Milena – Technical documentation

LRDE

Copyright

Contents

Chapter 1

Coding Style

The coding style is going to be updated. However, there are few reference documents available here:

- https://trac.lrde.org/olena/wiki/CodingStyle
- https://olena.lrde.epita.fr/cgi-bin/twiki/view/Olena/CodingStyle010

Chapter 2

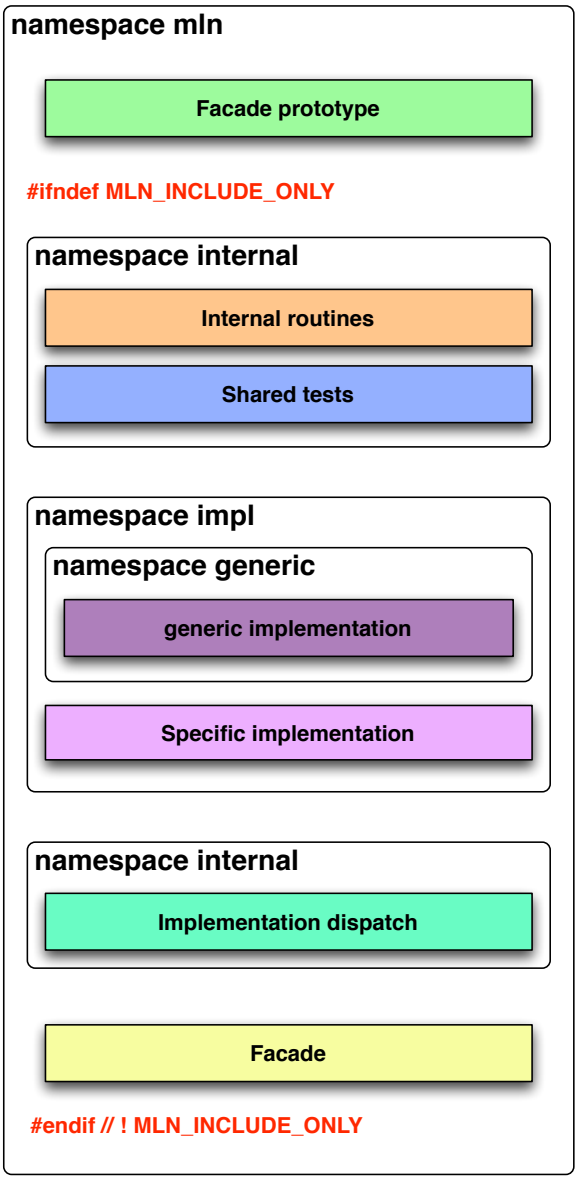
Extend the Library

2.1 Add a new image value type

2.2 Add a new image type

2.3 Add a new routine

Writing a new routine in Milena requires a specific layout in the code. This layout is described in figure 2.3.



For a better understanding, we are going to comment this figure (fig. 2.3) from the bottom to the top.

- **Facade**. The facade is the public routine that will be called by the user. It **should** not contains any processing code but call dispatch routines. It **must** be fully generic.

  An example:

```
template <typename I, typename N>
inline
mln_concrete(I)
erosion_tolerant(const Image<I>& input, const Neighborhood<N>& nbh,
                 unsigned rank)
{
  trace::entering("morpho::erosion_tolerant");
  mln_precondition(exact(input).is_valid());
  mln_precondition(exact(nbh).is_valid());

  mln_concrete(I) output
      = internal::erosion_tolerant_dispatch(input,
                                            nbh,
                                            rank);

  trace::exiting("morpho::erosion_tolerant");
  return output;
}
```

- **Implementation dispatch**. According to specific information, the dispatch will call the proper implementation of the algorithm. The dispatch is almost always static. It might contains dynamic dispatch in very specific cases. Dispatch function names are composed of the routine name with "_dispatch" appended to it. Dispatch entry point routine must have the same prototype as the facade. Dispatch routines are not intended to be called by the user.

  An example:

```
namespace internal
{

  // Generic case, whatever the image value and domain
  // types.
  template <typename I, typename N>
  mln_concrete(I)
  erosion_tolerant_dispatch(trait::image::kind::any,
                            trait::image::speed::any,
                            const I& input, const N& nbh,
                            unsigned rank)
  {
    return impl::generic::erosion_tolerant(input,
                                           nbh,
                                           rank);
```

```
  }

  // The image has bool values and is defined on a box.
  template <typename I, typename N>
  mln_concrete(I)
  erosion_tolerant_dispatch(trait::image::kind::logic,
                            trait::image::speed::fastest,
                            const I& input, const N& nbh,
                            unsigned rank)
  {
    return
     impl::erosion_tolerant_on_set_fastest(input,
                                           nbh,
                                           rank);
  }


  // Dispatch entry point.
  template <typename I, typename N>
  inline
  mln_concrete(I)
  erosion_tolerant_dispatch(const Image<I>& input,
                            const Neighborhood<N>& nbh,
                            unsigned rank)
  {
    return
     erosion_tolerant_dispatch(mln_trait_image_kind(I)(),
                               mln_trait_image_speed(I)(),
                               exact(input),
                               exact(nbh),
                               rank);
  }

} // end of namespace mln::internal
```

- **Specific implementation**. Implementation routine are intended to be called by the user. Thus, they must have an explicit name of what they do and what they take as parameter. They implement de processing code which may be specific for certain types. *namespaceimpl* may also have a sub namespace called *generic* for the generic algorithm.

  An example:

```
namespace impl
{

  namespace generic
  {

    template <typename I, typename N>
```

```
    mln_concrete(I)
    erosion_tolerant_on_set(const Image<I>& input_,
                            const Neighborhood<N>& nbh_,
                            unsigned rank)
    {
      trace::entering("impl::generic::erosion_tolerant");

      // Do it...

      trace::exiting("impl::generic::erosion_tolerant");
      return output;
    }

  }


  template <typename I, typename N>
  mln_concrete(I)
  erosion_tolerant_on_set_fastest(const Image<I>& input_,
                                  const Neighborhood<N>& nbh_,
                                  unsigned rank)
  {
    trace::entering("impl::erosion_tolerant_on_set_fastest");


    // Do it...

    trace::exiting("impl::erosion_tolerant_on_set_fastest");
    return output;
  }


} // end of namespace impl
```

- **Shared tests and internal routines**. This is the right place to move all the internal routines related to this new routine. They will be hidden to the user.
- **Facade prototype**. This is what the user should see first if the file is edited. The documentation should be written here.

Note that all parts of the file except the "facade prototype" are included between $MLN\_INCLUDE\_ONLY$ guards.

Chapter 3

Developer Ressources

3.1 Project Quality

- QA Center : https://trac.lrde.org/olena/wiki/QACenter
- Buildbot : https://buildfarm.lrde.org/buildfarm/oln/
- Trac : http://trac.lrde.org/olena

3.2 Questions and Answers

The best way to keep in touch with the latest patches or ask your questions is to subscribe to our mailing lists.

Currently four mailing-lists are available:

Olena	Discussion about the project Olena
Olena-bugs	Bugs from Olena projects
Olena-core	Internal list for the Olena project
Olena-patches	patches for the Olena project

You can subscribe to these mailing lists at the following address:

`https://www.lrde.epita.fr/mailman/listinfo/`

Just click on the name of the mailing list you want to subscribe to and fill out the form.