc</code> is <code>true</code>).
    void disconnectedFromServer();

    /// @brief Signal emitted when a connection has been successfully
    /// established between the client and the server.

    /// The signal is emitted exactly once when the first frame is
    /// recieved from the server.
    void connected();

    /// @brief Signal emitted when the server sends an abstract types list
    /// of a concrete type.

    /// @param types Abstract types list. Each element is a type.
    void abstractTypes(const QStringList & types);

    /// @brief Signal emitted when the server sends an concrete types list.

    /// @param types Concrete types list. Each element is a type.
    void concreteTypes(const QStringList & types);

    /// @brief Signal emitted when the server sends an ACK (acknowledgement)
    /// for a specified type of frame.

    /// @param type Type of the acknowledged frame.
    void ack(int type);

    /// @brief Signal emitted when the server sends an NACK
    /// (non-acknowledgement) for a specified type of frame.

    /// @param type Type of the non-acknowledged frame.
    void nack(int type);

    /// @brief Signal emitted when the server sends a directory contents.
    /// @param path Absolute path of the directoy of which contents belong
    /// to.
    /// @param dirs Directories list. Each element is a directory.
    /// @param files Files list. Each element is a file.
    void dirContent(const QString & path, const QStringList & dirs,
                    const QStringList & files);

  };

////////////////////////////////////////////////////////////////////////////////

 } // namspace client
} // namespace COOLFluiD

////////////////////////////////////////////////////////////////////////////////

#endif // COOLFluid_ClientServer_CommClient_h
```

1.2.2 CommClient.cxx

```
#include <QtCore>
#include <QtNetwork>
#include <QtXml>

#include "ClientServer/client/CommClient.h"
#include "ClientServer/network/ClientServerXMLParser.h"
#include "ClientServer/network/NetworkFrames.h"

using namespace COOLFluiD::client;
using namespace COOLFluiD::network;

CommClient::CommClient()
{
 this->socket = new QTcpSocket(this);

 connect(socket, SIGNAL(readyRead()), this, SLOT(newData()));
 connect(socket, SIGNAL(disconnected()), this, SLOT(disconnected()));
 connect(socket, SIGNAL(error(QAbstractSocket::SocketError)), this,
         SLOT(socketError(QAbstractSocket::SocketError)));

 this->blockSize = 0;
 this->requestDisc = false;
 this->connectedToServer = false;
 this->skipRefused = false;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

CommClient::~CommClient()
{
 delete this->socket;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommClient::sendActionGetTree() const
{
 QDomDocument doc = NetworkFrames::buildSimpleGetFrame(
   NetworkFrames::TYPE_GET_TREE);
 this->send(doc.toString());
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommClient::sendGetAbstractTypes(const QString & typeName)
{
 QDomDocument document = NetworkFrames::buildGetTypes(
   NetworkFrames::TYPE_GET_ABSTRACT_TYPES, typeName);

 this->send(document.toString());
}


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommClient::sendGetConcreteTypes(const QString & typeName)
{
 QDomDocument document = NetworkFrames::buildGetTypes(
   NetworkFrames::TYPE_GET_CONCRETE_TYPES, typeName);
 this->send(document.toString());
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommClient::sendAction(int action, const QDomDocument & data)
{
 QDomDocument doc = NetworkFrames::buildAction(action, data);

 // if the frame has not been built, the type does not exist
 if(doc.isNull())
  emit error("This␣type␣does␣not␣seem␣to␣exist.", false);
 else
  this->send(doc.toString());
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommClient::sendActionAddNode(const QDomNode & node,
                                   const QString & type,
                                   const QString & absType)
{
 QDomDocument doc = NetworkFrames::buildAddNode(node, type, absType);
 this->send(doc.toString());
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommClient::sendActionRenameNode(const QDomNode & node,
                                      const QString & newName)
{
 QDomDocument doc = NetworkFrames::buildRenameNode(node, newName);
 this->send(doc.toString());
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommClient::sendActionDeleteNode(const QDomNode & node) const
{
 QDomDocument doc = NetworkFrames::buildDeleteNode(node);
 this->send(doc.toString());
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommClient::connectToServer(const QString & hostAddress, quint16 port,
                                 bool skipRefused)
{
 this->skipRefused = skipRefused;
 this->socket->connectToHost(hostAddress, port);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommClient::disconnectFromServer(bool shutServer)
{
 if(shutServer)
 {
   QDomDocument doc = NetworkFrames::buildSimpleGetFrame(
     NetworkFrames::TYPE_SHUTDOWN_SERVER);
   this->send(doc.toString());
  }

  this->requestDisc = true;
  this->connectedToServer = false;

  // close the socket
  this->socket->abort();
  this->socket->close();
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommClient::sendGetFilesList() const
{
  QDomDocument doc = NetworkFrames::buildSimpleGetFrame(
    NetworkFrames::TYPE_GET_FILES_LIST);
  this->send(doc.toString());
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommClient::sendOpenFile(const QString & filename)
{
  QDomDocument doc = NetworkFrames::buildOpenFile(filename);
  this->send(doc.toString());
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommClient::sendRunSimulation()
{
  QDomDocument doc = NetworkFrames::buildSimpleGetFrame(
    NetworkFrames::TYPE_RUN_SIMULATION);
  this->send(doc.toString());
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommClient::sendOpenDir(const QString & dirname)
{
  QDomDocument doc = NetworkFrames::buildOpenDir(dirname);

  this->send(doc.toString());
}

/*************************************************************************

                                PRIVATE METHOD

*************************************************************************/

void CommClient::send(const QString & frame) const
{
  QByteArray block;
  QDataStream out(&block, QIODevice::WriteOnly);

  out.setVersion(QDataStream::Qt_4_4); // QDataStream version
  out << (quint16)0;     // reserve 16 bits for the frame data size
```

```
 out << frame;
 out.device()->seek(0);  // go back to the beginning of the frame
 out << (quint16)(block.size() - sizeof(quint16)); // write the frame data size

 this->socket->write(block);
 this->socket->flush();
}

/****************************************************************************

                                    SLOTS

****************************************************************************/

void CommClient::newData()
{
 ClientServerXMLParser handler;
 QXmlInputSource source;

 QString frame;
 QDataStream in(socket);
 in.setVersion(QDataStream::Qt_4_4); // QDataStream version

 // if the server sends two messages very close in time, it is possible that
 // the client never gets the second one.
 // So, it is useful to explicitly read the socket until the end is reached.
 while(!socket->atEnd())
 {
  // if the data size is not known
  if (this->blockSize == 0)
  {
   // if there are at least 2 bytes to read...
   if (this->socket->bytesAvailable() < (int)sizeof(quint16))
    return;

   // ...we read them
   in >> this->blockSize;
  }

  if (this->socket->bytesAvailable() < this->blockSize)
   return;

  in >> frame;

  source.setData(frame);


  QXmlSimpleReader reader;
  reader.setContentHandler( &handler );

  // if parse() returns false, the document is not valid
  if(!reader.parse(source))
  {
   QString errorStr = handler.errorString();

    // if error is empty, the document is not a well-formed XML document
   if(errorStr.isEmpty())
    errorStr = "not␣well-formed␣document.";

   emit error(handler.errorString(), false);
  }
  else
  {
   if(!this->connectedToServer)
   {
```

```
   this->connectedToServer = true;
   emit connected();
  }

  switch(handler.getTypeId())
  {
   // if the server sends a message
   case NetworkFrames::TYPE_MESSAGE :
    emit message(handler.get("value"));
    break;

   // if the server sends an error message
   case NetworkFrames::TYPE_ERROR :
    emit error(handler.get("value"), true);
    break;

   // if the server sends the tree
   case NetworkFrames::TYPE_TREE :
   {
    QDomDocument doc = handler.getDomDocument();
    emit newTree(doc);
    break;
   }

   // if the server sends the abstract types list
   case NetworkFrames::TYPE_ABSTRACT_TYPES :
    emit abstractTypes(handler.get("typesList").split(",␣"));
    break;

   // if the server sends the concrete types list
   case NetworkFrames::TYPE_CONCRETE_TYPES :
    emit concreteTypes(handler.get("typesList").split(",␣"));
    break;

   // if the server sends an ACK
   case NetworkFrames::TYPE_ACK:
    emit ack(handler.getAckType());
    break;

   // if the server sends a NACK
   case NetworkFrames::TYPE_NACK:
    emit nack(handler.getAckType());
    break;

   // if the server sends directory contents
   case NetworkFrames::TYPE_DIR_CONTENT:
   {
    QString dirs = handler.get("dirs");
    QString files = handler.get("files");
    QStringList dirsList;
    QStringList filesList;

    // file and directory names are separated by a '*'
    if(!dirs.isEmpty())
     dirsList = dirs.split("*");

    if(!files.isEmpty())
     filesList = files.split("*");

    emit dirContent(handler.get("path"), dirsList, filesList);
    break;
   }

  }
 }
```

```
   this->blockSize = 0;
  }
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommClient::disconnected()
{
 if(!this->requestDisc)
 {
  emit error("The connection has been closed", false);
  emit disconnectedFromServer();
 }

 this->connectedToServer = false;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommClient::socketError(QAbstractSocket::SocketError err)
{
 if(this->requestDisc)
  return;

 switch (err)
 {
  case QAbstractSocket::RemoteHostClosedError:
   emit error("Remote connection closed", false);
   break;

  case QAbstractSocket::HostNotFoundError:
   emit error("Host was not found", false);
   break;

  case QAbstractSocket::ConnectionRefusedError:
   if(!this->skipRefused)
    emit error("Connection refused. Please check if the server is running.",
false);
   else
    this->skipRefused = false;
   break;

  default:
   emit error(QString("The following error occurred: ") +
     this->socket->errorString(), false);
 }
}
```

1.3 *ConnectionDialog* class

1.3.1 ConnectionDialog.h

```
#ifndef COOLFluiD_client_ConnectionDialog_h
#define COOLFluiD_client_ConnectionDialog_h

////////////////////////////////////////////////////////////////////////////////

#include <QDialog>

class QCheckBox;
class QDialogButtonBox;
class QFormLayout;
class QHBoxLayout;
class QLabel;
class QLineEdit;
class QMainWindow;
class QSpinBox;

////////////////////////////////////////////////////////////////////////////////

namespace COOLFluiD
{
 namespace client
 {

////////////////////////////////////////////////////////////////////////////////

  struct TSshInformation;

  /// @brief Dialog used to gather information to connect to a server.

  /// This class inherits from <code>QDialog</code> and is used to show a
  /// dialog gathering the needed information to connect to a server. <b>It
  /// does not realise this connection</b>. The dialog is modal, wich means
  /// that once it is visible, the calling code execution is stopped until
  /// the dialog is invisible again. If the dialog has a parent window, it is
  /// centered on this parent. Otherwise, it is centered on the screen.<br>

  /// This dialog has two modes : basic and advanced. The difference is that
  /// in advanced mode, the user is able to choose the port number, but not
  /// in basic mode. In both modes, the user is invited to enter the hostname
  /// to connect to (default value: <i>localhost</i>) and, if he chose to
  /// start a new server instance, the username used to authenticate on the
  /// remote machine (default value: the username of the process owner).<br>

  /// After calling the constructor, the dialog is invisible. The show method
  /// has to be called to show it. This is a blocking method: it will not
  /// return until is invisible again. This method returns <code>true</code>
  /// if user clicked on "OK" to validate his entry or <code>false</code> if
  /// he clicked on "Cancel" or closed the dialog to cancel his entry.<br>

  /// If the user validates his entry, gathered information are stored in
  /// the TSshInformation structure parameter. If the user did not choose to
  /// launch a new server instance, <code>username</code> parameter is this
  /// structure is not modified. The method guarantees that all other
  /// attributes will be correctly set. If the user cancels his entry, the
  /// structure is not modified. If the user clicks on "OK" without typing
  /// any name, it is considered as a cancellation.<br>

  /// A typical use of this class is (assuming that <code>this</code> is a
  /// <code>QMainWindow</code> object) : <br>
  /// \code
/// ConnectionDialog dialog(this);
/// TSshInformation sshInfos;        // used to store gathered information
///
/// if(dialog.show(sshInfos)         // show advanced connection dialog
/// {                                // if user clicked on "OK"
///  // some treatements
/// }
/// \endcode

/// @author Quentin Gasper.

class ConnectionDialog : public QDialog
{
 Q_OBJECT

 private:

  /// @brief Label for the hostame line edit.
  QLabel * labHostname;

  /// @brief Label for the username line edit.
  QLabel * labUsername;

  /// @brief Label for the port number spin box.
  QLabel * labPortNumber;

  /// @brief Line edit for the hostame.
  QLineEdit * editHostname;

  /// @brief Line edit for the username.
  QLineEdit * editUsername;

  /// @brief Spin box for the port number.
  QSpinBox * spinPortNumber;

  /// @brief Main layout.
  QFormLayout * layout;

  /// @brief Button box containing "OK" and "Cancel" buttons.
  QDialogButtonBox * buttons;

  /// @brief Layout for hostname and port number components.
  QHBoxLayout * infosLayout;

  /// @brief Ckeck box to check of the user wants to launch a new
  /// server instance.
  QCheckBox * chkLaunchServer;

  /// @brief Indicates whether the user clicked on "OK" button.

  /// If the user clicked on "OK" button, the attribute value is
  /// <code>true</code>, (if the user closed the window or clicked on
  /// "Cancel" button) it is <code>false</code>.
  bool okClicked;

 public:

  /// @brief Constructor.

  /// @param parent Parent window.
  ConnectionDialog(QMainWindow * parent);

  /// @brief Desctructor.
  ~ConnectionDialog();
```

```
    /// @brief Shows the dialog.

    /// This is a blocking method. It will not return while the dialog
    /// is visible.

    /// @param hidePort If <code>true</code>, user will not be able to
    /// select the port number.
    /// @param sshInfos Reference to TSshInformation structure where grabbed
    /// information will be written if and only if the user clicked on "OK"
    /// and the name is not empty, otherwise the structure is unchanged.

    /// @return If the user clicked on "OK", returns "true". Otherwise,
    /// returns <code>false</code>.
    bool show(bool hidePort, TSshInformation & sshInfos);

   public slots:

    /// @brief Slot called when "OK" button is clicked.

    /// Sets <code>this->okClicked</code> to <code>true</code> and then sets
    /// the dialog to an invisible state.
    void btOkClicked();

    /// @brief Slot called when "Cancel" button is clicked.

    /// Sets <code>this->okClicked</code> to <code>false</code> and then
    /// sets the dialog to an invisible state.
    void btCancelClicked();

    /// @brief Slot called when <code>this->chkLaunchServer</code> has
    /// been checked or unchecked.

    /// If it is checked, username line edit will be enabled for
    /// modification, otherwise it will be disabled.

    /// @param state New state of <code>this->chkLaunchServer</code> (based
    /// on <code>Qt::CheckState</code> enum). If the value is
    /// <code>Qt::Checked</code>, the username line edit is set to enabled.
    void chkLaunchServerChecked(int state);
  };

////////////////////////////////////////////////////////////////////////////////

 }
}

////////////////////////////////////////////////////////////////////////////////

#endif // COOLFluiD_client_ConnectionDialog_h
```

1.3.2 ConnectionDialog.cxx

```
#include <QtGui>

#include "ClientServer/client/ConnectionDialog.h"
#include "ClientServer/client/TSshInformation.h"

using namespace COOLFluiD::client;

ConnectionDialog::ConnectionDialog(QMainWindow * parent)
 : QDialog(parent)
{
 QString username;
 QRegExp regex("^USER=");
 QStringList environment = QProcess::systemEnvironment().filter(regex);
 if(environment.size() == 1)
  username = environment.at(0);

 this->setWindowTitle("Connect␣to␣server");

 // create the components
 this->labHostname = new QLabel("Hostname:");
 this->labUsername = new QLabel("Username:");
 this->labPortNumber = new QLabel("Port␣number:");

 this->editHostname = new QLineEdit(this);
 this->editUsername = new QLineEdit(this);
 this->spinPortNumber = new QSpinBox(this);

 this->infosLayout = new QHBoxLayout();

 this->chkLaunchServer = new QCheckBox("Start␣a␣new␣server␣instance", this);

 this->layout = new QFormLayout(this);
 this->buttons = new QDialogButtonBox(QDialogButtonBox::Ok
   | QDialogButtonBox::Cancel);

 // the dialog is modal
 this->setModal(true);

 this->spinPortNumber->setMinimum(49150);
 this->spinPortNumber->setMaximum(65535);

 this->editHostname->setText("localhost");
 this->editUsername->setText(username.remove("USER="));
 this->spinPortNumber->setValue(62784);

 // add the components to the layout
 this->infosLayout->addWidget(this->labHostname);
 this->infosLayout->addWidget(this->editHostname);
 this->infosLayout->addWidget(this->labPortNumber);
 this->infosLayout->addWidget(this->spinPortNumber);

 this->chkLaunchServerChecked(this->chkLaunchServer->checkState());

 this->layout->addRow(this->infosLayout);
 this->layout->addRow(this->chkLaunchServer);
 this->layout->addRow(this->labUsername, this->editUsername);
 this->layout->addRow(this->buttons);

 // add the layout to the dialog
 this->setLayout(this->layout);

 // connect useful signals to slots
 connect(this->buttons, SIGNAL(accepted()), this, SLOT(btOkClicked()));
 connect(this->buttons, SIGNAL(rejected()), this, SLOT(btCancelClicked()));
 connect(this->chkLaunchServer, SIGNAL(stateChanged(int)),
         this, SLOT(chkLaunchServerChecked(int)));
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

ConnectionDialog::~ConnectionDialog()
{
 delete this->buttons;
 delete this->chkLaunchServer;
 delete this->editUsername;
 delete this->editHostname;
 delete this->infosLayout;
 delete this->labHostname;
 delete this->labPortNumber;
 delete this->labUsername;
 delete this->layout;
 delete this->spinPortNumber;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

bool ConnectionDialog::show(bool hidePort, TSshInformation & sshInfos)
{
 this->okClicked = false;

 this->labPortNumber->setVisible(!hidePort);
 this->spinPortNumber->setVisible(!hidePort);

 this->exec();
 if(this->okClicked)
 {
  sshInfos.hostname = this->editHostname->text();
  sshInfos.username = this->editUsername->text();
  sshInfos.launchServer = this->chkLaunchServer->isChecked();
  sshInfos.port = this->spinPortNumber->value();
 }
 return this->okClicked;
}

/****************************************************************************

                                   SLOTS

****************************************************************************/

void ConnectionDialog::btOkClicked()
{
 this->okClicked = true;
 this->setVisible(false);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void ConnectionDialog::btCancelClicked()
{
 this->setVisible(false);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void ConnectionDialog::chkLaunchServerChecked(int state)
{
 this->editUsername->setReadOnly(state != Qt::Checked);
}
```

1.4 *Display* class

1.4.1 Display.h

```
#ifndef COOLFluiD_client_Display_h
#define COOLFluiD_client_Display_h

////////////////////////////////////////////////////////////////////////////////

#include <QObject>

namespace COOLFluiD
{
 namespace client
 {

////////////////////////////////////////////////////////////////////////////////

   class CommClient;
   class DisplayConsole : public QObject
   {
    Q_OBJECT

    /// This class is a console interface between the network level and the
    /// user.

    /// @author Quentin Gasper.

    private:
   /// CommClient with the server
     CommClient * communication;

   /// @brief Reads a string on the standard input and sends it to the server
   /// by calling the <code>CommClient::send()</code> method.

   /// If the user enters "QUIT", then the application is exited by calling
   /// <code>QApplication::exit(0)</code>.
     void readAndSendString();

    public:
    /// @brief Constructor.

    /// Creates a new communication between the client and the server.
    /// @param hostAddress Server address
    /// @param port Port
     DisplayConsole(QString hostAddress = "127.0.0.1", quint16 port = 62784);

    /// @brief Destructor.

    /// Destroys (closes) the communication.
     ~DisplayConsole();

    public slots:
   /// @brief Slot called when an error in the network transmission
   /// protocol occurs (bad XML format).

   /// Displays the error message and quits the application by calling
   /// <code>QApplication::exit(-1)</code>. This slot is connected
   /// to <code>CommClient::error()</code> signal.
   /// @param error Error message
     void error(const QString & error);

   /// @brief Slot called when a response arrives.
```

```
  /// Displays the message and calls <code><b>this</b>->readAndSendString()
  /// </code>. This slot is connected to <code>CommClient::message()</code>
  /// signal.
  /// @param message Message
    void message(const QString & message);
  };

////////////////////////////////////////////////////////////////////////////////

 } // namespace client
} // namespace COOLFluiD

////////////////////////////////////////////////////////////////////////////////

#endif // COOLFluiD_client_Display_h
```

1.4.2 Display.cxx

```
#include <iostream>
#include <QApplication>

#include "client/CommClient.h"
#include "network/NetworkException.h"
#include "client/Display.h"

using namespace COOLFluiD::client;

DisplayConsole::DisplayConsole(QString hostAddress, quint16 port)
{
 connect(this->communication, SIGNAL(error(const QString &)),
         this, SLOT(error(const QString &)));
 connect(this->communication, SIGNAL(message(const QString &)),
         this, SLOT(message(QString)));

 this->communication->connectToServer(hostAddress, port);

 this->readAndSendString();
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

DisplayConsole::~DisplayConsole()
{
 delete this->communication;
}

/**************************************************************************

                                 SLOTS

**************************************************************************/

void DisplayConsole::error(const QString & error)
{
 std::cerr << error.toStdString() << std::endl;
 QApplication::exit(-1);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void DisplayConsole::message(const QString & message)
{
 std::cout << message.toStdString() << std::endl;
 this->readAndSendString();
}

/**************************************************************************

                             PRIVATE METHOD

**************************************************************************/

void DisplayConsole::readAndSendString()
{
 char buffer[256];
 std::cout << "Your␣string␣(\"QUIT\"␣to␣exit)␣:␣";
 std::cin.getline(buffer, 256);

 if(strcmp(buffer, "QUIT") == 0)
```

```
    QApplication::exit(0);

  //this->communication->sendMessage(buffer);
}
```

1.5 *FilesListItem* class

1.5.1 FilesListItem.h

```
#ifndef COOLFluiD_client_FilesListItem_h
#define COOLFluiD_client_FilesListItem_h

////////////////////////////////////////////////////////////////////////////////

#include <QStandardItem>

namespace COOLFluiD
{
 namespace client
 {

////////////////////////////////////////////////////////////////////////////////

  /// @brief Adds a functionnality to <code>QStandardItem</code> class.

  /// This class inherits from <code>QStandardItem</code> and add only one
  /// functionnality to its base class : the type of this item. An item can
  /// be either a file or a directory and it can be usefull to remember this,
  /// for exemple, to easily manage icons.<br>

  /// This class is used by <code>OpenFileDialog</code> to create items for
  /// the list view.<br>

  /// @author Quentin Gasper.

  class FilesListItem : public QStandardItem
  {
   private :

    /// @brief Indicates the type of this item.

    /// The value is either <code>DIRECTORY</code>or <code>FILE}</code>.
    int type;

   public:

    /// @brief Directory type.

    /// If the <code>this->type</code> value is equal to
    /// <code>DIRECTORY</code>, this item is a directory.
    static const int DIRECTORY = 0;

    /// @brief File type.

    /// If the <code>this->type</code> value is equal to <code>FILE</code>,
    /// this item is a directory.
    static const int FILE = 1;

    /// @brief Constructor.

    /// Calls the base class constructor with provided icon and text:
    /// <code>QStandardItem(icon, text)</code> and sets the provided type
    /// value to <code>this->type</code>.

    /// @param icon Item icon.
    /// @param text Item text.
    /// @param type Item type.
    FilesListItem(const QIcon & icon, const QString & text, int type);
    /// @brief Gives the type of this item.

    /// @return Returns <code>DIRECTORY</code> if this item is a directory,
    /// otherwise returns <code>FILE</code>.
    int getType() const;
  };

////////////////////////////////////////////////////////////////////////////////

 }
}

////////////////////////////////////////////////////////////////////////////////

#endif // COOLFluiD_client_FilesListItem_h
```

1.5.2 FilesListItem.cxx

```
#include <QtCore>
#include <stdexcept>

#include "ClientServer/client/FilesListItem.h"

using namespace COOLFluiD::client;

const int FilesListItem::DIRECTORY;
const int FilesListItem::FILE;

FilesListItem::FilesListItem(const QIcon & icon, const QString & text,
                             int type)
 : QStandardItem(icon, text)
{
 if(type != FilesListItem::DIRECTORY && type != FilesListItem::FILE)
  throw std::invalid_argument("Unknown␣item␣type");

 this->type = type;

}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

int FilesListItem::getType() const
{
 return this->type;
}
```

1.6 *GraphicalOption* class

1.6.1 GraphicalOption.h

```
#ifndef COOLFluiD_client_GraphicalOption_h
#define COOLFluiD_client_GraphicalOption_h

////////////////////////////////////////////////////////////////////////////////

class QFormLayout;
class QHBoxLayout;
class QLabel;
class QLineEdit;
class QWidget;

namespace COOLFluiD
{
 namespace client
 {

  /// @brief Displays an option graphically.

  /// The value component is adapted to the type of the option.

////////////////////////////////////////////////////////////////////////////////

  class GraphicalOption
  {
   private:

    /// @brief Label for the option name.
    QLabel * name;

    /// @brief Line edit for the option value.
    QWidget * value;

    /// @brief Type of the option, according to the type ids defined by
    /// OptionsTypes class.
    int type;

    /// @brief Indicates wether the value component is enabled (allows
    /// modification) or not.
    bool enabled;

   public:

    /// @brief Constructor.

    /// @param type Option type. Must be one of those defined by OptionsTypes
    /// class.
    GraphicalOption(int type);

    /// @brief Destructor.

    /// Frees all allocated memory.
    ~GraphicalOption();

    /// @brief Gives the option name.

    /// @return Returns the option name.
    QString getName() const;

    /// @brief Sets option name.
    /// @param name Option name.
    void setName(const QString & name);

    /// @brief Gives the option value

    /// Whatever is the option type, the value is return in a QString form.

    /// @return Returns the option value.
    QString getValue() const;

    /// @brief Adds this option to the provided layout.

    /// @param layout Layout to which the options has to be added.
    void addToLayout(QFormLayout * layout);

    /// @brief Sets a new value to the option

    /// @param v New value. Must be in a format compatible with the option
    /// type or nothing is done.
    void setValue(const QString & v);

    /// @brief Enables or disables the value component.

    /// If the component is enabled, its value is modifiable.

    /// @param enabled If <code>true</code>, the component is enabled.
    /// Otherwise it is disabled.
    void setEnabled(bool enabled);

    /// @brief Indicates wether the value component is enabled or not.

    /// @return Returns <code>true</code> if the component is enabled.
    bool isEnabled() const;

    /// @brief Sets a tooltip.

    /// @param toolTip Tool tip to set.
    void setToolTip(const QString & toolTip);
  };

////////////////////////////////////////////////////////////////////////////

 }
}

////////////////////////////////////////////////////////////////////////////

#endif // COOLFluiD_client_GraphicalOption_h
```

1.6.2 GraphicalOption.cxx

```
#include <QtCore>
#include <QtGui>

#include "ClientServer/client/GraphicalOption.h"
#include "ClientServer/client/OptionsTypes.h"

using namespace COOLFluiD::client;

GraphicalOption::GraphicalOption(int type)
{
 switch(type)
 {
  case OptionsTypes::TYPE_BOOL:
  {
   QCheckBox * checkBox = new QCheckBox();
   checkBox->setCheckState(Qt::Unchecked);
   this->value = checkBox;
   break;
  }

  case OptionsTypes::TYPE_STRING:
   this->value = new QLineEdit();
   break;

  case OptionsTypes::TYPE_DOUBLE:
   this->value = new QDoubleSpinBox();
   break;

  case OptionsTypes::TYPE_INT:
   this->value = new QSpinBox();
   break;

  case OptionsTypes::TYPE_UNSIGNED_INT:
   this->value = new QSpinBox();
   break;

  // default -> throw exception
 }

 this->name = new QLabel();

 this->type = type;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

GraphicalOption::~GraphicalOption()
{
 delete this->name;
 delete this->value;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QString GraphicalOption::getName() const
{
 return this->name->text();
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void GraphicalOption::setName(const QString & name)
{
 this->name->setText(name);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QString GraphicalOption::getValue() const
{
 QString return_string;
 switch(this->type)
 {
  case OptionsTypes::TYPE_BOOL:
  {
   Qt::CheckState state = ((QCheckBox *) this->value)->checkState();
   return_string = QVariant(state == Qt::Checked).toString();
   break;
  }

  case OptionsTypes::TYPE_STRING:
   return_string = ((QLineEdit *) this->value)->text();
   break;

  case OptionsTypes::TYPE_INT:
  {
   int val = ((QSpinBox *) this->value)->value();
   return_string = QVariant(val).toString();
   break;
  }

  case OptionsTypes::TYPE_UNSIGNED_INT:
  {
   unsigned int val = ((QSpinBox *) this->value)->value();
   return_string = QVariant(val).toString();
   break;
  }

  case OptionsTypes::TYPE_DOUBLE:
  {
   double val = ((QDoubleSpinBox *) this->value)->value();
   return_string = QVariant(val).toString();
   break;
  }
 }
 return return_string;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void GraphicalOption::setValue(const QString & v)
{
 bool ok;

 switch(this->type)
 {
  case OptionsTypes::TYPE_BOOL:
  {
   bool val = QVariant(v).toBool();
   if(val)
    ((QCheckBox *) this->value)->setCheckState(Qt::Checked);
   else
    ((QCheckBox *) this->value)->setCheckState(Qt::Unchecked);
    break;
   }

   case OptionsTypes::TYPE_STRING:
    ((QLineEdit *) this->value)->setText(v);
    break;

   case OptionsTypes::TYPE_INT:
   {
    int val = QVariant(v).toInt(&ok);
    if(!ok)
     ; // error
    ((QSpinBox *) this->value)->setValue(val);
   }

   case OptionsTypes::TYPE_UNSIGNED_INT:
   {
    int val = QVariant(v).toUInt(&ok);
    if(!ok)
     ; // error
    ((QSpinBox *) this->value)->setValue(val);
   }

   case OptionsTypes::TYPE_DOUBLE:
   {
    double val = QVariant(v).toDouble(&ok);
    if(!ok)
     ; // error
    ((QDoubleSpinBox *) this->value)->setValue(val);
   }
  }
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void GraphicalOption::addToLayout(QFormLayout * layout)
{
 if(layout != NULL)
 {
  layout->addRow(this->name, this->value);
 }
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void GraphicalOption::setEnabled(bool enabled)
{
 this->value->setEnabled(enabled);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

bool GraphicalOption::isEnabled() const
{
 return this->value->isEnabled();
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void GraphicalOption::setToolTip(const QString & toolTip)
```

```
{
 this->name->setToolTip(toolTip);
 this->value->setToolTip(toolTip);
}
```

1.7 *MainWindow* class

1.7.1 MainWindow.h

```
#ifndef COOLFluiD_client_MainWindow_h
#define COOLFluiD_client_MainWindow_h

////////////////////////////////////////////////////////////////////////////////

#include <QMainWindow>
#include <QHash>
#include <QList>
#include <QProcess>

#include "ClientServer/client/TSshInformation.h"

class QDockWidget;
class QDomDocument;
class QDomNode;
class QDomNamedNodeMap;
class QGridLayout;
class QMenu;
class QModelIndex;
class QTextEdit;
class QTimer;
class QScrollBar;
class QSortFilterProxyModel;
class QTreeView;

////////////////////////////////////////////////////////////////////////////////

namespace COOLFluiD
{
 namespace treeview
 {
  class TreeModel;
 }

 namespace client
 {

////////////////////////////////////////////////////////////////////////////////

  class CommClient;
  class ConnectionDialog;
  class OptionsPanel;
  class TSshInformation;

  /// @brief Main client window.

  /// @author Quentin Gasper.

  class MainWindow : public QMainWindow
  {
   Q_OBJECT

   private:

    /// @brief Emplacement in <code>this->actions</code> for the action
    /// used to connect to the server.
    const int ACTION_CONNECT_TO_SERVER;

    /// @brief Emplacement in <code>this->actions</code> for the action
    /// used to disconnect from the server.
    const int ACTION_DISC_FROM_SERVER;

    /// @brief Emplacement in <code>this->actions</code> for the action
    /// used to get the tree from the server.
    const int ACTION_GET_TREE;

    /// @brief Emplacement in <code>this->actions</code> for the action
    /// used to close the application.
    const int ACTION_QUIT;

    /// @brief Emplacement in <code>this->actions</code> for the action
    /// used to toggle between basic and advanced mode.
    const int ACTION_TOGGLE_ADVANCED_MODE;

    /// @brief Emplacement in <code>this->actions</code> for the action
    /// used to add a node to the tree.
    const int ACTION_ADD_NODE;

    /// @brief Emplacement in <code>this->actions</code> for the action
    /// used to rename a node.
    const int ACTION_RENAME_NODE;

    /// @brief Emplacement in <code>this->actions</code> for the action
    /// used to delete a node.
    const int ACTION_DELETE_NODE;

    /// @brief Emplacement in <code>this->actions</code> for the action
    /// used to display a tree node properties.
    const int ACTION_PROPERTIES;

    /// @brief Emplacement in <code>this->actions</code> for the action
    /// used to open a file.
    const int ACTION_OPEN_FILE;

    /// @brief Emplacement in <code>this->actions</code> for the action
    /// used to run the simulation.
    const int ACTION_RUN_SIMULATION;

    /// @brief The associated message is a normal message from the client.

    /// Used to differenciate messages types of messages in the log window.
    static const int TYPE_NORMAL = 0;

    /// @brief The associated message is a error message.

    /// Used to differenciate messages types of messages in the log window.
    static const int TYPE_ERROR = 1;

    /// @brief The associated message is a normal message from the server.

    /// Used to differenciate messages types of messages in the log window.
    static const int TYPE_SERVER = 2;

    /// @brief Indicates that the user wants to disconnect from the server.

    /// Used when the user does "Disconnect", "Quit", or closes the window.
    static const int CLOSE_DISC = 0;

    /// @brief Indicates that the user wants to shutdown the server.

    /// Used when the user does "Disconnect", "Quit", or closes the window.
    static const int CLOSE_SHUTDOWN = 1;

    /// @brief Indicates that the user wants cancel his request to close the
    /// connection/window.
    /// Used when the user does "Disconnect", "Quit", or closes the window.
    static const int CLOSE_CANCEL = 2;

    /// @brief The model to be displayed.
    COOLFluiD::treeview::TreeModel * treeModel;

    /// @brief The treeview that displays the model.
    QTreeView * treeView;

    /// @brief Panel used to display and modify options for a selected
    /// object.
    OptionsPanel * optionsPanel;

    /// @brief Layout used to display widgets. Layout of
    /// <code>this->centralWidget</code>.
    QGridLayout * widgetsLayout;

    /// @brief Main widget used to display widgets.
    QWidget * centralWidget;

    /// @brief Hashmap containing all available actions for menu items.

    /// The key is a number defined by one of the constant integer attributes
    /// of this class. The value is the action corresponding to this number.
    QHash<int, QAction *> actions;

    /// @brief List containing all actions for abstract types displayed in
    /// the context menu.

    /// These actions are not stored in <code>this->actions</code> because
    /// they are not identified by an integer and the list may be cleared
    /// several times during application runtime.
    QList<QAction *> abstractTypesActions;

    /// @brief "File" menu
    QMenu * mnuFile;

    /// @brief "View" menu
    QMenu * mnuView;

    /// @brief Context menu
    QMenu * mnuContext;

    /// @brief Abstract types menu.

    /// This is a sub-menu of the context menu.
    QMenu * mnuAbstractTypes;

    /// @brief Allows the communication with the server.
    CommClient * communication;

    /// @brief Log window, docked at the bottom of the window.
    QDockWidget * logWindow;

    /// @brief Text area displaying the log messages.
    QTextEdit * logList;

    /// @brief Currently selected abstract type.

    /// This string is empty if no abstract type is selected.
    QString currentAbstractType;

    /// @brief Filter for the treeview.
    /// Allows to switch between basic/advanced mode. The filter is used a
    /// as the treeview model. Its source is the tree model.
    QSortFilterProxyModel * modelFilter;

    /// @brief Information grabbed by the connection dialog to launch
    /// and connect to the server.
    TSshInformation sshInfos;

    /// @brief Process used to launch the server.
    QProcess * proLaunchServer;

    /// @brief Process used to check wether another server instance
    /// is already running on the remote machine.
    QProcess * proCheckServer;

    /// @brief Timer used to wait a few milliseconds between two attempts
    /// to connect to the server when launching a new instance of it.
    QTimer * timer;

    int logLinesCounter;

    ConnectionDialog * connectionDialog;

    bool connectedToServer;

    /// @brief Creates actions and menus
    void buildMenus();

    /// @brief Builds an action form the given parameter.

    /// @param text Action text
    /// @param index Hash map action index. If -1, the action is not added to
    /// the map.
    /// @param slot Slot to call if the action is triggered. If
    /// <code>NULL</code>, not slot is associated
    /// @param enabled If <code>true</code>, the action will be enabled,
    /// otherwise it will not.
    /// @param menu Menu to attach the action to. If <code>NULL</code>, the
    /// action is not attached to a menu
    /// @param shortcut Action shortcut. Default value is
    /// <code>QKeySequence()</code>.

    /// @return Returns the built action.
    QAction * initAction(const QString & text, int index,
                         const char * slot, bool enabled,
                         QMenu * menu = NULL,
                         const QKeySequence & shortcut = QKeySequence());

    /// @brief Appends a message to the log.

    /// @param string Message to append.
    /// @param type Type of message : <code>TYPE_NORMAL</code>,
    /// <code>TYPE_ERROR</code> or <code>TYPE_SERVER</code>.

    void appendToLog(const QString & string, int type);

    /// @brief Sets the client to a <i>connected</i> or a
    /// <i>non-connected</i> state by enabling or disabling certain options.

    /// @param connected If <code>true</code>, the client is set to a
    /// <i>connected</i> state, otherwise it is set to a <i>non-connected</i>
    /// state
    void setConnected(bool connected);

    /// @brief Sets the client to a <i>simulation running</i> or a
    /// <i>simulation not running</i> state by enabling or disabling
    /// certain options.

    /// @param simRunning If <code>true</code>, the client is set to a
    /// <i>sumulation running</i> state, otherwise it is set to a
    /// <i>sumulation not running</i> state.
    void setSimRunning(bool simRunning);

    /// @brief Method called if the user wants to launch the server.

    /// @return Returns <code>true</code> if the server has been
    /// successfully launched, otherwise <code>false</code> is returned.
    void launchServer();

    /// @brief Asks to the user to confirm his request to close the
    /// connection or window.

    /// @return Returns <code>CLOSE_DISC</code> if the user just wants to
    /// disconnect form the server, <code>CLOSE_SHUTDOWN</code> if the user
    /// wants to shutdown the server or <code>SHUT_CANCEL</code> if the user
    /// wants to cancel his action.
    int confirmClose();

   protected:
    /// @brief Overrides <code>QWidget::closeEvent()</code>.

    /// This method is called when the user closes the window. If a network
    /// communication is active, he is prompt to confirm his action.

    /// @param event Close event to manage the window closing.
    virtual void closeEvent(QCloseEvent * event);

   public:

    /// @brief Constructor.

    /// Builds all components used by the window. After the constructor, the
    /// window is visible and in a "<i>Not connected</i>" state.

    MainWindow();

    /// @brief Destructor.

    /// Frees the allocated memory.
    ~MainWindow();

   private slots:

    /// @brief Slot called when user commits changes to the selected
    /// node options.

    /// @param modOptions List of the modified options. Each child of this
    /// XML document is an option. May be empty.
    /// @param newOptions List of the new options. Each child of this XML
    /// document is an option. May be empty.
    void changesMade(const QDomDocument & modOptions,
                     const QDomDocument & newOptions);

    /// @brief Slot called when the user wants to connect to the server.

    /// This is a non-blocking slot. The connection request is sent but the
    /// slot returns without waiting for an answer.
    void connectToServer();

    /// @brief Slot called when the user wants to disconnect from the server.
    /// The user is invited to choose between shutdown the server, just
    /// disconnect from it or cancel the action. If the user don't select
    /// "Cancel", this slot destroys <code>this->communication</code> and
    /// <code>this->treeModel</code>. Bath pointers are set to
    /// <code>NULL</code>. If the user confirms the disconnection. This
    /// slot destroys the <code>CommClient</code> object and the model.
    void disconnectFromServer();

    /// @brief Slot called when the user wants to quit the application.

    /// The client disconnects form the server and exits immediately.
    void quit();

    /// @brief Slot called when the user want to get/update the tree.
    void getTree();

    /// @brief Slot called when an error in the network transmission
    /// protocol occurs (bad XML format) or when the server sends an
    /// error message.

    /// Calls <code>this->appendToLog()</code> to display the message.

    /// @param error Error message
    /// @param fromServer <code>true</code> if the error message comes
    /// from the server, otherwise <code>false</code>. If <code>true</code>,
    /// the string "<code>[SERVER]</code> " is prepended to the error
    /// message.
    void error(const QString & error, bool fromServer);

    /// @brief Slot called when a message arrives.

    /// Calls <code>this->appendToLog()</code> to display the message.

    /// @param message Message. The string "<code>[SERVER]</code> " is
    /// prepended to the error message.
    void message(const QString & message);

    /// @brief Sets a new TreeModel to the the treeview.

    /// The model is built from the given document.
    /// @param domDocument Document to use to build the new TreeView.
    void buildTree(const QDomDocument & domDocument);


    /// @brief Slot called when the client is connected to the server.
    void connected();

    /// @brief Slot called when the network level recieves abstract types
    /// list.

    /// @param types Recieved abstract types list.
    void abstractTypes(const QStringList & types);

    /// @brief Slot called when the network level recieves concrete types
    /// list.

    /// @param types Recieved concrete types list.
    void concreteTypes(const QStringList & types);

    /// @brief Slot called when an ACK (acknowledgement) arrives for a frame.

    /// @param type Type of the acknowledged frame (conforming to type
    /// defined by NetworkFrames class).
    void ack(int type);
    /// @brief Slot called when a NACK (non-acknowledgement) arrives for a
    /// frame.

    /// @param type Type of the non-acknowledged frame (conforming to type
    /// defined by NetworkFrames class).
    void nack(int type);

    /// @brief Slot called when an item in the treeview is selected.

    /// @param index Index of the selected item.
    void itemClicked(const QModelIndex & index);

    /// @brief Slot called when the user want to to toggle
    /// basic/advanced mode.
    void toggleAdvanced();

    /// @brief Slot called when the user makes a right-click on the tree
    /// view.

    /// If necessary (if an item has been clicked), a context menu is
    /// displayed.
    /// @param mousePos Mouse cursor position in pixels when the right-click
    /// occured. The origin point (0,0) is the top-left corner of the
    /// treeview.
    void contextMenu(const QPoint & mousePos);

    /// @brief Slot called when the user wants to add a child node of a
    /// selected abstract type.

    /// This slot sends a request to the server to get the concrete types
    /// corresponding to the selected abstract type. It returns without
    /// waiting an answer.
    void addNode();

    /// @brief Slot called when the user wants to add an option to the
    /// selected object.
    void addOption();

    /// @brief Slot called when the user wants to rename an object.

    /// A request is sent to the server if and only if the new name is not
    /// empty and it is different to the old one.
    void renameNode();

    /// @brief Slot called when the user wants to delete an object.

    /// The slot sends the request to the server without asking the user to
    /// confirm (should be fixed in the future).
    void deleteNode();

    /// @brief Slot called when the user wants to see an object properties.

    /// Properties are displayed in a message box.
    void showProperties();

    /// @brief Slot called when the user wants to open a file.

    void openFile();

    /// @brief Slot called when the user wants to run the simulation.

    void runSimulation();

    /// @brief Tries to connect to the server.
    /// During the waiting for the server to launch through an SSH
    /// connection, this slot is called at every timeout of
    /// <code>this->timer</code> and tries to connect to the server.
    void tryToConnect();

    /// @brief Slot called when there is an error in the launching server
    /// process.

    /// During the waiting for the server to launch through an SSH
    /// connection, this slot is called for any output on the process error
    /// output. A such output is considered as a fatal error in the
    /// launching process. This slot stops the timer and attempts to connect
    /// to the server are canceled.
    void sshError();

  };

//////////////////////////////////////////////////////////////////////////////

 } // namespace client
} // namespace COOLFluiD

//////////////////////////////////////////////////////////////////////////////

#endif // COOLFluiD_client_MainWindow_h
```

1.7.2 MainWindow.cxx

```
#include <QtCore>
#include <QtGui>
#include <QtXml>

#include "ClientServer/client/AddNodeDialog.h"
#include "ClientServer/client/CommClient.h"
#include "ClientServer/client/ConnectionDialog.h"
#include "ClientServer/client/OpenFileDialog.h"
#include "ClientServer/client/OptionsPanel.h"
#include "ClientServer/client/OptionsTypes.h"
#include "ClientServer/client/TSshInformation.h"
#include "ClientServer/network/NetworkFrames.h"
#include "ClientServer/treeview/TObjectProperties.h"
#include "ClientServer/treeview/TreeModel.h"

#include "ClientServer/client/MainWindow.h"

using namespace COOLFluiD::client;
using namespace COOLFluiD::network;
using namespace COOLFluiD::treeview;

MainWindow::MainWindow()
 : ACTION_CONNECT_TO_SERVER(0),
   ACTION_DISC_FROM_SERVER(1),
   ACTION_GET_TREE(2),
   ACTION_QUIT(3),
   ACTION_TOGGLE_ADVANCED_MODE(4),
   ACTION_ADD_NODE(5),
   ACTION_RENAME_NODE(6),
   ACTION_DELETE_NODE(7),
   ACTION_PROPERTIES(8),
   ACTION_OPEN_FILE(9),
   ACTION_RUN_SIMULATION(10)
{
 // create the components
 this->centralWidget = new QWidget(this);
 this->optionsPanel = new OptionsPanel(this);
 this->widgetsLayout = new QGridLayout();
 this->treeModel = new TreeModel(QDomDocument(), this);
 this->treeView = new QTreeView(this);
 this->logWindow = new QDockWidget("Log␣Window", this);
 this->logList = new QTextEdit(this->logWindow);
 this->timer = new QTimer(this);
 this->proLaunchServer = new QProcess(this);
 this->connectionDialog = new ConnectionDialog(this);

 this->logList->setReadOnly(true);

 this->logWindow->setWidget(this->logList);

 this->logWindow->setFeatures(QDockWidget::NoDockWidgetFeatures |
   QDockWidget::DockWidgetClosable);

 this->modelFilter = new QSortFilterProxyModel();

 this->modelFilter->setDynamicSortFilter(true);

 this->treeView->setHeaderHidden(true);
 this->treeView->setModel(this->modelFilter);
 this->treeView->setVisible(false);

 // add the components to the layout
 this->widgetsLayout->addWidget(this->treeView, 0, 0);
 this->widgetsLayout->addWidget(this->optionsPanel, 0, 1);
 this->widgetsLayout->setColumnStretch(1, 10);

 this->centralWidget->setLayout(this->widgetsLayout);

 this->setCentralWidget(this->centralWidget);
 this->addDockWidget(Qt::BottomDockWidgetArea, this->logWindow);

 this->connectedToServer = false;

 this->logLinesCounter = 0;

 setWindowTitle("Client␣window");
 this->buildMenus();

 this->communication = new CommClient();

 this->appendToLog("Client␣successfully␣launched.", MainWindow::TYPE_NORMAL);

 // connect useful signals to slots
 connect(this->treeView, SIGNAL(customContextMenuRequested(const QPoint &)),
         this, SLOT(contextMenu(const QPoint &)));

 connect(this->timer, SIGNAL(timeout()),
         this, SLOT(tryToConnect()));

 connect(this->communication, SIGNAL(error(const QString &, bool)),
         this, SLOT(error(const QString &, bool)));

 connect(this->communication, SIGNAL(message(const QString &)),
         this, SLOT(message(const QString &)));

 connect(this->communication, SIGNAL(newTree(const QDomDocument &)),
         this, SLOT(buildTree(const QDomDocument &)));

 connect(this->communication, SIGNAL(disconnectedFromServer()),
         this, SLOT(disconnectFromServer()));

 connect(this->communication, SIGNAL(connected()),
         this, SLOT(connected()));

 connect(this->communication, SIGNAL(abstractTypes(const QStringList &)),
         this, SLOT(abstractTypes(const QStringList &)));

 connect(this->communication, SIGNAL(concreteTypes(const QStringList &)),
         this, SLOT(concreteTypes(const QStringList &)));

 connect(this->communication, SIGNAL(ack(int)), this, SLOT(ack(int)));

 connect(this->communication, SIGNAL(nack(int)), this, SLOT(nack(int)));

 // when right clic on the treeview, a "Context menu event" must be generated
 this->treeView->setContextMenuPolicy(Qt::CustomContextMenu);
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

MainWindow::~MainWindow()
{
 delete this->treeView;
 delete this->treeModel;
 delete this->optionsPanel;
 delete this->widgetsLayout;
 delete this->centralWidget;
 delete this->logList;
 delete this->logWindow;
 delete this->mnuAbstractTypes;
 delete this->mnuContext;
 delete this->mnuView;
 delete this->mnuFile;
 delete this->modelFilter;

 this->proLaunchServer->terminate();

 delete this->communication; // TODO <--- segmentation fault here
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::buildTree(const QDomDocument & domDocument)
{
 TreeModel * newModel = new TreeModel(domDocument, this);

 // set the new model...
 this->modelFilter->setSourceModel(newModel);
 this->treeView->setModel(modelFilter);

 // ...and delete the old one
 delete this->treeModel;
 this->treeModel = newModel;

 connect(this->treeView, SIGNAL(clicked(const QModelIndex &)),
         this, SLOT(itemClicked(const QModelIndex &)));

 connect(this->optionsPanel,
         SIGNAL(changesMade(const QDomDocument &, const QDomDocument &)),
                 this, SLOT(changesMade(const QDomDocument &,
                           const QDomDocument &)));

 this->appendToLog("Treeview␣updated.", MainWindow::TYPE_NORMAL);

 this->toggleAdvanced();

 this->treeView->expandAll(); // temporary
 this->treeView->setVisible(true);
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::buildMenus()
{
 this->mnuFile = new QMenu("&File", this);

 this->initAction("&Connect␣to␣server", ACTION_CONNECT_TO_SERVER,
                  SLOT(connectToServer()), true, this->mnuFile,
                       tr("ctrl+O"));

 this->initAction("&Disconnect␣from␣server", ACTION_DISC_FROM_SERVER,
                  SLOT(disconnectFromServer()), false, this->mnuFile,
                       tr("CTRL+W"));

 this->initAction("&Get␣tree", ACTION_GET_TREE, SLOT(getTree()), false,
                  this->mnuFile, tr("CTRL+U"));

 this->initAction("&Open␣file", ACTION_OPEN_FILE, SLOT(openFile()),
                  false, this->mnuFile);
```

```
this->initAction("&Run␣simulation", ACTION_RUN_SIMULATION,
                 SLOT(runSimulation()), false, this->mnuFile);

this->mnuFile->addSeparator();

this->initAction("&Quit", this->ACTION_QUIT, SLOT(quit()), true,
                 this->mnuFile, tr("CTRL+Q"));

//-----------------------------------------------------
//-----------------------------------------------------

this->mnuView = new QMenu("&View", this);

this->mnuView->addAction(this->logWindow->toggleViewAction());

this->initAction("Toggle␣&advanced␣mode", ACTION_TOGGLE_ADVANCED_MODE,
                 SLOT(toggleAdvanced()), true, this->mnuView);

this->actions[ACTION_TOGGLE_ADVANCED_MODE]->setCheckable(true);

//-----------------------------------------------------
//-----------------------------------------------------

QMenu * mnuNewOption = new QMenu("Add␣an␣option");

QStringList typesList = OptionsTypes::getTypesList();
QStringList::iterator it = typesList.begin();

while(it != typesList.end())
{
 this->initAction(*it, -1, SLOT(addOption()), true, mnuNewOption);
 it++;
}

//-----------------------------------------------------
//-----------------------------------------------------

this->mnuAbstractTypes = new QMenu("Add␣a␣child␣node");
this->mnuContext = new QMenu("Context␣menu");

this->mnuContext->addMenu(this->mnuAbstractTypes);

this->mnuContext->addMenu(mnuNewOption);

this->mnuContext->addSeparator();

this->initAction("Rename", ACTION_RENAME_NODE,
                 SLOT(renameNode()), true, this->mnuContext);

this->mnuContext->addSeparator();

this->initAction("Delete", ACTION_DELETE_NODE,
                 SLOT(deleteNode()), true, this->mnuContext);

this->mnuContext->addSeparator();

this->initAction("Properties", ACTION_PROPERTIES,
                 SLOT(showProperties()), true, this->mnuContext);


//-----------------------------------------------------
//-----------------------------------------------------

this->menuBar()->addMenu(this->mnuFile);
```

```
 this->menuBar()->addMenu(this->mnuView);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QAction * MainWindow::initAction(const QString & text, int index,
                                 const char * slot, bool enabled,
                                 QMenu * menu, const QKeySequence & shortcut)
{
 QAction * action = new QAction(text, this);
 action->setEnabled(enabled);

 if(!shortcut.isEmpty())
  action->setShortcut(shortcut);

 if(slot != NULL)
  connect(action, SIGNAL(triggered()), this, slot);

 if(index != -1)
  this->actions[index] = action;

 if(menu != NULL)
  menu->addAction(action);

 action->setIconVisibleInMenu(true);

 return action;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::appendToLog(const QString & string, int type)
{
 QString date = QDate::currentDate().toString("MM/dd/yyyy␣");
 QString time = QTime::currentTime().toString("hh:mm:ss");

 QListWidgetItem * item = new QListWidgetItem(QString("[") + date + time +
   QString("]␣->␣") + string);


 QStringList list = string.split("\n", QString::SkipEmptyParts);

 for(int i = 0 ; i < list.size() ; i++)
 {
  QString str;

  // if log has 100000 lines, we clear it (to save memory)
  if(this->logLinesCounter == 100000)
  {
   this->logLinesCounter = 0;
   this->logList->clear();
  }
  else
   this->logLinesCounter++;

  if(type == MainWindow::TYPE_ERROR)
   str = "<font␣color=\"red\">" + list.at(i) + "</font>";

  else if(type == MainWindow::TYPE_SERVER)
   str = "<font␣color=\"darkgreen\">" + list.at(i) + "</font>";

  else
   str = list.at(i);
   if(i == 0)
    this->logList->append(QString("[") + date + time +
      QString("]␣->␣") + str);
   else
    this->logList->append(str);
 }
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::setConnected(bool connected)
{
 this->actions[ ACTION_CONNECT_TO_SERVER ]->setEnabled(!connected);
 this->actions[ ACTION_GET_TREE ]->setEnabled(connected);
 this->actions[ ACTION_DISC_FROM_SERVER ]->setEnabled(connected);
 this->actions[ ACTION_OPEN_FILE ]->setEnabled(connected);

 if(!connected)
 {
  this->optionsPanel->setReadOnly(false);
  this->actions[ ACTION_RUN_SIMULATION ]->setEnabled(false);
 }
 this->connectedToServer = connected;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::setSimRunning(bool simRunning)
{
 this->optionsPanel->setReadOnly(simRunning);

 this->actions[ ACTION_OPEN_FILE ]->setEnabled(!simRunning);
 this->actions[ ACTION_RUN_SIMULATION ]->setEnabled(!simRunning);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

int MainWindow::confirmClose()
{
 int answer;
 QMessageBox discBox(this);
 QPushButton * btDisc = NULL;
 QPushButton * btCancel = NULL;
 QPushButton * btShutServer = NULL;

 btDisc = discBox.addButton("Disconnect", QMessageBox::NoRole);
 btCancel = discBox.addButton(QMessageBox::Cancel);
 btShutServer = discBox.addButton("Shutdown␣server", QMessageBox::YesRole);

 discBox.setText("You␣are␣about␣to␣disconnect␣from␣the␣server.");
 discBox.setInformativeText("What␣do␣you␣want␣to␣do␣?");

 // show the message box
 discBox.exec();

 if(discBox.clickedButton() == btDisc)
  answer = CLOSE_DISC;
 else if(discBox.clickedButton() == btShutServer)
  answer = CLOSE_SHUTDOWN;
 else
  answer = CLOSE_CANCEL;
 delete btDisc;
 delete btCancel;
 delete btShutServer;

 return answer;
}

/************************************************************************

                              PROTECTED METHOD

************************************************************************/

void MainWindow::closeEvent(QCloseEvent * event)
{
 if(!this->connectedToServer)
  return;

 int answer = this->confirmClose();

 if(answer == CLOSE_DISC)
  this->communication->disconnectFromServer(false);
 else if(answer == CLOSE_SHUTDOWN)
  this->communication->disconnectFromServer(true);

 // if user clicked on "Cancel", we reject the event
 // (the window will not close)
 if(answer == CLOSE_CANCEL)
  event->ignore();
 // otherwise we accept the event (the window will close)
 else
  event->accept();
}

/************************************************************************

                                   SLOTS

************************************************************************/

void MainWindow::itemClicked(const QModelIndex & index)
{
 QModelIndex indexInModel = this->modelFilter->mapToSource(index);
 QDomNodeList options = this->treeModel->getOptions(indexInModel);
 this->optionsPanel->setOptions(options);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::changesMade(const QDomDocument & modOptions,
                             const QDomDocument & newOptions)
{
 QDomDocument doc;

 // get the index in the filter
 QModelIndex index = this->treeView->currentIndex();

 // get the corresponding index in the model
 QModelIndex indexInModel = this->modelFilter->mapToSource(index);

 doc = this->treeModel->modifyToDocument(indexInModel, modOptions,
                                         newOptions);
 this->communication->sendAction(NetworkFrames::TYPE_MODIFY_NODE, doc);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::connectToServer()
{
 bool advanced = this->actions[ ACTION_TOGGLE_ADVANCED_MODE ]->isChecked();

 TSshInformation sshInfo;

 if(!this->timer->isActive())
 {
  // show the connection dialog and wait for it to return
  if(!this->connectionDialog->show(!advanced, sshInfo))
   return;

  this->sshInfos = sshInfo;

  if(this->sshInfos.launchServer)
  {
   this->launchServer();
   return;
  }
//  delete this->communication;
  }
  this->communication->connectToServer(this->sshInfos.hostname,
                                       this->sshInfos.port,
                                       this->timer->isActive());

}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::disconnectFromServer()
{
 bool reallyDisc = true;

 QAction * action = static_cast<QAction *>(sender());

 if(action == this->actions[ ACTION_DISC_FROM_SERVER ])
 {
  int answer = this->confirmClose();

  if(answer != CLOSE_CANCEL)
   this->communication->disconnectFromServer(answer == CLOSE_SHUTDOWN);
  else
   return;
 }

//   delete this->communication;
//   this->communication = NULL;

 // destroy the tree
 this->modelFilter->setSourceModel(NULL);
 delete this->treeModel;
 this->treeModel = NULL;

 this->setConnected(false);

 this->appendToLog("Disconnected␣from␣server", TYPE_NORMAL);
}
```

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::quit()
{
 if(this->communication == NULL)
  this->disconnectFromServer();
 QApplication::exit(0);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::getTree()
{
 this->communication->sendActionGetTree();
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::error(const QString & error, bool fromServer)
{
 if(fromServer)
  this->appendToLog(QString("[SERVER]␣") + error, MainWindow::TYPE_ERROR);
 else
  this->appendToLog(error, MainWindow::TYPE_ERROR);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::message(const QString & message)
{
 this->appendToLog(QString("[SERVER]␣") + message, MainWindow::TYPE_SERVER);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::contextMenu(const QPoint & mousePoint)
{
 QModelIndex index = this->treeView->indexAt(mousePoint);

 // if right-clic on an item (not on the background)
 if(index.isValid())
 {
  if(index != this->treeView->currentIndex())
  {
   this->treeView->setCurrentIndex(index);
   this->itemClicked(index);
  }
  // the context menu is shown, with top left corner at
  // the mouse cursor position
  this->mnuContext->exec(QCursor::pos());
 }
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::addNode()
{
 QAction * action = static_cast<QAction *>(sender());
 if(action == NULL || !this->abstractTypesActions.contains(action))
  return;

 this->communication->sendGetConcreteTypes(action->text());

 this->currentAbstractType = action->text();
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::concreteTypes(const QStringList & types)
{
// get the index in the filter
 QModelIndex index = this->treeView->currentIndex();

 // get the corresponding index in the model
 QModelIndex indexInModel = this->modelFilter->mapToSource(index);

 AddNodeDialog add(this);
 QDomDocument doc;
 QDomNode node;
 QString str;
 QString name = add.show(types, str); // show the add node dialog

 // remove starting and ending spaces
 name = name.trimmed();
 // replace spaces by underscores
 name = name.replace("␣", "_");

 if(name == "")
  return;

 node = this->treeModel->newChildToNode(indexInModel, name, doc);
 this->communication->sendActionAddNode(node, str,
                                       this->currentAbstractType);
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::addOption()
{
 QAction * action = qobject_cast<QAction *>(sender());

 QString name = QInputDialog::getText(this, "New␣option",
                                     "Enter␣the␣name␣of␣the␣new␣option:");

 if(!name.isNull() && !name.isEmpty())
 {
  int type = OptionsTypes::getTypeId(action->text());
  this->optionsPanel->addOption(type, name);
 }
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::renameNode()
{
 QModelIndex index = this->treeView->currentIndex();
 QModelIndex indexInModel = this->modelFilter->mapToSource(index);
 QDomNode node = this->treeModel->indexToNode(indexInModel);

 if(!node.isNull())
 {
  QString name = QInputDialog::getText(this, "Rename␣node",
                                       "New␣name␣of␣the␣new␣node:",
                                       QLineEdit::Normal,
                                       node.nodeName());

 // remove starting and ending spaces
  name = name.trimmed();
 // replace spaces by underscores
  name = name.replace("␣", "_");

  if(!name.isNull() && !name.isEmpty() && name != node.nodeName())
  {
   QDomDocument doc;
   QDomNode node2 = this->treeModel->renameToNode(indexInModel, name, doc);
   this->communication->sendActionRenameNode(node2, name);
  }
 }
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::deleteNode()
{
// get the index in the filter
 QModelIndex index = this->treeView->currentIndex();

 // get the corresponding index in the model
 QModelIndex indexInModel = this->modelFilter->mapToSource(index);
 QDomNode node = this->treeModel->indexToNode(indexInModel);

 this->communication->sendActionDeleteNode(node);

 this->optionsPanel->setOptions(QDomNodeList());
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::showProperties()
{
 bool ok;
// get the index in the filter
 QModelIndex index = this->treeView->currentIndex();

 // get the corresponding index in the model
 QModelIndex indexInModel = this->modelFilter->mapToSource(index);

 TObjectProperties properties;

 properties = this->treeModel->getProperties(indexInModel, ok);

 QMessageBox::information(this, "Properties",
                          QString("Abstract␣type:␣") + properties.absType +
                            QString("\nType:␣") + properties.type +
                            QString("\nMode:␣") + (properties.basic ?
                            "basic" : "advanced") +
                            QString("\nDynamic:␣") +
                            QVariant(properties.dynamic).toString()
                         );
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void MainWindow::connected()
{
 if(this->timer->isActive())
 {
  // stop the process (send SIGKILL signal)
  this->proLaunchServer->kill();
  this->timer->stop();
  this->appendToLog("Server␣started!", MainWindow::TYPE_NORMAL);
 }
 this->appendToLog("Now␣connected␣to␣server.", MainWindow::TYPE_NORMAL);

 this->setConnected(true);

 this->communication->sendGetAbstractTypes("SubSystem");
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::toggleAdvanced()
{
 bool advanced = this->actions[ ACTION_TOGGLE_ADVANCED_MODE ]->isChecked();

 if(this->treeModel != NULL)
  this->treeModel->setAdvancedMode(advanced);

 this->optionsPanel->setAdvancedMode(advanced);

 // don't show empty strings
 QRegExp reg(QRegExp(".+", Qt::CaseInsensitive, QRegExp::RegExp));
 this->modelFilter->setFilterRegExp(reg);
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::abstractTypes(const QStringList & types)
{
 QStringList::const_iterator it = types.begin();

 // if the menu is not enabled, it can not be modified, even by code.
 this->mnuAbstractTypes->setEnabled(true);

 this->abstractTypesActions.clear();

 // clearing the menu will destroy all its actions (the ones that are not
 // linked to another menu, toolbar, etc...)...so no additional delete
 // needed
 this->mnuAbstractTypes->clear();

 while(it != types.end())
 {
  QString type = *it;
  QAction * action = this->initAction(type, -1, SLOT(addNode()), true,
                                      this->mnuAbstractTypes);
  this->abstractTypesActions.append(action);
  it++;
 }

 this->mnuAbstractTypes->setEnabled(!this->abstractTypesActions.isEmpty());
 this->appendToLog("Abstract␣types␣list␣updated.", TYPE_NORMAL);
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::openFile()
{
  OpenFileDialog open(this, this->communication);

  QString file = open.show();

 if(!file.isEmpty())
  this->communication->sendOpenFile(file);
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::runSimulation()
{
 this->communication->sendRunSimulation();
 this->setSimRunning(true);
 this->getTree();
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::ack(int type)
{
 switch(type)
 {
  case NetworkFrames::TYPE_OPEN_FILE :
   // if the file is open, the client can run the simulation
   this->actions[ ACTION_RUN_SIMULATION ]->setEnabled(true);
   this->getTree();
   break;

  case NetworkFrames::TYPE_SIMULATION_RUNNING :
   this->setSimRunning(true);
   break;

  case NetworkFrames::TYPE_RUN_SIMULATION :
   this->appendToLog("The␣server␣said␣that␣the␣simulation␣has␣finished.",
                     TYPE_NORMAL);
   this->setSimRunning(false);
   break;

  case NetworkFrames::TYPE_ADD_NODE :
  case NetworkFrames::TYPE_DELETE_NODE :
  case NetworkFrames::TYPE_RENAME_NODE :
  case NetworkFrames::TYPE_MODIFY_NODE :
   this->appendToLog("Action␣succeeded.", TYPE_NORMAL);
   break;
 }
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::nack(int type)
{
 switch(type)
 {
  case NetworkFrames::TYPE_OPEN_FILE :
   this->appendToLog("The␣server␣could␣not␣open␣this␣file.", TYPE_ERROR);
   break;
  case NetworkFrames::TYPE_RUN_SIMULATION :
   this->appendToLog("Simulation␣failed␣due␣to␣an␣error.", TYPE_ERROR);
   break;

  case NetworkFrames::TYPE_ADD_NODE :
  case NetworkFrames::TYPE_DELETE_NODE :
  case NetworkFrames::TYPE_RENAME_NODE :
  case NetworkFrames::TYPE_MODIFY_NODE :
   this->appendToLog("Action␣failed.", TYPE_NORMAL);
   break;
 }
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::launchServer()
{
 QByteArray errors;
 QByteArray stdout;
 QString cmd;
 QProcess checkIfRunning;

 cmd = QString("ssh␣%1@%2␣check_coolfluid_server.sh␣%3")
   .arg(this->sshInfos.username)
   .arg(this->sshInfos.hostname)
   .arg(this->sshInfos.port);

 this->appendToLog("Checking␣if␣no␣other␣server␣instance␣is␣running␣on␣this␣port...", TYPE_NORMAL);

 checkIfRunning.start(cmd);

 checkIfRunning.waitForFinished(-1);

 QString output = checkIfRunning.readAllStandardOutput();
 QString error = checkIfRunning.readAllStandardError();

 if(!error.isEmpty())
 {
  this->appendToLog(error, TYPE_ERROR);
  return ;
 }

 // if output is different from "0", a server is already running on this port
 if(output != "0")
 {
  this->appendToLog(QString("A␣server␣is␣already␣running␣on␣port␣%1␣on␣%2.␣"
    "Please␣change␣the␣port␣or␣the␣hostname.")
    .arg(this->sshInfos.hostname)
    .arg(this->sshInfos.port), TYPE_ERROR);
  return ;
 }

 cmd = QString("ssh␣-n␣%1@%2␣start_coolfluid_server.sh␣%3")
   .arg(this->sshInfos.username)
   .arg(this->sshInfos.hostname)
   .arg(this->sshInfos.port);

 this->appendToLog("Starting␣the␣server...", TYPE_NORMAL);

 this->timer->start(100);

 connect(this->proLaunchServer, SIGNAL(readyReadStandardError()),
         this, SLOT(sshError()));
 this->proLaunchServer->start(cmd);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::tryToConnect()
{
 this->connectToServer();
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void MainWindow::sshError()
{
 this->timer->stop();

 QString error = this->proLaunchServer->readAllStandardError();
 this->appendToLog(error, TYPE_ERROR);

 // stop the process (send SIGKILL signal)
 this->proLaunchServer->kill();
}
```

1.8 *OpenFileDialog* class

1.8.1 OpenFileDialog.h

```
#ifndef COOLFluiD_client_OpenFileDialog_h
#define COOLFluiD_client_OpenFileDialog_h

////////////////////////////////////////////////////////////////////////////////

#include <QObject>
#include <QDialog>
#include <QIcon>

class QDialogButtonBox;
class QDomDocument;
class QIcon;
class QLabel;
class QLineEdit;
class QListView;
class QMainWindow;
class QModelIndex;
class QVBoxLayout;
class QSortFilterProxyModel;
class QStandardItemModel;

namespace COOLFluiD
{

 namespace treeview
 {
  class TreeModel;
 }

 namespace client
 {
  class MyQStringListModel;

  class CommClient;

////////////////////////////////////////////////////////////////////////////////

  /// @brief Dialog used to select a file to open.

  /// This class inherits from <code>QDialog</code> and is used to show a
  /// dialog allowing the user to select a file to open. The dialog is modal,
  /// wich means that once it isvisible, the calling code execution is
  /// stopped until the dialog isinvisible again. If the dialog has a parent
  /// window, it is centered on this parent. Otherwise, it is centered on the
  /// screen.<br>

  /// This class allows user to browse server files system. Double-clicking
  /// on a directory will send a request to the server to open a directory
  /// and return its contents. Thus the class needs a <code>CommClient</code>
  /// object with an open socket to communicate with the server. The
  /// constructor sends a request to the server to open the default
  /// directory.<br>

  /// After calling the constructor, the dialog is invisible. The show
  /// method has to be called to show it. This is a blocking method: it will
  /// not return until the dialog is invisible again. This method returns
  /// either  the name (with path) of the selected file (if he clicked on
  /// "OK" to validate his selection) or an empty string (if he clicked on
  /// "Cancel" or closed the dialog to cancel his selection).<br>
/// A typical use of this class is (assuming that <code>this</code> is a
/// <code>QMainWindow</code> object): <br>
/// \code
/// OpenFileDialog dialog(this);
/// QString name = dialog.show();
///
/// if(name != "")
/// {
/// // some treatements
/// }
/// \endcode

/// @author Quentin Gasper.

class OpenFileDialog : public QDialog
{
 Q_OBJECT

 private:

  /// @brief Label for the filter.
  QLabel * labFilter;

  /// @brief Label for the files list.
  QLabel * labFilesList;

  /// @brief Line edit for the filter.
  QLineEdit * editFilter;

  /// @brief Button box for "OK" and "Cancel" buttons.
  QDialogButtonBox * buttons;

  /// @brief List view used to display files list.
  QListView * listView;

  /// @brief Model for the list view
  QStandardItemModel * model;

  /// @brief Filter for the model.
  QSortFilterProxyModel * filterModel;

  /// @brief Dialog layout.
  QVBoxLayout * layout;

  /// @brief Indicates whether the user clicked on "OK" button or not.

  /// If the user clicked on "OK" button, the attribute value is
  /// <code>true</code>, otherwise (if the user closed the window or
  /// clicked on "Cancel" button) it is <code>false</code>.
  bool okClicked;

  /// @brief Object used to communicate with the server.
  CommClient * communication;

  /// @brief Path of the current directory.
  QString currentPath;

  /// @brief File selected by the user.
  QString currentFile;

 public:

  /// @brief Constructor.

  /// @param parent Parent window of the dialog. May be <code>NULL</code>.
```

```
    /// @param communication {CommClient object used to communicate
    /// with the server.

    /// @throw std::invalid_argument If the pointer parameter is
    /// <code>NULL</code>.
    OpenFileDialog(QMainWindow * parent, CommClient * communication);

    /// @brief Destructor.

    /// Frees all allocated memory. Parent window and
    /// <code>communication</code> object are not destroyed.
    ~OpenFileDialog();

    /// @brief Shows the dialog.

    /// This is a blocking method. It will not return while the dialog is
    /// visible.

    /// @return Returns the absolute path of the selected file to open.
    QString show();

  private slots:

    /// @brief Slot called when "OK" button is clicked.

    /// Sets <code>this->okClicked</code> to <code>true</code> and then sets
    /// the dialog to an invisible state.
    void btOkClicked();

    /// @brief Slot called when "Cancel" button is clicked.

    /// Sets <code>this->okClicked</code> to <code>false</code> and then sets
    /// the dialog to an invisible state.
    void btCancelClicked();

    /// @brief Slot called each time the text in the line edit is modified.

    /// The filter is modified according to this text and the list view is
    /// updated.

    /// @param text New text in the line edit.
    void filterUpdated(const QString & text);

    /// @brief Slot called when the server sends the contents of a directory
    /// to the client.

    /// The model is updated and the line edit is cleared.

    /// @param path Absolute path of the directoy of which contents belong
    /// to.
    /// @param dirs Directories list. Each element is a directory.
    /// @param files Files list. Each element is a file.
    void dirContent(const QString & path, const QStringList & dirs,
                    const QStringList & files);

    /// @brief Slot called when an error comes from the network level.

    /// @param error
    /// @param fromServer <code>true</code> if the error message comes from
    /// the server, otherwise <code>false</code>. This parameter is never
    /// used.
    void error(const QString & error, bool fromServer);

    /// @brief Slot called when the user double-click on an item in the
    /// list view.
    /// If the item is a directory, a request to open this directory is sent
    /// to the serveur. If the item is a file, the dialog reacts as if the
    /// user clicked on "OK" button with this item selected.

    /// @param index Clicked item index in the model filter.
    void doubleClick(const QModelIndex & index);
  };

//////////////////////////////////////////////////////////////////////////////

 }
}

//////////////////////////////////////////////////////////////////////////////

#endif // COOLFluiD_client_OpenFileDialog_h
```

1.8.2 OpenFileDialog.cxx

```
#include <QtGui>

#include <stdexcept>

#include <QDomDocument>

#include "ClientServer/treeview/TreeModel.h"

#include "ClientServer/client/CommClient.h"
#include "ClientServer/client/OpenFileDialog.h"
#include "ClientServer/client/FilesListItem.h"

using namespace COOLFluiD::client;
using namespace COOLFluiD::treeview;

OpenFileDialog::OpenFileDialog(QMainWindow * parent,
                               CommClient * communication)
 : QDialog(parent)
{
 this->setWindowTitle("Open␣file");

 if(communication == NULL)
  throw std::invalid_argument("The␣communication␣is␣a␣NULL␣pointer");

 // create the components
 this->labFilter = new QLabel("Filter␣(wildcards␣allowed)␣:");
 this->labFilesList = new QLabel("Files␣:");
 this->model = new QStandardItemModel();
 this->listView = new QListView(parent);
 this->editFilter = new QLineEdit(this);
 this->filterModel = new QSortFilterProxyModel();

 this->layout = new QVBoxLayout(this);

 this->buttons = new QDialogButtonBox(QDialogButtonBox::Ok
   | QDialogButtonBox::Cancel);

 this->communication = communication;

 this->okClicked = false;

 this->setModal(true);

 this->filterModel->setDynamicSortFilter(true);

 this->filterModel->setSourceModel(this->model);
 this->listView->setModel(this->filterModel);

 this->listView->setEditTriggers(QAbstractItemView::NoEditTriggers);

 // add the components to the layout
 this->layout->addWidget(this->labFilter);
 this->layout->addWidget(this->editFilter);
 this->layout->addWidget(this->labFilesList);
 this->layout->addWidget(this->listView);
 this->layout->addWidget(this->buttons);

 // connect useful signals to slots
 connect(this->buttons, SIGNAL(accepted()), this, SLOT(btOkClicked()));
 connect(this->buttons, SIGNAL(rejected()), this, SLOT(btCancelClicked()));
 connect(this->editFilter, SIGNAL(textEdited(const QString &)),
         this, SLOT(filterUpdated(const QString &)));
 connect(this->listView, SIGNAL(doubleClicked(const QModelIndex &)),
         this, SLOT(doubleClick(const QModelIndex &)));

 connect(this->communication, SIGNAL(dirContent(const QString &,
         const QStringList &, const QStringList &)), this,
         SLOT(dirContent(const QString &, const QStringList &,
              const QStringList &)));

 connect(this->communication, SIGNAL(error(const QString &, bool)),
         this, SLOT(error(const QString &, bool)));

 this->communication->sendOpenDir("");
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

OpenFileDialog::~OpenFileDialog()
{
 delete this->buttons;
 delete this->editFilter;
 delete this->filterModel;
 delete this->labFilter;
 delete this->labFilesList;
 delete this->layout;
 delete this->listView;
 delete this->model;

 // disconnecting the signals connected by the constructor
 // (normally, this is automatically done when the object is destroyed,
 // but the documentation is not clear on this point)
 disconnect(this);
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QString OpenFileDialog::show()
{
 this->currentFile = "";

 this->exec();

 if(this->okClicked)
  return this->currentFile;
 return QString();
}

/***************************************************************************

                                  SLOTS

***************************************************************************/

void OpenFileDialog::btOkClicked()
{
 QModelIndex index = this->listView->currentIndex();
 QModelIndex indexInModel = this->filterModel->mapToSource(index);

 FilesListItem * item;
 item = static_cast<FilesListItem *>(this->model->itemFromIndex(indexInModel));

 if(item != NULL) // if an item is selected
 {
  if(item->getType() == FilesListItem::FILE)
  {
   this->currentFile = this->currentPath + item->text();
   this->okClicked = true;
   this->setVisible(false);
  }
  else // if it is a directory, it is like double-clicking on it
   this->doubleClick(index);
 }
 else
  QMessageBox::critical(this, "Error", "Please␣select␣an␣item␣first");
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void OpenFileDialog::btCancelClicked()
{
 this->okClicked = false;
 this->setVisible(false);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void OpenFileDialog::filterUpdated(const QString & text)
{
 QRegExp regex(text, Qt::CaseInsensitive, QRegExp::Wildcard);
 this->filterModel->setFilterRegExp(regex);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void OpenFileDialog::dirContent(const QString & path,
                                const QStringList & dirs,
                                const QStringList & files)
{
 QStringList list;

 QIcon dirIcon = QFileIconProvider().icon(QFileIconProvider::Folder);
 QIcon fileIcon = QFileIconProvider().icon(QFileIconProvider::File);

 QStringList::const_iterator itDirs = dirs.begin();
 QStringList::const_iterator itFiles = files.begin();

 this->currentPath = path;
 if(!this->currentPath.endsWith("/"))
  this->currentPath += "/";

 this->labFilesList->setText("Files␣in␣" + this->currentPath);

 // clear the list
 this->model->clear();

 // add directories to the list
 while(itDirs != dirs.end())
 {
  model->appendRow(new FilesListItem(dirIcon, *itDirs,
                   FilesListItem::DIRECTORY));
  itDirs++;
 }

 // add files to the list
 while(itFiles != files.end())
 {
  model->appendRow(new FilesListItem(fileIcon, *itFiles,
                   FilesListItem::FILE));
  itFiles++;
 }
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void OpenFileDialog::error(const QString & error, bool fromServer)
{
 QMessageBox::critical(this, "Error", error);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void OpenFileDialog::doubleClick(const QModelIndex & index)
{
 QModelIndex indexInModel = this->filterModel->mapToSource(index);
 FilesListItem * item;
 item = static_cast<FilesListItem *>(
                              this->model->itemFromIndex(indexInModel));

 if(item == NULL)
  return;

 if(item->getType() == FilesListItem::DIRECTORY)
  this->communication->sendOpenDir(this->currentPath + item->text());
 else
  this->btOkClicked();
}
```

1.9 *OptionsPanel* class

1.9.1 OptionsPanel.h

```
#ifndef COOLFluiD_client_OptionsPanel_h
#define COOLFluiD_client_OptionsPanel_h

////////////////////////////////////////////////////////////////////////////////

#include <QDomNamedNodeMap>
#include <QLabel>
#include <QLineEdit>
#include <QList>
#include <QObject>
#include <QWidget>

class QDomNodeList;
class QFormLayout;
class QGridLayout;
class QGroupBox;
class QPushButton;
class GraphicalOption;


////////////////////////////////////////////////////////////////////////////////

namespace COOLFluiD
{
 namespace client
 {

////////////////////////////////////////////////////////////////////////////////


  /// @brief Panel to view and modify options of an object.

  /// This class allows user to display and modify options of an object or
  /// add new options.

  /// @author Quentin Gasper.

  class OptionsPanel : public QWidget
  {
   Q_OBJECT

   private:

    /// @brief List containing new basic options components.
    QList<GraphicalOption *> newBasicOptions;

    /// @brief List containing basic options components.
    QList<GraphicalOption *> basicOptions;

    /// @brief List containing advanced options components.
    QList<GraphicalOption *> advancedOptions;

    /// @brief List containing new advanced options components.
    QList<GraphicalOption *> newAdvancedOptions;

    /// @brief Button used to commit changes made.
    QPushButton * btCommitChanges;

    /// @brief XML document containing basic options nodes.
    /// This document does not contain newly created options.
    QDomDocument basicOptionsNodes;

    /// @brief XML document containing advanced options nodes.

    /// This document does not contain newly created options.
    QDomDocument advancedOptionsNodes;

    /// @brief XML document containing new basic options nodes.

    /// This document does not contain already existing options.
    QDomDocument newBasicOptionsNodes;

    /// @brief XML document containing new advanced options nodes.

    /// This document does not contain already existing options.
    QDomDocument newAdvancedOptionsNodes;

    /// @brief Layout used to display basic options components.
    QFormLayout * basicOptionsLayout;

    /// @brief Layout used to display advanced options components.
    QFormLayout * advancedOptionsLayout;

    /// @brief Main layout containing all widgets.

    /// This layout is composed of two lines and one column.
    QGridLayout * mainLayout;

    /// @brief Groupbox used to display basic options components
    /// with a titled border.

    ///  Its layout is <code>this->basicOptionsLayout</code>.
    QGroupBox * gbBasicOptions;

    /// @brief Groupbox used to display advanced options components
    /// with a titled border.

    ///  Its layout is <code>this->advancedOptionsLayout</code>.
    QGroupBox * gbAdvancedOptions;

    /// @brief Indicates if the line edits are in read-only mode or not.

    /// If <code>true</code>, the panel is in read-only mode. Only
    /// options having <code>dynamic</code> attribute set to
    /// <code>true</code> are modifiable.
    bool readOnly;

    /// @brief Indicates if the panel is in advanced mode or not.

    /// If <code>true</code>, the panel is in advanced mode. Advanced
    /// options (if any) are displayed. Otherwise, they are hidden.
    bool advancedMode;

    /// @brief Builds a Unix-like path string to the given node.

    /// The string begins with a slash followed by the root node name and
    /// all given node parent nodes names, seperated by slashed (like in a
    /// Unix path).

    /// @param node Node from which the path will be extracted.

    /// @return Returns the built strings.
    QString getNodePath(QDomNode & node);
    /// @brief Builds an XML document containing all modified options.

    /// First the basic options and then the advanced ones.

    /// @return Returns the built XML document.
    QDomDocument getOptions();

    /// @brief Builds an XML document containing all new options.

    /// First the basic options and then the advanced ones.

    /// @return Returns the built XML document.
    QDomDocument getNewOptions();

    /// @brief Clears the given list by deleting the <code>TOption</code>
    /// objects its elements point to.

    /// After calling this method, the list is empty.

    /// @param list The list to clear.
    void clearList(QList<GraphicalOption *> & list);

    /// @brief Builds a part (basic or advanced options) of the XML document
    /// returned by <code>this->getOption()</code> and
    /// <code>this->getNewOption()</code>.

    /// This document is built by comparing original options nodes to
    /// corresponding options components, which may have different values.
    /// If the values differ, the node is considered to have been modified
    /// and components values are taken as new values. Only modified nodes
    /// are appended to the document, which means that the document may be
    /// empty (if no option has been modified).

    /// @param nodes Original options nodes.
    /// @param options Options components.
    /// @param document Document where built nodes will be stored. The
    /// presence of this parameter is due to the fact that a node can not
    /// exist if it does not belong to a document.}
    void buildOptions(const QDomDocument & nodes,
                      const QList<GraphicalOption *> & options,
                      QDomDocument & document);

    /// @brief Applies the basic/advanced modes to the panel.

    /// For each node in the given XML document, the corresponding
    /// option components (in the given list) <code>enabled</code> property
    /// is set to <code>false</code> is the panel is in read-only mode but
    /// the option is not dynamic. In all other cases, the property will be
    /// set to <code>true</code>.

    /// @param optionsNodes XML document.
    /// @param options Corresponding options components.
    void setEnabled(const QDomDocument & optionsNodes,
                    const QList<GraphicalOption *> & options);


   public:
    /// @brief Constructor.

    /// Builds a <code>QWidget</code> with no options. The panel is
    /// neither in read-only mode nor advanced mode.

    /// @param parent The parent widget. Default value is
    /// <code>NULL</code>
    OptionsPanel(QWidget * parent = NULL);
    /// @brief Destructor.

    /// Frees the allocated memory.  Parent is not destroyed.
    ~OptionsPanel();

    /// @brief Assigns new node options to the panel.

    /// Old options and options components are deleted.

    /// @param options List of new options.
    void setOptions(const QDomNodeList & options);

    /// @brief Toggles read-only mode.

    /// @param readOnly If <code>true</code>, the read-only mode is
    /// activated. Otherwise it is deactivated.
    void setReadOnly(bool readOnly);

    /// @brief Toggles advanced mode.

    /// @param advanced {If <code>true</code>, the advanced mode is
    /// activated. Otherwise it is deactivated.
    void setAdvancedMode(bool advanced);

    /// @brief Creates a new option.

    /// @param optionType New option type. This value must be one of those
    /// defined by <code>OptionsTypes</code> class. If the type is not valid
    /// (if <code>OptionsTypes::isValid(optionType)</code> returns
    /// <code>false</code>), this method returns directly without create
    /// any option.
    /// @param name New option name.
    /// @param basic If <code>true</code>, a new basic option will be
    /// created. Otherwise, a new advanced option is created.
    /// @param dynamic If <code>true</code>, a new dynamic option will be
    /// created. Otherwise, a new static option is created.
    void addOption(int optionType, const QString & name,
                   bool basic = true, bool dynamic = false);

    /// @brief Indicates wether the line edits are in read-only mode.

    /// @return Returns <code>true</code> if the panel is in read-only mode,
    /// otherwise returns <code>false</code>.
    bool getReadOnly() const;

    /// @brief Indicates wether the line edits are in read-only mode.

    /// @return Returns <code>true</code> if the panel is in advanced mode,
    /// otherwise returns <code>false</code>.
    bool getAdvancedMode() const;

  private slots:

    /// @brief Slot called when user clicks on "Commit changes" button.

    /// If at least one option has been modified, <code>changesMade</code>
    /// signal is emitted.
    void commitChanges();

  signals:

    /// @brief Signal emitted when user clicks on "Commit changes" button if
    /// at least one option has been modified.
    /// @param modOptions XML document representing all modified options.
    /// Each document child is a modified option.
    /// @param newOptions XML document representing all new options. Each
    /// document child is a new option.
    void changesMade(const QDomDocument & modOptions,
                     const QDomDocument & newOptions);
  };

////////////////////////////////////////////////////////////////////////////////

 } // client
} // COOLFluiD

////////////////////////////////////////////////////////////////////////////////

#endif // COOLFluiD_client_OptionsPanel_h
```

1.9.2 OptionsPanel.cxx

```
#include <iostream>
#include <QtCore>
#include <QtGui>

#include "ClientServer/client/GraphicalOption.h"
#include "ClientServer/client/OptionsPanel.h"
#include "ClientServer/client/OptionsTypes.h"

using namespace COOLFluiD::client;

OptionsPanel::OptionsPanel(QWidget * parent) : QWidget(parent)
{
 // create the components
 this->mainLayout = new QGridLayout(this);
 this->basicOptionsLayout = new QFormLayout();
 this->advancedOptionsLayout = new QFormLayout();
 this->btCommitChanges = new QPushButton("Commit␣changes...");
 this->gbBasicOptions = new QGroupBox();
 this->gbAdvancedOptions = new QGroupBox();

 this->gbBasicOptions->setLayout(this->basicOptionsLayout);
 this->gbAdvancedOptions->setLayout(this->advancedOptionsLayout);

 // add the components to the layout
 this->mainLayout->addWidget(this->gbBasicOptions, 0, 0);
 this->mainLayout->addWidget(this->gbAdvancedOptions, 1, 0);
 this->mainLayout->addWidget(this->btCommitChanges, 2, 0);
 this->mainLayout->setRowStretch(0, 1);

 // add the layout to the dialog
 this->setLayout(this->mainLayout);

 this->readOnly = false;
 this->advancedMode = false;

 this->gbBasicOptions->setVisible(false);
 this->gbAdvancedOptions->setVisible(false);
 this->btCommitChanges->setVisible(false);

 connect(this->btCommitChanges, SIGNAL(clicked()), this,
         SLOT(commitChanges()));
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

OptionsPanel::~OptionsPanel()
{
 this->clearList(this->basicOptions);
 this->clearList(this->advancedOptions);
 this->clearList(this->newBasicOptions);
 this->clearList(this->newAdvancedOptions);

 delete this->btCommitChanges;
 delete this->gbBasicOptions;
 delete this->gbAdvancedOptions;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void OptionsPanel::setReadOnly(bool readOnly)
{
 // if the parameter and the attribute are different...
 if(this->readOnly ^ readOnly)
 {
  this->readOnly = readOnly;

  // ...we change the editors readOnly property
  this->setEnabled(this->basicOptionsNodes, this->basicOptions);
  this->setEnabled(this->advancedOptionsNodes, this->advancedOptions);
 }
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void OptionsPanel::setEnabled(const QDomDocument & optionsNodes,
                              const QList<GraphicalOption *> & options)
{
 QDomNodeList nodes = optionsNodes.childNodes();

 for(int i = 0 ; i < nodes.count() ; i++)
 {
  QDomNode currentNode = nodes.at(i);
  QDomNamedNodeMap attributes = currentNode.attributes();
  bool isDynamic = attributes.namedItem("dynamic").nodeValue() == "true";

  if(this->readOnly && isDynamic)
   options.at(i)->setEnabled(true);

  else if(!this->readOnly)
   options.at(i)->setEnabled(true);

  else
   options.at(i)->setEnabled(false);
 }
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

bool OptionsPanel::getReadOnly() const
{
 return this->readOnly;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void OptionsPanel::addOption(int optionType, const QString & name,
                             bool basic, bool dynamic)
{
 QDomElement node = this->newBasicOptionsNodes.createElement(name);
 int typeId;
 GraphicalOption * newOption;
 QString mode = basic ? "basic" : "advanced";
 QString dynamicStr = QString("%1").arg(dynamic);

 if(node.isNull() || name.isNull() || name.isEmpty())
  return;

 QString typeString = OptionsTypes::getTypeString(optionType);

 if(typeString.isEmpty())
  ; // stop and rollback

 node.setAttribute("tree", "option");
 node.setAttribute("type", typeString);
 node.setAttribute("mode", mode);
 node.setAttribute("dynamic", dynamicStr);

 newOption = new GraphicalOption(optionType);
 newOption->setName(name + QString(":"));

 // if the option is basic...
 if(basic)
 {
  newOption->addToLayout(this->basicOptionsLayout);
  this->newBasicOptionsNodes.appendChild(node);
  this->newBasicOptions.append(newOption);
  this->gbBasicOptions->setVisible(true);
 }
 else // ...or advanced
 {
  newOption->addToLayout(this->advancedOptionsLayout);
  this->newAdvancedOptionsNodes.appendChild(node);
  this->newAdvancedOptions.append(newOption);
  this->gbAdvancedOptions->setVisible(true);
 }
 this->btCommitChanges->setVisible(true);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument OptionsPanel::getOptions()
{
 QDomDocument doc;

 this->buildOptions(this->basicOptionsNodes, this->basicOptions, doc);
 this->buildOptions(this->advancedOptionsNodes, this->advancedOptions, doc);

 return doc;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void OptionsPanel::buildOptions(const QDomDocument & nodes,
                                 const QList<GraphicalOption *> & options,
                                 QDomDocument & document)
{
 QDomNodeList childNodes = nodes.childNodes();

 for(int i = 0 ; i < childNodes.count() ; i++)
 {
  QDomElement child;
  QDomNodeList childrenList;
  QDomText text;
  QString newValue;

  child = document.importNode(childNodes.at(i), true).toElement();
  childrenList = child.childNodes();

  text = childrenList.item(0).toText();
  newValue = options.at(i)->getValue();

  if(text.isNull())
   text = document.createTextNode("");

  if(newValue != text.nodeValue().trimmed())
  {
    text.setData(newValue);
    child.appendChild(text);
    document.appendChild(child);
   }
  }
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument OptionsPanel::getNewOptions()
{
 QDomDocument doc;

 this->buildOptions(this->newBasicOptionsNodes, this->newBasicOptions, doc);
 this->buildOptions(this->newAdvancedOptionsNodes,
                    this->newAdvancedOptions, doc);

 return doc;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void OptionsPanel::clearList(QList<GraphicalOption *> & list)
{
 QList<GraphicalOption *>::iterator it = list.begin();

 while(it != list.end())
 {
  delete *it;
  it++;
 }

 list.clear();
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void OptionsPanel::setAdvancedMode(bool advanced)
{
 this->advancedMode = advanced;
 if(advanced && this->advancedOptions.count() > 0)
 {
  this->mainLayout->setRowStretch(1, 1);
  this->gbAdvancedOptions->setVisible(true);
 }
 else
 {
  this->mainLayout->setRowStretch(1, 0);
  this->gbAdvancedOptions->setVisible(false);
 }
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

bool OptionsPanel::getAdvancedMode() const
{
 return this->advancedMode;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void OptionsPanel::setOptions(const QDomNodeList & options)
{
 // delete old widgets
 this->clearList(this->basicOptions);
 this->clearList(this->advancedOptions);
 this->clearList(this->newBasicOptions);
 this->clearList(this->newAdvancedOptions);

 this->basicOptionsNodes.clear();
 this->advancedOptionsNodes.clear();
 this->newBasicOptionsNodes.clear();
 this->newAdvancedOptionsNodes.clear();

 // set the new widgets

 if(!options.isEmpty())
 {
  QDomNode parentNode = options.at(0).parentNode();
  QString parentPath = this->getNodePath(parentNode);


  this->gbBasicOptions->setTitle(QString("Basic␣options␣of␣") + parentPath);

  this->gbAdvancedOptions->setTitle(QString("Advanced␣options␣of␣") +
    parentPath);

  this->gbBasicOptions->setVisible(true);
  this->gbAdvancedOptions->setVisible(this->advancedMode);
  this->btCommitChanges->setVisible(true);
 }
 else
 {
  this->gbBasicOptions->setVisible(false);
  this->gbAdvancedOptions->setVisible(false);
  this->btCommitChanges->setVisible(false);
 }

 for(int i = 0 ; i < options.count() ; i++)
 {
  QDomNode option = options.at(i);
  QDomNode data;
  QDomNodeList childNodes = option.childNodes();


  if(!childNodes.isEmpty() &&
      option.attributes().namedItem("tree").nodeValue() == "option")
  {
   GraphicalOption * tOption;
   QString description;

   QString typeString = option.attributes().namedItem("type").nodeValue();
   int type = OptionsTypes::getTypeId(typeString);

   if(type == OptionsTypes::NO_TYPE) /// @todo stop and rollback
    return;

   tOption = new GraphicalOption(type);
   tOption->setName(option.nodeName() + QString(":"));
   tOption->setValue(option.toElement().text().trimmed());

   description = option.attributes().namedItem("description").nodeValue();
   tOption->setToolTip(description);

   if(option.attributes().namedItem("mode").nodeValue() == "basic")
   {
    this->basicOptions.append(tOption);

    tOption->addToLayout(this->basicOptionsLayout);
    this->basicOptionsNodes.appendChild(
                        this->basicOptionsNodes.importNode(option, true));
   }

   else if(option.attributes().namedItem("mode").nodeValue() == "advanced")
   {
    this->advancedOptions.append(tOption);
    tOption->addToLayout(this->advancedOptionsLayout);
    this->advancedOptionsNodes.
      appendChild(this->advancedOptionsNodes.importNode(
                  option, true));
   }
  }
 }

 if(this->advancedOptions.count() == 0)
  this->gbAdvancedOptions->setVisible(false);

 this->setEnabled(this->basicOptionsNodes, this->basicOptions);
 this->setEnabled(this->advancedOptionsNodes, this->advancedOptions);
}

/*****************************************************************************

                                PRIVATE METHOD

*****************************************************************************/

QString OptionsPanel::getNodePath(QDomNode & node)
{
 QDomNode parentNode = node.parentNode();
 QString name = node.nodeName();

 if(parentNode.isNull()) // if the node has no parent
  return QString();

 return this->getNodePath(parentNode) + QString("/") + name;
}

/*****************************************************************************

                                   SLOTS

*****************************************************************************/

void OptionsPanel::commitChanges()
{
 QDomDocument modOptions = this->getOptions();
 QDomDocument newOptions = this->getNewOptions();

 if(modOptions.hasChildNodes() || newOptions.hasChildNodes())
  emit changesMade(modOptions, newOptions);
}
```

1.10 *OptionsTypes* class

1.10.1 OptionsTypes.h

```
#ifndef COOLFLuiD_client_OptionsTypes_h
#define COOLFLuiD_client_OptionsTypes_h

////////////////////////////////////////////////////////////////////////////////

#include <QHash>

namespace COOLFluiD
{
 namespace client
 {

  /// @brief Defines and manages the option types.

  /// @author Quentin Gasper.

////////////////////////////////////////////////////////////////////////////////

  class OptionsTypes
  {
   private:

    /// @brief Hash map with all types.

    /// The key is the type id defined by one the public constant interger
    /// attributes of this class. The value is the type name for this id. All
    /// types ids have a name except <code>NO_TYPE</code>.
    static QHash<int, QString> types;

    /// @brief Builds the types hash map.

    /// This function builds the hash map at most once during runtime. If it
    /// is called a second time, it returns without doing anything.
    static void buildTypes();

   public:

    /// @brief Type id used to indicate an invalid type.
    static const int NO_TYPE = -1;

    /// @brief Type id used to indicate a <i>boolean</i> type.
    static const int TYPE_BOOL = 0;

    /// @brief Type id used to indicate a <i>signed integer</i> type.
    static const int TYPE_INT = 1;

    /// @brief Type id used to indicate a <i>integer</i> type.
    static const int TYPE_UNSIGNED_INT = 2;

    /// @brief Type id used to indicate a <i>double</i> type.
    static const int TYPE_DOUBLE = 3;

    /// @brief Type id used to indicate a <i>string</i> type.
    static const int TYPE_STRING = 4;

    /// @brief Type id used to indicate a <i>files</i> type.
    static const int TYPE_FILES = 5;

    /// @brief Checks if a type id is valid.
    /// A type id is valid if it exists and is it has a type name associated.
    /// Thus <code>OptionsTypes::NO_TYPE</code> will not be considered as
    /// valid by this function.

    /// @param id The type id to check.

    /// @return Returns <code>true</code> if the type id is valid,
    /// otherwise returns <code>false</code>.
    static bool isValid(int id);

    /// @brief Gives the type id of a given type name.

    /// @param type The type name.

    /// @return Returns the type id corresponding to the given type name, or
    /// <code>OptionsTypes::NO_TYPE</code> if the type name is unknown.
    static int getTypeId(const QString & type);

    /// @brief Gives the type name for a given type id.

    /// @param type The type id.

    /// @return Returns the type name for the provided type id, or an empty
    /// string if the type id does not exist or if it is
    /// <code>OptionsTypes::NO_TYPE</code>.
    static QString getTypeString(int type);

    /// @brief Gives a types list.

    /// This list contains all types that have a name associated to their id.
    /// This list is not sorted.

    /// @returns Returns the types list.
    static QStringList getTypesList();
  };

////////////////////////////////////////////////////////////////////////////////

 } // namespace client
} // namespace COOLFluiD

////////////////////////////////////////////////////////////////////////////////

#endif // COOLFLuiD_client_OptionsTypes_h
```

1.10.2 OptionsTypes.cxx

```
#include <QtCore>

#include "ClientServer/client/OptionsTypes.h"

using namespace COOLFluiD::client;

const int OptionsTypes::NO_TYPE;
const int OptionsTypes::TYPE_BOOL;
const int OptionsTypes::TYPE_INT;
const int OptionsTypes::TYPE_UNSIGNED_INT;
const int OptionsTypes::TYPE_DOUBLE;
const int OptionsTypes::TYPE_STRING;
const int OptionsTypes::TYPE_FILES;

QHash<int, QString> OptionsTypes::types;

bool OptionsTypes::isValid(int id)
{
 OptionsTypes::buildTypes();

 return OptionsTypes::types.contains(id);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

int OptionsTypes::getTypeId(const QString & type)
{
 QHash<int, QString>::iterator it;

 OptionsTypes::buildTypes();

 it = OptionsTypes::types.begin();

 while(it != OptionsTypes::types.end())
 {
  if(it.value() == type)
   return it.key();
  it++;
 }

 return OptionsTypes::NO_TYPE;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QString OptionsTypes::getTypeString(int type)
{
 OptionsTypes::buildTypes();

 if(OptionsTypes::isValid(type))
  return OptionsTypes::types[type];

 return QString();
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void OptionsTypes::buildTypes()
{
 static bool mapBuilt = false;
 if(mapBuilt) // if the map has already been built...
  return;     // the function returns (there no need to build it again)

 OptionsTypes::types[ TYPE_BOOL ] = "bool";
 OptionsTypes::types[ TYPE_INT ] = "int";
 OptionsTypes::types[ TYPE_UNSIGNED_INT ] = "unsigned␣int";
 OptionsTypes::types[ TYPE_DOUBLE ] = "double";
 OptionsTypes::types[ TYPE_STRING ] = "std::string";
 OptionsTypes::types[ TYPE_FILES ] = "files";

 mapBuilt = true;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QStringList OptionsTypes::getTypesList()
{
 static QStringList list;
 static bool listBuilt = false;
 QHash<int, QString>::iterator it;

 if(listBuilt)   // if the list has already been built...
  return list;   // the function returns (there no need to build it again)

 OptionsTypes::buildTypes();

 it = OptionsTypes::types.begin();

 while(it != OptionsTypes::types.end())
 {
  list << it.value();
  it++;
 }

 listBuilt = true;

 return list;
}
```

1.11 *TSshInformation* structure

1.11.1 TSshInformation.h

```
#ifndef COOLFluiD_client_TSshInformation_h
#define COOLFluiD_client_TSshInformation_h

////////////////////////////////////////////////////////////////////////////

namespace COOLFluiD
{
 namespace client
 {

////////////////////////////////////////////////////////////////////////////

  /// @author Quentin Gasper.

  struct TSshInformation
  {
   public :

    /// @brief Remote machine hostname.
    QString hostname;

    /// @brief Username to use to authenticate to the remote machine.
    QString username;

    /// @brief Socket port number.
    quint16 port;

    /// @brief If <code>true</code>, the user requests to launch a new server
    /// instance.
    bool launchServer;

    /// @brief Constructor.

    /// Provided for convinience.

    /// @param hostname Remote machine hostname.
    /// @param port Socket port number.
    /// @param launchServer If <code>true</code>, the user requests to
    /// launch a new server instance.
    /// @param username Username to use to authenticate to the remote
    /// machine.
    TSshInformation(const QString & hostname = QString("hostname"),
                    quint16 port = 62784,
                    bool launchServer = false,
                    const QString & username = QString())
    {
     this->hostname = hostname;
     this->username = username;
     this->port = port;
     this->launchServer = launchServer;
    }
  };

////////////////////////////////////////////////////////////////////////////

 } // client
} // COOLFluiD

////////////////////////////////////////////////////////////////////////////
```

```
#endif // COOLFluiD_client_TSshInformation_h
```

Chapter 2

Server

2.1 *CommServer* class

2.1.1 CommServer.h

```
#ifndef COOLFluid_server_CommServer_h
#define COOLFluid_server_CommServer_h

////////////////////////////////////////////////////////////////////////////////

#include <QObject>
#include <QAbstractSocket>
#include <QDomDocument>
#include <QList>
#include <QMutex>

class QHostAdress;
class QTcpServer;
class QTcpSocket;
class QString;


namespace COOLFluiD
{
 namespace network
 {
  class NetworkException;
 }

 namespace server
 {

  class ServerSimulation;

  ////////////////////////////////////////////////////////////////////////////////

  /// @brief This class is the server network level.

  /// @author Quentin Gasper.

  class CommServer : public QObject
  {
   Q_OBJECT
```

```
private:

 /// @brief The default path for the file browsing.

 /// The default path is the current directory (./).
 const QString DEFAULT_PATH;

 /// @brief The server socket.

 /// Used to accept connections.
 QTcpServer * server;

 /// @brief The server socket for the local loop.

 /// Used to accept connections coming from "localhost" (local loop).
 QTcpServer * localSocket;

  /// @brief Size of the frame that is being read.

 /// If the value is 0, no frame is currently being recieved.
 quint16 blockSize;

 /// @brief The client sockets.

 /// The key is pointer to the socket. ...
 QHash<QTcpSocket *, QDomNode> clients;

 /// @brief The simulation.
 ServerSimulation * srvSimulation;

 QDomDocument types;

 /// @brief List of clients that are requesting.
 /// <b>This attribute should be deleted when deleting TreeManager
 /// class</b>
 QList<QTcpSocket *> clientsRequesting;

 /// @brief Indicates wether a file is already open.

 /// If <code>true</code>, a file is already open.
 bool fileOpen;

 /// @brief Indicates wether the simulation is running.

 /// If <code>true</code>, the simulation is running.
 bool simRunning;

 /// @brief Number of bytes recieved.
 int bytesRecieved;

 /// @brief Number of bytes sent.
 int bytesSent;

 /// @brief Sends a message to a client.

 /// @param client Client socket to use. If <code>NULL</code>, the
 /// message will be sent to all clients.
 /// @param message Message to send.
 void sendMessage(QTcpSocket * client, const QString & message) ;

 /// @brief Sends an error message to a client.

 /// @param client Client socket to use. If <code>NULL</code>, the
 /// error message will be sent to all clients.
 /// @param message Error message to send.
```

```
    void sendError(QTcpSocket * client, const QString & message) ;

    /// @brief Sends a message to a client.

    /// @param client Client socket to use. If <code>NULL</code>, the
    /// frame will be sent to all clients.
    /// @param frame Frame to send.
    void send(QTcpSocket * client, const QString & frame);

    /// @brief Reads a directory contents.

    /// @param directory Directory to read.
    /// @param dirsList Reference of a <code>QStringList</code> where
    /// sub-directories names will be stored.
    /// @param filesList Reference of a <code>QStringList</code> where
    /// files names will be stored.

    /// @return Returns <code>true</code> if the directory has been correctly
    /// read. Otherwise, returns <code>false</code> (<code>dirsList</code>
    /// and <code>filesList</code> are not modified in this case).
    bool getDirContent(const QString & directory,
                       QStringList & dirsList,
                       QStringList & filesList) const;

    /// @brief Requests to the simulator to open a file.

    /// This method returns when the file is successfully opened or when an
    /// error has occured.

    /// @param client Client socket to use. If <code>NULL</code>, the frame
    /// will be sent to all clients.
    /// @param filename Name of the file to open.
    void openFile(QTcpSocket * client, const QString & filename);

    /// @brief Requests to the simulator to run the simulation.

    /// Starts the simulator thread and returns immediately.
    void runSimulation();

    /// @brief Sends an ACK/NACK.

    /// @param client Client socket to use. If <code>NULL</code>, the frame
    /// will be sent to all clients.
    /// @param success If <code>true</code> an "ACK" frame is built.
    /// Otherwise, a "NACK" frame is built.
    /// @param type Type of the frame to ACK/NACK.
    void sendAck(QTcpSocket * client, bool success, int type);

    /// @brief Sends the abstract types list.

    /// @param client Client socket to use. If <code>NULL</code>, the frame
    /// will be sent to all clients.
    /// @param typeName Type name of which the abstract types are requested.
    void sendAbstractTypes(QTcpSocket * client, const QString & typeName);

    /// @brief Sends the concrete types list.

    /// @param client Client socket to use. If <code>NULL</code>, the frame
    /// will be sent to all clients.
    /// @param typeName Abstract type name of which the abstract types are
    /// requested.
    /// @param typesList Concrete types list.
    void sendConcreteTypes(QTcpSocket * client, const QString & typeName,
                           const QStringList & typesList);
    /// @brief Sent the files list.

    /// @param client Client socket to use. If <code>NULL</code>, the frame
    /// will be sent to all clients.
    /// @param dirname Directory from which the files list is read.
    void sendFilesList(QTcpSocket * client, const QString & dirname);

  public:

    /// @brief Constructor.

    /// Creates a new socket with the address and port provided. The socket
    /// <code>client</code> is set to <code>NULL</code>.
    /// @param hostAddress Server address.
    /// @param port Socket port.
    /// @throw NetworkException Throws a NetworkException if the server
    /// cannot listen to the given address/port.
    CommServer(QString hostAddress = "127.0.0.1", quint16 port = 62784);

    /// @brief Destructor.

    /// Closes the sockets before the object is deleted.
    ~CommServer();

    /// @brief Gives the number of bytes recieved.

    /// @return Returns the number of bytes recieved.
    int getBytesRecieved() const;

    /// @brief Gives the number of bytes sent.

    /// @return Returns the number of bytes sent.
    int getBytesSent() const;

    /// @brief Sends the tree to a client.

    /// <b>This method should be deleted when deleting TreeManager class</b>

    /// @param clientId Client id in <code>clientsRequesting</code> list, or
    /// -1 to send to all clients.
    /// @param tree The tree.
    void sendTree(int clientId, const QDomDocument & tree) ;

    /// @brief Sends a message to a client.

    /// <b>This method should be deleted when deleting TreeManager class</b>

    /// @param clientId Client id in <code>clientsRequesting</code> list, or
    /// -1 to send to all clients.
    /// @param message The message.
    void sendMessage(int clientId, const QString & message) ;

    /// @brief Sends an error message to a client.

    /// <b>This method should be deleted when deleting TreeManager class</b>

    /// @param clientId Client id in <code>clientsRequesting</code> list, or
    /// -1 to send to all clients.
    /// @param message The error message.
    void sendError(int clientId, const QString & message) ;

    /// @brief Sends an ACK or a NACK to a client.

    /// <b>This method should be deleted when deleting TreeManager class</b>
    /// @param clientId Client id in <code>clientsRequesting</code> list, or
    /// -1 to send to all clients.
    /// @param type The type of the frame to ACK/NACK.
    /// @param success If <code>true</code> an ACK is sent, otherwise a NACK
    /// is sent.
    void sendAck(int clientId, int type, bool success);

   private slots :

    /// @brief Slot called when a new client tries to connect.
    void newClient();

    /// @brief Slot called when new data are available on one of the
    /// client sockets.
    void newData();

    /// @brief Slot called when the client has been disconnected.
    void clientDisconnected();

    /// @brief Slot called when the simulation is done.
    void simulationFinished();

    /// @brief Slot called when a message (i.e. output forwarding) comes
    /// from the simulator.

    /// Sends this message to all clients.

    /// @param message The message.
    void message(const QString & message);


    /// @brief Slot called when an error comes from the simulator.

    /// Sends this error message to all clients.

    /// @param message The error message.
    void error(const QString & message);


   signals:

    /// @brief Signal emitted to add a node to the tree.

    /// <b>This signal should be deleted when deleting TreeManager class</b>

    /// @param clientId Client id
    /// @param path Parent node path
    /// @param name Node name
    /// @param type Node type
    /// @param absType Node abstract type
    void addNode(int clientId, const QString & path, const QString & name,
                 const QString & type, const QString & absType);

    /// @brief Signal emitted to delete a node from the tree.

    /// <b>This signal should be deleted when deleting TreeManager class</b>

    /// @param clientId Client id
    /// @param path Node path
    void deleteNode(int clientId, const QString & path);

    /// @brief Signal emitted to add a node to the tree.

    /// <b>This signal should be deleted when deleting TreeManager class</b>
    /// @param clientId Client id
    void getTree(int clientId);

    /// @brief Signal emitted to modify a node in the tree.

    /// <b>This signal should be deleted when deleting TreeManager class</b>

    /// @param clientId Client id
    /// @param document XML data with the options to modify/add.
    void modifyNode(int clientId, const QDomDocument & document);

    /// @brief Signal emitted to rename a node in the tree.

    /// <b>This signal should be deleted when deleting TreeManager class</b>

    /// @param clientId Client id
    /// @param path Parent node path
    /// @param newName Node new name
    void renameNode(int clientId, const QString & path,
                    const QString & newName);

  };

////////////////////////////////////////////////////////////////////////////////

 } // namespace network
} // namespace COOLFluiD

////////////////////////////////////////////////////////////////////////////////

#endif // COOLFluid_server_CommServer_h
```

2.1.2 CommServer.cxx

```
#include <iostream>

#include <QtNetwork>
#include <QtXml>
#include <QtCore>

#include "ClientServer/network/ClientServerXMLParser.h"
#include "ClientServer/network/NetworkException.h"
#include "ClientServer/network/NetworkFrames.h"
#include "ClientServer/server/CommServer.h"
#include "ClientServer/server/ServerSimulation.h"

using namespace COOLFluiD::network;
using namespace COOLFluiD::server;

CommServer::CommServer(QString hostAddress, quint16 port)
 : DEFAULT_PATH(".")
{
 bool local = hostAddress == "127.0.0.1";

 this->server = new QTcpServer(this);

 if(!local)
  this->localSocket = new QTcpServer(this);
 else
  this->localSocket = NULL;

 // load available types from XML file
 /// TODO exception if error on reading the file

 QFile typesFile("./TypesList.xml"); // list of available types

 if (typesFile.open(QIODevice::ReadOnly) &&
     this->types.setContent(&typesFile))
  typesFile.close();

 if(!this->server->listen(QHostAddress(hostAddress), port))
 {
  throw NetworkException("Cannot␣listen␣" + hostAddress + "␣on␣port␣" +
    QVariant(port).toString() + ":␣" + this->server->errorString());
 }

 if(!local && !this->localSocket->listen(QHostAddress("127.0.0.1"), port))
 {
  throw NetworkException("Cannot␣listen␣" + hostAddress + "␣on␣port␣" +
    QVariant(port).toString() + ":␣" + this->server->errorString());
 }

 this->fileOpen = false;
 this->simRunning = false;

 this->srvSimulation = new ServerSimulation("Simulator");

 connect(this->srvSimulation, SIGNAL(message(const QString &)), this,
        SLOT(message(const QString &)));

 connect(this->srvSimulation, SIGNAL(error(const QString &)), this,
        SLOT(error(const QString &)));

 connect(this->srvSimulation, SIGNAL(finished()),
        this, SLOT(simulationFinished()));

 connect(this->server, SIGNAL(newConnection()), this, SLOT(newClient()));
 if(!local)
  connect(this->localSocket, SIGNAL(newConnection()), this,
          SLOT(newClient()));

 this->server->setMaxPendingConnections (1);
 this->blockSize = 0;
 this->bytesRecieved = 0;
 this->bytesSent = 0;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

CommServer::~CommServer()
{
 /// TODO delete sockets in CommServer destructor

/* for(int i = 0 ; i < this->clients.size() ; i++)
 {
 this->clients.at(i)->abort(); // cancel active data transfert
 this->clients.at(i)->close(); // close the socket
}*/
 this->server->close();
 delete this->srvSimulation;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::send(QTcpSocket * client, const QString & frame)
{
 QByteArray block;
 QDataStream out(&block, QIODevice::WriteOnly);

 out.setVersion(QDataStream::Qt_4_4);
 // reserving 2 bytes to store the data size
 // (frame size without these 2 bytes)
 out << (quint16)0;
 out << frame;
 out.device()->seek(0); // go back to the beginning of the frame
 out << (quint16)(block.size() - sizeof(quint16)); // store the data size

 if(client == NULL)
 {
  QHash<QTcpSocket *, QDomNode>::iterator it = this->clients.begin();

  while(it != this->clients.end())
  {
   client = it.key();
   this->bytesSent += client->write(block);
   client->flush();
   it++;
  }
 }
 else
 {
  this->bytesSent += client->write(block);
  client->flush();
 }
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
void CommServer::sendError(int clientId, const QString & message)
{
 QDomDocument doc = NetworkFrames::buildError(message);
 QTcpSocket * client;

 if(clientId == -1)
  client = NULL;

 else if(clientId < 0 || clientId > this->clientsRequesting.size())
  return;

 else
  client = this->clientsRequesting.at(clientId);

 this->send(client, doc.toString());
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::sendAbstractTypes(QTcpSocket * client,
                                   const QString & typeName)
{
 QDomNode node = this->types.namedItem(typeName);
 QDomDocument document;
 QDomNodeList childNodes;
 QStringList typesList;

 // if the node is null, typeName is not a existing type man
 if(node.isNull())
 {
  this->sendError(client, QString("Type␣'") + typeName +
    QString("'␣does␣not␣exist."));
  return;
 }

 childNodes = node.childNodes();

 // if no child, no types to send
 if(childNodes.isEmpty())
 {
  this->sendError(client, QString("No␣abstract␣type␣for␣type␣'%1'")
    .arg(typeName));

  return;
 }

 // building the types list
 for(int i = 0 ; i < childNodes.count() ; i++)
  typesList << childNodes.at(i).nodeName();

 document = NetworkFrames::buildTypesList(NetworkFrames::TYPE_ABSTRACT_TYPES,
                                          typeName, typesList);

 this->send(client, document.toString());

 // remember that this client works on this node
 this->clients[client] = node;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::sendConcreteTypes(QTcpSocket * client,
                                   const QString & typeName,
                                   const QStringList & typesList)
{
 QDomDocument document;

 document = NetworkFrames::buildTypesList(NetworkFrames::TYPE_CONCRETE_TYPES,
                                          typeName, typesList);

 this->send(client, document.toString());
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::sendTree(int clientId, const QDomDocument & tree)
{
 QDomDocument doc = NetworkFrames::buildTree(tree);
 QTcpSocket * client;

 if(clientId == -1)
  client = NULL;

 else if(clientId < 0 || clientId > this->clientsRequesting.size())
  return;

 else
  client = this->clientsRequesting.at(clientId);

 this->send(client, doc.toString());
}


//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::sendMessage(int clientId, const QString & message)
{
 QDomDocument doc = NetworkFrames::buildMessage(message);
 QTcpSocket * client;

 if(clientId == -1)
  client = NULL;

 else if(clientId < 0 || clientId > this->clientsRequesting.size())
  return;

 else
  client = this->clientsRequesting.at(clientId);

 this->send(client, doc.toString());
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::sendAck(int clientId, int type, bool success)
{
 QDomDocument doc = NetworkFrames::buildAck(success, type);

 QTcpSocket * client;

 if(clientId == -1)
  client = NULL;

 else if(clientId < 0 || clientId > this->clientsRequesting.size())
  return;
 else
  client = this->clientsRequesting.at(clientId);

 this->send(client, doc.toString());
}


//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

int CommServer::getBytesRecieved() const
{
 return this->bytesRecieved;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

int CommServer::getBytesSent() const
{
 return this->bytesSent;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::sendMessage(QTcpSocket * client, const QString & message)
{
 QDomDocument doc = NetworkFrames::buildMessage(message);
 this->send(client, doc.toString());
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::sendError(QTcpSocket * client, const QString & message)
{
 QDomDocument doc = NetworkFrames::buildError(message);
 this->send(client, doc.toString());
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::sendFilesList(QTcpSocket * client, const QString & dirname)
{
 QStringList directories;
 QStringList files;
 bool dotDot;

 QString directory;

 QDomDocument filesList;

 if(dirname.isEmpty())
  directory = this->DEFAULT_PATH;
 else
  directory = dirname;

 directory = QDir(directory).absolutePath();
 directory = QDir::cleanPath(directory);

 if(directory != "/")
  directories << "..";
 if(!this->getDirContent(directory, directories, files))
 {
  this->sendError(client, QString("'%1'␣is␣not␣an␣existing␣directory")
    .arg(directory));
  return;
 }

 QDomDocument doc = NetworkFrames::buildDirContent(directory, directories,
    files);

  this->send(client, doc.toString());
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::sendAck(QTcpSocket * client, bool success, int type)
{
 QDomDocument doc = NetworkFrames::buildAck(success, type);
 this->send(client, doc.toString());
}


//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

bool CommServer::getDirContent(const QString & directory,
                               QStringList & dirsList,
                               QStringList & filesList) const
{
 QStringList list;
 QDir dir(directory);

 dir.setFilter(QDir::Files | QDir::Dirs | QDir::Hidden | QDir::NoSymLinks);
 dir.setSorting(QDir::DirsFirst | QDir::Name);

 if(!dir.exists())
  return false;

 QFileInfoList files = dir.entryInfoList();
 QFileInfoList::iterator it = files.begin();

 while(it != files.end())
 {
  QFileInfo fileInfo = *it;
  QString filename = fileInfo.fileName();

  if (filename != "." && filename != "..")
  {
   if(fileInfo.isDir())
    dirsList << filename;
   else if(filename.endsWith(".xml") || filename.endsWith(".CFcase" ))
    filesList << filename;
  }
  it++;
 }
 return true;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::openFile(QTcpSocket * client, const QString & filename)
{
  if(this->srvSimulation->loadCaseFile(filename))
  {
   this->sendAck(client, true, NetworkFrames::TYPE_OPEN_FILE);
   this->fileOpen = true;
  }
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::runSimulation()
{
 this->simRunning = true;

 this->srvSimulation->start();
}

/****************************************************************************

                                SLOTS

****************************************************************************/

void CommServer::newClient()
{
 QTcpSocket * socket;

 socket = this->server->nextPendingConnection();

 if(socket == NULL)
  socket = this->localSocket->nextPendingConnection();

 // connect useful signals to slots
 connect(socket, SIGNAL(disconnected()), this, SLOT(clientDisconnected()));
 connect(socket, SIGNAL(readyRead()), this, SLOT(newData()));

 std::cout << "A␣new␣client␣is␣connected" << std::endl;

/*

 UNCOMMENT THIS TO SEND THE SERVER STATUS TO THE NEW CLIENT.
 AT THE TIME OF WRITING THESE LINES, THE SIMULATOR WAS CRASHING WHEN GETTING
 XML TREE.

 if(this->fileOpen)
   this->sendAck(socket, true, NetworkFrames::TYPE_OPEN_FILE);

 if(this->simRunning)
  this->sendAck(socket, true, NetworkFrames::TYPE_SIMULATION_RUNNING);

*/

 this->clients[socket] = QDomNode();

 // send a welcome message to the new client
 this->sendMessage(socket,"Welcome␣to␣the␣Client-Server␣project␣!");
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::newData()
{
// which client has sent data ?
 QTcpSocket * socket = qobject_cast<QTcpSocket *>(sender());
```

```
ClientServerXMLParser handler;
QXmlInputSource source;
QXmlSimpleReader reader;
int clientId;

QString frame;
QDataStream in(socket);
in.setVersion(QDataStream::Qt_4_4);

// if the client sends two messages very close in time, it is possible that
// the server never gets the second one.
// So, it is useful to explicitly read the socket until the end is reached.
while(!socket->atEnd())
{
 // if the data size is not known
 if (this->blockSize == 0)
 {
  // if there are at least 2 bytes to read...
  if (socket->bytesAvailable() < (int)sizeof(quint16))
   return;

  // ...we read them
  in >> this->blockSize;
 }

 if (socket->bytesAvailable() < this->blockSize)
  return;

 in >> frame;

 this->bytesRecieved += this->blockSize + (int)sizeof(quint16);

 source.setData(frame);
 reader.setContentHandler(&handler);

 // if parse() returns false, the document is not valid
 if(!reader.parse(source))
 {
  QString error = handler.errorString();

 // if error is empty, the document is not a well-formed XML document
  if(error.isEmpty())
   error = "not␣well-formed␣document.";
  this->sendError(socket, QString("XML␣parsing␣error␣:␣") + error);
 }
 else
 {
  QDomDocument doc = handler.getDomDocument();

  this->clientsRequesting.append(socket);
  clientId = this->clientsRequesting.size() - 1;

  switch(handler.getTypeId())
  {
   // if the client wants the tree
   case NetworkFrames::TYPE_GET_TREE :
   {
    QDomDocument d;
    d.setContent(this->srvSimulation->getTreeXML());
    this->sendTree(clientId, d);
    break;
   }

   // if the client requests to modify node options
```

```
    case NetworkFrames::TYPE_MODIFY_NODE :
     if(!this->fileOpen)
      this->sendError(socket, "No␣case␣file␣loaded␣!");
     else
      ; /// @todo forward to the simulator
     break;

    // if the client requests to add a node
    case NetworkFrames::TYPE_ADD_NODE :
     if(!this->fileOpen)
      this->sendError(socket, "No␣case␣file␣loaded␣!");
     else if(this->simRunning)
      this->sendError(socket, "A␣simulation␣is␣running.");
     else
      ; /// @todo forward to the simulator
     break;

    // if the client requests to rename a node
    case NetworkFrames::TYPE_RENAME_NODE :
     if(!this->fileOpen)
      this->sendError(socket, "No␣case␣file␣loaded␣!");
     else if(this->simRunning)
      this->sendError(socket, "A␣simulation␣is␣running.");
     else
      ; /// @todo forward to the simulator
     break;

    // if the client requests to delete a node
    case NetworkFrames::TYPE_DELETE_NODE :
     if(!this->fileOpen)
      this->sendError(socket, "No␣case␣file␣loaded␣!");
     else if(this->simRunning)
      this->sendError(socket, "A␣simulation␣is␣running.");
     else
      ; /// @todo forward to the simulator
     break;

    // if the client wants the abstract types
    case NetworkFrames::TYPE_GET_ABSTRACT_TYPES :
     this->sendAbstractTypes(socket, handler.get("typeName"));
     break;

    // if the client wants the concrete types
    case NetworkFrames::TYPE_GET_CONCRETE_TYPES :
    {
     QString typeName = handler.get("typeName");
     QStringList typesList = this->srvSimulation->getConcreteTypes(typeName);
     this->sendConcreteTypes(socket, typeName, typesList);
     break;
    }

    // if the client wants the XML files list
    case NetworkFrames::TYPE_OPEN_DIR :
     this->sendFilesList(socket, handler.get("dirname"));
     break;

    // if the client wants to open a file
    case NetworkFrames::TYPE_OPEN_FILE :
     this->openFile(socket, handler.get("filename"));
     break;

    // if the client wants to run the simulation
    case NetworkFrames::TYPE_RUN_SIMULATION :
     if(!this->fileOpen)
      this->sendError(socket, "Please␣open␣a␣case␣file␣before␣running␣a␣"
        "simulation.");
      else if(this->simRunning)
       this->sendError(socket, "The␣simulation␣is␣already␣running.␣"
         "You␣cannot␣run␣it␣twice␣at␣the␣same␣time.");
      else
       this->runSimulation();
      break;

     // if the client wants to shut the server down
     case NetworkFrames::TYPE_SHUTDOWN_SERVER :
      qApp->exit(0);
      break;
    }
   }
  this->blockSize = 0;
 }
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::clientDisconnected()
{
 // which client has been disconnected ?
 QTcpSocket * socket = qobject_cast<QTcpSocket *>(sender());

 this->clients.remove(socket);

 std::cout << "A␣client␣has␣gone␣(" << this->clients.size() << "␣left)"
   << std::endl;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::simulationFinished()
{
 this->simRunning = false;
 this->sendAck(NULL, NetworkFrames::TYPE_RUN_SIMULATION, true);
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::message(const QString & message)
{
 this->sendMessage((QTcpSocket*)NULL, message);
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void CommServer::error(const QString & message)
{
 this->sendError((QTcpSocket*)NULL, message);
}
```

2.2 *NetworkStatistics* class

2.2.1 NetworkStatistics.h

```
#ifndef COOLFluiD_server_NetworkStatistics_h
#define COOLFluiD_server_NetworkStatistics_h

////////////////////////////////////////////////////////////////////////////////

#include <QObject>

class QPushButton;

namespace COOLFluiD
{
 namespace server
 {

////////////////////////////////////////////////////////////////////////////////

  class CommServer;

  /// @brief Shows a message box with sent and recieved bytes.

  /// <b>This class is no longer used and should be deleted.</b>

  /// @author Quentin Gasper.

  class NetworkStatistics : public QObject
  {
   Q_OBJECT

   private:
    /// @brief Communication level
    CommServer * comm;

    /// @brief Button in which user clicks to display information.
    QPushButton * button;

   public:
    /// @brief Constructor.

    /// @param comm Communication level.
    NetworkStatistics(CommServer * comm);

   public slots :
    /// @brief Shows the message box.
    void showStats();
  };

////////////////////////////////////////////////////////////////////////////////

 } // namespace server
} // namespace COOLFluiD

////////////////////////////////////////////////////////////////////////////////

#endif // COOLFluiD_server_NetworkStatistics_h
```

2.2.2 NetworkStatistics.cxx

```
#include <QtGui>

#include "ClientServer/server/CommServer.h"
#include "ClientServer/server/NetworkStatistics.h"

using namespace COOLFluiD::server;

NetworkStatistics::NetworkStatistics(CommServer * comm)
{
 this->button = new QPushButton("Show␣stats");

 connect(this->button, SIGNAL(clicked()), this, SLOT(showStats()));
 this->comm = comm;
 this->button->show();
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void NetworkStatistics::showStats()
{
 QMessageBox::information(NULL, "Stats",
                          QString("Recieved␣bytes␣:␣") +
                            QVariant(this->comm->getBytesRecieved()).toString() +
                          QString("\nSent␣bytes␣:␣") +
                            QVariant(this->comm->getBytesSent()).toString()
                          );
}
```

2.3 *RemoteClientAppender* class

2.3.1 RemoteClientAppender.hh

```
#ifndef COOLFluiD_server_RemoteClientAppender_hh
#define COOLFluiD_server_RemoteClientAppender_hh

#include <string>
#include <iostream>

#include <QObject>

#include <logcpp/Portability.hh>
#include <logcpp/LayoutAppender.hh>

namespace COOLFluiD {
namespace server {

 /// Appends LoggingEvents to the remote client log window.

 class RemoteClientAppender : public QObject, public logcpp::LayoutAppender
 {
  Q_OBJECT

   public:

    RemoteClientAppender(const std::string& name);
    virtual ~RemoteClientAppender();

    virtual bool reopen();
    virtual void close();

   protected:
    virtual void _append(const logcpp::LoggingEvent& event);

   signals:
    void newData(const QString & data);
 };

} // server
} // coolfluid

#endif // COOLFluiD_server_RemoteClientAppender_hh
```

2.3.2 RemoteClientAppender.cxx

```
#include "logcpp/PortabilityImpl.hh"

#include "ClientServer/server/RemoteClientAppender.hh"

namespace COOLFluiD {
namespace server {

    RemoteClientAppender::RemoteClientAppender(const std::string& name) : logcpp::LayoutAppender(name)
    {}

    RemoteClientAppender::~RemoteClientAppender()
    {
        close();
    }

    void RemoteClientAppender::close()
    {
        // empty
    }

    void RemoteClientAppender::_append(const logcpp::LoggingEvent& event)
    {
      emit newData( _getLayout().format(event).c_str() );
    }

    bool RemoteClientAppender::reopen()
    {
        return true;
    }

} // server
} // coolfluid
```

2.4 *ServerSimulation* class

2.4.1 ServerSimulation.h

```
#ifndef COOLFluiD_server_ServerSimulation_h
#define COOLFluiD_server_ServerSimulation_h

////////////////////////////////////////////////////////////////////////////////

#include "Environment/CFEnv.hh"
#include "Framework/Simulator.hh"

#include <QObject>
#include <QThread>

namespace COOLFluiD {

  namespace Framework {  class Simulator;  }

namespace server {

  class ServerOutput;

////////////////////////////////////////////////////////////////////////////////

  /// @brief Interface between CommServer class and the simulator.

  /// @author Quentin Gasper.

  class ServerSimulation : public QThread
  {
   Q_OBJECT

   private:
    /// @brief The simulator
    COOLFluiD::Framework::Simulator * simulator;

    /// @brief If not empty, the name of the case file currently open.
    QString caseFile;

   public:
    /// @brief Constructor.

    /// @param simulatorName Simulator name.
    ServerSimulation(const QString & simulatorName = "Simulator");

    /// @brief Destructor.

    /// Destroys the simulator.
    ~ServerSimulation();

    /// @brief Thread excution.

    /// Runs the simualtion. This method should never be called directly.
    /// Call the method <code>start()</code> (inherited from base class)
    /// instead.
    void run();

    /// @brief Requests to the simulator to load a file.

    /// @param filename Name of the file to open.

    /// @return Returns <code>true</code> if the file was open with success,
    /// otherwise returns <code>false</code>.
    bool loadCaseFile(const QString & filename);

    /// @brief Requests to the XML tree to the simulator.

    /// @return Returns he tree in a QString.
    QString getTreeXML() const;

    /// @brief Gets the concrete types list of an abstract type.

    /// @param abstractType Abstract type.

    /// @return Returns the concrete types list.
    QStringList getConcreteTypes(const QString & abstractType) const;

   public slots:
    /// @brief Slot called when a message has been forwarded from the
    /// simulator.

    /// @param data The message
    void newData(const QString & data);

   signals:
    /// @brief Signal used to send a message.

    /// @param message The message
    void message(const QString & message);

    /// @brief Signal used to send an error message.

    /// @param message The error message
    void error(const QString & message);
  };

////////////////////////////////////////////////////////////////////////////////

} // namespace server
} // namespace COOLFluiD

////////////////////////////////////////////////////////////////////////////////

#endif // COOLFluiD_server_Simulation_h
```

2.4.2 ServerSimulation.cxx

```
#include <QtCore>

#include <exception>

#include "logcpp/PatternLayout.hh"
#include "Common/CFLog.hh"

#include "Environment/CFEnv.hh"
#include "Environment/FactoryRegistry.hh"
#include "Environment/FactoryBase.hh"

#include "Framework/Simulator.hh"

#include "ClientServer/server/RemoteClientAppender.hh"
#include "ClientServer/server/ServerSimulation.h"

using namespace COOLFluiD::Common;
using namespace COOLFluiD::Environment;
using namespace COOLFluiD::Framework;
using namespace COOLFluiD::server;

ServerSimulation::ServerSimulation(const QString & simulatorName)
{
 this->simulator = new Simulator(simulatorName.toStdString());

 logcpp::PatternLayout* f_layout = new logcpp::PatternLayout();
 f_layout->setConversionPattern( "%p␣%m" );

 logcpp::Appender* remote_appender = new RemoteClientAppender(
    "RemoteClientAppender" );
 remote_appender->setLayout(f_layout);

 CFLogger::getInstance().getMainLogger().addAppender( remote_appender );

 connect((RemoteClientAppender*)remote_appender,
          SIGNAL(newData(const QString &)), this,
                 SLOT(newData(const QString &)));
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

ServerSimulation::~ServerSimulation()
{
 delete this->simulator;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void ServerSimulation::run()
{
 try
 {
  if(!this->caseFile.isEmpty())
  {
   emit message("Starting␣the␣simulation");
   this->simulator->simulate();
   emit message("Simulation␣finished");
  }
  else
   emit error("No␣file␣to␣simulate");
 }
 catch ( std::exception& e )
 {
  emit error(e.what());
 }
 catch (...)
 {
  emit error("Unknown␣exception␣thrown␣and␣not␣caught␣!!!\nAborting␣...");
 }
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

bool ServerSimulation::loadCaseFile(const QString & filename)
{
 try
 {
  this->simulator->openCaseFile(filename.toStdString());
  this->caseFile = filename;
  emit message("File␣loaded␣:␣" + filename);

  return true;
 }
 catch ( std::exception& e )
 {
  emit error(e.what());
 }
 catch (...)
 {
  emit error("Unknown␣exception␣thrown␣and␣not␣caught␣!!!\nAborting␣...");
 }
 return false;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QString ServerSimulation::getTreeXML() const
{
 return this->simulator->getTreeXML().c_str();
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void ServerSimulation::newData(const QString & data)
{
 emit message(data);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QStringList ServerSimulation::getConcreteTypes(const QString & abstractType)
  const
{
 QStringList typesList;

 // what if the abstract type does not exist ????

 std::vector< ProviderBase* > registered_providers =
   CFEnv::getInstance().getFactoryRegistry()->
   getFactory(abstractType.toStdString())->getAllProviders();

 for(size_t i = 0; i < registered_providers.size(); ++i)
```

```
   typesList << QString(registered_providers[i]->getProviderName().c_str());

  return typesList;
}
```

2.5 *TreeManager* class

2.5.1 TreeManager.h

```
#ifndef COOLFluiD_server_TreeManager_h
#define COOLFluiD_server_TreeManager_h

////////////////////////////////////////////////////////////////////////////////

#include <QObject>
#include <QDomDocument>
#include <QMutex>

namespace COOLFluiD
{
 namespace server
 {
  class CommServer;

////////////////////////////////////////////////////////////////////////////////

  /// @brief Manages the tree.

  /// <b>This class is no longer used and should be deleted. When deleting
  /// this class, 4 methods in CommServer can be deleted to. They were
  /// created to be called by this class.</b>

  /// @author Quentin Gasper.

  class TreeManager : QObject
  {
   Q_OBJECT

   private:
    /// @brief The tree
    QDomDocument document;

    /// @brief A mutex to prevent concurrent access.
    QMutex mutex;

    /// @brief Communication level
    CommServer * communication;

    /// @brief Gives the node pointed by the path.

    /// @param path The path of the wanted node.

    /// @return Returns the node, or a null node if the path does not exist
    QDomNode getNode(const QString & path);

    /// @brief Requests to the communication level to send the tree.

    /// @param clientId Client id.
    /// @param notify If <code>true</code> a message is sent to clients to
    /// notify that the tree has been changed.
    void sendTree(int clientId, bool notify);

    bool tryLock(int clientId);

   public:

    /// @brief Constructor.

    /// @param communication Communication level
    TreeManager(CommServer * communication);

   public slots:

    /// @brief Slot called to add a node to the tree.

    /// @param clientId Client id
    /// @param path Parent node path
    /// @param name Node name
    /// @param type Node type
    /// @param absType Node abstract type
    void addNode(int clientId, const QString & path, const QString & name,
                 const QString & type, const QString & absType);

    /// @brief Slot called to delete a node from the tree.

    /// @param clientId Client id
    /// @param path Node path
    void deleteNode(int clientId, const QString & path);

    /// @brief Slot called to add a node to the tree.

    /// @param clientId Client id
    void getTree(int clientId);

    /// @brief Slot called to modify a node in the tree.

    /// @param clientId Client id
    /// @param document XML data with the options to modify/add.
    void modifyNode(int clientId, const QDomDocument & document);

    /// @brief Slot called to rename a node in the tree.

    /// @param clientId Client id
    /// @param path Parent node path
    /// @param newName Node new name
    void renameNode(int clientId, const QString & path,
                    const QString & newName);
  };

////////////////////////////////////////////////////////////////////////////////

 }
}

////////////////////////////////////////////////////////////////////////////////

#endif // COOLFluiD_server_TreeManager_h
```

2.5.2 TreeManager.cxx

```
#include <QtCore>
#include <QtXml>

#include "ClientServer/network/NetworkFrames.h"
#include "ClientServer/server/CommServer.h"
#include "ClientServer/server/TreeManager.h"

using namespace COOLFluiD::server;
using namespace COOLFluiD::network;

TreeManager::TreeManager(CommServer * communication)
 : mutex(QMutex::NonRecursive)
{

/// TODO throw exception if pointer is null
 this->communication = communication;

/// TODO exception if error on reading the file

 // Load tree data from XML file

 QFile treeFile("./Tree.xml"); // the tree

 if (treeFile.open(QIODevice::ReadOnly) &&
     this->document.setContent(&treeFile))
  treeFile.close();

 connect(this->communication, SIGNAL(sendTree(int)), this,
         SLOT(getTree(int)));

 connect(this->communication, SIGNAL(deleteNode(int, const QString &)), this,
         SLOT(deleteNode(int, const QString &)));

 connect(this->communication, SIGNAL(modifyNode(int, const QDomDocument &)),
         this, SLOT(modifyNode(int, const QDomDocument &)));

 connect(this->communication, SIGNAL(renameNode(int, const QString &,
         const QString &)), this, SLOT(renameNode(int, const QString &,
         const QString &)));

 connect(this->communication, SIGNAL(addNode(int, const QString &,
         const QString &, const QString &, const QString &)), this,
         SLOT(addNode(int, const QString &, const QString &,
              const QString &, const QString &)));
}

/****************************************************************************

                                   SLOTS

****************************************************************************/

void TreeManager::getTree(int clientId)
{
 this->sendTree(clientId, false);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void TreeManager::addNode(int clientId, const QString & path,
                          const QString & name, const QString & type,
                          const QString & absType)
```

```
{
 QDomNode parent = this->getNode(path);
 QDomElement element = this->document.createElement(name);
 bool success = true;

 if(!this->tryLock(clientId))
  return;

 // if parent is null, the path is invalid...
 if(parent.isNull())
 {
  this->communication->sendError(clientId, "Invalid␣path");
  success = false;
 }

 // ...otherwise, the new node is added to the tree
 // (if a node with the same name does not exist yet)
 else if(parent.namedItem(name).isNull())
 {
  parent.appendChild(element);

  // error if the child was not appended
  success = !parent.namedItem(name).isNull();

  element.setAttribute("tree", "object");
  element.setAttribute("type", type);
  element.setAttribute("abstype", absType);
  element.setAttribute("dynamic", "false");
  element.setAttribute("mode", "basic");

  this->sendTree(-1, true);
 }
 else
 {
  this->communication->sendError(clientId, "A␣node␣with␣the␣same␣parent␣and␣"
    "name␣already␣exists");
  success = true;
 }

 this->communication->sendAck(clientId, NetworkFrames::TYPE_ADD_NODE,
                              success);

 this->mutex.unlock();
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void TreeManager::deleteNode(int clientId, const QString & path)
{
 QDomNode node = this->getNode(path);
 bool success = true;

 if(!this->tryLock(clientId))
  return;

 if(node.isNull())
 {
  this->communication->sendError(clientId, "Invalid␣path");
  success = false;
 }

 // if removeChild() returns a null node, the node could not be deleted
 // (otherwise, the deleted node is returned)
 if(node.parentNode().removeChild(node).isNull())
 {
  this->communication->sendError(clientId, "Unable␣to␣remove␣this␣node");
  success = false;
 }
 else
  this->sendTree(-1, true);

 this->communication->sendAck(clientId, NetworkFrames::TYPE_DELETE_NODE,
                              success);

 this->mutex.unlock();
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void TreeManager::modifyNode(int clientId, const QDomDocument & document)
{
 /// TODO try to make this code smaller
 QDomNodeList childNodes = document.childNodes();
 bool success = true;

 if(!this->tryLock(clientId))
  return;

 for(int i = 0 ; i < childNodes.count() ; i++)
 {
  QDomNode child = childNodes.item(i);
  QDomNamedNodeMap attributes = child.attributes();
  QDomNode nodePath = attributes.namedItem("path");

  if(child.nodeName() == "modOptions" &&
     !nodePath.isNull())
  {
   QDomNode node = this->getNode(nodePath.nodeValue());

   success = !node.isNull();

   if(!success)
    this->communication->sendError(clientId, QString("Node␣'%1'␣not␣found!")
      .arg(nodePath.nodeValue()));
   else
   {
    QDomNodeList options = child.childNodes();

    for(int j = 0 ; j < options.count() ; j++)
    {
     QDomNode element = options.at(j);
     QDomElement option = node.namedItem(element.nodeName()).toElement();
     QDomNamedNodeMap attrs = element.attributes();

     for(int k = 0 ; k < attrs.count() ; k++)
     {
      QDomNode attribute = attrs.item(k);
      option.setAttribute(attribute.nodeName(), attribute.nodeValue());
      success = true;
     }

     QDomNode child = element.childNodes().item(0);
     if(child.isText() && option.firstChild().isText())
     {
      option.firstChild().toText().setData(child.nodeValue());
      success = true;
     }
    }
    }
   }

   else if(child.nodeName() == "addOptions" &&
           !attributes.namedItem("path").isNull())
   {
    QDomNode node = this->getNode(attributes.namedItem("path").nodeValue());

    success = !node.isNull();

    if(success)
    {
     QDomNodeList options = child.childNodes();

     for(int j = 0 ; j < options.count() ; j++)
     {
      QDomNode element = options.at(j);
      QDomElement option = this->document.createElement(element.nodeName());
      QDomNamedNodeMap attrs = element.attributes();

      for(int k = 0 ; k < attrs.count() ; k++)
      {
       QDomNode attribute = attrs.item(k);
       option.setAttribute(attribute.nodeName(), attribute.nodeValue());
       success = true;
      }

      if(element.firstChild().isText())
      {
       QDomText text = this->document.createTextNode(
         element.toElement().text());
       option.appendChild(text);
       success = true;
      }

      node.appendChild(option);
     }
    }
   }
  }

  if(success)
   this->sendTree(-1, true);

  this->communication->sendAck(clientId, NetworkFrames::TYPE_MODIFY_NODE,
                               success);


  this->mutex.unlock();
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void TreeManager::renameNode(int clientId, const QString & path,
                             const QString & newName)
{
 QDomNode node = this->getNode(path);
 QDomNode parent = node.parentNode();
 QDomElement element = this->document.createElement(newName);
 QDomNode tmpNode;

 if(!this->tryLock(clientId))
  return;
 // if parent is null, the path is invalid
 if(parent.isNull())
  return;

 tmpNode = parent.namedItem(newName);

 // !tmpNode.isNull() : check if the name does not exist for that parent
 // tmpNode != node : check if the two nodes are different (if they're equal,
 // it means that the user wants to rename the node to the same name, which
 // is not an error).
 if(!tmpNode.isNull() && tmpNode != node)
 {
  this->communication->sendError(clientId, "A␣node␣with␣the␣same␣parent␣and␣"
    "name␣already␣exists");
  return;
 }

 // finally, rename the node
 node.toElement().setTagName(newName);

 this->sendTree(-1, true);

 this->mutex.unlock();
}

/***************************************************************************

                              PRIVATE METHODS

***************************************************************************/

QDomNode TreeManager::getNode(const QString & path)
{
 QStringList parentsList = path.split("/", QString::SkipEmptyParts);
 QStringList::iterator it = parentsList.begin();
 QDomNode node = this->document;

 while(it != parentsList.end())
 {
  QDomNode tmpNode = node.namedItem(*it);

  // if the node does not exist, error
  if(tmpNode.isNull())
   return tmpNode;

  node = tmpNode;
  it++;
 }

 return node;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void TreeManager::sendTree(int clientId, bool notify)
{
 if(notify)
  this->communication->sendMessage(clientId, "The␣tree␣has␣been␣modified");

 this->communication->sendTree(clientId, this->document);
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
bool TreeManager::tryLock(int clientId)
{
 bool locked = this->mutex.tryLock();

 if(!locked)
  this->communication->sendError(clientId, "Service␣unavailable␣for␣the␣"
    "moment");

 return locked;
}
```

Chapter 3

Network

3.1 *ClientServerXMLParser* class

3.1.1 ClientServerXMLParser.h

```
#ifndef COOLFluiD_network_ClientServerXMLParser_h
#define COOLFluiD_network_ClientServerXMLParser_h

////////////////////////////////////////////////////////////////////////////////

#include <QDomDocument>
#include <QVector>
#include <QHash>

class QDomDocument;
class QXmlDefaultHandler;

////////////////////////////////////////////////////////////////////////////////

namespace COOLFluiD
{
 namespace network
 {

////////////////////////////////////////////////////////////////////////////////

  /// @brief Parses network frames and check their validity.

  /// This class inherits from <code>QXmlDefaultHandler</code> class. <br>
  /// The presence of the data is never checked. If data are found, they are
  /// copied to the data tree. Only presence of attributes is checked, not
  /// their values.<br>

  /// A typical use of this class is (assuming that <code>data</code> is a
  /// <code>QString</code> with XML data to parse):<br>
  /// \code
  /// ClientServerXMLParser handler;
  /// QXmlInputSource source;
  /// QXmlSimpleReader reader;
  ///
  /// source.setData(data);
  /// reader.setContentHandler(&handler);
  ///
  /// if(!reader.parse(source))
/// {
/// QString error = handler.errorString();
///
/// if(error.isEmpty())
///  error = "Not well-formed document.";
///
///  // display error
/// }
/// \endcode

/// See <i>Annexes volume 3 - Network Protocol</i> for futher
/// information about the network protocol.

/// @author Quentin Gasper.

class ClientServerXMLParser : public QXmlDefaultHandler
{
 private:

  /// @brief Indicates wether the frame root respect the protocol.

  /// If <code>true</code>, the frame begins by
  /// <code>NetworkFrames::FRAME_ROOT</code> and has been recognized as a
  /// frame of this protocol which do not means that this frame is valid.
  bool rootOk;

  /// @brief Indicates wether the class is waiting for the type tag.

  /// If <code>true</code>, the class is waiting for the type tag.
  /// Otherwise it is <code>false</code>. According to the network protocol,
  /// this attribute is set to <code>true</code> if the
  /// <code>NetworkFrames::FRAME_ROOT</code> has been found as
  /// the root tag. Once it is <code>true</code>, the next tag to be read
  /// <i>must</i> be the type tag.
  bool waitingForTypeTag;

  /// @brief Indicates wether the type tag has been found.

  /// If <code>true</code>, the type tag has been found and recognized.
  bool typeTagFound;

  /// @brief Index of the current tag type.

  /// This index is given by <code>NetworkFrames::getId(tagName)</code>
  /// where <code>tagName</code> is the read type tag name.
  int index;

  /// @brief Index of (non-)acknowledged frame .

  /// This index is given by <code>NetworkFrames::getId(tagName)</code>
  /// where <code>tagName</code> is the read type tag name.
  int ackType;

  /// @brief If not empty, contains the reason of a parsing failure.

  /// If there was a failure but this string is empty, the frame contains
  /// an XML format error.
  QString errorStr;

  /// @brief Hash map containing read attributes of type name.

  /// If an attribute is missing, the map is cleared. The key is the
  /// attibute name and the value is its value in a string form.
  QHash<QString, QString> attributes;
```

```
    /// @brief Hash map containing, for each type tag having attributes,
    /// all attributes to check.

    /// The key is this index of the tag type (according to
    /// <code>NetworkFrames</code> class) and the value is a list of its
    /// attributes.
    QHash<int, QStringList> mandAttributes;

    /// @brief XML document containing frame data (if any) which means all
    /// XML data between opening and closing type tag.

    /// This document may be empty is there was no data.
    QDomDocument domDocument;

    /// @brief Vector used to check text elements (non-tag data between an
    /// openig and a closing tag) and rebuild XML frame data.

    /// Each time an element is open, it is appended to the vector (thus
    /// the last element in the vector is the last element open). If a
    /// the text element is found while this vector is empty, there is an
    /// error. <br>

    /// When XML frame data are being rebuilt, each open element is appended
    /// to as a child of the last element of the vector or as a child of the
    /// document if the vector is empty. Thus the tree data tree structure is
    /// correctly respected
    QVector<QDomElement> elements;

  public:

    /// @brief Overrrides <code>QXmlDefaultHandler::startDocument()</code>.

    /// This method is called by the reader when the document parsing starts
    /// and initializes all attributes to their default values.

    /// @return Always returns <code>true</code>.
    bool startDocument();

    /// @brief Overrrides <code>QXmlDefaultHandler::endElement()</code>.

    /// This method is called by the reader when an element is closed. The
    /// element is popped from the vector (<code>this->elements</code>).}

    /// @param namespaceURI Namespace URI (<i><b>U</b>niform
    /// <b>R</b>esource <b>I</b>dentifier</i>). This parameter is never used.
    /// @param localName Local name. This parameter is never used.
    /// @param name Element name.

    /// @return Always returns <code>true</code>.
    bool endElement(const QString & namespaceURI, const QString & localName,
                    const QString & name);

    /// @brief Overrrides <code>QXmlDefaultHandler::endElement()</code>.

    /// This method is called by the reader when an element is started. The
    /// method has three different working modes : <ul>
    /// <li>If <code>this->frameOk</code> is <code>false</code>, this method
    /// checks that <code>name</code> parameter corresponds to
    /// <code>NetworkFrames::FRAME_ROOT</code> (if so, sets both
    /// <code>this->frameOk</code> and <code>this->waitingForTypeTag</code>
    /// to <code>true</code>).
    /// <li>If <code>this->frameOk</code> and
    /// <code>this->waitingForTypeTag</code> are both <code>true</code>, this
    /// method checks that <code>name</code> parameter corresponds to a type
    /// tag and, if so, checks that all attributes are present (if these
    /// checks pass, <code>this->typeTagFound</code> is set to
    /// <code>true</code> and <code>this->witingForTypeTag</code> is set to
    /// <code>false</code>).
    /// <li>If <code>this->frameOk</code> and
    /// <code>this->typeTagFound</code> are both <code>true</code>, the
    /// current element is appended to <code>this->document</code>. If one of
    /// these checks fails, the method returns <code>false</code>.

    /// @param namespaceURI Namespace URI (<i><b>U</b>niform
    /// <b>R</b>esource <b>I</b>dentifier</i>). This parameter is never used.
    /// @param localName Local name. This parameter is never used.
    /// @param name Element name.
    /// @param attrs Element name.
    bool startElement(const QString & namespaceURI, const QString & localName,
                      const QString & name, const QXmlAttributes & attrs);

    /// @brief Overrrides <code>QXmlDefaultHandler::characters()</code>.

    /// This method is called by the reader when non-XML characters are
    /// found characters. These characters are valid if there is at one
    /// character that is not a white space (including '<code>\\n</code>')
    /// and if we are "in an element" (an element has been open but not
    /// closed yet). Otherwise, there is an error.

    /// @param ch Read characters

    /// @return Returns <code>true</code> if the characters are valid,
    /// otherwise returns <code>false</code>.
    bool characters (const QString & ch);

    /// @brief Overrrides <code>QXmlDefaultHandler::errorString()</code>.

    /// Gives the last error that occured.

    /// @return Returns the last error
    QString errorString();

    /// @brief Gives the type id of the frame.

    /// @return Returns the type id of the frame or
    /// <code>NetworkFrames::NO_TYPE</code> if the type was not valid.
    int getTypeId() const;

    /// @brief Gives the type id of the ACK/NACK.

    /// @return Returns ths type id of the ACK NACK of
    /// <code>NetworkFrames::NO_TYPE</code> if the frame was not an ACK NACK.
    int getAckType() const;

    /// @brief Gives the XML data.

    /// @return Returns the frame data or an empty document if there was
    /// no data.
    QDomDocument getDomDocument() const;

    /// @brief Gives the value of a specified attribute.

    /// @param attributeName Attribute name.

    /// @return Returns this attribute value or an empty string if the
    /// specified attribute does not exist.
    QString get(QString attributeName) const;
  };

////////////////////////////////////////////////////////////////////////////////
```

```
 } // namespace network
} // namespace COOLFluiD

////////////////////////////////////////////////////////////////////////////////

#endif // COOLFluiD_network_ClientServerXMLParser_h
```

3.1.2 ClientServerXMLParser.cxx

```
#include <iostream>
#include <QtCore>
#include <QtXml>

#include "ClientServer/network/NetworkFrames.h"
#include "ClientServer/network/ClientServerXMLParser.h"

using namespace COOLFluiD::network;

bool ClientServerXMLParser::startDocument()
{
 this->rootOk = false;
 this->waitingForTypeTag = false;
 this->typeTagFound = false;
 this->index = NetworkFrames::NO_TYPE;

 this->ackType = NetworkFrames::NO_TYPE;

 this->mandAttributes[NetworkFrames::TYPE_MESSAGE] << "value";

 this->mandAttributes[NetworkFrames::TYPE_ERROR] << "value";

 this->mandAttributes[NetworkFrames::TYPE_ADD_NODE] << "path";
 this->mandAttributes[NetworkFrames::TYPE_ADD_NODE] << "name";
 this->mandAttributes[NetworkFrames::TYPE_ADD_NODE] << "type";
 this->mandAttributes[NetworkFrames::TYPE_ADD_NODE] << "absType";

 this->mandAttributes[NetworkFrames::TYPE_RENAME_NODE] << "path";
 this->mandAttributes[NetworkFrames::TYPE_RENAME_NODE] << "name";

 this->mandAttributes[NetworkFrames::TYPE_DELETE_NODE] << "path";

 this->mandAttributes[NetworkFrames::TYPE_GET_ABSTRACT_TYPES] << "typeName";

 this->mandAttributes[NetworkFrames::TYPE_GET_CONCRETE_TYPES] << "typeName";

 this->mandAttributes[NetworkFrames::TYPE_ABSTRACT_TYPES] << "typesList";

 this->mandAttributes[NetworkFrames::TYPE_CONCRETE_TYPES] << "typesList";

 this->mandAttributes[NetworkFrames::TYPE_FILES_LIST] << "filesList";

 this->mandAttributes[NetworkFrames::TYPE_OPEN_FILE] << "filename";

 this->mandAttributes[NetworkFrames::TYPE_ACK] << "type";

 this->mandAttributes[NetworkFrames::TYPE_NACK] << "type";

 this->mandAttributes[NetworkFrames::TYPE_OPEN_DIR] << "dirname";

 this->mandAttributes[NetworkFrames::TYPE_DIR_CONTENT] << "dirs";
 this->mandAttributes[NetworkFrames::TYPE_DIR_CONTENT] << "files";
 this->mandAttributes[NetworkFrames::TYPE_DIR_CONTENT] << "path";

 return true;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

bool ClientServerXMLParser::endElement(const QString &, const QString &,
                                       const QString & name)
{
 if(name == NetworkFrames::getType(NetworkFrames::FRAME_ROOT))
 {
  this->rootOk = false;
  this->waitingForTypeTag = false;
  this->typeTagFound = false;
 }
 else if(this->typeTagFound && !this->elements.empty())
 {
  this->elements.pop_back();
 }

 return true;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

bool ClientServerXMLParser::startElement(const QString &, const QString &,
                                         const QString & name,
                                         const QXmlAttributes & attrs)
{

 this->errorStr.clear();

 if(this->rootOk)
 {
  if(this->waitingForTypeTag)
  {
   this->index = NetworkFrames::getId(name);
   if(this->index == -1)
    this->errorStr = QString("'%1'␣is␣an␣unknown␣type").arg(name);

   // check the presence of all attributes
   QStringList list = this->mandAttributes.value(this->index);
   QList<QString>::iterator  it = list.begin();

   while(it != list.end() && this->errorStr.isEmpty())
   {
    QString attName = *it;

    if(attrs.index(attName) == -1)
    {
     this->errorStr = QString("'%1'␣attribute␣is␣missing").arg(attName);
    }
    else
     this->attributes[attName] = attrs.value(attName);
    it++;
   }

   // particular case if it is a (n)ack frame : the type attribute has to be
   // converted to an int (the correspondig type id)
   if((this->index == NetworkFrames::TYPE_ACK ||
       this->index == NetworkFrames::TYPE_NACK) &&
       this->attributes.contains("type"))
   {
    this->ackType = NetworkFrames::getId(attrs.value("type"));

    if(this->ackType == NetworkFrames::NO_TYPE)
     this->errorStr = "unknown␣frame␣type␣for␣ack";
   }

   // if there is an error, the list is cleared
   if(!this->errorStr.isEmpty())
    this->attributes.clear();
   this->typeTagFound = true;
   this->waitingForTypeTag = false;
  }
  else
  {
   QDomElement elt = this->domDocument.createElement(name);

   for(int i = 0 ; i < attrs.count() ; i++)
   {
    elt.setAttribute(attrs.qName(i), attrs.value(i));
   }

   // if the vector is empty, elt is a child of the tree
   // otherwise it is a child of the last element started
   if(this->elements.empty())
    this->domDocument.appendChild(elt);
   else
    this->elements.back().appendChild(elt);

   this->elements.push_back(elt);
  }
 }
 else if(name == NetworkFrames::getType(NetworkFrames::FRAME_ROOT))
 {
  this->rootOk = true;
  this->waitingForTypeTag = true;
 }

 return this->errorStr.isEmpty();
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

bool ClientServerXMLParser::characters (const QString & ch)
{
 // spaces at the beginning of each line are interpreted as
 // text but they are not. So if the string only contains
 // spaces characters (including '\n'), which means that the trimmmed
 // version is empty, there's nothing to do
 if(ch.trimmed().isEmpty())
  return true;

 // if we are "in" an element
 if(!this->elements.isEmpty())
 {

  QDomText text = this->domDocument.createTextNode(ch);
  this->elements.back().appendChild(text);
  return true;
 }

 this->errorStr = "illegal␣text␣was␣found";
 return false;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QString ClientServerXMLParser::errorString()
{
 return this->errorStr;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

int ClientServerXMLParser::getTypeId() const
{
 return this->index;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument ClientServerXMLParser::getDomDocument() const
{
 return this->domDocument;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

int ClientServerXMLParser::getAckType() const
{
 return this->ackType;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QString ClientServerXMLParser::get(QString attributeName) const
{
 return this->attributes[attributeName];
}
```

3.2 *NetworkException* class

3.2.1 NetworkException.h

```
#ifndef COOLFluiD_network_NetworkException_h
#define COOLFluiD_network_NetworkException_h

////////////////////////////////////////////////////////////////////////////////

#include <QString>

namespace COOLFluiD
{
 namespace network
 {

////////////////////////////////////////////////////////////////////////////////

   /// @brief Exception thrown when the server can not open its socket.

   /// @author Quentin Gasper.

   class NetworkException
   {
    private:

     /// @brief Exception message.
     QString message;
    public:

     /// @brief Constructor.

     /// If the provided message is empty, the string "Network error" is used
     /// has message.

     /// @param message Exception message. May be empty.

     NetworkException(QString message = QString())
     {
      if(message.isEmpty())
       this->message = "Network␣error";
      else
       this->message = message;
     }

     /// @brief Gives the exception message.

     /// @return Returns the exception message.
     QString getMessage() const
     {
      return this->message;
     }

   };

////////////////////////////////////////////////////////////////////////////////

 } //namespace Network
} // namespace COOLFLuiD

////////////////////////////////////////////////////////////////////////////////

#endif // COOLFluiD_network_NetworkException_h
```

3.3 *NetworkFrames* class

3.3.1 NetworkFrames.h

```
#ifndef COOLFluiD_network_NetWorkConstants_h
#define COOLFluiD_network_NetWorkConstants_h

////////////////////////////////////////////////////////////////////////////////

#include <QHash>

namespace COOLFluiD
{
 namespace network
 {

////////////////////////////////////////////////////////////////////////////////

  /// This class is used to build network frames.

  /// See <i>Annexes volume 3 - Network Protocol</i> for futher
  /// information about the network protocol. The documentation of this class
  /// contains many references to this annex.

  /// @author Quentin Gasper.

  class NetworkFrames
  {

   private :

    /// @brief Hash map with all types.

    /// The key is the type id defined by one the public constant interger
    /// attributes of this class. The value is the type name for this id. All
    /// types ids have a name except <code>NO_TYPE</code>.
    static QHash<int, QString> types;

    /// @brief Builds the types hash map.

    /// This function builds the hash map at most once during runtime. If it
    /// is called a second time, it returns without doing anything.
    static void buildTypes();

    /// @brief Builds a Unix-like path string to the given node.

    /// The string begins with a slash followed by the root node name and
    /// all given node parent nodes names, seperated by slashed (like in a
    /// Unix path).

    /// @param node Node from which the path will be extracted.
    /// @param addName If <code>true</code>, the node name is appended to the
    /// path

    /// @return Returns the built strings.
    static QString getNodePath(const QDomNode & node, bool addName);

    /// @brief Builds the skeleton of a frame.

    /// This function builds an XML document with two nodes : the frame root
    /// and the type node (with arguments, if any). The <code>typeNode</code>
    /// parameter is used to store the type node, so that the calling code
    /// can eventually append some additional data.
    /// @param type Type id of the frame.
    /// @param attrs Type node arguments. May be empty
    /// @param typeNode Node where the type node will be stored.

    /// @return Returns the built XML document.
    static QDomDocument buildFrame(int type,
                                   const QHash<QString, QString> & attrs,
                                   QDomElement & typeNode);

    /// @brief Builds the skeleton of a frame.

    /// This an overloaded function, provided for convinience. This function
    /// can be used to build skeletons for frames that do not need to add
    /// data after the type node. So getting this node is useless. This
    /// function calls <code>buildFrame(int, const QHash<QString, QString> &,
    /// QDomElement &)</code>.

    /// @param type Type id of the frame.
    /// @param attrs Type node arguments. May be empty

    /// @return Returns the built XML document.
    static QDomDocument buildFrame(int type,
                                   const QHash<QString, QString> & attrs =
                                   (QHash<QString, QString>()));

   public :

    /// @brief Type id used to indicate that a frame has no type (i.e. it
    /// does not respect the protocol).
    static const int NO_TYPE = -1;

    /// @brief Type id for the frame root.
    static const int FRAME_ROOT = 0;

    /// @brief Type id for "Error" frame.
    static const int TYPE_ERROR = 1;

    /// @brief Type id for "Get tree" frame.
    static const int TYPE_GET_TREE = 2;

    /// @brief Type id for "Message" frame.
    static const int TYPE_MESSAGE = 3;

    /// @brief Type id for "Modify node" frame.
    static const int TYPE_MODIFY_NODE = 4;

    /// @brief Type id for "Tree" frame.
    static const int TYPE_TREE = 5;

    /// @brief Type id for "Add node" frame.
    static const int TYPE_ADD_NODE = 6;

    /// @brief Type id for "Delete node" frame.
    static const int TYPE_DELETE_NODE = 7;

    /// @brief Type id for "Rename" frame.
    static const int TYPE_RENAME_NODE = 8;

    /// @brief Type id for "Get abstract types" frame.
    static const int TYPE_GET_ABSTRACT_TYPES = 9;

    /// @brief Type id for "Get concrete types" frame.
    static const int TYPE_GET_CONCRETE_TYPES = 10;

    /// @brief Type id for "Abstract types" frame.
    static const int TYPE_ABSTRACT_TYPES = 11;

    /// @brief Type id for "Concrete types" frame.
    static const int TYPE_CONCRETE_TYPES = 12;

    /// @brief Type id for "Get files list" frame.
    static const int TYPE_GET_FILES_LIST = 13;

    /// @brief Type id for "Files list" frame.
    static const int TYPE_FILES_LIST = 14;

    /// @brief Type id for "Open file" frame.
    static const int TYPE_OPEN_FILE = 15;

    /// @brief Type id for "Run simulation" frame.
    static const int TYPE_RUN_SIMULATION = 16;

    /// @brief Type id for "ACK" frame.
    static const int TYPE_ACK = 17;

    /// @brief Type id for "NACK" frame.
    static const int TYPE_NACK = 18;

    /// @brief Type id for "Shutdown server" frame.
    static const int TYPE_SHUTDOWN_SERVER = 19;

    /// @brief Type id for "Simulation running" frame.
    static const int TYPE_SIMULATION_RUNNING = 20;

    /// @brief Type id for "Open directory" frame.
    static const int TYPE_OPEN_DIR = 21;

    /// @brief Type id for "Directory contents" frame.
    static const int TYPE_DIR_CONTENT = 22;

    /// @brief Gives the type name for a given type id.

    /// @param id The type id.

    /// @return Returns the type name for the provided type id, or an empty
    /// string if the type id does not exist or if it is
    /// <code>NetworkFrames::NO_TYPE</code>.
    static QString getType(int id);

    /// @brief Checks if a type id is valid.

    /// A type id is valid if it exists and is it has a type name associated.
    /// Thus <code>NetworkFrames::NO_TYPE</code> will not be considered as
    /// valid by this function.

    /// @param id The type id to check.

    /// @return Returns <code>true</code> if the type id is valid,
    /// otherwise returns <code>false</code>.
    static bool isValid(int id);

    /// @brief Gives the type id of a given type name.

    /// @param type The type name.

    /// @return Returns the type id corresponding to the given type name, or
    /// <code>NetworkFrames::NO_TYPE</code> if the type name is unknown.
    static int getId(const QString & type);

    /// @brief Builds an action frame.
```

```
/// An action frame is a frame of which the type tag does not have any
/// attributes but may have data.

/// @param type Action type id.
/// @param data Frame data. May be empty.

/// @return Returns the built frame, or an empty document if the provided
/// type is not valid.
static QDomDocument buildAction(int type, const QDomDocument & data);

/// @brief Builds a "Simple get" frame.

/// A "Simple get" frame is a frame of which the type tag does not
/// have any attributes and has no data. Only the folowing types
/// considered as valid and are accepted:
/// <ul>
///  <li><code>NetworkFrames::TYPE_GET_TREE</code>
///  <li><code>NetworkFrames::TYPE_GET_FILES_LIST</code>
///  <li><code>NetworkFrames::TYPE_RUN_SIMULATION</code>
///  <li><code>NetworkFrames::TYPE_SIMULATION_RUNNING</code>
///  <li><code>NetworkFrames::TYPE_SHUTDOWN_SERVER</code>
/// </ul>

/// @param type Action type id.

/// @return Returns the built frame, or an empty document if the provided
/// type is not valid.
static QDomDocument buildSimpleGetFrame(int type);

/// @brief Builds a "Message" frame.

/// @param message The message.

/// @return Return the built frame in an XML document.
static QDomDocument buildMessage(const QString & message);

/// @brief Builds a "Error" frame.

/// @param error The error message.

/// @return Return the built frame in an XML document.
static QDomDocument buildError(const QString & error);

/// @brief Builds a "Tree" frame.

/// @param tree The tree.

/// @return Return the built frame in an XML document.
static QDomDocument buildTree(const QDomDocument & tree);

/// @brief Builds a "Add node" frame.

/// @param node The new node. Its parents reprensent the path of the
/// new node in the tree.
/// @param type Concrete type name for the new node.
/// @param absType Abstract type name for the new node.

/// @return Return the built frame in an XML document.
static QDomDocument buildAddNode(const QDomNode & node,
                                 const QString & type,
                                 const QString & absType);

/// @brief Builds a "Delete node" frame.
```

```
    /// @param node The new node to delete. Its parents reprensent the path
    /// of this node in the tree.

    /// @return Return the built frame in an XML document.
    static QDomDocument buildDeleteNode(const QDomNode & node);

    /// @brief Builds a "Rename node" frame.

    /// @param node The new node to rename. Its parents reprensent the path
    /// of this node in the tree.
    /// @param newName The node new name.

    /// @return Return the built frame in an XML document.
    static QDomDocument buildRenameNode(const QDomNode & node,
                                        const QString & newName);

    /// @brief Builds a "Get abstract types" or a "Get concrete types" frame.

    /// @param type Type of the frame. Only two types are accepted :
    /// <code>TYPE_GET_ABSTRACT_TYPES</code> and
    /// <code>TYPE_GET_CONCRETE_TYPES</code>.
    /// @param typeName If <code>type</code> is
    /// <code>TYPE_GET_ABSTRACT_TYPES</code>, this parameter is the type name
    /// from which the abstract types list is wanted. If <code>type</code> is
    /// <code>TYPE_GET_CONCRETE_TYPES</code>, this parameter is the abstract
    /// type name from which the concrete types list is wanted.

    /// @return Return the built frame in an XML document or an empty
    /// document if the <code>type</code> is from
    /// <code>TYPE_GET_ABSTRACT_TYPES</code> and
    /// <code>TYPE_GET_CONCRETE_TYPES</code>.
    static QDomDocument buildGetTypes(int type, const QString & typeName);

    /// @brief Builds an "Abstract types list" or a "Concrete types list"
    /// frame.

    /// @param type Type of the frame. Only two types are accepted :
    /// <code>TYPE_ABSTRACT_TYPES</code> and
    /// <code>TYPE_CONCRETE_TYPES</code>.
    /// @param typeName If <code>type</code> is
    /// <code>TYPE_ABSTRACT_TYPES</code>, this parameter is the type name
    /// from which the abstract types list comes. If <code>type</code> is
    /// <code>TYPE_CONCRETE_TYPES</code>, this parameter is the abstract
    /// type name from which the concrete types list comes.
    /// @param typesList If <code>type</code> is
    /// <code>TYPE_ABSTRACT_TYPES</code>, this parameter is an abstract types
    /// list. If <code>type</code> is <code>TYPE_CONCRETE_TYPES</code>, this
    /// parameter is a concrete types list.

    /// @return Return the built frame in an XML document or an empty
    /// document if the <code>type</code> is from
    /// <code>TYPE_ABSTRACT_TYPES</code>
    /// and <code>TYPE_CONCRETE_TYPES</code>.
    static QDomDocument buildTypesList(int type, const QString & typeName,
                                       const QStringList & typesList);

    /// @brief Builds a "Files list" frame.

    /// @param filesList The files list.

    /// @return Return the built frame in an XML document.
    static QDomDocument buildFilesList(const QStringList & filesList);

    /// @brief Builds a "Open file" frame.
    /// @param fileName The file to open.

    /// @return Return the built frame in an XML document.
    static QDomDocument buildOpenFile(const QString & fileName);

    /// @brief Builds an "ACK"  or a "NACK" frame.

    /// @param success If <code>true</code> an "ACK" frame is built.
    /// Otherwise, a "NACK" frame is built.
    /// @param type Type of the frame to ACK/NACK.

    /// @return Return the built frame in an XML document or an empty
    /// document if the type does not exist.
    static QDomDocument buildAck(bool success, int type);

    /// @brief Builds a "Open directory" frame.

    /// @param dirname The directory to open and read.

    /// @return Return the built frame in an XML document.
    static QDomDocument buildOpenDir(const QString & dirname);

    /// @brief Builds a "Directory contents" frame.

    /// @param path The directory from which the contents are taken.
    /// @param dirs List of directories.
    /// @param files List of files.

    /// @return Return the built frame in an XML document.
    static QDomDocument buildDirContent(const QString & path,
                                        const QStringList & dirs,
                                        const QStringList & files);

  };

////////////////////////////////////////////////////////////////////////////////

 } // namespace network
} // namespace COOLFluiD

////////////////////////////////////////////////////////////////////////////////

#endif // COOLFluiD_network_NetWorkConstants_h
```

3.3.2 NetworkFrames.cxx

```cpp
#include <QtXml>

#include "ClientServer/network/NetworkFrames.h"

using namespace COOLFluiD::network;

// the following is necessary to allocate the static members

const int NetworkFrames::FRAME_ROOT;
const int NetworkFrames::TYPE_ERROR;
const int NetworkFrames::TYPE_GET_TREE;
const int NetworkFrames::TYPE_MESSAGE;
const int NetworkFrames::TYPE_MODIFY_NODE;
const int NetworkFrames::TYPE_TREE;
const int NetworkFrames::TYPE_ADD_NODE;
const int NetworkFrames::TYPE_DELETE_NODE;
const int NetworkFrames::TYPE_RENAME_NODE;
const int NetworkFrames::NO_TYPE;
const int NetworkFrames::TYPE_GET_ABSTRACT_TYPES;
const int NetworkFrames::TYPE_GET_CONCRETE_TYPES;
const int NetworkFrames::TYPE_ABSTRACT_TYPES;
const int NetworkFrames::TYPE_CONCRETE_TYPES;
const int NetworkFrames::TYPE_GET_FILES_LIST;
const int NetworkFrames::TYPE_FILES_LIST;
const int NetworkFrames::TYPE_OPEN_FILE;
const int NetworkFrames::TYPE_RUN_SIMULATION;
const int NetworkFrames::TYPE_ACK;
const int NetworkFrames::TYPE_NACK;
const int NetworkFrames::TYPE_SHUTDOWN_SERVER;
const int NetworkFrames::TYPE_SIMULATION_RUNNING;
const int NetworkFrames::TYPE_OPEN_DIR;
const int NetworkFrames::TYPE_DIR_CONTENT;

QHash<int, QString> NetworkFrames::types;

QString NetworkFrames::getType(int id)
{
 NetworkFrames::buildTypes();

 if(NetworkFrames::isValid(id))
  return NetworkFrames::types[id];

 return QString();
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

bool NetworkFrames::isValid(int id)
{
 NetworkFrames::buildTypes();

 return NetworkFrames::types.contains(id);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument NetworkFrames::buildAction(int type, const QDomDocument & data)
{
 QDomElement typeNode;
 QDomNodeList childNodes = data.childNodes();
 QDomDocument doc = buildFrame(type, QHash<QString, QString>(), typeNode);

 for(int i = 0 ; i < childNodes.count() ; i++)
  typeNode.appendChild(doc.importNode(childNodes.item(i), true));

 return doc;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

int NetworkFrames::getId(const QString & type)
{
 QHash<int, QString>::iterator it;

 NetworkFrames::buildTypes();

 it = NetworkFrames::types.begin();

 while(it != NetworkFrames::types.end())
 {
  if(it.value() == type)
   return it.key();
  it++;
 }

 return NetworkFrames::NO_TYPE;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument NetworkFrames::buildSimpleGetFrame(int type)
{
 switch(type)
 {
  case TYPE_GET_TREE :
  case TYPE_GET_FILES_LIST :
  case TYPE_RUN_SIMULATION :
  case TYPE_SIMULATION_RUNNING :
  case TYPE_SHUTDOWN_SERVER :
   break;
  default :
   return QDomDocument();
 }

 return buildFrame(type);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument NetworkFrames::buildMessage(const QString & message)
{
 QHash<QString, QString> attrs;

 attrs["value"] = message;

 return buildFrame(TYPE_MESSAGE, attrs);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument NetworkFrames::buildError(const QString & error)
```

```
{
 QHash< QString , QString > attrs;

 attrs["value"] = error;

 return buildFrame(TYPE_ERROR, attrs);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument NetworkFrames::buildTree(const QDomDocument & tree)
{
 // if the first node is the xml tag (<?xml...), it's removed (the
 // second node becomes the first one) : there's no need to show it.
 if(tree.firstChild().nodeName().compare("xml") == 0)
 {
  QDomDocument document = tree.cloneNode(true).toDocument();
  QDomNodeList childNodes = document.childNodes();
  document.replaceChild(childNodes.item(1), childNodes.item(0));

  return buildAction(TYPE_TREE, document);
 }

 return buildAction(TYPE_TREE, tree);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument NetworkFrames::buildAddNode(const QDomNode & node,
                                         const QString & type,
                                         const QString & absType)
{
 QHash< QString , QString > attrs;

 attrs["path"] = getNodePath(node, false);
 attrs["name"] = node.nodeName();
 attrs["type"] = type;
 attrs["absType"] = absType;

 return buildFrame(TYPE_ADD_NODE, attrs);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument NetworkFrames::buildDeleteNode(const QDomNode & node)
{
 QHash< QString , QString > attrs;

 attrs["path"] = getNodePath(node, true);

 return buildFrame(TYPE_DELETE_NODE, attrs);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument NetworkFrames::buildRenameNode(const QDomNode & node,
                                            const QString & newName)
{
 QHash< QString , QString > attrs;

 attrs["path"] = getNodePath(node, true);
 attrs["name"] = newName;

 return buildFrame(TYPE_RENAME_NODE, attrs);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument NetworkFrames::buildGetTypes(int type, const QString & typeName)
{
 QHash<QString, QString> attrs;

 if(type != TYPE_GET_ABSTRACT_TYPES && type != TYPE_GET_CONCRETE_TYPES)
  return QDomDocument();

 attrs["typeName"] = typeName;

 return buildFrame(type, attrs);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument NetworkFrames::buildTypesList(int type,
                                           const QString & typeName,
                                           const QStringList & typesList)
{
 QHash<QString, QString> attrs;

 if(type != TYPE_ABSTRACT_TYPES && type != TYPE_CONCRETE_TYPES)
  return QDomDocument();

 attrs["typeName"] = typeName;
 attrs["typesList"] = typesList.join(",␣");

 return buildFrame(type, attrs);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument NetworkFrames::buildFilesList(const QStringList & filesList)
{
 QHash<QString, QString> attrs;

 attrs["filesList"] = filesList.join("*");

 return buildFrame(TYPE_FILES_LIST, attrs);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument NetworkFrames::buildOpenFile(const QString & fileName)
{

 QHash<QString, QString> attrs;

 attrs["filename"] = fileName;

 return buildFrame(TYPE_OPEN_FILE, attrs);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
QDomDocument NetworkFrames::buildAck(bool success, int type)
{
 QHash<QString, QString> attrs;

 NetworkFrames::buildTypes();

 if(!NetworkFrames::isValid(type))
  return QDomDocument();

 attrs["type"] = NetworkFrames::getType(type);

  if(success)
   return buildFrame(TYPE_ACK, attrs);
  else
   return buildFrame(TYPE_NACK, attrs);
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument NetworkFrames::buildOpenDir(const QString & dirname)
{
 QHash<QString, QString> attrs;

 attrs["dirname"] = dirname;

 return NetworkFrames::buildFrame(NetworkFrames::TYPE_OPEN_DIR, attrs);
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument NetworkFrames::buildDirContent(const QString & path,
                                            const QStringList & dirs,
                                            const QStringList & files)
{
 QHash<QString, QString> attrs;

 attrs["path"] = path;
 attrs["dirs"] = dirs.join("*");
 attrs["files"] = files.join("*");

 return NetworkFrames::buildFrame(NetworkFrames::TYPE_DIR_CONTENT, attrs);
}
/***************************************************************************

                             PRIVATE FUNCTIONS

***************************************************************************/

void NetworkFrames::buildTypes()
{
 static bool mapBuilt = false;

 if(mapBuilt) // if the map has already been built...
  return;     // the function returns (there no need to build it again)

 NetworkFrames::types[ FRAME_ROOT ] = "ClientServerXML";
 NetworkFrames::types[ TYPE_ERROR ] = "error";
 NetworkFrames::types[ TYPE_GET_TREE ] = "getTree";
 NetworkFrames::types[ TYPE_MESSAGE ] = "message";
 NetworkFrames::types[ TYPE_MODIFY_NODE ] = "modifyNode";
 NetworkFrames::types[ TYPE_TREE ] = "tree";
 NetworkFrames::types[ TYPE_ADD_NODE ] = "addNode";
 NetworkFrames::types[ TYPE_DELETE_NODE ] = "deleteNode";
 NetworkFrames::types[ TYPE_RENAME_NODE ] = "renameNode";
 NetworkFrames::types[ TYPE_GET_ABSTRACT_TYPES ] = "getAbstractTypes";
 NetworkFrames::types[ TYPE_GET_CONCRETE_TYPES ] = "getConcreteTypes";
 NetworkFrames::types[ TYPE_ABSTRACT_TYPES ] = "abstractTypes";
 NetworkFrames::types[ TYPE_CONCRETE_TYPES ] = "concreteTypes";
 NetworkFrames::types[ TYPE_GET_FILES_LIST ] = "getFilesList";
 NetworkFrames::types[ TYPE_FILES_LIST ] = "filesList";
 NetworkFrames::types[ TYPE_OPEN_FILE ] = "openFile";
 NetworkFrames::types[ TYPE_RUN_SIMULATION ] = "runSimulation";
 NetworkFrames::types[ TYPE_ACK ] = "ack";
 NetworkFrames::types[ TYPE_NACK ] = "nack";
 NetworkFrames::types[ TYPE_SHUTDOWN_SERVER ] = "shutdownServer";
 NetworkFrames::types[ TYPE_SIMULATION_RUNNING ] = "simulationRunning";
 NetworkFrames::types[ TYPE_OPEN_DIR ] = "openDir";
 NetworkFrames::types[ TYPE_DIR_CONTENT ] = "dirContent";

 mapBuilt = true; // now the map is built
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QString NetworkFrames::getNodePath(const QDomNode & node, bool addName)
{
 QDomNode parentNode = node.parentNode();

 if(parentNode.isNull()) // if the node has no parent
  return QString();

 QString path = getNodePath(parentNode, true) + QString("/");

 if(addName)
  path += node.nodeName();

 return path;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument NetworkFrames::buildFrame(int type,
                                       const QHash<QString, QString> & attrs)
{
 QDomElement node;
 return buildFrame(type, attrs, node);
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument NetworkFrames::buildFrame(int type,
                                       const QHash<QString, QString> & attrs,
                                       QDomElement & typeNode)
{
 QDomDocument doc;
 QDomElement root;
 QHash<QString, QString>::const_iterator it = attrs.begin();

 if(NetworkFrames::types.count() == 0)
  NetworkFrames::buildTypes();

 if(!NetworkFrames::isValid(type))
  return QDomDocument();
 // set the frame root
 root = doc.createElement(getType(FRAME_ROOT));
 typeNode = doc.createElement(getType(type));

 while(it != attrs.end())
 {
  typeNode.setAttribute(it.key(), it.value());
  it++;
 }

 doc.appendChild(root);
 root.appendChild(typeNode);

 return doc;
}
```

Chapter 4

Treeview

4.1 *TObjectProperties* class

4.1.1 TObjectProperties.h

```
#ifndef COOLFluiD_treeview_TObjectProperties_h
#define COOLFluiD_treeview_TObjectProperties_h

////////////////////////////////////////////////////////////////////////////////

namespace COOLFluiD
{
 namespace treeview
 {

////////////////////////////////////////////////////////////////////////////////

  /// @brief Handles object properties.

  struct TObjectProperties
  {
   public :

    /// @brief Object type name.
    QString type;

    /// @brief Object abstract type name.
    QString absType;

    /// @brief If <code>true</code>, the object is basic, otherwise it is
    /// advanced.
    bool basic;

    /// @brief If <code>true</code>, the object is static, otherwise it is
    /// dynamic.
    bool dynamic;

    /// @brief Constructor.

    /// Provided for convinience.

    /// @param type Object type name.
    /// @param absType Object abstract type name.
    /// @param basic If <code>true</code>, the object is basic, otherwise
    /// it is advanced.
    /// @param dynamic If <code>true</code>, the object is dynamic, otherwise
    /// it is not.
    TObjectProperties(const QString & type = QString(),
                      const QString & absType = QString(),
                      bool basic = false,
                      bool dynamic = false)
    {
     this->type = type;
     this->absType = absType;
     this->basic = basic;
     this->dynamic = dynamic;
    }
  };

//////////////////////////////////////////////////////////////////////////////

 } // namespace treeview
} // namespace COOLFluiD

//////////////////////////////////////////////////////////////////////////////

#endif // COOLFluiD_treeview_TObjectProperties_h
```

4.2 *TreeItem* class

4.2.1 TreeItem.h

```
#ifndef COOLFluiD_treeview_TreeItem_h
#define COOLFluiD_treeview_TreeItem_h

//////////////////////////////////////////////////////////////////////////////

#include <QHash>

class QDomNode;

namespace COOLFluiD
{
 namespace treeview
 {

  ////////////////////////////////////////////////////////////////////////////

  /// @brief TreeItem class represents an item of the tree model.

  /// @author Quentin Gasper.

  class TreeItem
  {
   private:
    /// @brief The node represented by this item.
    QDomNode domNode;

    /// @brief Children of this item.

    /// The key is an integer representing the row number of the associated
    /// child (see <code>rowNumber</code> attribute). The value is a
    /// pointer to this child.
    QHash<int,TreeItem *> childItems;

    /// @brief A pointer to the parent item.

    /// This pointer may be null, if this item is the root of the tree.
    TreeItem * parentItem;

    /// @brief Number of this item in its parent children.

    /// This number corresponds to the emplacement of this item in the
    /// <code>QDomNodeList</code> return by
    /// <code>parent->childNodes()</code>.
    int rowNumber;

   public:
    /// @brief Consructor.

    /// @param node The node this item represents.
    /// @param row Number of this item in the parent children.
    /// @param parent A pointer to the parent. May be null of this item
    /// is the root of the tree.
    TreeItem(QDomNode & node, int row, TreeItem * parent = NULL);

    /// @brief Destructor.

    /// Free all allocated memory.
    ~TreeItem();

    /// @brief Gives the child having the given row number.
    /// @param i Row number of the wanted child
    /// @return Returns the corresponding TreeItem, or a null pointer
    /// if the row number is not valid.
    TreeItem * getChild(int i);

    /// @brief Gives the parent item of this item.

    /// @return Returns the parent item.
    TreeItem * getParentItem();

    /// @brief Gives the node of this item.

    /// @return Returns the node.
    QDomNode getDomNode() const;

    /// @brief Gives the row number of this item.

    /// @return Returns the row number.
    int getRowNumber();
  };

////////////////////////////////////////////////////////////////////////////////

 } // namespace treeview
} // namespace COOLFluiD

////////////////////////////////////////////////////////////////////////////////

#endif // COOLFluiD_treeview_TreeItem_h
```

4.2.2 TreeItem.cxx

```
#include <QtXml>
#include <QDomNamedNodeMap>

#include "ClientServer/treeview/TreeItem.h"

using namespace COOLFluiD::treeview;

TreeItem::TreeItem(QDomNode & node, int row, TreeItem * parent)
{
 this->domNode = node;
 this->rowNumber = row;
 this->parentItem = parent;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

TreeItem::~TreeItem()
{
 QHash<int,TreeItem *>::iterator it = this->childItems.begin();

 while(it != this->childItems.end())
 {
  delete it.value();
  it++;
 }
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomNode TreeItem::getDomNode() const
{
 return this->domNode;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

TreeItem * TreeItem::getParentItem()
{
 return this->parentItem;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

TreeItem * TreeItem::getChild(int i)
{
 // if the TreeItem corresponding to this child has already been created,
 // it is returned...
 if (childItems.contains(i))
  return childItems[i];

 // ...otherwise, if the index is valid, it is created and returned...
 if (i >= 0 && i < this->domNode.childNodes().count())
 {
  QDomNode childNode = this->domNode.childNodes().item(i);
  TreeItem *childItem = new TreeItem(childNode, i, this);
  this->childItems[i] = childItem;

  return childItem;
 }

 // ...if the index is not valid, return a NULL pointer
 return NULL;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

int TreeItem::getRowNumber()
{
 return this->rowNumber;
}
```

4.3 *TreeModel* class

4.3.1 TreeModel.h

```
#ifndef COOLFLuiD_treeview_TreeModel_h
#define COOLFLuiD_treeview_TreeModel_h

////////////////////////////////////////////////////////////////////////////////

#include <QAbstractItemModel>
#include <QDomDocument>
#include <QList>

class QModelIndex;
class QVariant;

namespace COOLFluiD
{
 namespace treeview
 {

  struct TObjectProperties;

////////////////////////////////////////////////////////////////////////////////

  class TreeItem;

  /// @brief This class provides tools to manipulate an XML tree.

  /// This class also provides a model (inherits
  /// <code>QAbstractItemModel</code> class) that can be used to display the
  /// tree in a graphical view using the "Model/View Programming" concept.

  /// @author Quentin Gasper.

  class TreeModel : public QAbstractItemModel
  {

   private:
    /// @brief The tree
    QDomDocument domDocument;

    /// @brief Root of the tree (used to display the tree in a view)
    TreeItem * rootItem;

    /// @brief Indicates wether the model is in advanced mode.

    /// If <code>true</code>, the model is in advanced mode. See data() for
    /// further infomation.
    bool advancedMode;

    /// @brief Recursive method that builds a <code>QStingList</code> with
    /// all parent nodes names of a given node.

    /// The first string is the root of the tree.

    /// @param node Node from which the parents are returned
    /// @return Returns the built list
    QStringList getParentNodeNames(const QDomNode & node);

    /// @brief Appends to a document given options of a node.

    /// @param tagName Tag name. "modOptions" for modified options and
    /// "addOptions" for new options.
    /// @param parent Parent of the options (used to know the path)
    /// @param options Options to append.
    /// @param doc A reference to the document to which the options will
    /// be appended.
    /// @param keepAttrs If <code>true</code>, XML attributes are kept.
    /// Otherwise they are removed.
    void buildModification(const QString & tagName,const QDomNode & parent,
                           const QDomDocument & options, QDomDocument & doc,
                           bool keepAttrs);

  public:
    /// Constructor.

    /// @param document XML document on which this model is based.
    /// @param parent Parent of this model.
    TreeModel(QDomDocument document, QObject * parent = NULL);

    /// Destructor.
    ~TreeModel();

    /// @brief Implementation of <code>QAbstractItemModel::data()</code>.

    /// Only the role <code>Qt::DisplayRole</code> is accepted. Other
    /// roles will result to the return of an empty QVariant object
    /// (built with the default construtor).

    /// @param index Concerned item index.
    /// @param role Role of the returned value (only
    /// <code>Qt::DisplayRole</code>).

    /// @return Returns an empty QVariant object if the role is not
    /// <code>Qt::DisplayRole</code> or if the <code>index.isValid()</code>
    /// returns <code>false</code>. Otherwise, returns the nodename of the
    /// the item at the specified index.
    QVariant data(const QModelIndex & index, int role) const;

    /// @brief Implementation of <code>QAbstractItemModel::index()</code>.

    /// Gives the index of the item at the given row and column under
    /// the given parent. If the parent index is not valid, the root item
    /// is taken as parent.

    /// @param row Item row from the parent.
    /// @param column Item column.
    /// @param parent Item parent.
    /// @return Returns the requested index, or a null index if
    /// <code><b>this</b>->hasIndex(row, column, parent)</code> returns
    /// <code>false</code>.
    QModelIndex index(int row, int column,
                      const QModelIndex & parent = QModelIndex()) const;

    /// @brief Implementation of <code>QAbstractItemModel::parent()</code>.

    /// @param child Item index of which we would like to know the parent.

    /// @return Returns the parent index of the given child or a null
    /// index if the child is not a valid index.
    QModelIndex parent(const QModelIndex &child) const;

    /// @brief Implementation of <code>QAbstractItemModel::rowCount()</code>.

    /// If the parent index is not valid, the root item is taken as parent.

    /// @return Returns the row count (number of children) of a given parent.
    int rowCount(const QModelIndex & parent = QModelIndex()) const;
    /// @brief Implementation of
    /// <code>QAbstractItemModel::columnCount()</code>.

    /// @return Always returns 1.
    int columnCount(const QModelIndex & parent = QModelIndex()) const;

    /// @brief Gives the child nodes of the given index.

    /// @param index The parent index.

    /// @return Returns a <code>QDomNodeList</code> containing the children,
    /// or an empty list if the provided index is not valid.
    QDomNodeList getOptions(const QModelIndex & index) const;

    /// @brief Builds a document that can be used as data for "modifyOption"
    /// action.

    /// @param index Index from which options are taken.
    /// @param options Document cotaining modified options. These options
    /// must be at the root of the root of the document.
    /// @param newOptions Document cotaining new options. These options must
    /// be at the root of the root of the document.

    /// @return Returns the built document.
    QDomDocument modifyToDocument(const QModelIndex & index,
                                  const QDomDocument options,
                                  const QDomDocument newOptions =
                                    QDomDocument());

    /// @brief Gives the node associated to the given index.

    /// @param index Index.

    /// @return Returns the node associated, or a null node if the given
    /// index is not valid.
    QDomNode indexToNode(const QModelIndex & index) const;

    /// @brief Builds a node that can be used as data for "addNode"
    /// action.

    /// @param index Index of the parent node.
    /// @param newNode New node name.
    /// @param doc The document the node will be added to. The presence of
    /// this parameter is due to the fact that a node can not exist if it
    /// does not belong to a document.

    /// @return Returns the built node.
    QDomNode newChildToNode(const QModelIndex & index,
                            const QString & newNode, QDomDocument & doc);

    /// @brief Builds a node that can be used as data for "renameNode"
    /// action.

    /// @param index Index of the node to rename.
    /// @param newName New name of the node.
    /// @param doc The document the node will be added to. The presence of
    /// this parameter is due to the fact that a node can not exist if it
    /// does not belong to a document.

    /// @return Returns the built node.
    QDomNode renameToNode(const QModelIndex & index,
                          const QString & newName, QDomDocument & doc);

    /// @brief Sets or resets the advanced mode.
    /// @param advanced Advanced mode state.
    void setAdvancedMode(bool advanced);

    /// @brief Gives the advanced mode state

    /// @return Returns the advanced mode state.
    bool getAdvancedMode() const;

    /// @brief Give the properties (XML attributes) of a given item.

    /// @param index Item index.
    /// @param ok Reference to a bool variable. After this method returns,
    /// the bool value is <code>false</code> if there was an error, otherwise
    /// value is <code>true</code>.

    /// @return Returns the properties of the item. Properties values are
    /// undefined if there was an error (<code>ok</code> =
    /// <code>false</code>).
    TObjectProperties getProperties(const QModelIndex & index,
                                    bool & ok) const;

  };

//////////////////////////////////////////////////////////////////////////////

 } // namespace treeview
} // namespace COOLFluiD

//////////////////////////////////////////////////////////////////////////////

#endif // COOLFLuiD_treeview_TreeModel_h
```

4.3.2 TreeModel.cxx

```
#include <iostream>
#include <QtGui>
#include <QtXml>

#include "ClientServer/treeview/TreeItem.h"
#include "ClientServer/treeview/TObjectProperties.h"
#include "ClientServer/treeview/TreeModel.h"

using namespace COOLFluiD::treeview;

TreeModel::TreeModel(QDomDocument document, QObject *parent)
 : QAbstractItemModel(parent)
{
 QDomNodeList nodeList;
 domDocument = document;

 nodeList = domDocument.childNodes();

 // if the first node is the xml tag (<?xml...), it's removed (the
 // second node becomes the first one) : there's no need to show it.
 if(nodeList.item(0).nodeName().compare("xml") == 0)
  domDocument.replaceChild(nodeList.item(1), nodeList.item(0));

 rootItem = new TreeItem(domDocument, 0);

 this->advancedMode = false;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

TreeModel::~TreeModel()
{
 delete rootItem;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

int TreeModel::columnCount(const QModelIndex & parent) const
{
 return 1;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QVariant TreeModel::data(const QModelIndex & index, int role) const
{
 TreeItem *item;
 QDomNode node;

 if (!index.isValid() || role != Qt::DisplayRole)
  return QVariant();

 item = static_cast<TreeItem*>(index.internalPointer());

 if(item == NULL)
  return QVariant();

 node = item->getDomNode();

 if(index.column() == 0)
 {
  QDomNamedNodeMap attributes = node.attributes();

  if(attributes.namedItem("tree").nodeValue() == "object")
  {
   if(!this->advancedMode && attributes.namedItem("mode").nodeValue() ==
       "advanced")
    return QVariant();

   return node.nodeName() + QString("␣[") +
     attributes.namedItem("type").nodeValue() + QString("]");
  }
  return QVariant();
 }
 else
  return QVariant();
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QModelIndex TreeModel::index(int row, int column,
                             const QModelIndex &parent) const
{
 if (!this->hasIndex(row, column, parent))
  return QModelIndex();

 TreeItem *parentItem;

 if (!parent.isValid())
  parentItem = this->rootItem;
 else
  parentItem = static_cast<TreeItem*>(parent.internalPointer());

 TreeItem * childItem = parentItem->getChild(row);
 if (childItem != NULL)
  return createIndex(row, column, childItem);
 else
  return QModelIndex();
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QModelIndex TreeModel::parent(const QModelIndex & child) const
{
 if (!child.isValid())
  return QModelIndex();

 TreeItem * childItem = static_cast<TreeItem*>(child.internalPointer());
 TreeItem * parentItem = childItem->getParentItem();

 if (parentItem == NULL || parentItem == this->rootItem)
  return QModelIndex();

 return createIndex(parentItem->getRowNumber(), 0, parentItem);
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

int TreeModel::rowCount(const QModelIndex &parent) const
{
 TreeItem *parentItem;
 if (parent.column() > 0)
  return 0;

 if (!parent.isValid())
  parentItem = rootItem;
 else
  parentItem = static_cast<TreeItem*>(parent.internalPointer());

 return parentItem->getDomNode().childNodes().count();
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomNodeList TreeModel::getOptions(const QModelIndex & index) const
{
 TreeItem * item;

 if(!index.isValid())
  return QDomNodeList();

 item = static_cast<TreeItem*>(index.internalPointer());

 if(item == NULL)
  return QDomNodeList();

 return item->getDomNode().childNodes();
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomDocument TreeModel::modifyToDocument(const QModelIndex & index,
                                         const QDomDocument options,
                                         const QDomDocument newOptions)
{
 QDomDocument doc;
 TreeItem * item;

 if(!index.isValid())
  return QDomDocument();

 item = static_cast<TreeItem *>(index.internalPointer());

 if(item == NULL)
  return QDomDocument();

 this->buildModification("modOptions", item->getDomNode(), options, doc,
                         false);

 this->buildModification("addOptions", item->getDomNode(), newOptions, doc,
                         true);

 return doc;
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void TreeModel::buildModification(const QString & tagName,
                                  const QDomNode & parent,
                                  const QDomDocument & options,
                                  QDomDocument & doc, bool keepAttrs)
{
 QDomElement node;
 QStringList parents = this->getParentNodeNames(parent);
 QString parentsString = QString("/") + parents.join("/");
 QDomNodeList childNodes = options.childNodes();

 if(!childNodes.isEmpty())
 {
  node = doc.createElement(tagName);
  node.setAttribute("path", parentsString);

  for(int i = 0 ; i < childNodes.count() ; i++)
  {
   QDomElement option = doc.importNode(childNodes.item(i), true).toElement();

   if(option.isNull())
    continue;

   if(!keepAttrs)
   {
    QDomNamedNodeMap attributes = option.attributes();

    while(attributes.count() > 0)
     option.removeAttribute(attributes.item(0).nodeName());
   }

   node.appendChild(option);
  }

  doc.appendChild(node);
 }
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

TObjectProperties TreeModel::getProperties(const QModelIndex & index,
                                           bool & ok) const
{
 TObjectProperties properties;
 TreeItem * item;
 QDomNamedNodeMap attributes;

 if(!index.isValid())
 {
  ok = false;
  return TObjectProperties();
 }

 item = static_cast<TreeItem *>(index.internalPointer());

 if(item == NULL)
 {
  ok = false;
  return TObjectProperties();
 }

 attributes = item->getDomNode().attributes();

 properties.type = attributes.namedItem("type").nodeValue();
 properties.absType = attributes.namedItem("abstype").nodeValue();
 properties.dynamic = attributes.namedItem("dynamic").nodeValue() == "true";
 properties.basic = attributes.namedItem("mode").nodeValue() == "basic";

 return properties;
}
```

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomNode TreeModel::newChildToNode(const QModelIndex & index,
                                   const QString & newNode,
                                   QDomDocument & doc)
{
 //QDomDocument doc;
 QDomElement lastElement;
 QDomElement elt;

 TreeItem * item;
 QDomNode node;
 QDomNode indexNode;

 if(!index.isValid())
  return QDomDocument();

 if(newNode.isNull() || newNode.isEmpty())
  return QDomDocument();

 item = static_cast<TreeItem *>(index.internalPointer());
 indexNode = item->getDomNode();

 if(item == NULL)
  return QDomDocument();

 QStringList parents = this->getParentNodeNames(indexNode.parentNode());

 if(parents.count() > 0)
 {
  lastElement = doc.createElement(parents.at(0));
  doc.appendChild(lastElement);

  for(int i = 1 ; i < parents.count() ; i++)
  {
   QDomElement element = doc.createElement(parents.at(i));
   lastElement.appendChild(element);
   lastElement = element;
  }

  elt = doc.createElement(indexNode.nodeName());
  node = lastElement.appendChild(elt);

  QDomElement elem = doc.createElement(newNode);
  node.appendChild(elem);
  return elem;
 }

 return QDomElement();
}

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomNode TreeModel::renameToNode(const QModelIndex & index,
                                 const QString & newName,
                                 QDomDocument & doc)
{
 //QDomDocument doc;
 QDomElement lastElement;
 QDomElement elt;

 TreeItem * item;
 QDomNode node;
 QDomNode indexNode;

 if(!index.isValid())
  return QDomDocument();

 if(newName.isNull() || newName.isEmpty())
  return QDomDocument();

 item = static_cast<TreeItem *>(index.internalPointer());
 indexNode = item->getDomNode();

 if(item == NULL)
  return QDomDocument();

 QStringList parents = this->getParentNodeNames(indexNode.parentNode());

 if(parents.count() > 0)
 {
  lastElement = doc.createElement(parents.at(0));
  doc.appendChild(lastElement);

  for(int i = 1 ; i < parents.count() ; i++)
  {
   QDomElement element = doc.createElement(parents.at(i));
   lastElement.appendChild(element);
   lastElement = element;
  }

  elt = doc.createElement(indexNode.nodeName());
  elt.setAttribute("newName", newName);
  node = lastElement.appendChild(elt);
  return elt;
 }

 return QDomNode();
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

QDomNode TreeModel::indexToNode(const QModelIndex & index) const
{
 TreeItem * item;

 if(!index.isValid())
  return QDomNode();

 item = static_cast<TreeItem*>(index.internalPointer());

 if(item == NULL)
  return QDomNode();

 return item->getDomNode();
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

void TreeModel::setAdvancedMode(bool advanced)
{
 this->advancedMode = advanced;
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
bool TreeModel::getAdvancedMode() const
{
 return this->advancedMode;
}


/***************************************************************************

                                PRIVATE METHOD

***************************************************************************/

QStringList TreeModel::getParentNodeNames(const QDomNode & node)
{
 QDomNode parentNode = node.parentNode();
 QStringList list;

 if(parentNode.isNull()) // if the node has no parent
  return list;
 else
 {
  list = this->getParentNodeNames(parentNode);
  list << node.nodeName();
  return  list;
 }
}
```

Part II

Code maintenance