

# Debug, Error, Warning and Message Output in NMM

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.

## 1. Available Output Streams

The class NamedObject provides streams for debug-, warning-, error- and simple output-messages. Each stream is divided into 3 message-level to control the output. The message-level 0 is reserved for the node programmer and the message-level 1 and 2 are reserved for NMM-internals. Each stream and message-

level can be dynamically switched on or off. The following methods can be used to access the streams, where X is a value from 0..2 for a specific message-level.

- `errorStreamX()` : This method returns a pointer to the error-stream with message-level X.
- `debugStreamX()` : This method returns a pointer to the debug-stream with message-level X.
- `warningStreamX()` : This method returns a pointer to the warning-stream with message-level X.
- `messageStreamX()` : This method returns a pointer to the message-stream with message-level X.

A little example shows how to use these methods.

```
FooNode::foo{
    if (messageStream0())
        *messageStream0 << "Example for class Message" << endl;

    if (debugStream0())
        *debugStream0() << "Start function foo" << endl;
    ...
    if (true != true)
        if (errorStream0())
            *errorStream0 << "Impossible error true != true !!!" << endl;
    ...
    if (requestBuffer() == NULL)
        if (warningStream0())
            *warningStream() << "Could not allocate a buffer" << endl;
    ...
    if (debugStream0())
        *debugStream() << "End function foo" << endl;
}
```

This generates the following output:

```
Example for class Message
Start function foo
Impossible error true != true !!! //(this output should never occur)
Could not allocate a buffer
End function foo
```

## 2. Output Streams with Macros

Well, neither the code in the first example nor the output looks fine because you have to check each stream against NULL and for this output you can also use cout, cerr and clog. To access streams a bit more sophisticated you should use the following macros.

```

ERROR_STREAM(x), WARNING_STREAM(x), DEBUG_STREAM(x) and MESSAGE_STREAM(x),
  where x is the output message

FooNode::foo{
  MESSAGE_STREAM("Example for class Message" << endl);

  DEBUG_STREAM("Start function foo" << endl);
  ...
  if (true != true)
    ERROR_STREAM("Impossible error true != true !!!" << endl);
  ...
  if (requestBuffer() == NULL)
    WARNING_STREAM("Could not allocate a buffer" << endl);
  ...
  DEBUG_STREAM("End function foo" << endl);
}

```

```

NMM MESSAGE      : FooNode> Example for class Message
NMM DEBUG_MESSAGE : FooNode> Start function foo
NMM ERROR_MESSAGE : FooNode> Impossible error true != true !!! //(this
                        output should never occur)
NMM WARNING_MESSAGE : FooNode> Could not allocate a buffer
NMM DEBUG_MESSAGE : FooNode> End function foo

```

Now the source-code and the output looks nice. The macro prints also some additional information like the node name and message type. These macros can only be used in objects which inherit from NamedObject! For output-messages in other objects you can choose one of the following macros.

```

NODE_ERROR_STREAM(x, y), NODE_WARNING_STREAM(x, y), NODE_DEBUG_STREAM(x, y) and
  MESSAGE_STREAM(x,y), where x is a pointer to a NamedObject and y the output message.

```

The NODE-Macros can be used in objects which has a NamedObject-Pointer. A little example shows how to use these macros.

```

FooNode::FooNode(NamedObject* no) {
  my_no = no; //my_no is an internal pointer to the NamedObject

  NODE_MESSAGE_STREAM(my_no, "Example for streams" << endl);

  NODE_DEBUG_STREAM(no, "begin constructor" << endl);
  ...
  if (true != true)
    NODE_ERROR_STREAM(my_no, "Impossible error true != true !!!" << endl);
  ...
}

```

```
if (requestBuffer() == NULL)
    NODE_WARNING_STREAM(my_no, "Could not allocate a buffer" << endl);
...
NODE_DEBUG_STREAM(my_no, "End function foo" << endl);
}
```

For objects without a NamedObject you can use the GLOBAL-Macros. These Macros use a static NamedObject instance.

```
GLOBAL_ERROR_STREAM(x), GLOBAL_WARNING_STREAM(x), GLOBAL_DEBUG_STREAM(x) and
GLOBAL_MESSAGE_STREAM(x), where x is the output message
```

Another advantage from is that you can compile your application without any message.

## 3. Additional Properties of Streams

The class NamedObject provides additional methods to redirect a stream or to change the output color

### 3.1. Redirecting Streams

The class NamedObject provides methods to redirect streams to each ostream, like a file-streams. The methods are

- setErrorStream(ostream\* stream, MessageLevel level);
- setDebugStream(ostream\* stream, MessageLevel level);
- setMessageStream(ostream\* stream, MessageLevel level);
- setWarningStream(ostream\* stream, MessageLevel level);

The first argument is a pointer to the stream and the second one the MessageLevel. The MessageLevel is an enum which defines the values LEVEL0, LEVEL1, LEVEL2 and ALL\_LEVELS. ALL\_LEVELS means that the operation is executed on all message-level. To disable a stream you can set a stream to NULL. The following example shows how to use these methods.

```
void main() {
    FooNode* node = new FooNode("FooNode"); //create new Node
    fstream o_file("debug_messages.log", ios::out); //create a output file

    node->setDebugStream(&o_file, NamedObject::LEVEL0); //write debug-messages
                                                with level 0 to a file
    node->setMessageStream(0, NamedObject::ALL_LEVELS) //disables all message streams

    //Now we redirect the global stream
```

```
NamedObject::getGlobalInstance().setDebugStream(&o_file, NamedObject::ALL_LEVELS)
//all Messages using the GLOBAL-Macro were written to the output file o_file

....

//You must reset the global stream because o_file is deleted at the end of this
//method but the static instance still exist.
NamedObject::getGlobalInstance().setDebugStream(0, NamedObject::ALL_LEVELS);
}
```

## 3.2. Colored Output Messages

To change the output color you can use the following methods.

- `setErrorColor(Color c, MessageLevel level);`
- `setDebugColor(Color c, MessageLevel level);`
- `setMessageColor(Color c, MessageLevel level);`
- `setWarningColor(Color c, MessageLevel level);`

The first argument is the Color which is an enum in NamedObject and the second one the message level. The enum defines the values `BLACK_COLOR`, `RED_COLOR`, `GREEN_COLOR`, `ORANGE_COLOR`, `BLUE_COLOR`, `PINK_COLOR`, `LBLUE_COLOR`, `GREY_COLOR` and `NOCOLOR`. `nocolor` disables the output color. A little example shows how to use these methods

```
void main() {
    FooNode* node = new FooNode("FooNode"); //create new Node
    node->setWarningColor(NamedObject::blue, NamedObject::LEVEL0);
    node->setMessageColor(NamedObject::green, NamedObject::ALL_LEVELS);

    //Now we change color of the global stream
    NamedObject::getGlobalInstance().setDebugColor(NamedObject::GREY_COLOR,
        NamedObject::ALL_LEVELS)
    NamedObject::getGlobalInstance().setWarningColor(NamedObject::NOCOLOR,
        NamedObject::ALL_LEVELS)
    ....
}
```