

Clic - An Application for Setting up NMM Multimedia Flow Graphs

**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 describes the usage of the application 'clic' (command line interaction and configuration) used in the NMM project to build up a flow graph from a textual description. Furthermore, clic allows to use the graph builder of NMM that automatically creates a distributed flow graph from a given URL.

1. Introduction

The NMM framework is used to build up multimedia applications. The basic components in this framework are nodes which perform a certain functionality, like reading a video file or decoding it. Such nodes can be connected to a flow graph to perform a certain task like watching a movie or transcoding it into another format.

Especially transcoder jobs do not need any additional user interaction if the application is running. Writing such an application without any user interaction is currently straightforward. First, the application requests these nodes from the registry, connects them to a flow graph and finally starts the entire flow graph.

The application `clic` is used to build such a standard NMM application automatically from a textual description which is called 'graph description'. This document describes the syntax of this graph description and the usage of the clic-application.

Furthermore, clic allows to use the graph builder of NMM that automatically creates a distributed flow graph from a given URL: the source of media, the audio output, and the video output can be distributed.

Please note that all following examples were tested on Linux only, unless otherwise indicated. Therefore, some node names or settings need to be changed when running the examples on other operating systems. Examples for other operating systems are provided with the corresponding NMM package.

2. Syntax of the graph description

A graph description consists of two major parts. The first part is an (optional) comment introduced by the character `%`. The second part specifies the flow graph which describes how to connect the nodes. The following example describes a simple graph to play an WAV audio file.

```
% This graph description realizes a simple WAV player.  
% Use the -i option of clic to specify an WAV file
```

```
WavReadNode ! ALSAPlaybackNode
```

To run this graph description, copy it into a file, e.g. `wavplay.gd`, and start clic with the following command:

```
./clic wavplay.gd -i /home/bob/audio/song.wav
```

The option `-i` is used to specify the input file. To exit the application, enter `q` at the command line and press enter. Note: If you call clic on linux host with an input file located on a windows host, you need to quote the path to the input file.

```
./clic wavplay.gd -i "c:\home\bob\audio\song.wav"
```

For detailed information about all options simply type:

```
./clic -h
```

Appendix A describes the location of the graph descriptions that are included in the current NMM release.

2.1. Creating flow graphs

The flow graph section of a graph description describes how to connect the nodes. A node is identified by its name, which is (normally) the C++ class name. The exclamation mark (!) is used to denote a connection between two nodes as seen in the graph description for playing WAV files. In this example the statement

```
WavReadNode ! ALSAPlaybackNode
```

results in clic to request the nodes WavReadNode and ALSAPlaybackNode. If all nodes can be successfully requested, the WavReadNode is connected to the ALSAPlaybackNode, and the MPEGAudioDecodeNode to the PlaybackNode. Then, clic sets the specified parameters from the command line, for example an input file, and starts the entire flow graph.

To specify more complex flow graphs that include a demultiplexer, we need an extended syntax to describe the different input or output jacks which a (de)multiplexer can provide. Therefore, a unique string which is called jack tag is associated with each jack to identify it. For example these jack tags are used to distinguish between audio and video in/output of a multiplexer or a demultiplexer node. If a node provides only a single input or output jack the string "default" is used as jack tag. This jack tag is used in graph descriptions if no other jack tag is specified. In order to use a specific input jack that should be used for a connection, the corresponding jack tag must be written before the node name, and behind the node name to specify an output jack respectively. The following example shows the WAV player graph description with specified "default" jack tags:

```
WavReadNode "default"  
! "default" ALSAPlaybackNode
```

In this example, the specification of jack tags is not required because clic uses the jack "default" to connect nodes if no other jack tag is specified.

Furthermore, we need an extended syntax to describe branches of connected nodes that must be connected to a specific output jack of a demultiplexer. To specify a branch within a flow graph, the corresponding nodes are encapsulated in braces ({}) as seen in the following example, for reading mpeg files.

```
% This graph description realizes a simple mpeg video player with ac3 audio.  
% Use the -i option of clic to specify the mpeg video file.
```

```
GenericReadNode
```

```
! MPEGDemuxNode
{
  { ["mpeg_video0"] ! MPEGVideoDecodeNode
    ! XDisplayNode }
  { ["ac3_audio0"] ! AC3DecodeNode
    ! ALSAPlaybackNode }
}
```

The MPEGDemuxNode demultiplexes incoming messages and forwards audio buffer to the "ac3_audio0" output jack and video buffer to the "mpeg_video0" output jack, respectively. After the MPEGDemuxNode, braces encapsulates all branches connected to this node. Each branch itself is encapsulated in braces too and starts with the output jack tag of the MPEGDemuxNode, encapsulated in square brackets. This output jack is connected to the leading node of the branch. So in this example we have two branches. The first one consists of the MPEGVideoDecodeNode, connected to the XDisplayNode. The MPEGVideoDecodeNode is the first node of this branch and is connected to the output jack of the MPEGDemuxNode with jack tag "mpeg_video0". The second branch consist in the AC3DecodeNode connected to the ALSAPlaybackNode. The first node of this branch is the AC3DecodeNode which is connected to the output jack of the MPEGDemuxNode with jack tag "mpeg_audio0".

The following example extends the above .gd file to attach an additional XDisplayNode. Nodes that only have a single output (or input), such as the MPEGVideoDecodeNode, typically name this input 'default'. This .gd file demonstrates that clic supports nested sub-branches.

```
% This graph description plays an MPEG file with AC3 audio
% using the ALSAPlaybackNode and two XDisplayNodes.
% It can only be run on Linux.

% Command:
% clic mpegplay_ac3_two_displays.gd -i file

GenericReadNode
! MPEGDemuxNode
{
  { ["mpeg_video0"] ! MPEGVideoDecodeNode
    {
      { ["default"] ! XDisplayNode }
      { ["default"] ! XDisplayNode }
    }
  }
  { ["ac3_audio0"] ! AC3DecodeNode
    ! ALSAPlaybackNode
  }
}
```

To add a multiplexer node is very similar to a demultiplexer, as seen in the following example.

```
% This graph description converts an MPEG video file with AC3
% audio into an avi file.
```

```
% Unfortunately the flow graph does not work because the FFMpegEncodeNode
% requires a bitrate.
```

```
GenericReadNode
! MPEGDemuxNode
{
  { ["mpeg_video0"] ! MPEGVideoDecodeNode
                                ! FFMpegEncodeNode
                                !
    ["video"]
  }
  {
    ["ac3_audio0"] !
    ["audio"]
  }
}
AVMuxNode
! AVIWriteNode
```

This example can be used to convert an MPEG file into an AVI file with DivX-video and ac3 audio. The first branch consists of an MPEGVideoDecodeNode which is connected to the FFMpegEncodeNode. The input of the first node in this branch, the MPEGVideoDecodeNode, is connected to the output jack of the MPEGDemuxNode with jack tag "mpeg_video0". The output of the last node in this branch, the FFMpegVideoEncodeNode, is connected to the input jack of the AVMuxNode with jack tag "video". The second branch has no nodes. In this branch output of the MPEGDemuxNode jack with jack tag "ac3_audio0" is connected to the input jack of AVmuxNode with jack tag "audio". Therefore, the audio buffers are directly passed through from the MPEGDemuxNode to the AVMuxNode without any conversion.

The syntax of this example is correct but the flow graph can not run because the FFMpegEncodeNode needs the encoding bitrate which is specified in the connection format. Thus, we need a possibility to (partially) specify a connection format that is used between two nodes. The statement `Format` prefaced by the character '@' is used to specify the connection format. This statement expects a comma separated list of format types, which are defined in the `Format.hpp` followed by a corresponding value. A sole exception is the format type which is written as a single string. Note: The connection format must be specified in front of the connection symbol (!). The following example shows how to define the connection format including the format type and the bitrate of the FFMpegEncodeNode.

```
% This graph description converts an MPEG video file with AC3 audio
% into an AVI file. Furthermore the desired bitrate of the video is
% specified using the connection format.
% Use the -i option of clic to specify the mpeg video file and -o option
% to specify the name of the output file.
```

```
GenericReadNode
! MPEGDemuxNode
{
  { ["mpeg_video0"] ! MPEGVideoDecodeNode
                                ! FFMpegEncodeNode
                                @ Format ("video/mpeg4", bitrate = 1200000)
                                !
    ["video"]
  }
}
```

```
}  
{  
  ["ac3_audio0"] !  
  ["audio"]  
}  
} AVMuxNode  
! AVIWriteNode
```

To run this flow graph, copy the graph description into a file, e.g. `mpeg_to_avi.gd`, and enter

```
./clic mpeg_to_avi.gd -i /home/bob/video/movie.mpeg -o /home/bob/video/movie.avi
```

whereas the option `-i` specifies the input file and `-o` the output file.

In addition, you can specify two additional parameters to set the transport protocol used to transmit data between two nodes, which is especially useful if they are running on different hosts. Both parameters are prefaced by the character `'@'` as well. To configure a specific transport protocol used for instream communication the parameter `setBufferStrategy` and `setEventStrategy` can be used, as seen in the following graph description.

```
% This graph description describes a simple MP3 player which specifies the  
% used transport protocols between the nodes.  
% Use the -i option of clic to specify the MP3 file  
  
GenericReadNode      @ setEventStrategy("TCPStrategy")  
                    @ setBufferStrategy("RTPStrategy") !  
MPEGAudioDecodeNode @ setBufferStrategy("RTPStrategy") !  
PlaybackNode
```

The parameter `setBufferStrategy` specifies the transport protocol used to transmit multimedia buffers and the parameter `setEventStrategy` how to transport instream events. In this example the multimedia protocol RTP is used to transmit the MP3 buffers and the reliable TCP protocol to transmit instream events. To transport buffers between the `MPEGAudioDecodeNode` and `PlaybackNode`, the RTP protocol is used, but no protocol for instream events is specified. If one of these parameters or both are not set, the default transport strategy proposed by the NMM is used, which is a `LocalStrategy` to transport buffers and events between nodes in the same address space efficiently via pointer forwarding and `TCPStrategy` between nodes running on different hosts or address spaces.

3. Node specific Parameters

In several cases you want to configure some node specific parameters like the location of a node. Thus, the clic syntax allows to set such parameters which are prefaced by the symbol `'#'` followed by a parameter which is identical to the corresponding C++ method provided by the node description of the component.

These parameters must be written after the node name and its jack tag. Currently the following parameter types are supported:

- `setLocation(<string>)`: The `setLocation` parameter expects a string as argument and thereby allows to specify the host, from where the node is requested. The string can either include the host name of the system or the IP address in dotted-decimal notation (e.g. "127.0.0.1") Note: The application `serverregistry` must be running on this host.
- `setPort(<int>)`: The `setPort` parameter expects an integer as argument and allows to specify the port a server registry is listening on. By default the port 22801 is used.
- `setSharingType(<sharing-type>)`: This parameter expects a sharing type as argument and allows to specify whether the node can be reused from other applications or not.

The rest of this section describes the usage of these parameters. For simplicity, the WAV player example is used to describe this features.

```
WavReadNode
! ALSAPlaybackNode
```

3.1. setLocation

The clic application normally requests a node from the local system, but the NMM framework allows transparent access to nodes distributed in a network. To support this feature in the clic-application, the location of a node inside the network can be specified with the parameter `setLocation`. Instead of requesting a node from the local system it is requested from the specified one. This is especially helpful if a resource intensive job, like video encoding, should be done on a powerful computer. To request a node from another host the application `serverregistry` must be running on this system. This parameter expects a single string as argument which specifies the hostname as seen in the following example. This parameter expects a single string as argument which specifies the hostname or the IP address in dotted-decimal notation (e.g. "127.0.0.1") as seen in the following examples.

```
% This graph description describes a simple WAV player where the WavReadNode
% is running on host "server1". The serverregistry on host "server1"
% is listening on port 22801.
% Use the -i option of clic to specify the WAV file

WavReadNode # setLocation("server1")
! ALSAPlaybackNode
```

In this graph description the `WavReadNode` is not requested from the local system. Instead the node is requested from a `serverregistry` running on host "server1" which accepts incoming requests on port 22801. If the node can not be requested from this `serverregistry` the application terminates with a corresponding error message.

The following graph description shows how to request a node by using an IP address:

```
% This graph description describes a simple WAV player where the WavReadNode
% is running on host with IP address 192.168.1.1 . The serverregistry on host with IP address
% is listening on port 22801.
% Use the -i option of clic to specify the WAV file

WavReadNode # setLocation("192.168.1.1")
! ALSAPlaybackNode
```

In this graph description the WavReadNode is not requested from the local system. Instead the node is requested from a serverregistry running on host with IP address 192.168.1.1 which accepts incoming requests on port 22801. If the node can not be requested from this serverregistry the application terminates with a corresponding error message.

3.2. setPort

If the serverregistry application is listening on a different port for some reason, e.g. 3000, you can set it using the setPort parameter as seen in the following example.

```
% This graph description describes a simple WAV player where the WavReadNode
% is running on host "server1". The serverregistry on host "server1"
% is listening on port 3000
% Use the -i option of clic to specify the WAV file

WavReadNode # setLocation("server1")
              # setPort(3000)
! ALSAPlaybackNode
```

3.3. setSharingType

NMM provides a service called *Session Sharing* which allows to share parts of a running flow graph between several applications. This is especially useful to use limited hardware devices, like TV cards, efficiently: Several user can share the same TV card, but use different displays in a different room to watch TV. In contrast to a streaming application each user is able to control all nodes of the entire flow graph and playback is synchronized when using NMM. In general a user will not share all nodes of a running flowgraph. Especially the sink nodes will be used exclusively in most cases. Thus, NMM allows a user to specify a so called *sharing type* for each node of a flowgraph that specifies whether a node can, must or must not be shared. The following sharing types are supported by NMM:

- EXCLUSIVE: A new node is requested and other applications are not allowed to share it.
- EXCLUSIVE_THEN_SHARED: A new node is requested, but other applications can use this running node as well.

- `SHARED`: Using this sharing type no new node is allocated, instead the node is used from an application that requested the node with sharing type `EXCLUSIVE_THEN_SHARED`.
- `EXCLUSIVE_OR_SHARED`: First clic tries to request an `EXCLUSIVE` node. If this fails, it tries to request a node with sharing type `SHARED`.
- `SHARED_OR_EXCLUSIVE`: First clic tries to request a `SHARED` node. If this fails, it tries to request a node with sharing type `EXCLUSIVE`.
- `EXCLUSIVE_THEN_SHARED_OR_SHARED`: First clic tries to request an `EXCLUSIVE` node, that can be shared with other applications. If this fails, it tries to request a node with sharing type `SHARED`.
- `SHARED_OR_EXCLUSIVE_THEN_SHARED`: First clic tries to request a `SHARED` node. If this fails, it tries to request a node with sharing type `EXCLUSIVE_THEN_SHARED`

If no sharing type is specified clic uses the sharing type `EXCLUSIVE_OR_SHARED`. The following example shows how to use this parameter.

```
% This graph description describes a simple WAV player where the WavReadNode
% can be reused by other applications. The remaining node is exclusive and
% cannot be accessed by other applications.
% Use the -i option of clic to specify a WAV file. Furthermore you must start clic
% with the command line argument '-S' or you must start a serverregistry to enable
% access from other applications to this flow graph.
```

```
WavReadNode      # setSharingType(EXCLUSIVE_THEN_SHARED)
! ALSAPlaybackNode # setSharingType(EXCLUSIVE)
```

For testing this graph description copy it into a file, e.g. `shared_wavplay.gd`, and enter the following command.

```
./clic shared_wavplay.gd -i /home/bob/audio/song.wav -S
```

Please note that you must also start a server registry on the local host or you must start clic with the command line argument `'-S'` to enable remote hosts access to the clic application. If you start this graph description on host "host1" clic allows other applications to share the `WavReadNode` and to receive the same WAV file, but the `ALSAPlaybackNode` must not be shared. Therefore, other application running on additional hosts can access the `WavReadNode` (at its current position within the file) and synchronously listen to the same WAV file, e.g. in different rooms of your house using the following graph description.

```
% This graph description describes a simple WAV player which requests an
% WavReadNode by an already running application. In this case no WAV file
% can be specified because it has been set by the application, which is
% already running
```

```
WavReadNode  # setLocation("host1")
              # setSharingType(SHARED)
! ALSAPlaybackNode
```

To run this graph description, copy it into a file, e.g. `shared_wavplay2.gd`, and enter the following command on a second host, e.g. "host2", while the first instance of clic is still running on host1.

```
./clic shared_wavplay2.gd
```

In this case no additional arguments must be specified. If you start clic using this graph description on another host, e.g. "host2", the running WavReadNode from "host1" will be re-used and you can listen to the same music as played back on "host1". In contrast to common streaming applications NMM synchronizes the playback between "host1" and "host2" so that no disturbing echo effects occur during the playback of the same audio stream on different hosts.

4. Configuration parameters

4.1. Interface methods

In several cases a node must be configured before it can be used. In the previous examples, the name of a WAV file needs to be specified for the WavReadNode to read data from this file. Even though the clic-application offers a command line arguments for such a scenario, other nodes might require additional parameters, which can not be specified this way. In this case, you must call a node specific method, offered by an NMM interface specified in NMM-IDL (Interface Definition Language). Each method of an NMM interface can be called from graph description as well. For this purpose you have to write the corresponding IDL method with all arguments prefaced by the character '\$'. Furthermore you have to specify the state in which the method must be executed. For example the WavReadNode offers the method `setFilename` in one of its interfaces to set a WAV file, which must be called in the state `INITIALIZED`. Furthermore you might want to change the queue of the ALSAPlaybackNode for incoming buffers, which can be done using the method `setDownstreamMaxSize` which must be called in the state `CONSTRUCTED`. Note: All configuration parameters must be written after the node configuration parameters which are prefaced by the character '#' and before the connection parameters introduced by the character '@'. The following example shows how to call these two methods from a graph description.

```
% This graph description describes a simple WAV player and shows how to
% call interface methods from a graph description. It is not required
% to set the input file using the option -i of clic.
  WavReadNode      $ setFilename("/home/bob/audio/song.wav")  INITIALIZED
! ALSAPlaybackNode $ setDownstreamMaxSize(1, "default")      CONSTRUCTED
```

As can be seen in this example, the methods to be called are specified with their common name and additional parameters followed by the state of the node in which they should be executed.

Be careful using this feature. The given arguments are read as strings and converted to the expected type. Thus, it can cause errors if you set invalid arguments. Furthermore, you must specify all arguments. Default arguments, like in C++, are currently not supported.

4.2. setEventStrategy

You can also specify the transport protocol that is used to control the node. The parameter `setEventStrategy` prefaced by the character '@' is used to specify the transport strategy that is used to forward interface method calls to its node. If this parameter is not set, the default transport strategy proposed by the NMM middleware is used, which is a `LocalStrategy` for nodes in the address space of the application and `TCPStrategy` for nodes running on a different host or address space. The following example shows how to use this parameter.

```
WavReadNode          @ setEventStrategy("TCPStrategy")
! ALSAPlaybackNode
```

In this graph description the transport strategy "TCPStrategy" is specified to communicate with the `WavReadNode`, even if it runs in the address space of the application.

5. Using the Graph Builder

`clic` allows to use the graph builder of NMM that automatically creates a (distributed) flow for playing back the content from a given URL:

```
./clic <url>?<option>=<value>
```

In general, all file formats and codecs supported by NMM are also available when using the graph builder. Following URLs are currently supported:

5.1. file

```
file://<hostname>/<path and filename>
```

For your convenience, local file URLs can be specified directly, i.e. use '`<path and filename>`' instead of '`file://<path and filename>`'

For example:

```
./clic file:///home/bob/mp3/song.mp3
```

or:

```
./clic ~/mp3/song.mp3
```

Reads an MP3 file and plays back audio via your local audio device. Notice the three '/': two for the URL, one for the root path.

```
./clic file://host1/home/bob/mp3/song.mp3
```

Reads an MP3 file from host host1 and plays back audio via your local audio device. Notice: you need a NMM serverregistry running on host host1. However, no shared file system is needed.

Other possible examples for file URLs are:

```
./clic file:///home/bob/audio/song.wav
./clic file:///home/bob/audio/song.ogg
./clic file:///home/bob/audio/song.mp3
./clic file:///home/bob/audio/song.ac3
./clic file:///home/bob/video/movie.mpeg
./clic file:///home/bob/video/movie.avi
./clic file:///home/bob/video/movie.ogm
./clic file:///home/bob/video/movie.vdr
./clic file:///home/bob/image/picture.png
./clic file:///home/bob/image/picture.jpg
```

5.2. audiocd

```
audiocd://<path to audio cdrom device>?track=<track number>
```

For example:

```
./clic audiocd:///dev/cdrom
```

Reads the first audio track from an audio CD-ROM in /dev/cdrom and plays back audio via your local audio device.

```
./clic "audiocd:///dev/cdrom?track=4"
```

Reads the fourth audio track from an audio CD-ROM in /dev/cdrom. Notice that the URL is enclosed in ".

```
./clic "audiocd://host1/dev/cdrom?track=4"
```

Reads the fourth audio track from an audio CD-ROM in /dev/cdrom of host host1. Notice: you need a NMM serverregistry running on host host1.

5.3. dvd

dvd://<hostname>/<path to dvd device>?title=<number>&chapter=<number>&angle=<number>

For example:

```
./clic "dvd:///dev/cdrom?title=1&chapter=4&angle=1"
```

Reads the specified title, chapter, and angle, from an DVD in /dev/cdrom and plays back audio and video via your local audio device and display, respectively.

```
./clic "dvd://host1/dev/cdrom?title=1&chapter=4&angle=1"
```

Does the same as above, but reads from an DVD drive in host host1. Notice: you need a NMM serverregistry running on host host1.

5.4. tv

tv://<hostname>/<path to video device>

For example:

```
./clic tv://
```

```
./clic tv://host1
```

5.5. dvb tv

dvb tv://<hostname>/<path to video device>

For example:

```
./clic dvb tv://
```

```
./clic dvb tv://host1
```

```
./clic "dvb tv://host1?channel=3"
```

Tunes in the 3rd channel.

```
./clic "dvbvtv://host1?channelConfig=/home/bob/channels.conf"
```

This option sets the channels-configuration-file if it should be different from default. Uses channel-configuration-file located in /home/bob

5.6. ivtv

```
ivtv://<hostname>/<path to video device>
```

For example:

```
./clic ivtv://
```

```
./clic ivtv://host1
```

```
./clic "ivtv://host1?channel=5"
```

Tunes in the 5th channel.

```
./clic "ivtv://host1?channel=7&freqtable=pal-europe"
```

Sets the frequency-table to 'pal-europe' and tunes in the 7th channel.

```
./clic "ivtv://host1?input=4"
```

Sets the input to 4, which means the tuner of the IVTV-Card (see IVTV-Dokumentation for details of available inputs)

```
./clic "ivtv://host1?channelConfig=/home/bob/ivtv.conf"
```

This option sets the channels-configuration-file if it should be different from default. Uses channel-configuration-file located in /home/bob

5.7. mpeg tv

```
mpeg tv://<hostname>/<path to video device>
```

For example:

```
./clic mpeg tv://
```

```
./clic mpegTV://host1
```

5.8. http

```
http://<url of streaming audio>
```

For example:

```
./clic http://213.84.19.220:8060/steamcast_high.ogg
```

```
./clic http://mozart.soundportal.at:8000/live160
```

6. Using the Graph Builder for Distributed Media Playback

clic allows to use the graph builder of NMM that automatically creates a distributed flow for playing back the content from a given URL:

```
./clic <url>?<option>=<value> -A <host for audio output> -V <host for video output>
```

By specifying a host for audio or video output, clic allows to distribute the playback of media. For example:

```
./clic file://host1/home/bob/video/movie.mpeg  
-A host2 -V host3
```

Reads an MPEG audio/video file from the host host1. Audio is played back on host host2, video is played back on host host3. Notice: you need a NMM serverregistry running on all three host. Notice: for synchronized media playback, you need to setup NTP properly.

7. Keyboard Control

While clic is playing an URL, the following keyboard controls are available:

Table 1. clic Keyboard Control

Left, Right	Seek forward/backward
Space	Pause
f	Toggle fullscreen
q	Quit

A. Graph descriptions

Each NMM release includes several graph descriptions, which cover the most common flow graphs for media playback, capture, or transcoding. If NMM was installed as binary, graph description files can be in the subdirectory called `<installation directory>/share/nmm/gd` under Linux/PS3/MacOSX and `<installation directory>/gd` under Windows. If you compiled NMM from a source package, the `.gd` files can be found in the subdirectory `<nmm directory>/apps/clic/gd`. All graph descriptions are sorted by the supported operating system or systems. This means there are various graph descriptions for Linux, MacOS, and Windows. Then, there are graph descriptions for cross-platform usage, e.g. for using a Linux and Windows system together. Within the various directories, you will typically find following hierarchy:

A.1. Playback

The directory `gd/playback` stores all graph descriptions which play back different media, like MP3's, AVI's and even images.

- `gd/playback/audio`: This directory includes graph descriptions for audio playback, like MP3 and AC3.
- `gd/playback/video`: This directory includes graph descriptions for video playback, like DivX and DVD.
- `gd/playback/image`: This directory includes graph descriptions for manipulation and display of images
- `gd/playback/tv`: This directory includes graph descriptions for watching TV, like DVB or ivtv

A.2. Recording

The directory `gd/recording` stores all graph descriptions which enable to record from different media, e.g. CD grabbing or TV recording and includes the following graph descriptions.

A.3. Transcoding

The directory `gd/transcode` stores all graph descriptions which transcode media, like "DivX to MPEG" or conversion between image formats and includes the following graph descriptions.

A.4. Session Sharing

The directory `gd/session_sharing` stores all graph descriptions which demonstrate different scenarios of session sharing for different types of media. Files with corresponding names can be used together, e.g. `wav_play.gd` and `wav_play2.gd`, as described in the example above.