

Hello World! Welcome to NMM Application Development

Motama GmbH, Saarbruecken, Germany
(<http://www.motama.com>)

April 2010

Copyright (C) 2005-2010
Motama GmbH, Saarbruecken, Germany
<http://www.motama.com>

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with the Invariant Sections being all sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license can be found in the file COPYING.FDL.

THE DOCUMENT IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE DOCUMENT BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE DOCUMENT OR THE USE OR OTHER DEALINGS IN THE DOCUMENT.

This document provides some simple examples of how to develop applications with NMM. All examples are available in the examples/helloworld/ sub-directory of NMM.

1. Motivation

Developing (distributed) multimedia application with NMM is easy. This tutorial includes following examples.

- helloworld1: simple wav player using GraphDescription (with error handling, with access to 'serverregistry')
- helloworld2: simple wav player creating flow graph manually with error handling and access to 'serverregistry'
- helloworld3: simple wav player using GraphDescription with error handling and access to 'serverregistry'; same code as helloworld1, but added listener for startTrack and endTrack events
- helloworld4: simple wav player using GraphDescription with error handling and access to 'serverregistry'; same code as helloworld1, but implemented simple seeking functionality
- helloworld5: simple wav player using GraphDescription with error handling and access to 'serverregistry' same code as helloworld1, but using advanced node descriptions for audio sink
- hellonmm1 (modified version of helloworld5): distributed wav player using GraphDescription with error handling and access to 'serverregistry' : playback on remote host
- hellonmm2 (modified version of helloworld5) : distributed wav player using GraphDescription with error handling and access to 'serverregistry' : playback on local and remote host
- hellonmm3 (modified version of helloworld5) : optimized distributed wav player using GraphDescription with error handling and access to 'serverregistry': playback on remote host
- hellonmm4 and hellonmm41 : these two examples use a client server approach. hellonmm4 is a server streaming mp3 files using RTP multicast. hellonmm41 is a streaming client. To demonstrate the capabilities of NMM, several streaming clients can be started on different hosts and synchronized audio rendering can be enabled.
- hellonmm5 and hellonmm51 : these two examples use a client server approach. hellonmm5 is a server streaming video files using RTP multicast. hellonmm51 is a streaming client. To demonstrate the capabilities of NMM, several streaming clients can be started on different hosts and synchronized video rendering can be enabled.
- hellonmm9: demonstrates discovery of running serverregistries in the local network.
- hellonmm10 and hellonmm101 : these two examples use a client server approach. hellonmm10 is a server streaming MPEG TS files using RTP multicast. hellonmm101 is a streaming client. To demonstrate the capabilities of NMM, several streaming clients can be started on different hosts and synchronized video rendering can be enabled.
- hellographbuilder1 : shows how to use the graph builder of NMM that automatically creates a flow graph from a given URL.
- hellographbuilder2 : shows how to use the graph builder of NMM that automatically creates a distributed flow graph from a given URL.

For all examples that can access a 'serverregistry', you can optionally start the program `apps/registry/serverregistry`. However, examples `helloworld1`, `helloworld2`, `helloworld3`, `helloworld4`, `helloworld5`, and `hellographbuilder1` will work without a running 'serverregistry' as well. Notice: for examples `hellonmm1`, `hellonmm2` and `hellonmm3` you do need a running serverregistry on a second

remote host. For example `helloworldbuilder2` you do need running serverregistries on all remote hosts given.

'`clic`' (command line interaction and configuration) is an NMM application that allows to set up a distributed flow graph from a textual description. For `helloworld1`, `hellonmm1`, `hellonmm2`, and `hellonmm3` we also provide the corresponding graph description files to be used with '`clic`'.

- `helloworld1.gd`: simple wav player
- `hellonmm1.gd`: distributed wav player
- `hellonmm2.gd`: distributed wav player : playback on local and remote host
- `hellonmm3.gd`: optimized distributed wav player : playback on remote host. In contrast to the application `hellonmm3` the TCP buffer size (or any parameter of a network connection) can not be changed because this feature is currently not implemented in `clic`.

2. helloworld1

The simple WAV player will be used as our first example. The corresponding flow graph consists of two NMM nodes: reading and rendering audio. For creating a request that can be used to query the registry service, following lines of code need to be provided.

```
GraphDescription graph;  
  
NodeDescription readfile_descr("WavReadNode");  
NodeDescription audioplay_descr(DEFAULT_AUDIO_SINK);  
  
graph.addEdge(&readfile_descr, &audioplay_descr);
```

`DEFAULT_AUDIO_SINK` is using the default name of the audio sink of the used platform, which is different for Linux, MacOS, Windows, etc. In `helloworld5` you will learn to request nodes in a platform independent way.

Such a graph description is forwarded to the registry service by forwarding it to a client registry, which can be accessed by creating a central NMM application:

```
NMMApplication* app = 0;  
  
app = ProxyApplication::getApplication(argc, argv);  
ClientRegistry& registry = app->getRegistry();  
registry.requestGraph(graph);
```

If the last method invocation succeeded, all nodes were created. Before the flow graph can be started, we need to set the WAV file to be played, e.g. the file given as command line parameter: First, we need to request the generic interface `INode` of the node description of the data source. Second, the specific

interface `IFileHandler` is tried to be acquired. If it is not available for the instantiated node, an exception is thrown. Third, the set the filename by a simple method invocation. Together, this results in following lines of source code.

```
INode* readfile = graph.getINode(readfile_descr);
IFileHandler_var filehandler(readfile->getParentObject()->
                             getCheckedInterface<IFileHandler>());
filehandler->setFilename(argv[1]);
```

The flow graph is configured and started by two additional commands:

```
graph.realizeGraph();
graph.startGraph();
```

For terminating and releasing the flow graph, following two lines are required:

```
graph.stopGraph();
registry.releaseGraph(graph);
```

The complete source code of this application: `helloworld1: simple wav player using GraphDescription`, also including error handling:

```
// simple wav player using GraphDescription, with error handling

#include <iostream>

// include NMM
#include "nmm/comm/ProxyObject.hpp"
#include "nmm/services/registry/ProxyApplication.hpp"
#include "nmm/services/DefaultSinks.hpp"

// include used interfaces
#include "nmm/plugins/interfaces/IFileHandler.hpp"

// import namespace
using namespace NMM;

int main(int argc, char ** argv)
{
    if(argc != 2) {
        cerr << "Usage:" << endl
             << " " << argv[0] << " <wav file> " << endl
             << endl;
        return 1;
    }
    cerr << argv[0] << " running ... " << endl << endl;

    // disable all debug output (error, warning, message, debug)
    // see documentation Debug, Error, Warning and Message Output in NMM
    NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);
    NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);
```

Hello World! Welcome to NMM Application Development

```
NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);

// the graph description for the example
GraphDescription graph;

// the NMM application for the example
NMMApplication* app = 0;

// NMM-operations may throw exception, in this simple example, all exceptions
// are just caught at the end, and the application is terminated
try {
    // create application object for example
    app = ProxyApplication::getApplication(argc, argv);

    // create the descriptions for the WavReadNode and platform specific PlaybackNode
    // using default name for used platform (different for Linux, MacOS, Windows, etc.)
    NodeDescription readfile_descr("WavReadNode");
    NodeDescription audioplay_descr(DEFAULT_AUDIO_SINK);

    // create a flow graph: readfile -> audioplay
    graph.addEdge(readfile_descr, audioplay_descr);

    // request complete graph
    cerr << "request GraphDescription from registry" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.requestGraph(graph);

    // set the filename by requesting appropriate interface
    cerr << "set filename to " << argv[1] << endl;
    INode* readfile = graph.getINode(readfile_descr);
    IFileHandler_var filehandler (readfile-> getParentObject()-> getCheckedInterface<IFileH
filehandler-> setFilename(argv[1]);

    // realize and start graph
    cerr << "realizeGraph()" << endl;
    graph.realizeGraph();
    cerr << "startGraph()" << endl;
    graph.startGraph();

    // wait ...
    cerr << "Playing audio ... enter 'q <enter> ' to exit." << endl;
    string s;
    cin >> s;

    // stop and release graph
    cerr << "stopGraph()" << endl;
    graph.stopGraph();

    cerr << "releaseGraph()" << endl;
    registry.releaseGraph(graph);

    delete app;
}
```

```
    return 0;
}
catch(const Exception& e) { // catch NMM-exception and print it
    cerr << e << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
catch(const SerializedException& e) { // catch serialized NMM-exception and print it
    cerr << e << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
catch(...) { // any other exception
    cerr << "something went wrong!" << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
}
```

3. helloworld2

helloworld2: simple wav player creating flow graph manually with error handling and access to 'server-registry'

```
// simple wav player creating flow graph manually with error handling and access to 'server-registry'

#include <iostream>

// include NMM
#include "nmm/comm/ProxyObject.hpp"
```

Hello World! Welcome to NMM Application Development

```
#include "nmm/services/registry/ProxyApplication.hpp"
#include "nmm/services/DefaultSinks.hpp"
#include "nmm/multimedia/connect.hpp"
#include "nmm/multimedia/NodeDescription.hpp"
#include "nmm/services/registry/RegistryLock.hpp"

// include used interfaces
#include "nmm/plugins/interfaces/IFileHandler.hpp"

// import namespace
using namespace NMM;

int main(int argc, char ** argv)
{
    if(argc != 2) {
        cerr << "Usage:" << endl
             << " " << argv[0] << " <wav file> " << endl
             << endl;
        return 1;
    }
    cerr << argv[0] << " running ... " << endl << endl;

    // disable all debug output (error, warning, message, debug)
    // see documentation Debug, Error, Warning and Message Output in NMM
    NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);
    NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);
    NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);
    NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);

    // the NMM application for the example
    NMMApplication* app = 0;

    // the _var variables for the INodes (C++ auto_ptr<> )
    INode_var readfile;
    INode_var audioplay;

    // NMM-operations may throw exception, in this simple example, all exceptions
    // are just caught at the end, and the application is terminated
    try {
        // create application object for example
        app = ProxyApplication::getApplication(argc, argv);

        // create the descriptions for the WavReadNode and platform specific PlaybackNode
        // using default name for used platform (different for Linux, MacOS, Windows, etc.)
        NodeDescription readfile_descr("WavReadNode");
        NodeDescription audioplay_descr(DEFAULT_AUDIO_SINK);

        // request Nodes:
        // lock registry with guard, init request for each node, instantiate first result for e
        cerr << "request Nodes from registry" << endl;
        ClientRegistry& registry = app-> getRegistry();
        {
            RegistryLock lock(registry);
```

Hello World! Welcome to NMM Application Development

```
list<Response> readfile_response = registry.initRequest(readfile_descr);
list<Response> audioplay_response = registry.initRequest(audioplay_descr);

if (readfile_response.empty() || audioplay_response.empty() ) {
    cerr << "size of readfile_response " << readfile_response.size() << endl;
    cerr << "size of audioplay_response " << audioplay_response.size() << endl;
    throw Exception("a required node is not available");
}

readfile.reset(registry.requestNode(readfile_response.front()));
audioplay.reset(registry.requestNode(audioplay_response.front()));
} // end of scope for lock -> will release lock

// set the filename by requesting appropriate interface
cerr << "set filename to " << argv[1] << endl;
IFileHandler_var filehandler (readfile-> getParentObject()-> getCheckedInterface<IFileH
filehandler-> setFilename(argv[1]);

// create a flow graph: readfile -> audioplay

// initialize all three nodes ...
readfile-> init();
audioplay-> init();

// initialize output of source node
readfile-> initOutput();

// now, output of source node can be connected to initialized input of audio sink
cerr << "connect(readfile, audioplay)" << endl;
connect(readfile, audioplay);

// activate source node so that audio sink can initialize its output
readfile-> activate();
audioplay-> initOutput();

// audio sink can be activated now
audioplay-> activate();

// ... and finally start them!
cerr << "start() all Nodes" << endl;
readfile-> start();
audioplay-> start();

// wait ...
cerr << "Playing audio ... enter 'q <enter> ' to exit." << endl;
string s;
cin >> s;

// stop nodes, reset and destroy nodes
cerr << "stop(), flush(), deactivate(), deinitOutput(), deinit() all Nodes" << endl;
readfile-> stop();
audioplay-> stop();
```

```
readfile-> flush();
audioplay-> flush();

readfile-> deactivate();
audioplay-> deactivate();

readfile-> deinitOutput();
audioplay-> deinitOutput();

readfile-> deinit();
audioplay-> deinit();

// Release the reserved nodes
cerr << "release all Nodes" << endl;
registry.releaseNode(*readfile);
registry.releaseNode(*audioplay);

// need to first release INodes before deleting app
readfile.reset();
audioplay.reset();
delete app;

return 0;
}
catch(const Exception& e) { // catch NMM-exception and print it
    cerr << e << endl;

    // try to release the reserved nodes
    cerr << "release all Nodes" << endl;
    ClientRegistry& registry = app-> getRegistry();
    RegistryLock lock(registry);
    if(readfile.get()) {
        registry.releaseNode(*readfile);
    }
    if(audioplay.get()) {
        registry.releaseNode(*audioplay);
    }
}

// need to first release INodes before deleting app
readfile.reset();
audioplay.reset();
delete app;

return 1;
}
catch(const SerializedException& e) { // catch serialized NMM-exception and print it
    cerr << e << endl;

    // try to release the reserved nodes
    cerr << "release all Nodes" << endl;
    ClientRegistry& registry = app-> getRegistry();
    RegistryLock lock(registry);
```

```
if(readfile.get()) {
    registry.releaseNode(*readfile);
}
if(audioplay.get()) {
    registry.releaseNode(*audioplay);
}

// need to first release INodes before deleting app
readfile.reset();
audioplay.reset();
delete app;

return 1;
}
catch(...) { // any other exception
    cerr << "something went wrong!" << endl;

    // try to release the reserved nodes
    cerr << "release all Nodes" << endl;
    ClientRegistry& registry = app-> getRegistry();
    RegistryLock lock(registry);
    if(readfile.get()) {
        registry.releaseNode(*readfile);
    }
    if(audioplay.get()) {
        registry.releaseNode(*audioplay);
    }

    // need to first release INodes before deleting app
    readfile.reset();
    audioplay.reset();
    delete app;

    return 1;
}
}
```

4. helloworld3

helloworld3: simple wav player using GraphDescription with error handling and access to 'serverregistry'; same code as helloworld1, but added listener for startTrack and endTrack event (parts of code typed **bold**)

```
// simple wav player using GraphDescription with error handling and access to 'serverregist
// same code as helloworld1, but added listener for endTrack event
```

```
#include <iostream>
```

```
// include NMM
#include "nmm/comm/ProxyObject.hpp"
#include "nmm/services/registry/ProxyApplication.hpp"
#include "nmm/services/DefaultSinks.hpp"
#include "nmm/comm/serialize/SerializeResult.hpp"

// include used interfaces
#include "nmm/plugins/interfaces/IFileHandler.hpp"
#include "nmm/multimedia/interfaces/ITrack.hpp"

// import namespace
using namespace NMM;

// a simple listener class
class TrackListener {
public:
    Result endTrack()
    {
        cerr << endl << "endTrack" << endl << endl;
        return SUCCESS;
    }

    Result startTrack(const string& filename)
    {
        cerr << endl << "startTrack " << filename << endl << endl;
        return SUCCESS;
    }
};

int main(int argc, char ** argv)
{
    if(argc != 2) {
        cerr << "Usage:" << endl
            << " " << argv[0] << " <wav file> " << endl
            << endl;
        return 1;
    }
    cerr << argv[0] << " running ... " << endl << endl;

    // disable all debug output (error, warning, message, debug)
    // see documentation Debug, Error, Warning and Message Output in NMM
    NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);
    NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);
    NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);
    NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);

    // the graph description for the example
    GraphDescription graph;

    // the NMM application for the example
    NMMApplication* app = 0;

    // NMM-operations may throw exception, in this simple example, all exceptions
```

```
// are just caught at the end, and the application is terminated
try {
    // create application object for example
    app = ProxyApplication::getApplication(argc, argv);

    // create the descriptions for the WavReadNode and platform specific PlaybackNode
    // using default name for used platform (different for Linux, MacOS, Windows, etc.)
    NodeDescription readfile_descr("WavReadNode");
    NodeDescription audioplay_descr(DEFAULT_AUDIO_SINK);

    // create a flow graph: readfile -> audioplay
    graph.addEdge(readfile_descr, audioplay_descr);

    // request complete graph
    cerr << "request GraphDescription from registry" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.requestGraph(graph);

    // set the filename by requesting appropriate interface
    cerr << "set filename to " << argv[1] << endl;
    INode* readfile = graph.getINode(readfile_descr);
    IFileHandler_var filehandler (readfile-> getParentObject()-> getCheckedInterface<IFileH
filehandler-> setFilename(argv[1]);

    // register listener for events startTrack and endTrack of ITrack interface at audio si
    // -> the method TrackListener::startTrack will get triggered when the stream starts a
    // -> the method TrackListener::endTrack will get triggered when the stream ended at a
    //
    // if you are unsure about the correct syntax for TEDObjects,
    // please refer to hpp/cpp sources generated from corresponding
    // .idl file
    INode* audioplay = graph.getINode(audioplay_descr);
    TrackListener end_track_listener;
    audioplay-> getParentObject()-> registerEventListener(ITrack::endTrack_event,
        new TEDObject0<TrackListener> (&end_track_listener, &TrackListener::endTrack));
    audioplay-> getParentObject()-> registerEventListener(ITrack::startTrack_event,
        new TEDObject1<TrackListener, string,
        Result(TrackListener:: * )(const string&), Result>
        (&end_track_listener, &TrackListener::startTrack));

    // realize and start graph
    cerr << "realizeGraph()" << endl;
    graph.realizeGraph();
    cerr << "startGraph()" << endl;
    graph.startGraph();

    // wait ...
    cerr << "Playing audio ... enter 'q <enter> ' to exit." << endl;
    string s;
    cin >> s;

    // stop and release graph
    cerr << "stopGraph()" << endl;
```

```
graph.stopGraph();

cerr << "releaseGraph()" << endl;
registry.releaseGraph(graph);

delete app;
return 0;
}
catch(const Exception& e) { // catch NMM-exception and print it
    cerr << e << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
catch(const SerializedException& e) { // catch serialized NMM-exception and print it
    cerr << e << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
catch(...) { // any other exception
    cerr << "something went wrong!" << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
}
```

5. helloworld4

helloworld4: simple wav player using GraphDescription with error handling and access to 'serverregistry'; same code as helloworld1, but implemented simple seeking functionality (parts of code typed **bold**)

```
// simple wav player using GraphDescription with error handling and access to 'serverregist  
// same code as helloworld1, but implemented simple seeking functionality
```

```
#include <iostream>  
  
// include NMM  
#include "nmm/comm/ProxyObject.hpp"  
#include "nmm/services/registry/ProxyApplication.hpp"  
#include "nmm/services/DefaultSinks.hpp"  
  
// include used interfaces  
#include "nmm/plugins/interfaces/IFileHandler.hpp"  
#include "nmm/plugins/file/interfaces/ISeekable.hpp"  
  
// import namespace  
using namespace NMM;  
  
int main(int argc, char ** argv)  
{  
    if(argc != 2) {  
        cerr << "Usage:" << endl  
             << " " << argv[0] << " <wav file> " << endl  
             << endl;  
        return 1;  
    }  
    cerr << argv[0] << " running ... " << endl << endl;  
  
    // disable all debug output (error, warning, message, debug)  
    // see documentation Debug, Error, Warning and Message Output in NMM  
    NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);  
    NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);  
    NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);  
    NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);  
  
    // the graph description for the example  
    GraphDescription graph;  
  
    // the NMM application for the example  
    NMMApplication* app = 0;  
  
    // NMM-operations may throw exception, in this simple example, all exceptions  
    // are just caught at the end, and the application is terminated  
    try {  
        // create application object for example  
        app = ProxyApplication::getApplication(argc, argv);  
  
        // create the descriptions for the WavReadNode and platform specific PlaybackNode  
        // using default name for used platform (different for Linux, MacOS, Windows, etc.)  
        NodeDescription readfile_descr("WavReadNode");  
        NodeDescription audioplay_descr(DEFAULT_AUDIO_SINK);  
  
        // create a flow graph: readfile -> audioplay  
        graph.addEdge(readfile_descr, audioplay_descr);
```

```
// request complete graph
cerr << "request GraphDescription from registry" << endl;
ClientRegistry& registry = app-> getRegistry();
registry.requestGraph(graph);

// set the filename by requesting appropriate interface
cerr << "set filename to " << argv[1] << endl;
INode* readfile = graph.getInode(readfile_descr);
IFileHandler_var filehandler (readfile-> getParentObject()-> getCheckedInterface<IFileH
filehandler-> setFilename(argv[1]);

// get ISeekable interface from file source
// (see nmm/interfaces/file/ISeekable.idl for more information)
ISeekable_var readfile_seek(readfile-> getParentObject()-> getCheckedInterface<ISeekabl

// realize and start graph
cerr << "realizeGraph()" << endl;
graph.realizeGraph();
cerr << "startGraph()" << endl;
graph.startGraph();

// wait ...
while(true) {
    cerr << "Playing audio ... enter 'q <enter> ' to exit " << endl
         << "or '<positive number between 0..100> <enter> ' to seek to position in percent
    string s;
    cin >> s;
    if(s == "q") {
        break;
    }
    else {
        if(readfile_seek-> hasPercentSeek()) { // check, if file source supports this speci
            int seek_percent = atoi(s.c_str());
            readfile_seek-> seekPercentTo(Rational(seek_percent,100));
            cerr << "seeked to " << seek_percent << " percent" << endl;
        }
        else {
            cerr << "file source cannot seek" << endl;
        }
    }
}

// stop and release graph
cerr << "stopGraph()" << endl;
graph.stopGraph();

cerr << "releaseGraph()" << endl;
registry.releaseGraph(graph);

delete app;
return 0;
}
```

```
catch(const Exception& e) { // catch NMM-exception and print it
    cerr << e << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
catch(const SerializedException& e) { // catch serialized NMM-exception and print it
    cerr << e << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
catch(...) { // any other exception
    cerr << "something went wrong!" << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
}
```

6. helloworld5

helloworld5: simple wav player using GraphDescription with error handling and access to 'serverregistry' same code as helloworld1, but using advanced node descriptions for audio sink (parts of code typed **bold**.)

```
// simple wav player using GraphDescription, with error handling
// same code as helloworld1, but using advanced node descriptions for audio sink

#include <iostream>

// include NMM
#include "nmm/comm/ProxyObject.hpp"
#include "nmm/services/registry/ProxyApplication.hpp"
```

```
// include used interfaces
#include "nmm/plugins/interfaces/IFileHandler.hpp"
#include "nmm/plugins/audio/interfaces/IAudioDevice.hpp"

// import namespace
using namespace NMM;

int main(int argc, char ** argv)
{
    if(argc != 2) {
        cerr << "Usage:" << endl
             << " " << argv[0] << " <wav file> " << endl
             << endl;
        return 1;
    }
    cerr << argv[0] << " running ... " << endl << endl;

    // disable all debug output (error, warning, message, debug)
    // see documentation Debug, Error, Warning and Message Output in NMM
    NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);
    NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);
    NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);
    NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);

    // the graph description for the example
    GraphDescription graph;

    // the NMM application for the example
    NMMApplication* app = 0;

    // NMM-operations may throw exception, in this simple example, all exceptions
    // are just caught at the end, and the application is terminated
    try {
        // create application object for example
        app = ProxyApplication::getApplication(argc, argv);

        // create the descriptions for the WavReadNode and platform specific PlaybackNode
        NodeDescription readfile_descr("WavReadNode");
        // create the description for audio sink:
        // ... some sink node
        // ... that supports the interface IAudioDevice
        // notice that this description will provide an audio sink
        // on all supported platforms
        NodeDescription audioplay_descr;
        audioplay_descr.setNodeType(INode::SINK);
        audioplay_descr.addInterface(IAudioDevice::IAudioDevice_type);

        // create a flow graph: readfile -> audioplay
        graph.addEdge(readfile_descr, audioplay_descr);

        // request complete graph
        cerr << "request GraphDescription from registry" << endl;
    }
```

```
ClientRegistry& registry = app-> getRegistry();
registry.requestGraph(graph);

// set the filename by requesting appropriate interface
cerr << "set filename to " << argv[1] << endl;
INode* readfile = graph.getINode(readfile_descr);
IFileHandler_var filehandler (readfile-> getParentObject()-> getCheckedInterface<IFileH
filehandler-> setFilename(argv[1]);

// realize and start graph
cerr << "realizeGraph()" << endl;
graph.realizeGraph();
cerr << "startGraph()" << endl;
graph.startGraph();

// wait ...
cerr << "Playing audio ... enter 'q <enter> ' to exit." << endl;
string s;
cin >> s;

// stop and release graph
cerr << "stopGraph()" << endl;
graph.stopGraph();

cerr << "releaseGraph()" << endl;
registry.releaseGraph(graph);

delete app;
return 0;
}
catch(const Exception& e) { // catch NMM-exception and print it
    cerr << e << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
catch(const SerializedException& e) { // catch serialized NMM-exception and print it
    cerr << e << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
catch(...) { // any other exception
```

```
cerr << "something went wrong!" << endl;

// release graph
cerr << "releaseGraph()" << endl;
ClientRegistry& registry = app-> getRegistry();
registry.releaseGraph(graph);

delete app;
return 1;
}
}
```

7. hellonmm1

Since NMM flow graphs can be distributed transparently, only a few additional lines of source code are required to redirect the audio output to a remote host. In particular, the sink node for rendering audio needs to be requested from a remote host.

In our example, only one additional line is required: the setLocation() call. The host name of the remote system is given as second argument when starting the application. On this remote host, the required node need to be available, i.e. the corresponding names need to be printed out when calling './serverregistry -s'. During runtime, a serverregistry needs to be started on the remote host:

```
user@linux:~/nmm/bin> ./serverregistry
ServerRegistry successfully started!
```

The complete source code of the application hellonmm1: (**modified** version of helloworld5) : distributed wav player using GraphDescription with error handling and access to 'serverregistry' : playback on remote host.

```
// distributed wav player using GraphDescription with error handling
// and access to 'serverregistry' playback on remote host
// notice: you need a running 'serverregistry' on the remote host

#include <iostream>

// include NMM
#include "nmm/comm/ProxyObject.hpp"
#include "nmm/services/registry/ProxyApplication.hpp"

// include used interfaces
#include "nmm/plugins/interfaces/IFileHandler.hpp"
#include "nmm/plugins/audio/interfaces/IAudioDevice.hpp"

// import namespace
using namespace NMM;
```

```
int main(int argc, char ** argv)
{
    if(argc != 3) {
        cerr << "Usage:" << endl
            << " " << argv[0] << " <wav file> <remote host where serverregistry is running>"
            << endl;
        return 1;
    }
    cerr << argv[0] << " running ... " << endl << endl;

    // disable all debug output (error, warning, message, debug)
    // see documentation Debug, Error, Warning and Message Output in NMM
    NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);
    NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);
    NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);
    NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);

    // the graph description for the example
    GraphDescription graph;

    // the NMM application for the example
    NMMApplication* app = 0;

    // NMM-operations may throw exception, in this simple example, all exceptions
    // are just caught at the end, and the application is terminated
    try {
        // create application object for example
        app = ProxyApplication::getApplication(argc, argv);

        // create the descriptions for the WavReadNode and platform specific PlaybackNode
        NodeDescription readfile_descr("WavReadNode");

        // create the description for audio sink:
        // ... some sink node
        // ... that supports the interface IAudioDevice
        // notice that this description will provide an audio sink
        // on all supported platforms
        NodeDescription audioplay_descr;
        audioplay_descr.setNodeType(INode::SINK);
        audioplay_descr.addInterface(IAudioDevice::IAudioDevice_type);

        // audio renderer will be requested from serverregistry
        // application running on given host
        audioplay_descr.setLocation(argv[2]);

        // create a flow graph: readfile -> audioplay
        graph.addEdge(readfile_descr, audioplay_descr);

        // request complete graph
        cerr << "request GraphDescription from registry" << endl;
        ClientRegistry& registry = app-> getRegistry();
        registry.requestGraph(graph);
    }
```

```
// set the filename by requesting appropriate interface
cerr << "set filename to " << argv[1] << endl;
INode* readfile = graph.getINode(readfile_descr);
IFileHandler_var filehandler (readfile-> getParentObject()-> getCheckedInterface<IFileH
filehandler-> setFilename(argv[1]);

// realize and start graph
cerr << "realizeGraph()" << endl;
graph.realizeGraph();
cerr << "startGraph()" << endl;
graph.startGraph();

// wait ...
cerr << "Playing audio ... enter 'q <enter> ' to exit." << endl;
string s;
cin >> s;

// stop and release graph
cerr << "stopGraph()" << endl;
graph.stopGraph();

cerr << "releaseGraph()" << endl;
registry.releaseGraph(graph);

delete app;
return 0;
}
catch(const Exception& e) { // catch NMM-exception and print it
    cerr << e << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
catch(const SerializedException& e) { // catch serialized NMM-exception and print it
    cerr << e << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
catch(...) { // any other exception
    cerr << "something went wrong!" << endl;
```

```

// release graph
cerr << "releaseGraph()" << endl;
ClientRegistry& registry = app-> getRegistry();
registry.releaseGraph(graph);

delete app;
return 1;
}
}

```

8. hellonmm2

hellonmm2 (**modified** version of helloworld5) : distributed wav player using GraphDescription with error handling and access to 'serverregistry' : playback on local and remote host

Want a networked wav player that plays back audio on the local **and** the remote host synchronously? Simply add some lines of code: specify an additional second audio output to be located on a remote host, and add an additional edge to your graph description - that was easy, wasn't? Remember: You need to start the NMM serverregistry on the remote host. In addition, the two hosts should be synchronized by the Network Time Protocol (NTP) as described in the NMM NTP documentation.

```

// distributed wav player using GraphDescription with error handling
// and access to 'serverregistry' playback on local and remote host
// notice: you need a running 'serverregistry' on the remote host

#include <iostream>
// include NMM
#include "nmm/comm/ProxyObject.hpp"
#include "nmm/services/registry/ProxyApplication.hpp"

// include used interfaces
#include "nmm/plugins/interfaces/IFileHandler.hpp"
#include "nmm/plugins/audio/interfaces/IAudioDevice.hpp"

// import namespace
using namespace NMM;

int main(int argc, char ** argv)
{
    if(argc != 3) {
        cerr << "Usage:" << endl
             << " " << argv[0] << " <wav file> <remote host where serverregistry is running> "
             << endl;
        return 1;
    }
    cerr << argv[0] << " running ... " << endl << endl;
}

```

Hello World! Welcome to NMM Application Development

```
// disable all debug output (error, warning, message, debug)
// see documentation Debug, Error, Warning and Message Output in NMM
NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);

// the graph description for the example
GraphDescription graph;

// the NMM application for the example
NMMApplication* app = 0;

// NMM-operations may throw exception, in this simple example, all exceptions
// are just caught at the end, and the application is terminated
try {
    // create application object for example
    app = ProxyApplication::getApplication(argc, argv);

    // create the descriptions for the WavReadNode and platform specific PlaybackNode
    NodeDescription readfile_descr("WavReadNode");

    // create the description for audio sink:
    // ... some sink node
    // ... that supports the interface IAudioDevice
    // notice that this description will provide an audio sink
    // on all supported platforms
    NodeDescription audioplay_descr;
    audioplay_descr.setNodeType(INode::SINK);
    audioplay_descr.addInterface(IAudioDevice::IAudioDevice_type);

    NodeDescription audioplay_descr2;
    audioplay_descr2.setNodeType(INode::SINK);
    audioplay_descr2.addInterface(IAudioDevice::IAudioDevice_type);
    // audio renderer will be requested from serverregistry
    // application running on given host
    audioplay_descr2.setLocation(argv[2]);

    // create a flow graph: readfile -> audioplay
    graph.addEdge(readfile_descr, audioplay_descr);
    // and add additional connection to remote audio sink
    graph.addEdge(readfile_descr, audioplay_descr2);

    // request complete graph
    cerr << "request GraphDescription from registry" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.requestGraph(graph);

    // setup distributed inter-stream synchronization for synchronous playback
    cerr << "setup synchronization" << endl;
    graph.connectSinkSynchronizer();

    // set the filename by requesting appropriate interface
```

```
cerr << "set filename to " << argv[1] << endl;
INode* readfile = graph.getInode(readfile_descr);
IFileHandler_var filehandler (readfile-> getParentObject()-> getCheckedInterface<IFileH
filehandler-> setFilename(argv[1]));

// realize and start graph
cerr << "realizeGraph()" << endl;
graph.realizeGraph();
cerr << "startGraph()" << endl;
graph.startGraph();

// wait ...
cerr << "Playing audio ... enter 'q <enter> ' to exit." << endl;
string s;
cin >> s;

// stop and release graph
cerr << "stopGraph()" << endl;
graph.stopGraph();

cerr << "releaseGraph()" << endl;
registry.releaseGraph(graph);

delete app;
return 0;
}
catch(const Exception& e) { // catch NMM-exception and print it
    cerr << e << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
catch(const SerializedException& e) { // catch serialized NMM-exception and print it
    cerr << e << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
catch(...) { // any other exception
    cerr << "something went wrong!" << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
```

```
ClientRegistry& registry = app-> getRegistry();
registry.releaseGraph(graph);

delete app;
return 1;
}
}
```

9. hellonmm3

hellonmm3 (**modified** version of hellonmm1) : An optimized distributed wav player using GraphDescription with error handling and access to 'serverregistry' : playback on remote host

In the following example, we show how to reduce the amount of data that is buffered inside the part of the flow graph that is running on the remote host. For example, this is especially important for mobile devices that only provide a limited amount of memory, or to reduce latency on the network.

To test this application with different parameters, they can be specified at the command line. Since incoming buffers in downstream direction are stored in a queue before they are processed by a node, NMM allows to adjust the size for each queue by using the method **setDownstreamMaxSize**. The queue for incoming downstream buffers can be set to 1, which is obviously the smallest value for a queue.

Since audio is played back on a remote host, all buffers must be send across a network connection. Transport protocols like TCP also use a sending and receiving buffer that often stores more than 100 kByte of data. NMM provides access to all parameters of the used transport protocol, which allows to adjust all these parameters to specific requirements. Since we only need to limit the amount of data that is buffered inside the part of the flow graph that is running on the remote host, we reduce the receiving buffer of the tcp connection to 8192 bytes (8 kByte). Please notice that most Linux systems don't allow to set smaller values than 256 bytes. To configure the parameters of a network connection that receives downstream messages, we add a corresponding event to the GraphDescription.

Start hellonmm3 to get a list of possible options:

```
hellonmm3 -h
```

Using the following command line, the application is started and the audio sink nodes uses a queue of size 1 for incoming NMM buffers and the buffer size for incoming data of the TCP connection between source and sink is set to 8192 bytes:

```
hellonmm3 -i <WAV-file> -H <remote host> -p 1 -t 8192
```

The complete source code:

```
// distributed wav player using GraphDescription with error handling
// and access to 'serverregistry' playback on remote host
// notice: you need a running 'serverregistry' on the remote host

#include <iostream>
#include <getopt.h>

// include NMM
#include "nmm/comm/ProxyObject.hpp"
#include "nmm/services/registry/ProxyApplication.hpp"

// include used interfaces
#include "nmm/plugins/interfaces/IFileHandler.hpp"
#include "nmm/plugins/audio/interfaces/IAudioDevice.hpp"
#include "nmm/plugins/interfaces/IBufferSize.hpp"
#include "nmm/comm/netstrategy/ITCPControl.hpp"
#include "nmm/multimedia/IBiDirQueuedNode.hpp"

// import namespace
using namespace NMM;

void usage(const string& name) {
    cerr << "Usage :" << endl
        << " " << name << " [options] -H <Host> -i <wav file> " << endl
        << endl
        << "Options:" << endl
        << " -i <wav file>      : Sets an input file " << endl
        << " -H <Host>          : Sets remote host for audio playback " << endl
        << " -p <queue size>    : Sets the queue size of the audio sink node for incoming buf
        << " -t <buffer size>  : Sets the buffer size (in bytes) for incoming data of the TC
        << " -h                : Show this help" << endl
        << endl;
}

int main(int argc, char ** argv)
{
    if(argc < 3) {
        usage(argv[0]);
        return 1;
    }

    const char *opt_string = "i:H:p:t:h";
    struct option options [] = {
        {"input-file",    1, NULL, 'i'},
        {"host",          1, NULL, 'H'},
    }
```

```
    {"playback-queue", 1, NULL, 'p'},
    {"tcp-queue",      1, NULL, 't'},
    {"help",           0, NULL, 'h'},
    {NULL,             0, NULL, 0}
};

int ch, long_option_index;

string input_file;
string remote_host;
int playback_queue_size = 0;
int tcp_buffer_size     = 0;

while((ch=getopt_long(argc, argv, opt_string, options, &long_option_index)) != -1){
    // which option did we get?
    switch (ch) {
        case 'i':
            input_file = optarg;
            break;
        case 'H':
            remote_host = optarg;
            break;
        case 'p':
            playback_queue_size = atoi(optarg);
            break;
        case 't':
            tcp_buffer_size = atoi(optarg);
            break;
        case 'h':
            usage(argv[0]);
            exit(0);
        default:
            usage(argv[0]);
            throw Exception("Invalid option");
    }
}

cerr << argv[0] << " running ... " << endl << endl;

// disable all debug output (error, warning, message, debug)
// see documentation Debug, Error, Warning and Message Output in NMM
NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);

// the graph description for the example
GraphDescription graph;

// the NMM application for the example
NMMApplication* app = 0;

// NMM-operations may throw exception, in this simple example, all exceptions
```

```
// are just caught at the end, and the application is terminated
try {
    // create application object for example
    app = ProxyApplication::getApplication(argc, argv);

    // create the descriptions for the WavReadNode and platform specific PlaybackNode
    NodeDescription readfile_descr("WavReadNode");

    // create the description for audio sink:
    // ... some sink node
    // ... that supports the interface IAudioDevice
    // notice that this description will provide an audio sink
    // on all supported platforms
    NodeDescription audioplay_descr;
    audioplay_descr.setNodeType(INode::SINK);
    audioplay_descr.addInterface(IAudioDevice::IAudioDevice_type);

    // audio renderer will be requested from serverregistry
    // application running on given host
    audioplay_descr.setLocation(remote_host);

    // create a flow graph: readfile -> audioplay
    graph.addEdge(readfile_descr, audioplay_descr);

    // request complete graph
    cerr << "request GraphDescription from registry" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.requestGraph(graph);

    // set the filename by requesting appropriate interface
    cerr << "set filename to " << input_file << endl;
    INode* readfile = graph.getINode(readfile_descr);
    IFileHandler_var filehandler (readfile-> getParentObject()-> getCheckedInterface<IFileH
    filehandler-> setFilename(input_file);

    // set the queue size of the audio sink node for incoming NMM buffers in downstream dir
    if (playback_queue_size > 0) {
        cerr << "set queue size to " << playback_queue_size << endl;
        IBiDirQueuedNode_var audioplay_queue(graph.getINode(audioplay_descr)-> getParentObjec
        audioplay_queue-> setDownstreamMaxSize(playback_queue_size, "default");
    }

    // set the buffer size of the tcp connection for incoming data, if it is specified
    if (tcp_buffer_size > 0) {
        cerr << "set tcp buffer size " << tcp_buffer_size << endl;

        // create event for setting tcp receive buffer size
        // this event will be forwarded to binding upon creation of the flow graph
        Event event(ITCPControl::setRecvBufferSize_event, new TInValue<nmm_int32> (tcp_buffer

    // get binding description for the edge in the graph.
    ssize_t edge_id = graph.getEdgeID(readfile_descr, audioplay_descr);
    InstreamBindingDescription& binding = graph.getEdge(edge_id).getBindingDescription();
```

```
// add a TCPStrategy to make sure we are using TCP.
unsigned int ts_index = binding.addTransportStrategy("TCPStrategy", InstreamChannel::);

// add the event to the downstream configuration of the TCPStrategy.
// This means it only increases the receive buffer for
// downstream messages.
binding.getTransportStrategy(ts_index).addDownstreamConfigEvent(event);
}

// realize and start graph
cerr << "realizeGraph()" << endl;
graph.realizeGraph();
cerr << "startGraph()" << endl;
graph.startGraph();

// wait ...
cerr << "Playing audio ... enter 'q <enter>' to exit." << endl;
string s;
cin >> s;

// stop and release graph
cerr << "stopGraph()" << endl;
graph.stopGraph();

cerr << "releaseGraph()" << endl;
registry.releaseGraph(graph);

delete app;
return 0;
}
catch(const Exception& e) { // catch NMM-exception and print it
    cerr << e << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
catch(const SerializedException& e) { // catch serialized NMM-exception and print it
    cerr << e << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
}
```

```
catch(...) { // any other exception
    cerr << "something went wrong!" << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
}
```

10. hellonmm4 and hellonmm41

These two examples use a client server approach. hellonmm4 is a server streaming mp3 files using RTP multicast. hellonmm41 is a streaming client. To demonstrate the capabilities of NMM, several streaming clients can be started on different hosts and synchronized audio rendering can be enabled.

First, start the server

```
./hellonmm4 -i <mp3 file>
```

Then, start the first client

```
./hellonmm41
```

Then, you can start a second client

```
./hellonmm41 -H <hostname of the first client>
```

In order to start audio playback, you need to enter any key and press enter. The second client then starts rendering audio synchronously with the first client.

Following is the source code of hellonmm4.cpp

```
// rtp streaming server for mp3

#include <iostream>
#include <getopt.h>
#include "nmm/utils/directory_utils.hpp"
#include "nmm/comm/netstrategy/RTPAddress.hpp"
#include "nmm/comm/netstrategy/MulticastIPAddress.hpp"

// include NMM
#include "nmm/multimedia/connect.hpp"
```

```
#include "nmm/comm/ProxyObject.hpp"
#include "nmm/services/registry/ProxyApplication.hpp"
#include "nmm/services/registry/GraphDescription.hpp"

// include used interfaces
#include "nmm/plugins/interfaces/IFileHandler.hpp"
#include "nmm/plugins/net/INet.hpp"

// import namespace
using namespace NMM;

void usage(const string& name) {
    cerr << "Usage :" << endl
        << " " << name << " [options] -i <input file> -a <audio-port> " << endl
        << endl
        << "Options:" << endl
        << " -i <input file> : Sets an input file (must be in mp3 format) " << endl
        << " -a <audio-port> : Sets the multicast port to stream audio " << endl
        << " -h : Show this help" << endl
        << endl;
}

// RTP and RTCP ports to send audio data
int rtp_port = 44672;
int rtcp_port = 44673;

string input_file;

int main(int argc, char ** argv)
{
    if(argc < 3) {
        usage( argv[0] );
        return 1;
    }
    cerr << argv[0] << " running ... " << endl << endl;

    const char *opt_string = "n:i:a:hv";
    struct option options [] = {
        {"input-file", 1, NULL, 'i'},
        {"audio-port", 1, NULL, 'a'},
        {"verbose", 0, NULL, 'v'},
        {"help", 0, NULL, 'h'},
        {NULL, 0, NULL, 0}
    };

    int ch, long_option_index;
    int counter = 1;
    int verbose = 0;
    while((ch=getopt_long(argc,argv,opt_string,options,&long_option_index))!=-1){
        // which option did we get?
        switch (ch) {
            case 'i':
```

```
        input_file = getAbsolutePath(optarg);
        break;
    case 'a':
        rtp_port = atoi(optarg);
        rtcp_port = rtp_port+1;
        break;
    case 'n':
        counter = atoi(optarg);
        break;
    case 'v' :
        ++verbose;
        break;
    case 'h':
        usage(argv[0]);
        exit(0);
    default:
        usage(argv[0]);
        throw Exception("Invalid option");
    }
}

// Use default verbose level if no -v option was specified
if(verbose == 0) {
    verbose = NamedObject::getVerboseLevel();
}

// disable all debug output (error, warning, message, debug)
// see documentation Debug, Error, Warning and Message Output in NMM
// maybe turn off debug messages
if (verbose > 0) {
    NamedObject::getGlobalInstance().setErrorStream(&cout, NamedObject::ALL_LEVELS);
} else {
    NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);
}
if (verbose > 1) {
    NamedObject::getGlobalInstance().setWarningStream(&cout, NamedObject::ALL_LEVELS);
} else {
    NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);
}
if (verbose > 2) {
    NamedObject::getGlobalInstance().setMessageStream(&cout, NamedObject::ALL_LEVELS);
} else {
    NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);
}
if (verbose > 3) {
    NamedObject::getGlobalInstance().setDebugStream(&cout, NamedObject::ALL_LEVELS);
} else {
    NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);
}

// the graph description for the example
GraphDescription* graph = new GraphDescription();
```

```
// the NMM application for the example
NMMApplication* app = ProxyApplication::getApplication(argc, argv);

// create the descriptions for the GenericReadNode and NetSinkNode
NodeDescription readfile_descr("GenericReadNode");
NodeDescription netsink_descr("NetSinkNode");

// create a flow graph: readfile -> net_sink
graph-> addEdge(readfile_descr, netsink_descr);

// request complete graph
cerr << "request GraphDescription from registry" << endl;
ClientRegistry& registry = app-> getRegistry();
registry.requestGraph(*graph);

// set the filename by requesting appropriate interface
cerr << "set filename to " << input_file << endl;
INode* readfile = graph-> getINode(readfile_descr);
IFileHandler_var filehandler (readfile-> getParentObject()-> getCheckedInterface<IFileHan
filehandler-> setFilename(input_file);

// set the port number:
INode* netsink = graph-> getINode(netsink_descr);

// set multicast address and rtp/rtcp port to send audio data to
INet_var i_net (netsink-> getParentObject()-> getCheckedInterface<INet> ());
MulticastIPAddress ip_addr("239.0.0.2");
RTPAddress rtp_addr (ip_addr, rtp_port, rtcp_port);
i_net-> setTransportStrategy(&rtp_addr);
cerr << "Using RTP port : " << rtp_port << endl;
cerr << "Using RTCP port : " << rtcp_port << endl;

// realize and start graph
cerr << "realizeGraph()" << endl;
graph-> realizeGraph();
cerr << "startGraph()" << endl;
graph-> startGraph();

// wait ...
cerr << "Playing audio ... enter 'q <enter> ' to exit." << endl;
string s;
cin >> s;

// stop and release graph
cerr << "stopGraph()" << endl;
graph-> stopGraph();

cerr << "releaseGraph()" << endl;
app-> getRegistry().releaseGraph(*graph);

delete app;
return 0;
}
```

Following is the source code of hellonmm41.cpp

```
// rtp streaming client for mp3

#include <iostream>#include <getopt.h>#include "nmm/utils/directory_utils.hpp"
#include "nmm/comm/netstrategy/RTPAddress.hpp"
#include "nmm/comm/netstrategy/MulticastIPAddress.hpp"

// include NMM
#include "nmm/multimedia/connect.hpp"
#include "nmm/comm/ProxyObject.hpp"
#include "nmm/services/registry/ProxyApplication.hpp"
#include "nmm/services/registry/GraphDescription.hpp"

// include used interfaces
#include "nmm/plugins/interfaces/IFileHandler.hpp"
#include "nmm/plugins/net/INet.hpp"

// include used interfaces
#include "nmm/multimedia/sync/interfaces/ISynchronizedSink.hpp"
#include "nmm/multimedia/sync/MultiAudioVideoSynchronizer.hpp"
#include "nmm/utils/time/Clock.hpp"
#include "nmm/comm/ReferenceManager.hpp"
#include "nmm/comm/ServiceProxy.hpp"
#include "nmm/comm/netstrategy/TCPAddress.hpp"
#include "nmm/comm/netstrategy/TCPAcceptor.hpp"
#include "nmm/multimedia/IBiDirQueuedNode.hpp"

// import namespace
using namespace NMM;

void usage(const string& name) {
    cerr << "Usage :" << endl
        << " " << name << " [options] " << endl
        << endl
        << "Options:" << endl
        << " -a <audio-port> : Sets the multicast port to stream audio " << endl
        << " -s : Disable synchronization " << endl
        << " -q <millisecs> : Set queue duration in milliseconds" << endl
        << " -p <millisecs> : Set prefetching duration in milliseconds" << endl
        << " -H <hostname> : Synchronize to specified host " << endl
        << " -h : Show this help" << endl
        << endl;
}

int main(int argc, char ** argv)
{
    // RTP and RTCP ports to connect to
    int rtp_port = 44672;
    int rtcp_port = 44673;

    MultiAudioVideoSynchronizer* synchronizer = 0;
```

```
ISinkSynchronizer_var      i_sink_synchronizer;

bool master = true;
TCPAcceptor* acceptor = 0;
bool is_synchronization_enabled = true;

Interval downstream_max_duration(0,300000000); // 300ms
Interval prefetch_queue_duration(0,200000000); // 200ms

string host;

cerr << argv[0] << " running ... " << endl << endl;

const char *opt_string = "H:a:sp:q:vh";
struct option options [] = {
    {"audio-port",    1,NULL,'a'},
    {"host",          1,NULL,'H'},
    {"nosync",        0,NULL,'s'},
    {"queue",         1,NULL,'q'},
    {"prefetch",      1,NULL,'p'},
    {"verbose",       0,NULL,'v'},
    {"help",          0,NULL,'h'},
    {NULL,            0,NULL,0}
};

int ch,long_option_index;
int verbose = 0;
while((ch=getopt_long(argc,argv,opt_string,options,&long_option_index))!=-1){
    // which option did we get?
    switch (ch) {
        case 'a':
            rtp_port = atoi(optarg);
            rtcp_port = rtp_port+1;
            break;
        case 'H':
            host = optarg;
            master = false;
            break;
        case 's':
            is_synchronization_enabled = false;
            break;
        case 'q':
            {
                unsigned int milliseconds = atoi(optarg);
                downstream_max_duration = Interval(milliseconds / 1000, milliseconds * 1000000);
                break;
            }
        case 'p':
            {
                unsigned int milliseconds = atoi(optarg);
                prefetch_queue_duration = Interval(milliseconds / 1000, milliseconds * 1000000);
                break;
            }
    }
}
```

```
    case 'v':
        ++verbose;
        break;
    case 'h':
        usage(argv[0]);
        exit(0);
    default:
        usage(argv[0]);
        throw Exception("Invalid option");
}
}

// Use default verbose level if no -v option was specified
if(verbose == 0) {
    verbose = NamedObject::getVerboseLevel();
}

// disable all debug output (error, warning, message, debug)
// see documentation Debug, Error, Warning and Message Output in NMM
// maybe turn off debug messages
if (verbose > 0 ) {
    NamedObject::getGlobalInstance().setErrorStream(&cout, NamedObject::ALL_LEVELS);
} else {
    NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);
}
if (verbose > 1) {
    NamedObject::getGlobalInstance().setWarningStream(&cout, NamedObject::ALL_LEVELS);
} else {
    NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);
}
if (verbose > 2) {
    NamedObject::getGlobalInstance().setMessageStream(&cout, NamedObject::ALL_LEVELS);
} else {
    NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);
}
if (verbose > 3) {
    NamedObject::getGlobalInstance().setDebugStream(&cout, NamedObject::ALL_LEVELS);
} else {
    NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);
}

// the graph description for the example
GraphDescription graph;

// the NMM application for the example
NMMApplication* app = ProxyApplication::getApplication(argc, argv);

if (master) {
    acceptor = new TCPAcceptor(9000);
}

// create the descriptions for the GenericReadNode, MPEGAudioDecodeNode, and PlaybackNode
NodeDescription netsource_descr("NetSourceNode");
```

```
NodeDescription mpegdec_descr("MPEGAudioDecodeNode");
NodeDescription playback_descr("ALSAPlaybackNode");

Format in_format("audio/mpeg");

// define the audio data format we want to receive
Format mp3_format (Format::audio_mpeg_type);
mp3_format.addIntParamValue(Format::sample_rate_param, 44100) ;
mp3_format.addIntParamValue(Format::channels_param, 2);
mp3_format.addIntParamValue(Format::layer_param, 3);
mp3_format.addIntParamValue(Format::bitpersample_param, 16);

// create a flow graph: readfile -> net_sink
graph.addEdge(netsource_descr, mpegdec_descr, &mp3_format);
graph.addEdge(mpegdec_descr, playback_descr);

// request complete graph
cerr << "request GraphDescription from registry" << endl;
ClientRegistry& registry = app-> getRegistry();
registry.requestGraph(graph);

// Set queue and prefetching durations
INode* playback = graph.getINode(playback_descr);
IBidirQueuedNode_var iqueue(playback-> getParentObject()-> getCheckedInterface<IBidirQueuedNode>());
if(downstream_max_duration != Interval()) {
    cerr << downstream_max_duration << endl;
    iqueue-> setDownstreamMaxDuration(downstream_max_duration);
}
if(prefetch_queue_duration != Interval()) {
    cerr << prefetch_queue_duration << endl;
    playback-> setPrefetchQueueDuration(prefetch_queue_duration);
    graph.setPrefetchingEnabled(true);
}

// Try to allocate or to request a Synchronizer
try {
    if ( master ) {
        // Create a synchronizer that can also be used by other applications
        synchronizer = new MultiAudioVideoSynchronizer();
        i_sink_synchronizer.reset( synchronizer-> getInterface<ISinkSynchronizer> ( ) );

        // Register the Synchronizer as service
        SReferenceManager::getInstance().registerService( "SinkSynchronizer", synchronizer );
    } else {
        ServiceProxy service_proxy;

        try {
            // Get synchronizer of host given per commandline option.
            TCPAddress tcp_address( host, 9000 );
            service_proxy.connectTo(tcp_address, "SinkSynchronizer");

            // Get the interface of synchronizer on given host
            i_sink_synchronizer.reset( service_proxy.getCheckedInterface<ISinkSynchronizer> ( ) )
        }
    }
}
```

```
    } catch (const BindingException& e) {
        GLOBAL_ERROR_STREAM("Could not connect to location: " << host << ":" << 9000 << endl;
        cerr << e << endl;
        return 0;
    }
    catch(...) {
        throw;
    }
}
}
catch ( const SerializedException& e ) {
    cerr << e << endl;
}
catch ( const Exception& e ) {
    cerr << e << endl;
}
catch(...) {
    cerr << "Unknown error" << endl;
}

// Set RTP Payloadtype on NetSourceNode
INode* netsource = graph.getINode(netsource_descr);
INet_var          i_net (netsource-> getParentObject()-> getCheckedInterface<INet> ());
i_net-> setPayloadType(14);

// set Multicast address to connect to

MulticastIPAddress ip_addr("239.0.0.2");
RTPAddress          rtp_addr (ip_addr, rtp_port, rtcp_port);

i_net-> setTransportStrategy(&rtp_addr);
i_net-> setPayloadType(14);

cerr << "Using Port " << rtp_port << endl;
cerr << "Realize Graph " << endl;
graph.realizeGraph();
cerr << "Realize Graph " << endl;

Clock clock;
Time start_time;
Time graph_started;

if (! master ) {
    cerr << "Enter any key and press <Enter> to start 'listen in' " << endl;
    string s;
    cin >> s;
}

start_time = clock.getTime();

try {
```

```
if(is_synchronization_enabled) {
    // Use the allocated synchronizer
    ISession* i_session = graph.getISession();
    i_session-> setSinkSynchronizer( *i_sink_synchronizer.get() );

    graph.connectSinkSynchronizer();
}
else {
    graph.setSinkSynchronizationEnabled(false);
}

// start graph
cerr << "startGraph()" << endl;
graph.startGraph();
graph_started = clock.getTime();
if (! master ) {
    cerr << "Start Time : " << start_time << endl;
    cerr << "Graph Started : " << graph_started << endl;
}

}
catch (const SerializedException& e) {
    cerr << e << endl;
}
catch (const Exception& e) {
    cerr << e << endl;
}
catch(...) {
    cerr << "Unknown error" << endl;
}

// wait ...

cerr << "Playing audio ... enter 'q <enter> ' to exit." << endl;
string s;

while (1) {
    cin >> s;
    if ( s == "q" ) {
        break;
    }
}
// stop and release graph
cerr << "stopGraph()" << endl;
graph.stopGraph();

cerr << "releaseGraph()" << endl;
registry.releaseGraph(graph);
SReferenceManager::getInstance().removeService( "SinkSynchronizer");

// do cleanup
delete synchronizer;
delete acceptor;
```

```
delete app;
return 0;
}
```

11. hellonmm5 and hellonmm51

These two examples use a client server approach. hellonmm5 is a server streaming video files using RTP multicast. hellonmm51 is a streaming client. To demonstrate the capabilities of NMM, several streaming clients can be started on different hosts and synchronized video rendering can be enabled.

First, start the server

```
./hellonmm5 -i <video file>
```

Then, start the first client

```
./hellonmm51
```

Then, you can start a second client

```
./hellonmm51 -H <hostname of the first client>
```

In order to start video playback, you need to enter any key and press enter. The second client then starts rendering video synchronously with the first client.

Following is the source code of hellonmm5.cpp

```
// rtp streaming server for video

#include <iostream>#include <getopt.h>#include "nmm/utils/directory_utils.hpp"
#include "nmm/comm/netstrategy/RTPAddress.hpp"
#include "nmm/comm/netstrategy/MulticastIPAddress.hpp"
// include NMM
#include "nmm/multimedia/connect.hpp"
#include "nmm/comm/ProxyObject.hpp"
#include "nmm/services/registry/ProxyApplication.hpp"
#include "nmm/services/registry/GraphDescription.hpp"

// include used interfaces
#include "nmm/plugins/interfaces/IFileHandler.hpp"
#include "nmm/plugins/net/INet.hpp"

// import namespace
using namespace NMM;
```

```
void usage(const string& name) {
    cerr << "Usage :" << endl
        << " " << name << " [options] -i <input-file> -a <audio-port> -v <video-port> " << endl
        << endl
        << "Options:" << endl
        << " -i <input file> : Sets an input file " << endl
        << " -a <audio-port> : Sets the multicast port to stream audio " << endl
        << " -v <video-port> : Sets the multicast port to stream video " << endl
        << " -V : enable verbose mode" << endl
        << " -h : Show this help" << endl
        << " -n : specifies the number of streamed videos" << endl
        << endl;
}

int audio_rtp_port = 0;
int audio_rtcp_port = 0;

int video_rtp_port = 0;
int video_rtcp_port = 0;

int transcoded_video_rtp_port = 0;
int transcoded_video_rtcp_port = 0;

string input_file;

string transcoding_host = "localhost";

void start_graph(bool /*ac3_mpeg*/, GraphDescription& /*graph*/, NMMApplication* /*app*/) {
}

int main(int argc, char ** argv)
{
    if(argc < 4) {
        usage( argv[0] );
        return 1;
    }

    bool ac3_mpeg = false;

    const char *opt_string = "n:i:a:v:dVh";
    struct option options [] = {
        {"input-file", 1, NULL, 'i'},
        {"audio-port", 1, NULL, 'a'},
        {"video-port", 1, NULL, 'v'},
        {"Verbose", 0, NULL, 'V'},
        {"dvd", 0, NULL, 'd'},
        {"help", 0, NULL, 'h'},
    }
```

```
{NULL,          0, NULL, 0}
};

int ch, long_option_index;
int counter = 1;
int verbose = 0;
while((ch=getopt_long(argc,argv,opt_string,options,&long_option_index))!=-1){
    // which option did we get?
    switch (ch) {
        case 'i':
            input_file = getAbsolutePath(optarg);
            break;
        case 'a':
            audio_rtp_port = atoi(optarg);
            audio_rtcp_port = audio_rtp_port+1;
            break;
        case 'v':
            video_rtp_port = atoi(optarg);
            video_rtcp_port = video_rtp_port+1;
            transcoded_video_rtp_port =video_rtp_port+2;
            transcoded_video_rtcp_port = video_rtp_port+3;
            break;
        case 'V':
            ++verbose;
            break;
        case 'd':
            ac3_mpeg = true;
            break;
        case 'n':
            counter = atoi(optarg);
            break;
        case 'h':
            usage(argv[0]);
            exit(0);
        default:
            usage(argv[0]);
            throw Exception("Invalid option");
    }
}

cerr << argv[0] << " running ... " << endl << endl;

// Use default verbose level if no -v option was specified
if(verbose == 0) {
    verbose = NamedObject::getVerboseLevel();
}

// disable all debug output (error, warning, message, debug)
// see documentation Debug, Error, Warning and Message Output in NMM
NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);
```

```
if (verbose > 0) {
    NamedObject::getGlobalInstance().setErrorStream(&cout, NamedObject::ALL_LEVELS);
}
if (verbose > 1) {
    NamedObject::getGlobalInstance().setWarningStream(&cout, NamedObject::ALL_LEVELS);
}
if (verbose > 2) {
    NamedObject::getGlobalInstance().setMessageStream(&cout, NamedObject::ALL_LEVELS);
}
if (verbose > 3) {
    NamedObject::getGlobalInstance().setDebugStream(&cout, NamedObject::ALL_LEVELS);
}

// the NMM application for the example
NMMApplication* app = ProxyApplication::getApplication(argc, argv);

// the description of the graph we use
GraphDescription* graph = new GraphDescription();

// NodeDescriptions for streaming flowgraph without transcoding.
NodeDescription readfile_descr;
readfile_descr.setNodeName("GenericReadNode");

NodeDescription mpeg_demux("MPEGDemuxNode");
NodeDescription mpeg_vsh("MPEGVSHDetectionNode");

NodeDescription ac3_dec("AC3DecodeNode");
NodeDescription mpeg_enc("MPEGAudioEncodeNode");

NodeDescription audio_netsink_descr("NetSinkNode");
NodeDescription video_netsink_descr("NetSinkNode");

// NodeDescriptions for streaming flowgraph with transcoding video
NodeDescription mpeg_dec("MPEGVideoDecodeNode");
mpeg_dec.setLocation(transcoding_host);

NodeDescription deinterlace_descr("YUVDeInterlaceNode");
deinterlace_descr.setLocation(transcoding_host);

NodeDescription scaler ("VideoScalerNode");
scaler.setLocation(transcoding_host);

NodeDescription ffmpeg_enc("FFMpegEncodeNode");
ffmpeg_enc.setLocation(transcoding_host);

NodeDescription video_transcode_netsink("NetSinkNode");
video_transcode_netsink.setLocation(transcoding_host);

// create a flow graph: readfile -> net_sink
graph-> addEdge(readfile_descr, mpeg_demux);
```

```
if ( ac3_mpeg ) { // build graph with ac3 audio

    graph-> addEdge(mpeg_demux, "ac3_audio0", ac3_dec);
    Format enc_format(Format::audio_mpeg_type);
    enc_format.addIntParamValue(Format::bitrate_param, 320000);
    graph-> addEdge(ac3_dec, mpeg_enc);

    graph-> addEdge(mpeg_enc, "default", mpeg_vsh, "audio_input_stream", &enc_format);
    graph-> addEdge(mpeg_vsh, "audio_output_stream", audio_netsink_descr);

    graph-> addEdge(mpeg_demux, "mpeg_video0", mpeg_vsh, "video_input_stream");
    graph-> addEdge(mpeg_vsh, "video_output_stream", video_netsink_descr);

} else { // build graph with mpeg audio

    graph-> addEdge(mpeg_demux, "mpeg_audio0", mpeg_vsh, "audio_input_stream");
    graph-> addEdge(mpeg_demux, "mpeg_video0", mpeg_vsh, "video_input_stream");
    graph-> addEdge(mpeg_vsh, "audio_output_stream", audio_netsink_descr);
    graph-> addEdge(mpeg_vsh, "video_output_stream", video_netsink_descr);

    // define the formats for transcoding nodes
    Format video_format(Format::video_mpeg_type);
    video_format.addIntParamValue(Format::version_param, 2);
    video_format.addIntParamValue(Format::x_resolution_param, 720);
    video_format.addIntParamValue(Format::y_resolution_param, 540);
    video_format.addIntParamValue(Format::ratio_x_param, 4);
    video_format.addIntParamValue(Format::ratio_y_param, 3);
    video_format.addFloatParamValue(Format::framerate_param, 25.0);

    Format scaler_format(Format::video_raw_type);
    scaler_format.addIntParamValue(Format::x_resolution_param, 320);
    scaler_format.addIntParamValue(Format::y_resolution_param, 240);
    scaler_format.addIntParamValue(Format::ratio_x_param, 320);
    scaler_format.addIntParamValue(Format::ratio_y_param, 240);
    scaler_format.addFloatParamValue(Format::framerate_param, 25.0);

    Format video_encoder_format(Format::video_h263plus_type);
    video_encoder_format.addIntParamValue(Format::bitrate_param, 1000000);

    // Setup the transcoding flow graph
    graph-> addEdge(mpeg_vsh, "video_output_stream", mpeg_dec, &video_format);
    graph-> addEdge(mpeg_dec, scaler);
    graph-> addEdge(scaler, ffmpeg_enc, &scaler_format);
    graph-> addEdge(ffmpeg_enc, video_transcode_netsink, &video_encoder_format);
}

// request complete graph
cerr << "request GraphDescription from registry" << endl;
ClientRegistry& registry = app-> getRegistry();
registry.requestGraph(*graph);
```

```
// set the filename by requesting appropriate interface
cerr << "set filename to " << input_file << endl;
INode* readfile = graph-> getINode(readfile_descr);
IFileHandler_var filehandler (readfile-> getParentObject()-> getCheckedInterface<IFileHan
filehandler-> setFilename(input_file);

// define and set the Multicast IP adres for audio
INode* netsink = graph-> getINode(audio_netsink_descr);
INet_var i_net (netsink-> getParentObject()-> getCheckedInterface<INet> ());
MulticastIPAddress ip_addr("239.0.0.2");
RTPAddress rtp_addr (ip_addr, audio_rtp_port, audio_rtcp_port);
i_net-> setTransportStrategy(&rtp_addr);

cerr << "Using audio rtp port : " << audio_rtp_port << endl;
cerr << "Using audio rtcp port : " << audio_rtcp_port << endl;

// define and set the Multicast IP adres for video
INode* netsink2 = graph-> getINode(video_netsink_descr);
INet_var i_net2 (netsink2-> getParentObject()-> getCheckedInterface<INet> ());
MulticastIPAddress ip_addr2("239.0.0.2");
RTPAddress rtp_addr2 (ip_addr2, video_rtp_port, video_rtcp_port);
i_net2-> setTransportStrategy(&rtp_addr2);

cerr << "Using video rtp port : " << video_rtp_port << endl;
cerr << "Using video rtcp port : " << video_rtcp_port << endl;

// define and set the Multicast IP adres for transcoded video
INode* netsink3 = graph-> getINode(video_transcode_netsink);
INet_var i_net3 (netsink3-> getParentObject()-> getCheckedInterface<INet> ());
MulticastIPAddress ip_addr3("239.0.0.2");
RTPAddress rtp_addr3 (ip_addr3, transcoded_video_rtp_port, transcoded_video_rtcp_port);
i_net3-> setTransportStrategy(&rtp_addr3);

cerr << "Using transcoded video rtp port : " << transcoded_video_rtp_port << endl;
cerr << "Using transcoded video rtcp port : " << transcoded_video_rtcp_port << endl;

// realize and start graph
cerr << "realizeGraph()" << endl;
graph-> realizeGraph();
cerr << "startGraph()" << endl;
graph-> startGraph();

// wait ...
cerr << "Playing audio ... enter 'q <enter> ' to exit." << endl;
string s;
cin >> s;

// stop and release graph
```

```
cerr << "stopGraph()" << endl;
graph-> stopGraph();

cerr << "releaseGraph()" << endl;
app-> getRegistry().releaseGraph(*graph);

delete app;
return 0;
}
```

Following is the source code of `hellonmm51.cpp`

```
// rtp streaming client for video

#include <iostream>
#include <getopt.h>
#include "nmm/utils/directory_utils.hpp"
#include "nmm/comm/netstrategy/RTPAddress.hpp"
#include "nmm/comm/netstrategy/MulticastIPAddress.hpp"
// include NMM
#include "nmm/multimedia/connect.hpp"
#include "nmm/comm/ProxyObject.hpp"
#include "nmm/services/registry/ProxyApplication.hpp"
#include "nmm/services/registry/GraphDescription.hpp"

// include used interfaces
#include "nmm/plugins/interfaces/IFileHandler.hpp"
#include "nmm/plugins/net/INet.hpp"

// include used interfaces
#include "nmm/multimedia/sync/interfaces/ISynchronizedSink.hpp"
#include "nmm/multimedia/sync/MultiAudioVideoSynchronizer.hpp"
#include "nmm/utils/time/Clock.hpp"
#include "nmm/comm/ReferenceManager.hpp"
#include "nmm/comm/ServiceProxy.hpp"
#include "nmm/comm/netstrategy/TCPAddress.hpp"
#include "nmm/comm/netstrategy/TCPAcceptor.hpp"

// import namespace
using namespace NMM;

void usage(const string& name) {
    cerr << "Usage :" << endl
        << " " << name << " [options] -a <audio-port> -v <video-port> " << endl
        << endl
        << "Options:" << endl
        << " -a <audio-port> : Sets the multicast port to stream audio " << endl
        << " -v <video-port> : Sets the multicast port to stream video " << endl
        << " -V : increase verbose mode " << endl
}
```

```
<< " -t                : receive transcoded video " << endl
<< " -H <Host>         : connects to specified host to request a Synchronizer" << endl
<< " -h : Show this help" << endl
<< endl;
}

int main(int argc, char ** argv)
{
    if(argc < 3) {
        usage(argv[0]);
        return 1;
    }

    cerr << argv[0] << " running ... " << endl << endl;

    int audio_rtp_port = 0;
    int audio_rtcp_port = 0;
    int video_rtp_port = 0;
    int video_rtcp_port = 0;
    int transcoded_video_rtp_port = 0;
    int transcoded_video_rtcp_port = 0;

    bool ac3_mpeg = false;

    string host;
    bool is_master = true;

    MultiAudioVideoSynchronizer* synchronizer = 0;
    ISinkSynchronizer_var        i_sink_synchronizer;
    TCPAcceptor* acceptor = 0;

    bool build_transcoding_graph = false;

    const char *opt_string = "H:a:v:dVth";
    struct option options [] = {
        {"audio-port",    1,NULL,'a'},
        {"video-port"    ,1,NULL,'v'},
        {"verbose",      0,NULL,'V'},
        {"transcoding",  0,NULL,'t'},
        {"dvd",          0,NULL,'d'},
        {"host",         1,NULL,'H'},
        {"help",         0,NULL,'h'},
        {NULL,           0,NULL,0}
    };

    int ch,long_option_index;
    int verbose = 0;

    while((ch=getopt_long(argc,argv,opt_string,options,&long_option_index))!=-1){
        // which option did we get?
        switch (ch) {
            case 'a':
```

```
        audio_rtp_port = atoi(optarg);
        audio_rtcp_port = audio_rtp_port+1;
        break;
    case 'v':
        video_rtp_port = atoi(optarg);
        video_rtcp_port = video_rtp_port+1;
        transcoded_video_rtp_port = video_rtp_port+2;
        transcoded_video_rtcp_port = video_rtp_port+3;
        break;
    case 'd':
        ac3_mpeg = true;
        break;
    case 'V':
        ++verbose;
        break;
    case 'H':
        host = optarg;
        is_master = false;
        break;
    case 't':
        build_transcoding_graph = true;
        break;
    case 'h':
        usage(argv[0]);
        exit(0);
    default:
        usage(argv[0]);
        throw Exception("Invalid option");
}
}

// Use default verbose level if no -v option was specified
if(verbose == 0) {
    verbose = NamedObject::getVerboseLevel();
}

// disable all debug output (error, warning, message, debug)
// see documentation Debug, Error, Warning and Message Output in NMM
NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);
if (verbose > 0) {
    NamedObject::getGlobalInstance().setErrorStream(&cout, NamedObject::ALL_LEVELS);
}
if (verbose > 1) {
    NamedObject::getGlobalInstance().setWarningStream(&cout, NamedObject::ALL_LEVELS);
}
if (verbose > 2) {
    NamedObject::getGlobalInstance().setMessageStream(&cout, NamedObject::ALL_LEVELS);
}
if (verbose > 3) {
    NamedObject::getGlobalInstance().setDebugStream(&cout, NamedObject::ALL_LEVELS);
}
```

```
}

// the graph description for the example
GraphDescription graph;

// the NMM application for the example
NMMApplication* app = ProxyApplication::getApplication(argc, argv);

// create the descriptions
NodeDescription audio_netsource_descr("NetSourceNode");
NodeDescription video_netsource_descr("NetSourceNode");
NodeDescription mpeg_video_dec("MPEGVideoDecodeNode");
NodeDescription ffmpeg_dec("FFMpegDecodeNode");
NodeDescription display("XDisplayNode");
NodeDescription mpeg_audio_dec("MPEGAudioDecodeNode");
NodeDescription playback_dec("ALSAPlaybackNode");

// create a flow graph:

// define receiving audio format
Format mp3_format (Format::audio_mpeg_type);
mp3_format.addIntParamValue(Format::sample_rate_param, 48000) ;
mp3_format.addIntParamValue(Format::channels_param, 2);
mp3_format.addIntParamValue(Format::layer_param, 3);
mp3_format.addIntParamValue(Format::bitpersample_param, 16);

// build audio part of graph
graph.addEdge(audio_netsource_descr, mpeg_audio_dec, &mp3_format);
graph.addEdge(mpeg_audio_dec, playback_dec);

// define receiving video format
Format video_transcode_format(Format::video_h263plus_type);
video_transcode_format.addIntParamValue(Format::x_resolution_param, 320);
video_transcode_format.addIntParamValue(Format::y_resolution_param, 240);
video_transcode_format.addIntParamValue(Format::ratio_x_param, 320);
video_transcode_format.addIntParamValue(Format::ratio_y_param, 240);
video_transcode_format.addFloatParamValue(Format::framerate_param, 25.0);

// define alternative receiving video format
Format video_format(Format::video_mpeg_type);
video_format.addIntParamValue(Format::version_param, 2);
video_format.addIntParamValue(Format::x_resolution_param, 720);
video_format.addIntParamValue(Format::y_resolution_param, 540);
video_format.addIntParamValue(Format::ratio_x_param, 4);
video_format.addIntParamValue(Format::ratio_y_param, 3);
video_format.addFloatParamValue(Format::framerate_param, 25.0);

// build video part of graph
if (!build_transcoding_graph) {
    graph.addEdge(video_netsource_descr, mpeg_video_dec, &video_format);
    graph.addEdge(mpeg_video_dec, display);
} else {
```

```
graph.addEdge(video_netsource_descr, ffmpeg_dec, &video_transcode_format);
graph.addEdge(ffmpeg_dec, display);
}

// request complete graph
cerr << "request GraphDescription from registry" << endl;
ClientRegistry& registry = app-> getRegistry();
registry.requestGraph(graph);

// Try to allocate or to request a Synchronizer
try {
    if ( is_master ) {
        acceptor = new TCPAcceptor(9000);

        cerr << "master application " << endl;
        // Create a synchronizer that can also be used by other applications
        synchronizer = new MultiAudioVideoSynchronizer();
        i_sink_synchronizer.reset( synchronizer-> getInterface<ISinkSynchronizer> ( ) );

        // Register the Synchronizer as service
        SReferenceManager::getInstance().registerService( "SinkSynchronizer", synchronizer );
    } else {
        cerr << "Connect to host " << host << endl;
        ServiceProxy service_proxy;

        try {
            TCPAddress tcp_address( host, 9000 );
            service_proxy.connectTo(tcp_address, "SinkSynchronizer");

            // Get the interface of the remote rtp client
            i_sink_synchronizer.reset( service_proxy.getCheckedInterface<ISinkSynchronizer> ( )

        } catch (const BindingException& e) {
            GLOBAL_ERROR_STREAM("Could not connect to location: " << host << ":" << 9000 << endl;
            cerr << e << endl;
            return 0;
        }
        catch(...) {
            throw;
        }
    }
}
catch ( const SerializedException& e ) {
    cerr << e << endl;
}
catch ( const Exception& e ) {
    cerr << e << endl;
}
catch(...) {
    cerr << "Unknown error" << endl;
}

Clock clock;
```

```
Time start_time;
Time graph_started;

// set the audio rtp payload type
INode* netsource1 = graph.getINode(audio_netsource_descr);
INet_var i_net (netsource1-> getParentObject()-> getCheckedInterface<INet> ());
i_net-> setPayloadType(14); //MP3 Audio

// set the video rtp payload type
INode* netsource2 = graph.getINode(video_netsource_descr);
INet_var i_net2 (netsource2-> getParentObject()-> getCheckedInterface<INet> ());
if ( build_transcoding_graph ) {
    i_net2-> setPayloadType(34);
} else {
    i_net2-> setPayloadType(32);
}

// define Multicast IP address for audio
MulticastIPAddress ip_addr("239.0.0.2");
RTPAddress rtp_addr (ip_addr, audio_rtp_port, audio_rtcp_port);
i_net-> setTransportStrategy(&rtp_addr);

// define Multicast IP address for video
MulticastIPAddress ip_addr2("239.0.0.2");
if ( build_transcoding_graph ) {
    RTPAddress rtp_addr2 (ip_addr2, transcoded_video_rtp_port, transcoded_video_rtcp_port);
    i_net2-> setTransportStrategy(&rtp_addr2);
} else {
    RTPAddress rtp_addr2 (ip_addr2, video_rtp_port, video_rtcp_port);
    i_net2-> setTransportStrategy(&rtp_addr2);
}

cerr << "realizeGraph()" << endl;
graph.realizeGraph();

if (! is_master ) {
    cerr << "Enter any key and press <Enter>  to start 'listen in' " << endl;
    string s;
    cin >> s;
}

start_time = clock.getTime();

try {
    // Use the allocated synchronizer
    ISession* i_session = graph.getISession();
    i_session-> setSinkSynchronizer( *i_sink_synchronizer.get() );
    graph.connectSinkSynchronizer();

    // start graph
```

```
    cerr << "startGraph()" << endl;
    graph.startGraph();
    graph_started = clock.getTime();
    if (! is_master ) {
        cerr << "Start Time : " << start_time << endl;
        cerr << "Graph Started : " << graph_started << endl;
    }
}
catch (const SerializedException& e) {
    cerr << e << endl;
}
catch (const Exception& e) {
    cerr << e << endl;
}
catch(...) {
    cerr << "Unknown error" << endl;
}

// wait ...
cerr << "Playing audio ... enter 'q <enter> ' to exit." << endl;
string s;
cin >> s;

// stop and release graph
cerr << "stopGraph()" << endl;
graph.stopGraph();

cerr << "releaseGraph()" << endl;
registry.releaseGraph(graph);

// do cleanup
delete synchronizer;
delete acceptor;
delete app;
return 0;
}
```

12. hellonmm10 and hellonmm101

These two examples use a client server approach. hellonmm10 is a server streaming MPEG TS files using RTP multicast. hellonmm101 is a streaming client. To demonstrate the capabilities of NMM, several streaming clients can be started on different hosts and synchronized video rendering can be enabled.

First, start the server

```
./hellonmm10 -i <video file>
```

Then, start the first client

```
./hellowmm101
```

Then, you can start a second client

```
./hellowmm101 -H <hostname of the first client>
```

In order to start video playback, you need to enter any key and press enter. The second client then starts rendering video synchronously with the first client.

Following is the source code of `hellowmm10.cpp`

```
// rtp streaming server for mpeg2 ts

#include <iostream>
#include <getopt.h>
#include "nmm/utils/directory_utils.hpp"
#include "nmm/comm/netstrategy/RTPAddress.hpp"
#include "nmm/comm/netstrategy/MulticastIPAddress.hpp"
// include NMM
#include "nmm/multimedia/connect.hpp"
#include "nmm/comm/ProxyObject.hpp"
#include "nmm/services/registry/ProxyApplication.hpp"
#include "nmm/services/registry/GraphDescription.hpp"

// include used interfaces
#include "nmm/plugins/interfaces/IFileHandler.hpp"
#include "nmm/plugins/net/INet.hpp"
#include "nmm/plugins/interfaces/IBufferSize.hpp"

// import namespace
using namespace NMM;

void usage(const string& name) {
    cerr << "Usage :" << endl
         << " " << name << " [options] <input file> " << endl
         << endl
         << "Options:" << endl
         << " -v : Increase verbosity, can be given up to four times, i.e. -vvvv" << endl
         << " -h : Show this help" << endl
         << endl;
}

int main(int argc, char ** argv)
{
    if(argc < 2) {
        usage( argv[0] );
        return 1;
    }
    cerr << argv[0] << " running ... " << endl << endl;
}
```

```
// RTP and RTCP ports to connect to
int rtp_port = 1234;
int rtcp_port = 1235;

const char *opt_string = "hv";
struct option options [] = {
    {"verbose",      0, NULL, 'v'},
    {"help",         0, NULL, 'h'},
    {NULL,           0, NULL, 0}
};

int ch, long_option_index;
int verbose = 0;
while((ch=getopt_long(argc,argv,opt_string,options,&long_option_index))!=-1){
    // which option did we get?
    switch (ch) {
        case 'v' :
            ++verbose;
            break;
        case 'h':
            usage(argv[0]);
            exit(0);
        default:
            usage(argv[0]);
            throw Exception("Invalid option");
    }
}

string input_file = argv[optind];

// Use default verbose level if no -v option was specified
if(verbose == 0) {
    verbose = NamedObject::getVerboseLevel();
}

// disable all debug output (error, warning, message, debug)
// see documentation Debug, Error, Warning and Message Output in NMM
// maybe turn off debug messages
if (verbose > 0) {
    NamedObject::getGlobalInstance().setErrorStream(&cout, NamedObject::ALL_LEVELS);
} else {
    NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);
}

if (verbose > 1) {
    NamedObject::getGlobalInstance().setWarningStream(&cout, NamedObject::ALL_LEVELS);
} else {
    NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);
}

if (verbose > 2) {
    NamedObject::getGlobalInstance().setMessageStream(&cout, NamedObject::ALL_LEVELS);
} else {
    NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);
}
```

```
}
if (verbose > 3) {
    NamedObject::getGlobalInstance().setDebugStream(&cout, NamedObject::ALL_LEVELS);
} else {
    NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);
}

// the graph description for the example
GraphDescription graph;

// the NMM application for the example
NMMApplication* app = ProxyApplication::getApplication(argc, argv);

try {

    // create the descriptions for the GenericReadNode and NetSinkNode
    NodeDescription readfile_descr("GenericReadNode");
    NodeDescription netsink_descr("NetSinkNode");

    // create a flow graph: readfile -> net_sink
    graph.addEdge(readfile_descr, netsink_descr);

    // request complete graph
    cerr << "Request GraphDescription from registry" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.requestGraph(graph);

    // set the filename by requesting appropriate interface
    cerr << "Set input filename to " << input_file << endl;
    INode* readfile = graph.getINode(readfile_descr);
    IFileHandler_var filehandler (readfile-> getParentObject()-> getCheckedInterface<IFileH
    filehandler-> setFilename(input_file);

    IBufferSize_var buffersize(readfile-> getParentObject()-> getCheckedInterface<IBufferSi
    buffersize-> setBufferSize(1316);

    // set multicast address and rtp/rtcp port to send audio data to
    INode* netsink = graph.getINode(netsink_descr);
    INet_var i_net (netsink-> getParentObject()-> getCheckedInterface<INet> ());
    MulticastIPAddress ip_addr("239.255.42.42");
    RTPAddress rtp_addr (ip_addr, rtp_port, rtcp_port);
    i_net-> setTransportStrategy(&rtp_addr);

    // realize and start graph
    cerr << "realizeGraph()" << endl;
    graph.realizeGraph();

    cerr << "startGraph()" << endl;
    graph.startGraph();

    // wait ...
    cerr << "Sending MPEG2 TS ... enter 'q <enter> ' to exit." << endl;
    string s;
```

```
cin >> s;

// stop and release graph
cerr << "stopGraph()" << endl;
graph.stopGraph();

cerr << "releaseGraph()" << endl;
app-> getRegistry().releaseGraph(graph);

delete app;
return 0;
}
catch(const Exception& e) { // catch NMM-exception and print it
    cerr << e << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
catch(const SerializedException& e) { // catch serialized NMM-exception and print it
    cerr << e << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
catch(...) { // any other exception
    cerr << "something went wrong!" << endl;

    // release graph
    cerr << "releaseGraph()" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.releaseGraph(graph);

    delete app;
    return 1;
}
}
```

Following is the source code of `hellonmm101.cpp`

```
// rtp streaming client for mpeg2 ts; dumps received stream to file
```

```
#include <iostream>
#include <getopt.h>
#include "nmm/utils/directory_utils.hpp"
#include "nmm/comm/netstrategy/RTPAddress.hpp"
#include "nmm/comm/netstrategy/MulticastIPAddress.hpp"
// include NMM
#include "nmm/multimedia/connect.hpp"
#include "nmm/comm/ProxyObject.hpp"
#include "nmm/services/registry/ProxyApplication.hpp"
#include "nmm/services/registry/GraphDescription.hpp"

// include used interfaces
#include "nmm/plugins/interfaces/IFileHandler.hpp"
#include "nmm/plugins/net/INet.hpp"

// include used interfaces
#include "nmm/multimedia/sync/interfaces/ISynchronizedSink.hpp"
#include "nmm/multimedia/sync/MultiAudioVideoSynchronizer.hpp"
#include "nmm/utils/time/Clock.hpp"
#include "nmm/comm/ReferenceManager.hpp"
#include "nmm/comm/ServiceProxy.hpp"
#include "nmm/comm/netstrategy/TCPAddress.hpp"
#include "nmm/comm/netstrategy/TCPAcceptor.hpp"
#include "nmm/multimedia/IBidirQueuedNode.hpp"

// import namespace
using namespace NMM;

void usage(const string& name) {
    cerr << "Usage :" << endl
         << " " << name << " [options] <output file> " << endl
         << endl
         << "Options:" << endl
         << " -v : Increase verbosity, can be given up to four times, i.e. -vvvv" << endl
         << " -h : Show this help" << endl
         << endl;
}

int main(int argc, char ** argv)
{
    if(argc < 2) {
        usage( argv[0] );
        return 1;
    }
    cerr << argv[0] << " running ... " << endl << endl;

    // RTP and RTCP ports to connect to
    int rtp_port = 1234;
    int rtcp_port = 1235;

    const char *opt_string = "hv";
    struct option options [] = {
        {"verbose",          0, NULL, 'v'},

```

```
{ "help",          0, NULL, 'h' },
{ NULL,           0, NULL, 0 }
};

int ch, long_option_index;
int verbose = 0;
while((ch=getopt_long(argc,argv,opt_string,options,&long_option_index))!=-1){
    // which option did we get?
    switch (ch) {
        case 'v':
            ++verbose;
            break;
        case 'h':
            usage(argv[0]);
            exit(0);
        default:
            usage(argv[0]);
            throw Exception("Invalid option");
    }
}

string output_file = argv[optind];

// Use default verbose level if no -v option was specified
if(verbose == 0) {
    verbose = NamedObject::getVerboseLevel();
}

// disable all debug output (error, warning, message, debug)
// see documentation Debug, Error, Warning and Message Output in NMM
// maybe turn off debug messages
if (verbose > 0 ) {
    NamedObject::getGlobalInstance().setErrorStream(&cout, NamedObject::ALL_LEVELS);
} else {
    NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);
}
if (verbose > 1) {
    NamedObject::getGlobalInstance().setWarningStream(&cout, NamedObject::ALL_LEVELS);
} else {
    NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);
}
if (verbose > 2) {
    NamedObject::getGlobalInstance().setMessageStream(&cout, NamedObject::ALL_LEVELS);
} else {
    NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);
}
if (verbose > 3) {
    NamedObject::getGlobalInstance().setDebugStream(&cout, NamedObject::ALL_LEVELS);
} else {
    NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);
}

// the graph description for the example
```

```
GraphDescription graph;

// the NMM application for the example
NMMApplication* app = ProxyApplication::getApplication(argc, argv);

try {
    // create the descriptions for the GenericReadNode, MPEGAudioDecodeNode, and PlaybackNode
    NodeDescription netsource_descr("NetSourceNode");
    NodeDescription sink_descr("GenericWriteNode");

    Format format(Format::av_mpegts_type);

    // create a flow graph: readfile -> net_sink
    graph.addEdge(netsource_descr, sink_descr, &format);

    // request complete graph
    cerr << "Request GraphDescription from registry" << endl;
    ClientRegistry& registry = app-> getRegistry();
    registry.requestGraph(graph);

    // Set RTP Payloadtype on NetSourceNode
    INode* netsource = graph.getINode(netsource_descr);
    INet_var i_net (netsource-> getParentObject()-> getCheckedInterface<INet> ())
    i_net-> setPayloadType(33);

    // set Multicast address to connect to
    MulticastIPAddress ip_addr("239.255.42.42");
    RTPAddress rtp_addr (ip_addr, rtp_port, rtcp_port);
    i_net-> setTransportStrategy(&rtp_addr);

    // set the filename by requesting appropriate interface
    cerr << "Set output filename to " << output_file << endl;
    INode* sink = graph.getINode(sink_descr);
    IFileHandler_var filehandler (sink-> getParentObject()-> getCheckedInterface<IFileHandler> ())
    filehandler-> setFilename(output_file);

    cerr << "realizeGraph()" << endl;
    graph.realizeGraph();

    // start graph
    cerr << "startGraph()" << endl;
    graph.startGraph();

    // wait ...

    cerr << "Receiving MPEG2 TS ... enter 'q <enter> ' to exit." << endl;
    string s;

    while (1) {
        cin >> s;
        if ( s == "q" ) {
            break;
        }
    }
}
```

```
    }  
    // stop and release graph  
    cerr << "stopGraph()" << endl;  
    graph.stopGraph();  
  
    cerr << "releaseGraph()" << endl;  
    registry.releaseGraph(graph);  
  
    // do cleanup  
    delete app;  
    return 0;  
}  
catch(const Exception& e) { // catch NMM-exception and print it  
    cerr << e << endl;  
  
    // release graph  
    cerr << "releaseGraph()" << endl;  
    ClientRegistry& registry = app-> getRegistry();  
    registry.releaseGraph(graph);  
  
    delete app;  
    return 1;  
}  
catch(const SerializedException& e) { // catch serialized NMM-exception and print it  
    cerr << e << endl;  
  
    // release graph  
    cerr << "releaseGraph()" << endl;  
    ClientRegistry& registry = app-> getRegistry();  
    registry.releaseGraph(graph);  
  
    delete app;  
    return 1;  
}  
catch(...) { // any other exception  
    cerr << "something went wrong!" << endl;  
  
    // release graph  
    cerr << "releaseGraph()" << endl;  
    ClientRegistry& registry = app-> getRegistry();  
    registry.releaseGraph(graph);  
  
    delete app;  
    return 1;  
}  
}
```

13. hellonmm9

This example demonstrates the discovery of serverregistries somewhere in the local network.

First, start some serverregistries on arbitrary hosts in the local network

```
./serverregistry
```

Then, start the example

```
./hellonmm9
```

and watch the output.

Following is the source code of hellonmm9.cpp

```
// This example demonstrates how to discover running server registries on other hosts than  
// In order to take full advantage of this feature, the names and  
// ports printed out could be used to request nodes.  
// notice: you need a running 'serverregistry' on the remote host(s)  
  
#include <iostream>  
  
// include NMM  
#include "nmm/comm/ProxyObject.hpp"  
#include "nmm/services/registry/ProxyApplication.hpp"  
#include "nmm/comm/ProcessUserInfo.hpp"  
  
// include used interfaces  
#include "nmm/plugins/interfaces/IFileHandler.hpp"  
#include "nmm/plugins/audio/interfaces/IAudioDevice.hpp"  
  
// import namespace  
using namespace NMM;  
  
int main(int argc, char ** argv)  
{  
  
    cerr << "Usage:" << endl  
         << " " << argv[0] << " [registry port]" << endl  
         << endl << endl;  
  
    int registry_port = getDefaultRegistryPort();  
  
    if(argc >= 2) {  
        registry_port = atoi(argv[1]);  
    }  
  
    cout << "Scanning for registries running on port " << registry_port << " ... " << endl <<
```

Hello World! Welcome to NMM Application Development

```
// disable all debug output (error, warning, message, debug)
// see documentation Debug, Error, Warning and Message Output in NMM
NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);

// the graph description for the example
GraphDescription graph;

// the NMM application for the example
NMMApplication* app = 0;

// NMM-operations may throw exception, in this simple example, all exceptions
// are just caught at the end, and the application is terminated
try {
    // create application object for example
    app = ProxyApplication::getApplication(argc, argv, SProcessUserInfo::getInstance().getH

    //request client registry from application
    ClientRegistry& registry = app-> getRegistry();

    // simple NMM discovery service: scan for running serverregistries that are using a cer
    // The first argument of this method specifies this registry por. The second argument s
    // method waits to receive responses from running serverregistries before it returns. T
    list <RegistryName> result = registry.findServerRegistries(registry_port, 50);

    // print all available serverregistries
    // in order to take full advantage of this feature, the names and
    // ports printed out could be used to request nodes
    cerr << "Available registries " << endl;
    for ( list<RegistryName> ::iterator it = result.begin(); it != result.end(); ++it)
    {
        cerr << "Found registry on " << *it << endl;
    }

    // wait ...
    cerr << "Searching for running server registries ... enter 'q <enter> ' to exit." << en
    string s;
    cin >> s;
    cerr << "Quit ... " << endl;

    delete app;
    return 0;
}
catch(const Exception& e) { // catch NMM-exception and print it
    cerr << e << endl;

    delete app;
    return 1;
}
catch(const SerializedException& e) { // catch serialized NMM-exception and print it
```

```
    cerr << e << endl;

    delete app;
    return 1;
}
catch(...) { // any other exception
    cerr << "something went wrong!" << endl;

    delete app;
    return 1;
}
}
```

14. hellographbuilder1

hellographbuilder1: shows how to use the graph builder of NMM that automatically creates a flow graph from a given URL. All supported codecs and file formats supported by NMM can be played with this simple example. E.g. start with './hellographbuilder1 file:///home/bob/music.wav'. For more examples of possible URLs, see the corresponding section of the documentation for 'clit'.

```
// hellographbuilder1: shows how to use the graph builder of NMM that
// automatically creates a flow graph from a given URL. All supported
// codecs and file formats supported by NMM can be played with this
// simple example. E.g. start with './hellographbuilder1
// file:///home/bob/music.wav'. For more examples of possible URLs,
// see the corresponding section of the documentation for 'clit'.
```

```
#include <iostream>

// include NMM
#include "nmm/comm/ProxyObject.hpp"
#include "nmm/services/registry/ProxyApplication.hpp"
#include "nmm/services/graphbuilder/GraphBuilder.hpp"
#include "nmm/multimedia/sync/MultiAudioVideoSynchronizer.hpp"

// include used interfaces
#include "nmm/plugins/interfaces/IFileHandler.hpp"
#include "nmm/plugins/audio/interfaces/IAudioDevice.hpp"
#include "nmm/plugins/video/display/interfaces/IVideoDisplay.hpp"

using namespace NMM;

int main(int argc, char * argv[])
{
    if(argc < 2) {
        cerr << "Usage:" << endl
             << " " << argv[0] << " <url> " << endl
             << endl;
        return 1;
    }
}
```

```
}
cerr << argv[0] << " running ... " << endl << endl;

// disable all debug output (error, warning, message, debug)
// see documentation Debug, Error, Warning and Message Output in NMM
NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);
NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);

// the NMM application for the example
NMMApplication* app = 0;
// the composite node that holds the flow graph created by the graph builder
CompositeNode* composite = 0;

// NMM-operations may throw exception, in this simple example, all exceptions
// are just caught at the end, and the application is terminated
try {
    // create application object for example
    app = ProxyApplication::getApplication(argc, argv);

    // create node descriptions for possible sink nodes for audio and video
    // create the description for audio sink:
    // ... some sink node
    // ... that supports the interface IAudioDevice
    // notice that this description will provide an audio sink
    // on all supported platforms
    NodeDescription audioplay_descr;
    audioplay_descr.setNodeType(INode::SINK);
    audioplay_descr.addInterface(IAudioDevice::IAudioDevice_type);
    // create the description for video sink:
    // ... some sink node
    // ... that supports the interface IVideoDisplay
    // notice that this description will provide a video sink
    // on all supported platforms
    NodeDescription display_descr;
    display_descr.setNodeType(INode::SINK);
    display_descr.addInterface(IVideoDisplay::IVideoDisplay_type);

    // create synchronizer for audio/video
    MultiAudioVideoSynchronizer av_sync;
    IMultiAudioVideoSynchronizer_var sync(av_sync.getCheckedInterface<IMultiAudioVideoSynch

    // setup graph builder
    GraphBuilder gb;
    if(!gb.setURL(argv[1])) {
        throw Exception("Invalid URL given");
    }
    gb.setMultiAudioVideoSynchronizer(sync.get());
    gb.setAudioSink(audioplay_descr);
    gb.setVideoSink(display_descr);

    // create flow graph for given URL
```

```
composite = gb.createGraph(*app);

// start all internal nodes of composite node
composite-> reachStarted();

// wait ...
cerr << "Playing ... enter 'q <enter> ' to exit." << endl;
string s;
cin >> s;
}
catch (const Exception& e) {
    cerr << e << endl;
}
catch(const SerializedException& e) { // catch serialized NMM-exception and print it
    cerr << e << endl;
}
catch(...) {
    cerr << "something went wrong..." << endl;
}

// release composite node
if(composite) {
    composite-> reachActivated();
    composite-> flush();
    composite-> reachConstructed();
    delete composite;
}

delete app;
return 0;
}
```

15. hellographbuilder2

hellographbuilder2 (**modified** version of hellographbuilder1): shows how to use the graph builder of NMM that automatically creates a **distributed** flow graph from a given URL; the audio and video sinks are located on the given hosts. All supported codecs and file formats supported by NMM can be played with this simple example. E.g. start with `./hellographbuilder1 file:///home/bob/video.mpg host1 host2`. Notice: you need a running serverregistry on host1 and host2. For more examples of possible URLs, see the corresponding section of the documentation for `'clie'`.

```
// hellographbuilder2 (modified version of hellographbuilder1): shows  
// how to use the graph builder of NMM that automatically creates a  
// distributed flow graph from a given URL; the audio and video sinks  
// are located on the given hosts. All supported codecs and file  
// formats supported by NMM can be played with this simple  
// example. E.g. start with './hellographbuilder1  
// file:///home/bob/video.mpg host1 host2'. Notice: you need a running
```

```
// serverregistry on host1 and host2. For more examples of possible
// URLs, see the corresponding section of the documentation for
// 'clirc'.

#include <iostream>

// include NMM
#include "nmm/comm/ProxyObject.hpp"
#include "nmm/services/registry/ProxyApplication.hpp"
#include "nmm/services/graphbuilder/GraphBuilder.hpp"
#include "nmm/multimedia/sync/MultiAudioVideoSynchronizer.hpp"

// include used interfaces
#include "nmm/plugins/interfaces/IFileHandler.hpp"
#include "nmm/plugins/audio/interfaces/IAudioDevice.hpp"
#include "nmm/plugins/video/display/interfaces/IVideoDisplay.hpp"

using namespace NMM;

int main(int argc, char * argv[])
{
    if(argc < 4) {
        cerr << "Usage:" << endl
            << " " << argv[0] << " <url> <location of audio sink> <location of video sink> "
            << endl;
        return 1;
    }
    cerr << argv[0] << " running ... " << endl << endl;

    // disable all debug output (error, warning, message, debug)
    // see documentation Debug, Error, Warning and Message Output in NMM
    NamedObject::getGlobalInstance().setErrorStream(0, NamedObject::ALL_LEVELS);
    NamedObject::getGlobalInstance().setWarningStream(0, NamedObject::ALL_LEVELS);
    NamedObject::getGlobalInstance().setMessageStream(0, NamedObject::ALL_LEVELS);
    NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);

    // the NMM application for the example
    NMMApplication* app = 0;
    // the composite node that holds the flow graph created by the graph builder
    CompositeNode* composite = 0;

    // NMM-operations may throw exception, in this simple example, all exceptions
    // are just caught at the end, and the application is terminated
    try {
        // create application object for example
        app = ProxyApplication::getApplication(argc, argv);

        // create node descriptions for possible sink nodes for audio and video
        // create the description for audio sink:
        // ... some sink node
        // ... that supports the interface IAudioDevice
        // notice that this description will provide an audio sink
        // on all supported platforms
    }
```

```
NodeDescription audioplay_descr;
audioplay_descr.setNodeType(INode::SINK);
audioplay_descr.addInterface(IAudioDevice::IAudioDevice_type);
// create the description for video sink:
// ... some sink node
// ... that supports the interface IVideoDisplay
// notice that this description will provide a video sink
// on all supported platforms
NodeDescription display_descr;
display_descr.setNodeType(INode::SINK);
display_descr.addInterface(IVideoDisplay::IVideoDisplay_type);

// set location for audio sink to argv[2] and for video sink to argv[3]
audioplay_descr.setLocation(argv[2]);
display_descr.setLocation(argv[3]);

// create synchronizer for audio/video
MultiAudioVideoSynchronizer av_sync;
IMultiAudioVideoSynchronizer_var sync(av_sync.getCheckedInterface<IMultiAudioVideoSynch

// setup graph builder
GraphBuilder gb;
if(!gb.setURL(argv[1])) {
    throw Exception("Invalid URL given");
}
gb.setMultiAudioVideoSynchronizer(sync.get());
gb.setAudioSink(audioplay_descr);
gb.setVideoSink(display_descr);

// create flow graph for given URL
composite = gb.createGraph(*app);

// start all internal nodes of composite node
composite-> reachStarted();

// wait ...
cerr << "Playing ... enter 'q <enter> ' to exit." << endl;
string s;
cin >> s;
}
catch (const Exception& e) {
    cerr << e << endl;
}
catch(const SerializedException& e) { // catch serialized NMM-exception and print it
    cerr << e << endl;
}
catch(...) {
    cerr << "something went wrong..." << endl;
}

// release composite node
if(composite) {
    composite-> reachActivated();
}
```

```
    composite-> flush();
    composite-> reachConstructed();
    delete composite;
}

delete app;
return 0;
}
```

16. helloworld1.gd

helloworld1.gd: simple wav player

```
% simple wav player
% use the -i option of clic to specify the wav file
% usage: ./clic helloworld1.gd -i <my-wav-file>
% notice: if your system does not provide ALSA, use the DirectX or MacOSX node instead

WavReadNode
! ALSAPlaybackNode
```

17. hellonmm1.gd

hellonmm1.gd: distributed wav player

```
% distributed wav player
% replace <remote host where serverregistry is running>
% use the -i option of clic to specify the wav file
% usage: ./clic hellonmm1.gd -i <my-wav-file>
% notice: if your system does not provide ALSA, use the DirectX or MacOSX node instead

WavReadNode
! ALSAPlaybackNode
#setLocation("<remote host where serverregistry is running> ")
```

18. hellonmm2.gd

hellonmm2.gd: distributed wav player : playback on local and remote host

```
% distributed wav player: playback on local and remote host
% replace <remote host where serverregistry is running>
% use the -i option of clic to specify the wav file
% usage: ./clic hellonmm2.gd -i <my-wav-file>
% notice: if your system does not provide ALSA, use the DirectX or MacOSX node instead

WavReadNode
{
  { ! ALSAPlaybackNode
    #setLocation("<remote host where serverregistry is running> ")
  }
  { ! ALSAPlaybackNode
  }
}
```

19. hellonmm3.gd

hellonmm3.gd: optimized distributed wav player: playback on remote host. In contrast to the application hellonmm3 the TCP buffer size for incoming data can not be changed, because the current version of clic does not allow to adjust the parameters of a used network connection.

```
% distributed wav player
% replace <remote host where serverregistry is running>
% use the -i option of clic to specify the wav file
% The wav player reduces the amount of data that are
% buffered inside the flow graph
% usage: ./clic hellonmm3.gd -i <my-wav-file>
% notice: if your system does not provide ALSA, use the DirectX or MacOSX node instead

WavReadNode
! ALSAPlaybackNode
# setLocation("<remote host where serverregistry is running> ")
$ setDownstreamMaxSize(1, "default") CONSTRUCTED
```