Format Programming in NMM

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

April 2010

1. Introduction

In order to be able to connect nodes, their input and output formats have to match. The supported input and output formats have to be specified by the programmer of a plug-in. If a class is a subclass of GenericSourceNode, GenericFilterNode, GenericConverterNode, GenericProcessorNode, GenericDemultiplexerNode, GenericMultiplexer or GenericSinkNode (for short: 'the generic-classes'), you do not need to do a lot of programming. There are two places where you have to setup the formats of your node. First, the generally supported formats should be specifies in doInit() (use the methods getOwnInputProperty() or getOwnOutputProperty()). Secondly, formats specific to this instance of the node should be specified in doInitOutput() (use the methods getOwnWorkingInputProperty() or getOwnWorkingOutputProperty()).

2. Format Specification

Format specification in the doInit and doInitOutput method of a node, e.g. the VideoDecodeNode (if your class is a subclass of a generic-class)

```
Result VideoDecodeNode::doInit() {
  // set static in and out formats in own property
  Format *myInputFormat = getOwnInputProperty()->addNewFormat("video/png");
  myInputFormat->addWildcard(Format::x_resolution_param);
  myInputFormat->addWildcard(Format::y_resolution_param);
  myInputFormat->addStringParamValue(Format::colorspace_param,"rgb24");
  myInputFormat->addStringParamValue(Format::channelorder_param,"bgr");
  myInputFormat->addIntParamValue(Format::bitperpixel_param,24);
  myInputFormat->addStringParamValue(Format::endian_param, "little");
  myInputFormat->addWildcard(Format::framerate_param);
  getOwnInputProperty()->setDefaultFormat(myInputFormat);

  Format *myOutputFormat = getOwnOutputProperty()->addNewFormat("video/raw");
  myOutputFormat->addWildcard(Format::x_resolution_param);
  myOutputFormat->addWildcard(Format::y_resolution_param);
  myOutputFormat->addStringParamValue(Format::colorspace_param,"rgb24");
  myOutputFormat->addStringParamValue(Format::channelorder_param,"rgb");
  myOutputFormat->addIntParamValue(Format::bitperpixel_param,24);
  myOutputFormat->addStringParamValue(Format::endian_param, "little");
  myOutputFormat->addWildcard(Format::framerate_param);
  myOutputFormat->addStringParamValue(Format::format_param, "packet");
  getOwnOutputProperty()->setDefaultFormat(myOutputFormat);

  myInputFormat->setIOPartner(myOutputFormat);
  myOutputFormat->setIOPartner(myInputFormat);

  return SUCCESS;
}

Result VideoDecodeNode::doInitOutput() {
  // set in and out formats in working property

  const Format* in_format = getInputFormat();
  if (in_format == NULL) { return FAILURE; }

  int x_resolution = in_format->getIntValue(Format::x_resolution_param);
  int y_resolution = in_format->getIntValue(Format::y_resolution_param);
  float framerate = in_format->getFloatValue(Format::framerate_param);

  Format* myInputFormat = new Format( *in_format );
  getOwnWorkingInputProperty()->addNewFormat(myInputFormat);
  getOwnWorkingInputProperty()->setDefaultFormat(myInputFormat);

  Format *myOutputFormat = getOwnWorkingOutputProperty()->addNewFormat("video/raw");
  myOutputFormat->addIntParamValue(Format::x_resolution_param, x_resolution);
  myOutputFormat->addIntParamValue(Format::y_resolution_param, y_resolution);
  myOutputFormat->addStringParamValue(Format::colorspace_param,"rgb24");
```

```
  myOutputFormat->addStringParamValue(Format::channelorder_param,"rgb");
  myOutputFormat->addIntParamValue(Format::bitperpixel_param,24);
  myOutputFormat->addStringParamValue(Format::endian_param, "little");
  myOutputFormat->addFloatParamValue(Format::framerate_param, framerate);
  myOutputFormat->addStringParamValue(Format::format_param, "packet");
  getOwnWorkingOutputProperty()->setDefaultFormat(myOutputFormat);

  myInputFormat->setIOPartner(myOutputFormat);
  myOutputFormat->setIOPartner(myInputFormat);

  return SUCCESS;
}
```

3. Parameter Values

Instead of single values, sets, ranges or wildcards can be used:

```
// single value
OFormat1->addFloatParamValue("framerate", 7.5);
...
// set with two entries
OFormat2->addFloatParamValue("framerate", 7.5);
OFormat2->addFloatParamValue("framerate", 15.0);
...
// range of values
OFormat3->addRFloatParamValue("framerate", 7.5, 30.0);
...
// wildcard
OFormat4->addWildcard("framerate");
...
```

4. Assigning Preferences

There are two possibilities to specify that a certain format provides better quality than a other: First, each property has a default format (which is the desired or best format of the node), second a weight for each possible value of a parameter can be specified, This weight is a float, where 0 means no quality, and 1 best quality. Per default, the value 1 is assigned to all values. The weights of all values of all parameters are used to compute a total weight of the format. The following code snippet shows the two possibilities:

```
Format *IFormat;
```

```
IFormat=getOwnInputProperty()->addNewFormat("video/raw");
...
// Set the default format of the input property
getInputProperty()->setDefaultFormat(IFormat);

// Set individual weights
Format *OFormat1;
OFormat1=getOwnOutputProperty()->addNewFormat("video/raw");
OFormat1->addIntParamValue("x_resolution",640, 0.9);
OFormat1->addIntParamValue("y_resolution",480, 0.9);
...

Format *OFormat2;
OFormat2=getOwnOutputProperty()->addNewFormat("video/raw");
OFormat2->addIntParamValue("x_resolution",320, 0.5);
OFormat2->addIntParamValue("y_resolution",240, 0.5);
// OFormat2 supports a range of floats 7.5 to 30.0 fps which is
// which is weighted with 0.1 to 1.0
OFormat2->addRFloatParamValue("framerate", 7.5, 30.0, 0.1, 1.0);
...
```