Petri Net API

Encapsulating Petri Net-Related Functions in a C++ API

`http://service-technology.org/pnapi`

Version 4.02, 28 July 2010

Niels Lohmann

About this document:

This manual is for the Petri Net API, version 4.02, encapsulating Petri net-related functions in a C++ API, last updated 28 July 2010.

Table of Contents

1 First Steps

1.1 Setup and Installation

1. Go to `http://service-technology.org/files/pnapi` and download the latest release version of the Petri Net API, say `pnapi-4.02.tar.gz`. To setup and compile the Petri Net API, change into your download directory and type

   ```
   tar xfz pnapi-4.02.tar.gz
   cd pnapi-4.02
   ./configure
   make
   ```

   After compilation, a library '`src/libpnapi.a`' and a frontend tool '`utils/petri`' are generated.[1] If you experience any compiler warnings, don't panic: the Petri Net API contains some generated or third party code that we cannot influence.
2. To test whether everything went fine, type

   ```
   make check
   ```

   to execute the testcases located in `tests`. If everything went fine, you should see something like:[2]

   ```
   ## ------------- ##
   ## Test results. ##
   ## ------------- ##

   All 104 tests were successful.
   ```

   If an error occurs, please send the output to `pnapi@service-technology.org`.
3. To install the library, the frontend binary, the manpage, and the documentation, type

   ```
   make install
   ```

   You might need superuser permissions to do so.

If you need any further information, see file `INSTALL` for detailed instructions.

1.2 Contents of the Distribution

The distribution contains several directories:

`doc` The Texinfo documentation of PNAPI. The documentation can be created using '`make pdf`'. Note you need to have PNAPI properly installed before (see installation description above).

`man` The manpage of PNAPI which can be displayed using '`man pnapi`' after having PNAPI installed (see installation description above).

`src` The source code of PNAPI.

`tests` Testcases for PNAPI which check the generated library by using the frontend. Some test scripts use external tools (e.g. Fiona or Wendy) to check whether the calculated operating guidelines are correct. If a needed tool was not found by the configure script, theses tests are skipped.

[1] On Microsoft Windows, the file will be called `petri.exe`.

[2] Some tests use external tools (Fiona, Wendy, Petrify, Genet) to check whether the calculated results are correct. If a needed tool was not found by the configure script, theses tests are skipped.

1.3 Requirements

In order to run, compile, and develop PNAPI, several tools are required.

1.3.1 Runtime

- Petrify (`http://www.lsi.upc.edu/~jordicf/petrify/`) to convert service automata in petri nets (used by default if found).
- Genet (`http://genet.sourceforge.net/`) to convert service automata in petri nets (used by default, if Petrify was not found but Genet was).

The configure script will search for them an set the default paths.

1.3.2 Compilation

To compile the source code from a source distribution, the following tools are required.

- GCC (`http://gcc.gnu.org/`)
- GNU Make (`http://www.gnu.org/software/make/`)
- Libtool (`http://www.gnu.org/software/libtool/`)

1.3.3 Development

In case you want to make changes to the source code, the following tools are required to reconfigure and compile PNAPI.

- Autoconf, `http://www.gnu.org/software/autoconf/`
- Automake, `http://www.gnu.org/software/automake/`
- Bison, `http://www.gnu.org/software/bison/`
- flex, `http://flex.sourceforge.net/`
- Gengetopt, `http://www.gnu.org/software/gengetopt/`
- help2man, `http://www.gnu.org/software/help2man/`
- Texinfo, `http://www.gnu.org/software/texinfo/`

Please check our nightly build server at `http://service-technology.org/nightly` for the versions we use during development.

2 How to adjust your tool when updating the Petri Net API.

When changes in the interface of the Petri Net API have been made, a tool updating its version of the API need to be adjusted. Below will be a short guide written as FAQ which changes are necessary. If you still have problems, updating your tool, please report to `pnapi@service-technology.org`.

2.1 Upgrading vom 3.00 to 4.00

2.1.1 Some functions I used to use are missing, now.

Please check the functions' names. Some functions have been renamed, according to a uniformly naming scheme. If you miss a getter like e.g. `PetriNet::finalCondition()`, it most likely will now begin with "get": `PetriNet::getFinalCondition()`. Appropriately, if you miss a function changing its object, it propably now starts with a "set". Below is a list of functions, that have been renamed (most likely incomplete):

- `PetriNet::finalCondition()` => `PetriNet::getFinalCondition()`

2.1.2 What happened to PetriNet::getInternalPlaces() and PetriNet::getInterfacePaces()?

Since Petri Net API version 4.00 interface places have become labels organized in an interface class, so these functions are no longer needed. If you needed `PetriNet::getInternalPlaces()` you now can use `PetriNet::getPlaces()`; if you needed `PetriNet::getInterfacePlaces()` you now have to get the net's interface by `PetriNet::getInterface()` and to get the former interface places seperately by `Interface::getInputLabels()` and `Interface::getOutputLabels()`, or in one set by using `Interface::getAsynchronousLabels()`. Note, that these sets will be generated by each call, so you shoud use a cache for better performance when e.g. iterating through all labels.

2.1.3 What about place types or node types?

Since there are now only internal places in a net, nodes and places no longer have types. The enum `Node::Type` determining a node's type has been moved to `Transition::Type`.

2.1.4 What about searching places?

If you earlier got a place name and searched for the appropriate place by `PetriNet::findPlace()` you now should perform a search for the appropriate label, if the place search failed by using `Interface::findLabel()`.

2.1.5 I used to catch pnapi::io::InputError when parsing a net.

Then you have to catch `pnapi::exception::InputError` now.

2.1.6 How do I get the preset of a former output place or the postset of a former input place, appropriately?

By `Label::getTransitions()` you will get a set of transitions connected to this label. In case of an input or output label these transition previously where consuming from or producing to the former place.

3 How to use the API in other programs

The Petri Net API provides structures for working with Petri nets.
For using it, simply include the header file `pnapi.h` in your code:

```
#include "pnapi.h"
```

Everything you will use can be found in the following namespaces:

- `pnapi`
- `pnapi::exception`
- `pnapi::formula`
- `pnapi::io`
- `pnapi::verbose`

The classes '`pnapi::PetriNet`' and '`pnapi::Automaton`' provide the main functionality and can serve as an entry point for getting an overview of the API's functionality. For more information, please go to the online API documentation reachable at `http://esla.informatik.uni-rostock.de:8080/job/pnapi/doclinks/1/`.

4 Error Handling

Whenever an error occurs within the API, an exception will be thrown. Here is a list of existing exceptions an when they will be thrown.

4.1 Error

Parent class of all exceptions containing a message, what happened. Catch this to catch everything the API ever will throw.

4.2 InputError

Exception thrown by parsing a Petri Net or a Service Automaton, containing filename, line and token, where the error occured, as well as the error kind, whether it was a syntactic error, violating the grammar below or a semantic error e.g. by giving two nodes the same name.

4.3 NotImplementedError

Thrown when use cases happend we not thought about, yet. If you ever catch such an exception, please report to `pnapi@service-technology.org` which feature is missing.

4.4 AssertionFailedError and UserCausedError

There are several sanity checks within the API e.g. to ensure that there a no conflicts with node names (i.e. there are no two places with the same name). When such a check failes, the API should not cause the program to fail but give it the chance to decide how to handle this error. Hence an exception is thrown instead of using the C '`assert`' macro.

There exist two macros within the API: The first one is used like '`assert`', throws an '`AssertionFailedError`' when the assertion failed, and can be disabled by defining the symbol '`NDEBUG`'. The second one checks user inputs (like node names), throws a '`UserCausedError`' when the assertion failed, and can not be disabled by defining '`NDEBUG`'. Both macros can be replaced by '`assert`' by defining the symbol '`PNAPI_USE_C_ASSERTS`'. This way, also the second macro will be disabled by defining '`NDEBUG`'.

5 Verbose Output

Sometimes verbose output about processes within the API are desired, like

```
TOOL: parsing Service Automaton from file 'myAutomaton.sa'
TOOL: writing Service Automaton to temporary file '/tmp/temp-004200'
TOOL: calling petrify
TOOL: parsing petrify output
TOOL: generating Petri Net from petrify output
```

For this purpose there exists the function pointer'`pnapi::verbose::status`', by default pointing at '`pnapi::verbose::quiet`'. If you wish to print verbose output you only have to provide an own output function and set '`pnapi::verbose::status`' to it.

Your output function must have the signature

```
void myStatusFunction(const char *, ...);
```

beeing a variadic function to be used like '`printf`'.

If you prefer to let the API write the output by itself, set the pointer to '`pnapi::verbose::defaultStatus`'. This way all messages will be written like

```
PNAPI: writing net to file 'result.lola'
```

to standard error.

Verbose output will be written at these situations:

- opening a temporary file
- closing a temporary file

6 Some Examples

Creating a small Petrinet

```
PetriNet net;
Place & p1 = net.createPlace();
p1.mark();
Place & p2 = net.createPlace();
Transition & t = net.createTransition();
net.createArc(p1,t);
net.createArc(t,p2);
```

Assigning a final condition

```
// net from the previous example is recycled here
net.getFinalCondition() = ((p1 == 0) && (p2 == 1));
Place & p3 = net.createPlace();
net.getFinalCondition() = (net.getFinalCondition().getFormula() && (p3 == 0));
```

Reading from stream and aborting if an error occurs

```
istream is;
try
{
  is >> io::owfn >> net;
}
catch (pnapi::exception::InputError error)
{
  std::cerr << error;
  exit(EXIT_FAILURE);
}
```

Writing to a stream

```
ostream os;
// LoLA without formulae
os << io::lola << net;
// LoLA with formulae
os << io::lola << io::formula << net;
```

Reducing by applying some rules

```
net.reduce(PetriNet::SET_STARKE | PetriNet::KEEP_NORMAL);
```

Creating a service automaton

```
Automaton sa(net);
```

Query structural information

```
net.isNormal();
net.isWorkflow();
```

Getting verbose output only when normalizing

```
pnapi::verbose::status = myStatusFunction;
net.normalize();
pnapi::verbose::status = pnapi::verbose::quiet;
```

7 Wildcards in Formulae

Since Version 3.00 the formula classes have been refactored. I.e. from now on they only concern internal places and only two wildcards are left: '`ALL_OTHER_PLACES_EMPTY`' and '`ALL_PLACES_EMPTY`'[1].

When parsing a net, the wildcard will be unfolded by using

```
net.getFinalCondition().allOtherPlacesEmpty(net);
```

A wildcard must be given top level, i.e. the final condition must have the form

```
(...) AND ALL_OTHER_PLACES_EMPTY;
```

or

```
ALL_PLACES_EMPTY;
```

Furthermore a wildcard can be transformed in disjunctive normal form by using

```
net.getFinalCondition().dnf();
```

[1] Other wildcards are left due to compatibility reasons.

8 Open Net File Format

Below is the EBNF grammar for open net files. Comments are not part of the grammar, and hence are not included here.

```
petrinet = interface
           "INITIALMARKING" marking_list ";" finalcondition
           { transition }
         ;

interface = "INTERFACE" interface_ports "PLACE" places ";" roles
          | "PLACE" typed_places roles ports
          ;

interface_ports = input_places output_places synchronous
                | "PORT" identifier input_places output_places synchronous
                  { "PORT" identifier input_places output_places synchronous }
                ;

places = [ "SAFE" [ number ] ":" ] place_list { ";" [ "SAFE" [ number ] ":" ] place_lis

roles = [ "ROLES" identifier { "," identifier } ";" ] ;


typed_places = internal_places input_places output_places synchronous
             | places ";"
             ;

internal_places = [ "INTERNAL" places ";" ] ;

input_places = [ "INPUT" places ";" ] ;

output_places = [ "OUTPUT" places ";" ] ;

synchronous = [ "SYNCHRONOUS" identifier { "," identifier } ";" ] ;

place_list = [ identifier [ "{$" commands "$}" ] ] { "," identifier [ "{$" commands "$]

identifier = id
           | number
           ;

commands = { "MAX_UNIQUE_EVENTS" "=" number
           | "ON_LOOP" "=" ( "TRUE" | "FALSE" )
           | "MAX_OCCURRENCES" "=" ( number | negative_number ) }
           ;

ports = [ "PORTS" identifier ":" identifier { "," identifier } ";" { identifier ":" ide


transition = "TRANSITION" identifier
             [ "COST" number ";" ]
```

```
                [ "ROLES" identifier { "," itentifier } ";" ]
                "CONSUME" [ identifier [ ":" number ] ] { "," identifier [ ":" number ] }
                "PRODUCE" [ identifier [ ":" number ] ] { "," identifier [ ":" number ] }
                [ "SYNCHRONIZE" identifier { "," identifier } ";" ]
                [ "CONSTRAIN" identifier { "," identifier } ";" ]
              ;


marking_list = [ identifier [ ":" number ] ] { "," identifier [ ":" number ] } ;

finalcondition = "FINALMARKING" marking_list ";" { marking_list ";" }
               | "NOFINALMARKING"
               | "FINALCONDITION" [ formula ] ";"
               ;

formula = "(" formula ")"
        | "TRUE"
        | "FALSE"
        | "ALL_PLACES_EMPTY"
        | "NOT" formula
        | formula ( "AND" | "OR" ) formula
        | formula "AND" "ALL_OTHER_PLACES_EMPTY"
        | identifier ( "=" | "#" | "<>" | "<" | "<=" | ">" | ">=" ) number
        ;
```

Terminals are defined below as regular expressions:

```
id                [A-Za-z._=\-\[\]]+
number            [0-9]+
negative_number   -[0-9]+
```

9 Developing the API

The guidelines below will be useful when developing the API.

9.1 How to add a new parser?

For each file below there is a sample file in the templates folder taken from the lola parser, that can be used as template. Just replace "FORMAT" by the actual format name and change these files to your desires.

When introducing a new parser to the API, you have to create a new namespace within the namespace '`pnapi::parser`', named to your new format. Whithin this namespace there will be created a lexer class, a parser class from Bison and an encapsulation of everything ready to be used by IO framework. Everything generated by Bison or Flex will be generated in the namespace "yy". So after all we will have at least the following classes:

- '`pnapi::parser::FORMAT::Parser`' - the encapsulation
- '`pnapi::parser::FORMAT::yy::BisonParser`' - the parser generated by Bison
- '`pnapi::parser::FORMAT::yy::Lexer`' - the lexer class from Flex

9.1.1 The lexer file

The API lexers use the FlexLexer class provided by Flex (`http://flex.sourceforge.net/`) at version 2.5.35. To specify a new lexer you can use the file `parser-FORMAT-lexer.ll` as template.

The lexer file is divided in three parts.

The first part sets some options for Flex. Most options are equal in every lexer; the only options you have to set individually are "yyclass", specifying the class the method "yylex" will be generated in, and "prefix", specifying how to rename the base class.

In the header `FlexLexer.h` there are defined an abstract class named "FlexLexer" and a class named "yyFlexLexer" deriving from FlexLexer. When defining a prefix, the prefix yy from yyFlexLexer will be replaced by the defined prefix to "FORMATFlexLexer". Since we have to extend these lexers by some Methods, we will not use these lexers directly, but derive a third class from FORMATFlexLexer, storing it in the namespace '`pnapi::parser::FORMAT::yy`'. So we have to tell Flex our final class using the option "yyclass".

The second part of the lexer file includes the wrapper header `parser-FORMAT-wrapper.h` providing the parser class. When called by the parser, the lexer has to return values defined in the parser class. Since the namespace path '`pnapi::parser::FORMAT::yy::BisonParser::token`' is rather long, we define an alias "tt" (shot for "token type").

The third part defines lexer states (%s), regular expressions defining the token classes, and their respective actions. Note that there should be called '`LexerError()`' when an error occurs (e.g. an unexpected character). When the parser defines semantic values for tokens, there will be provided a pointer named "yylval" to a union storing these values. So you can pass a value by accessing e.g.

```
yylval->str = strdup(yytext);
```

9.1.2 The parser file

The actual parser will be generated by Bison (`http://www.gnu.org/software/bison/`) at version 2.4.2. To specify a new parser you can use the file `parser-FORMAT.yy` as template.

The parser is divided in four parts.

The first part sets some settings for Bison. Most options are equal in every parser; the only definition you have to set individually is "namespace", specifying the namespace the parser class will be generated in.

The second part forward declares the encapsulating parser class and includes the header `parser-FORMAT-wrapper.h` such as other headers needed by the rules below.

The third part defines the tokens the lexer has to return, semantic types and the types of some tokens and rules.

The last part contains the actual grammar and the code to be executed when matching a rule. Note that every variable or funktion used in such a code block will be stored in the encapsulating parser class. There is a reference to the corresponding class called "parser_", so if you have to access them you do it like

```
$$ = parser_.foo(parser.bar, $1);
```

DO NOT DEFINE ANY VARIABLES OR HELPING FUNCTIONS IN THIS FILE!

9.1.3 The wrapper framework

Since the headers "parser-FORMAT.h" generated by Bison always start with the same "#ifdef", only one Bison generated header can be included at once. So for each parser we have to do the following:

- In the header `parser.h` within the appropriate namespace we have to provide a function named "parse", taking an input stream and returning a PetriNet.
- In the header `parser-FORMAT-wrapper.h` we first have to include the header `FlexLexer.h`. Note that this wrappuing header will be included both from files already including `FlexLexer.h` and from those ones that do not. So we have to make sure we only include this header once. And remember that we defined a prefix in the file `parser-FORMAT-lexer.ll`, so we have to do so here, too, by defining '`#define yyFlexLexer FORMATFlexLexer`'.

  Next, canonical to all parsers, the following classes and functions are defined:
  - '`yy::yylex`' the function called by BisonParser and calling '`yy::Lexer::yylex`'
  - '`yy::Lexer`' the class Flex will generate its yylex in. Note that this class will derive from an abstract lexer class, passing its own base class as third template argument.
  - '`Parser`' the encapsulating parser class. Each variable or helping function needed by the actual parser will be defined here.
- In the file `parser-FORMAT-wrapper.cc` the following functions have to be implemented:
  - '`pnapi::parser::FORMAT::parse`'
  - '`pnapi::parser::FORMAT::yy::yylex`'
  - '`pnapi::parser::FORMAT::yy::Lexer::Lexer`'
  - '`pnapi::parser::FORMAT::yy::BisonParser::error`'
  - '`pnapi::parser::FORMAT::Parser::Parser`' - initializing all variables
- In the header `myio.h` add an appropriate value to the enum "Format" and a stream manipulator (i.e. a function taking and returning an ios_base reference) with an appropriate name.
- In the file `myio.cc` add this value to the switch in '`operator>>`' according to the existing parsers.
- In the file `io-format.cc` implement the manipulator defined in `myio.h` (see the other format manipulators for samples).

Finally add this new input format to the frontend "Petri".

9.2 How to add a new output format?

9.2.1 Changes in the header `myio.h`

To add a new output format first add a new value to the "Format" enum. After doing so, create a new namespace named "__FORMAT" within the "io" namespace. Here, for each Petri Net component to be written out, you have to add a function named "output" returning an output stream reference and taking such a reference and the appropriate component.

9.2.2 Changes in the file `myio.cc`

For each '`operator<<`', i.e. for each component you want to be written out, add a case for the new format to the appropriate switches calling the appropriate output method. You do not have to provide a case for every component in the Petri Net, i.e. if you e.g. do not need to print the Final Condition or communication labels you can leave them out.

9.2.3 Changes in the file `io-format.cc`

Here you have to implement the various output functions. Note that since you overloaded the '`operator<<`' and provided output functions for every component to be written, you now can easily use them and delegate printing of subcomponents to their respective output function.

9.2.4 Changes in the file `petrinet.h`

Add the output function to print a Petri Net as a frient to the PetriNet class, so you can even print components that are not ment to be accessed by others than the PetriNet itself.

9.2.5 Printing sets of pointers

Most components of a Petri Net are stored in sets storing pointers to the actual objects, such as '`places_`', '`transitions_`' or '`arcs_`'. To write out such sets you do not have to iterate through them by yourself, just pass the whole set to the output stream. This will call the appropriate output function for each element of the set. By using the stream mainpulator '`delim`' you can specify a delimiter to be printed between two elements of the set.

An example:

```
std::ostream & output(std::ostream & os, const PetriNet & net)
{
  os << "PLACES\n  " << delim(", ") << net.places_ << ";\n";
  return os;
}
```

This will print all places, as specified in the output function for places, in a comma separated list finishing the list by a semicolon. Note that changing the delimiter within the output functions of the elements of the set does not affect the delimiter of the set itself. So you can write a set of transitions as linebreak seperated list and within a transition its pre- and postset as comma seperated list.

This does not only work for sets but also for vectors and multimaps.

9.2.6 Output modes

Sometimes you have to write a certain component more than once, e.g. a place in the oWFN format will be named both in the PLACES section, to declare there is a place with this name in the net, and in the pre- or postset of each transition connected with this place. And sometimes you have to print a component differently, depending on the context it will be written in. That is why there is a enum named "Mode" defined in the header `myio.h`.

Before passing a component (or a set of components, see above) to the output stream you can set the mode by using the stream manipulator '`mode`'.

```
os << mode(io::util::INNER) << net.places_;
```

So in the output function of a component that will be printed in various modes you first have to get the recent mode and then do the appropriate output. For example:

```
switch(ModeData::data(os))
{
case io::util::PLACE { ... }; break;
case io::util::INNER { ... }; break;
default: /* do nothing */ ;
}
```

9.3 How to add a new stream manipulator?

Sometimes one wants to pass additional information to the output functions, e.g. whether to print the Final Condition or whether to remove role information. These information do not belong to the net itself but to the output stream, so you have to provide a manipulator for telling the stream, what to do.

9.3.1 A manipulator setting binary information

First you have to create a manipulator in the header `myio.h` to write this data to the stream. Simply add a function taking and returning an '`std::ostream`' reference above the definition of the "Format" enum.

Then you have to create a new type for the stream data itself. Even if you just want to store a basic type you have to create a new one or it would interfere with others; a simple typedef does not suffice. Below the various output function some structs are already defined, so you can define your struct (i.e. the data type) here, too. By providing a constructor to your struct you can define a default value.

Next you create the stream meta data out of your recently created data type by defining a shotcut. Search for "TYPE NAME SHORTCUTS" to find the appropriate position and some samples. Now your data has a place in the stream to be stored and you can access your struct whith something like '`util::FooData::data(myOutputStream)`'.

Now you only have to implement the manipulator in the file `io-format.cc`. Simply set the information e.g. like

```
std::ostream & foo(std::ostream & os)
{
  util::FooData::data(os).value = true;
  return os;
}
```

9.3.2 A manipulator setting more than binary information

If you do not only want to set a flag, "do this" or "do not do this", you have to pass some information to your manipulator. I.e. instead of calling something like

```
myOutputStream << foo << net;
```

you have to call something like

```
myOutputStream << bar(42) << net;
```

Creating such a manipulator is pretty much the same, but instead of adding a function taking and returning an output stream reference you now have to create a function taking the information and returning an appropriate instanciation of the Manipulator template class. This is done below the StreamMetaData typedefs; see '`mode`' or '`delim`' for samples.

This function will be implemented in the file `myio.cc`, below the implementation of most operators. Also see '`mode`' or '`delim`' for samples here.

9.4 What to do after introducing new classes?

After introducing a new class that has to be written out, first open the header `myio.h` and add an output function for each format printing this new class. Afterwards add a new '`operator<<`' to the list of output operators. If you want to print sets of this class, you also have to provide a '`compareContainerElements`' function for pointers to objects of this class, above the operator list.

Now you only have to open the file `myio.cc`, implement the operator and the comparison function and you are ready to fit the various output function in the file `io-format.cc` to the new class.

9.5 Coding style

The API has a relative consistent coding style and should keep this uniform look for a long time. So if you are developing the API, please follow the policies below.

9.5.1 Indentation

Everytime a new block of code is entered (i.e. after each opening brace) the code will be indented by two spaces more than the line above.

Exception: Within a switch statement or within a namespace will not be indented.

```
foo = 42;
{
  int bar = 23 + foo;
}

namespace foo
{
class Bar;
}

switch(foo)
{
case 23: break;
case 42: break;
default: /* do nothing */ ;
}
```

9.5.2 Operators and commas

There should be a single space before and after each operator, especially before and after each "*" or "&" when defining a pointer or a reference.

Exception: The is no space between the unary operators ! (negation), * (dereferring) and & (getting the adress of a variable) and their appropriate operand. There is also no space between a comma (or semicolon) and the value before.

Always use logical operators (&&, ||, !) instead of mnemonics (AND, OR, NOT).

```
int i = foo(42, 23);
int * p = &i;
i = foo(i + 42, *p);
bool b = ((bar(i) && bar(*p)) || (!baz(i)));
```

9.5.3 Braces

An opening brace is *always* below the first character of the respective keyword or the return value, one line below the keyword or the function head. The closing brace ist in the same column as well.

```
int foo()
{
}

if(b)
{
}
else
{
}

Bar::Bar() :
  Baz()
{
}
```

9.5.4 Parameters and return values

Basic types, such as '`int`', '`char`' or pointers can always be called by value. Everything else should be passed as const reference to a function, if it is not to be changed within it.

If the function returns a member, the result should be passed as a copy (call by value) or as a const reference as well, unless the member is allowed to be changed by others. In this case return a reference.

If a function creates a new object without storing a pointer to it (e.g. the '`parser::FORMAT::parse`' functions) generate the object on the stack and return a copy instead of creating it on the heap an returning a reference.

In the header files belongs only the signature of a function, so spare the parameter names there. If a function has a parameter not needed by the implementation, you can spare the name there, too.

IMPORTANT: When implementing a function always use the same type spelling as in the declaration, even if you areusing a "using" statement. I.e. if you have defined a function taking an '`std::set`' and your source file begins with

```
using std::set;
```

anyway use the spelling '`std::set`' instead of only '`set`' in the function head. Otherwise doxygen will not be able to match function declaration and implementation.

```
int foo(int, int &, int *);
```

9.5.5 Comments and class description

When introducing new variables, functions or types, *immediately* add a short description about what you just did, or even better, about what you are about to do. In headers, a full description of a type or an introduced namespace and a short discription of each member whould be nice.

The full description of a function, at least using the "brief" keyword, belongs above its implementation. If the behaviour of a function is more complicated than a few simple lines, please add a more verbose description below the brief one. Also, if the parameters are not explained enough by their names, comment them by using the "param" keyword.

When defining a class, group its members in the following order: types, static constants, static variables, constants, variables, static methods, methods; each of them in the order public, protected, private.

If such a group (e.g. public methods) holds too much members, group them by common properties, like e.g. constructors/destructors, getters or object changing methods.

```
/*!
 * \brief A sample class
 */
class Foo
{
private: /* private variables */
  /// a member variable
  int bar_;

public: /* public methods */
  /*!
   * \name constructors/destructors
   */
  //@{
  /// constructor
  Foo();
  /// copy constructor
  Foo(const Foo &);
  /// destructor
  ~Foo();
  //@}

  /*!
   * \name getters
   */
  //@{
  /// get bar
  int getBar() const;
  //@}
};


/*!
 * \brief get bar
 */
int Foo::getBar() const
{
  return bar_;
}
```

9.5.6 Naming

Since the API is to be used by other tools, the global namespace should be kept as clean as possible. So everything defined within the API should be defined in the namespace "pnapi" or a subnamespace. However, preprocessor makros can only be defined at a global scope, so use them as rare as possible.

- namespaces: Lower case, singular, if possible only one word, like "pnapi", "parser" or "formula".

- types: Upper camel case, singular, like "PetriNet", "AbstractParser" or "StreamMetaData".
- member variables/constants: Lower camel case, ending with an underscore if private, like "places_", "pathToPetrify_" or "nodesByName_".
- functions or not-member variables/constants: Lower camel case, like "getPlaces", "isNormal" or "printToSTG".
- enum values: Upper case, like "OWFN", "LOLA" or "WOFLAN".
- makros: Upper case with underscores, beginning with "PNAPI", like "PNAPI_FOREACH" or "PNAPI_PARSER_LOLA_WRAPPER_H".
- files: Lower case, seperated by a dash, like "petrinet.h" or "parser-lola-wrapper.h".

9.5.7 Constructors

Sometimes you have to initialize member variables or call the constructor of a parent class within a constructor. If you have to, write the function head, a single space, a colon and one line below, indented by 2 spaces, first the parental constructor call and then the member initializations.

Note that the order members are initialized depends on the order they are declared in the class description, not on the order you name them here.

```
Foo::Foo(int i, int j) :
  ParentClass(i), bar_(j),
  baz_(42)
{
}
```

9.5.8 Long lines

Sometimes, when calculating complicated boolean expressions or when using long access paths, a line can become too long. If so, you can add a linebreak after each comma or *after* each logical operator. The next line will be indented as much as necessary, so that the first character of the following line is below the first character of the first argument or subexpression.

Another candidate for too long lines is the for statement. To shorten such a line, please use the '`PNAPI_FOREACH`' makro defined in the header `util.h` or break after the first semicolon and indent the second line by one space more than descriped above.

```
bool b = (very.very.long->path->to.aValue() &&
          anotehr.very->long.pathTo->a.value());

for(std::set<Place *>::iterator p = places_.begin();
     p != places_.end(); ++p)
{
}

PNAPI_FOREACH(p, places_) // this does the same as the for statement above
{
}
```

9.5.9 Error handling

Since the API will be included by other programs, it is not the job of the API to terminate the program. So if an error occurs, do not call '`exit(EXIT_FAILURE);`' but throw an exception, so actual program can catch it and decide by itself, how to handle this error.

This also applies to assertions. If something you want to make sure within the API went wrong, do not terminate the whole program by using asserts but inform the program by using appropriate exceptions.

There exists two macros that can be used like '`assert(expr)`'. The first one is '`PNAPI_ASSERT(expr)`'. It evaluates the given expression and throws an '`AssertionFailedError`' containing the file and line of the assertion, when evaluation to false. Like assert, this macro can be disabled by declaring the symbol NDEBUG.

Some errors, however, depend on the user input. These errors should still be reportet when usual assertions are disabled. For this purpose you can use the macros '`PNAPI_ASSERT_USER(expr)`', '`PNAPI_ASSERT_USER(expr, msg)`' or '`PNAPI_ASSERT_USER(expr, msg, type)`', optionally specifying a proper error message and an error type (for a better handling in a catch block). Default message will be the expression itself and the default type will be '`UE_NONE`'.

Note that, since these kind of errors are to be caused by the user, no debugging in the API itself is expected, so there will be no further information (file, line), where the error occurred.

9.5.10 Inclusion and "using"

Within a source file (*.cc) you can do as you wish, but keep the headers as clean as possible. I.e. only include an other header if necessary and if forward declaration does not suffice. Also *never* use a "using" statement in a header file; allways use the full path instead.

10 Theory

The Petri Net API implements several algorithms and result published in several scientific articles.

10.1 Open Nets

Basic Definitions

Karsten Wolf. **Does my service have partners?**. *LNCS ToPNoC*, 5460(II):152-171, March 2009. Special Issue on Concurrency in Process-Aware Information Systems.

Structural Reduction Rules

Tadao Murata. **Petri nets: Properties, analysis and applications**. *Proceedings of the IEEE*, 77(4):541–580, 1989.

Peter H. Starke. **Analyse von Petri-Netz-Modellen**. Teubner Verlag (1990).

Thomas Pillat. **Gegenüberstellung struktureller Reduktionstechniken für Petrinetze**. Diplomarbeit, Humboldt-Universität zu Berlin, March 2008.

Normalization

Niels Lohmann, Peter Massuthe, and Karsten Wolf. **Operating guidelines for finite-state services**. In Jetty Kleijn and Alex Yakovlev, editors, *28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN 2007, Siedlce, Poland, June 25–29, 2007, Proceedings*, volume 4546 of Lecture Notes in Computer Science, pages 321–341. Springer-Verlag, June 2007.

Constraints (product operation)

Niels Lohmann, Peter Massuthe, and Karsten Wolf. **Behavioral constraints for services**. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24–28, 2007, Proceedings*, volume 4714 of Lecture Notes in Computer Science, pages 271–287. Springer-Verlag, September 2007.

See `http://service-technology.org/publications` for more information.

11 How to run PNAPI frontend "Petri"

Petri is a command line tool and does not come with a fancy GUI. However, Petri can be easily scripted and combined with other tools.

11.1 Command Line Options

For an overview of the command line options, type '`petri -h`' to see the following help screen:

```
petri 4.02

A command-line frontend for the Petri Net API

Usage: petri [OPTIONS]... [FILES]...

  -h, --help                    Print help and exit
      --detailed-help           Print help, including all details and hidden
                                  options, and exit
      --full-help               Print help, including hidden options, and exit
  -V, --version                 Print version and exit

Input and Output:
  -i, --input=FORMAT            Select the input file format  (possible
                                  values="owfn", "lola", "pnml", "sa",
                                  "tpn" default=‘owfn’)
  -o, --output=FORMAT           Select the output file format(s)  (possible
                                  values="owfn", "lola", "pnml", "sa",
                                  "tpn", "dot", "png", "pdf", "eps",
                                  "svg")
      --canonicalNames[=FILENAME]
                                Renames nodes to avoid problems with conversion
                                  to other formats. Mapping may be written to a
                                  given file.
  -f, --formula                 Places a final condition at the end of LoLA
                                  files. Has no effect to other output formats.
                                   (default=off)
      --removePorts             Removes ports in output files.  (default=off)
      --guessFormula            Derive the final condition from the structure
                                  (i.e., the sink place) of the net.
                                  (default=off)

 Group: Automaton Converter
  Defining a program, to convert Service Automata to Petri Nets
      --genet[=FILENAME]        Use Genet; specify the path  (default=‘genet’)
      --petrify[=FILENAME]      Use Petrify; specify the path
                                  (default=‘petrify’)

Composition and Product:
      --compose=NET             Compose each net given in the input section
                                  with this net.
      --produce=CONSTRAINT      Calculate the product.

Check for Structural Properties:
      --isFreeChoice            Check if the Petri net is free choice, i.e. if
                                  conflicting transitions share all pre-places.
                                   (default=off)
      --isNormal                Check if the Petri net is normal, i.e. every
                                  transition is connect to at most one
                                  interface place.  (default=off)
      --isWorkflow              Check if the Petri net has workflow structure,
                                  i.e. it (i) has a distinct source place, (ii)
                                  has a district sink place, and (iii) every
                                  node lays on a path between the source place
                                  and the sink place.  (default=off)
```

```
  -c, --check=PROPERTY          Check a structural property of the Petri net,
                                  see ‘isFreeChoice’, ‘isNormal’, and
                                  ‘isWorkflow’ for details.  (possible
                                  values="freechoice", "normal",
                                  "workflow")

Modifications:
  -r, --reduce=RULES            Apply structural reduction rules, i.e. reduce
                                  the structure of the net while preserving
                                  liveness and boundedness.  (possible
                                  values="0", "1", "2", "3", "4",
                                  "5", "6", "starke", "pillat",
                                  "dead_nodes", "identical_places",
                                  "identical_transitions", "series_places",
                                  "series_transitions", "self_loop_places",
                                  "self_loop_transitions", "equal_places",
                                  "starke3p", "starke3t", "starke4",
                                  "starke5", "starke6", "starke7",
                                  "starke8", "starke9", "once",
                                  "k_boundedness", "boundedness",
                                  "liveness")
  -n, --normalize               Normalize the Petri net, i.e. change to
                                  structure such that every transition is
                                  connected to at most one interface place.
                                  (default=off)
  -N, --negate                  Negate the final condition of the net, i.e.
                                  every specified final marking is now
                                  non-final. The result is the 'anti open net'.
                                   (default=off)
  -M, --mirror                  Mirror the interface, i.e. change the direction
                                  of communication  (default=off)
      --dnf                     Convert final condition to disjunctive normal
                                  form  (default=off)

Configuration:
      --config=FILE             Read configuration from file.
      --dot=FILENAME            Set the path an binary of dot.  (default=‘dot’)
      --tmpfile=FILENAME        Set the path and name of temporary files
                                  (default=‘/tmp/petri-XXXXXX’)
      --noClean                 Do not delete temporary files.  (default=off)

Debugging:
  -v, --verbose                 Show verbose output  (default=off)
      --suffix_owfn=SUFFIX      Suffix for open net files
      --suffix_sa=SUFFIX        Suffix for service automaton files
      --stats                   Display time and memory consumption on
                                  termination.  (default=off)
```

12 Error Codes

In case any error occurs, Petri aborts with exit code '1' and prints a message with an error code to the standard error stream.

#01 A wrong command-line parameter was given or there was a problem with the combination of command-line parameters. This message is usually accompanied by another message describing the exact problem. Run '`petri --help`' for an overview of the valid command-line parameters.

```
pnapi: unrecognized option '--foo'
pnapi: invalid command-line parameter(s) -- aborting [#01]
```

#02 An error occurred while parsing the net. The message will give further information.

```
pnapi: Input Error: stdin:1: error: syntax error, unexpected $end, expecting KEY_INTERFACE o
```

#03 Could not open input file.

```
pnapi: could not read from file 'foo' -- aborting [#03]
```

#04 Too many nets used with '`--produce`' parameter.

```
pnapi: at most one net can be used with '--produce' parameter -- aborting [#04]
```

#05 Graphviz dot was not found by configure script; see README. Necessary for option '`--output=FORMAT`' where FORMAT is 'png', 'eps', 'pdf' or 'svg'.

```
pnapi: Graphviz dot was not found by configure script -- aborting [#05]
```

#06 Cannot open UNIX pipe to Graphviz dot. Create dot file with '`--output=dot`' and call Graphviz dot manually.

```
pnapi: cannot open UNIX pipe to Graphviz dot -- aborting [#06]
```

#07 Exception caught.

```
pnapi: Exception caught: node name conflict: node 'p2' already exists -- aborting [#07]
```

#11 Error opening a file to write. Usually, this error occurs in case Petri is called in a directory without writing permissions or the output file is already present, but write protected. Output files can be the target files of the `--output` parameter.

```
pnapi: could not write to file 'foo.owfn' -- aborting [#11]
```

#13 Petri cannot create a temporary file `/tmp/petri-XXXXXX`, where '`XXXXXX`' is replaced by a unique name. If this error occurs, check whether the `/tmp` folder exists and you have the rights to write there. One reason for this error can be that you are

running Petri under Windows (outside Cygwin), where UNIX-style path names are not recognized. In that case, try using the `--tmpfile` parameter.

```
pnapi: could not create a temporary file ‘/tmp/petri-k4CS4x’ -- aborting [#13]
```

13 Version History

PNAPI is developed under the "Release Early, Release Often" maxime (see `http://catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ar01s04.html`): Whenever enough integrated or a non-trivial changes have summed up, a new version is published. Though this releases might now always mark significant changes, they at least allow to quickly fix bugs and avoid infinite procrastination.

Version 4.02 (2010-07-28)

- made petri configurable (paths to external tools, configuration file)
- removed sa2sm and sa2owfn
- allowed parsing of Final Conditions only
- introduced method to guess place relations
- refactored interface classes (labels without a port will get their own ports)
- refactored compose according to the tpp glossar (i.e. two ports be composed iff they match perfectly)
- added parameter `--removePorts` to remove ports in output files (dot, owfn)
- using dot output, the final condition will now be written below the net
- removed bug with node IDs in dot output

Version 4.01 (2010-05-11)

- removed memory leak (thanks to wendy)
- implemented Woflan parser
- caged lexer and parser in classes (i.e. cleaned up global namespace)
- added a parameter `--guessFormula` to the frontend tool to guess the final condition from the structure of the net (i.e., use the sink place)
- added parameter `--canonicalNames` to rename nodes

Version 4.00 (2010-03-14)

- HEAVY INTERFACE CHANGES
- make check now tests, whether customer tools still work with the API (see `https://gna.org/task/?6813`)
- added Makefile.am.customer for customer tools (will be generated by compiling the API)
- added make target for zcov
- copied parameter `--stats` from Wendy to Petri
- integrated `Output.*` and `verbose.*` to Petri (see `https://gna.org/task/?6837`)
- reimplemented open net normalization function `normalize()`) according to [LohmannMW_2007_icatpn]
- implemented PNML import and export based on the code of `http://www.w3.org/XML/9707/XML-in-C`
- added a utility that adds a configuration interface to a labeled Petri net
- added a utility that forms the final conditions in CDNF (see `https://gna.org/task/index.php?6845`)
- added a utility that unfolds HL-oWFN to LL-oWFN (see `https://gna.org/task/index.php?6876`)

- extended oWFN by role information (see `https://gna.org/task/index.php?6887`)
- adapted scripts etc. to finally use the PNAPI as shared library
- added pnapi::version() to get PNAPI package string in customer tools
- introduced Interface class to handle input, output and synchronous labels
- added handle for verbose output
- rearranged exceptions
- fixed bug #14274: PNAPI: Memory leaks
- fixed bug #14856: PNAPI: Reduction by Starke rule #9 crashes under certain conditions
- fixed bug #14887: PNAPI: make check fails under MinGW
- fixed bug #15179: PNAPI: avoid using mktemp in Output.cc
- fixed bug #15415: PNAPI: Assertion failure when reducing net

Version 3.00 (2009-11-03)

- final condition can now be converted in disjunctive normal form
- added type getter to formula classes
- started proper versioning
- started proper documentation
- formulae now only concern internal places; only left wildcard is '`ALL_OTHER_PLACES_EMPTY`'
- fixed bug #14648: PNAPI: Shift/reduce conflict in `parser-pn.cc`

Version 2.00

- removed memory leaks
- integrated Output class from Wendy
- added make target "win32" to compile the api without linking against the cygwin1.dll
- added manpage for petri tool
- API can be compiled with foreign config.h (removed macro dependencies)
- added scripts to use LCOV ('http://ltp.sourceforge.net/coverage/lcov.php') to determine test case coverage (use 'make cover' in 'tests' directory)
- new features:
  - Transition costs (https://gna.org/task/index.php?6615)
  - test cases and integration of new owfn parser (https://gna.org/task/index.php?6658)
  - Function to add a prefix to all places and transitions (https://gna.org/task/index.php?6749)
  - Integrate Genet for SA2ON conversion (https://gna.org/task/index.php?6767)
- fixed bugs:
  - PetriNet does not know Synchronous Labels of its transitions (https://gna.org/bugs/index.php?13936)
  - sa2sm/sa2owfn ignore synchronous labels (https://gna.org/bugs/index.php?13968)
  - Net reduction invalidates finalCondition and leads to core dump (https://gna.org/bugs/index.php?13969)
  - Composition of more than two nets fails because of prefixes to I/O places (https://gna.org/bugs/index.php?13990)
  - No (public) method to remove nodes (https://gna.org/bugs/index.php?14003)
  - Removal of dead nodes also deletes interface place (https://gna.org/bugs/index.php?14064)

    - Arc weights to/from interface places are ignored by isNormal() function (https://gna.org/bugs/index.php?14116)
    - Normalization yields unequivalent nets (https://gna.org/bugs/index.php?14119)
    - No error when using SYNCHRONIZE with undeclared label in oWFN file (https://gna.org/bugs/index.php?14127)
    - LoLA-Parser is more strict than LoLA's parser (https://gna.org/bugs/index.php?14206)
    - synchronous steps become tau-transitions (https://gna.org/bugs/index.php?14335)
    - product with constraint does not work with synchronous communication (https://gna.org/bugs/index.php?14411)
    - When composing nets, costs of a transition get lost (https://gna.org/bugs/index.php?14417)
- further (undocumented) bugs fixed
- further (undocumented) changes

Version 1.1

- implemented scanner and parser for oWFN files
- structural changes
- added formula class
- overworked reduction rules
    - murata rules will only be applied if the weight of all involved arcs is 1
    - some murata rules are extended by arc weights according to Thomas Pillat's proofs (see [pillat2008])
    - remaining rules have been deactivated
    - normalized nets can be kept normal
    - implemented new functions
    - findLivingTransition
    - calcSuccessorMarking
    - marking2Places
    - calcCurrentMarking
    - initMarking
    - isWorkflowNet
    - isFreeChoice
    - set and get methods for P, P_in, P_out, T and F
    - reevaluateType
    - isNormal
    - normalize
    - makeInnerStructure
    - set and get methods for invocation_string and package_string
- bugfixes
    - added missing "=NULL" after use of delete
    - searching for a place which has been removed or renamed won't cause a crash anymore
- added files
    - generic Doxygen configuration file
    - testcases

Version 1.0

- copied api from fiona
- created Makefile.am and configure.ac for autotools usage
- it's compilable running autoreconf, configure and make

The most recent change log is available at PNAPI's website at `http://service-technology.org/files/pnapi/ChangeLog`.

Appendix A The GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both

covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

   B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

   C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

   D. Preserve all the copyright notices of the Document.

   E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

   F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

   G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

   H. Include an unaltered copy of this License.

   I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its

Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.