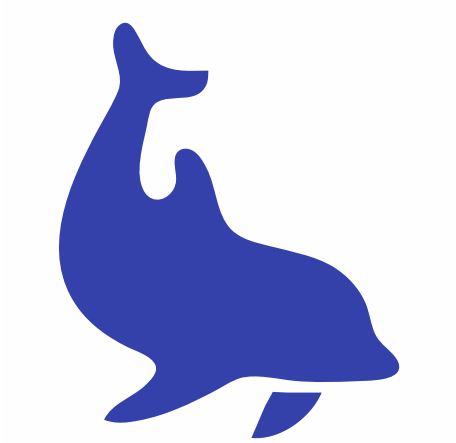
DOLFIN User Manual

September 23, 2003



Hoffman, Logg, et al.

This manual has been written by:
Johan Hoffman, Johan Jansson, and Anders Logg.

Contents

1 Introduction

This is a first draft for a DOLFIN manual. Contributions are most welcome.

2 Installation

In preparation.

3 Linear algebra

In preparation.

4 Grid management

In preparation.

5 The log system

The purpose of the log system is to provide a simple and clean interface for logging messages, including warnings and errors.

The following functions / macros are provided for logging:

```
dolfin_info();
dolfin_debug();
dolfin_warning();
dolfin_error();
dolfin_assert();
```

Examples of usage:

```
dolfin_info("Created vector of size %d.", x.size());
dolfin_debug("Opened file");
dolfin_warning("Unknown cell type.");
dolfin_error("Out of memory.");
dolfin_assert(i < m);
```

Note that in order to pass additional arguments to the last three functions (which are really macros, in order to automatically print information about file names, line numbers and function names), the variations `dolfin_debug1()`, `dolfin_debug2()` and so on, must be used.

As an alternative to `dolfin_info()`, C++ style output to cout (`dolfin::cout`, and not `std::cout`) can be used. These messages will be delivered to the same destination as messages by use of the function `dolfin_info()`.

Examples of usage:

```
cout << "Assembling matrix: " << A << endl;
cout << "Refining grid: " << grid << endl;
```

The `dolfin_assert()` macro should be used for simple tests that may occur often, such as checking indexing in vectors. The check is turned on only if DEBUG is defined.

To notify progress by a progress session, use the class Progress.

Examples of usage:

```
Progress p("Assembling", grid.noCells());

for (CellIterator c(grid); !c.end(); ++c) {
  ...
  p++;
}
```

Progress also supports the following usage:

```
p = i;    // Specify step number
p = 0.5;  // Specify percentage
p.update(t/T, "Time is t = %f", t);
```

6 Handling parameters

In preparation.

7 Writing a new module / solver

In preparation.

7.1 Contributing code to DOLFIN

If you have created a new module, fixed a bug somewhere, or have done a small change somewhere which you want to contribute to DOLFIN, then currently the best way to do so is to send a patch to the maintainer. A patch is a file which describes how to transform one file or directory structure into another. The patch is built by comparing a version which both parties have against the modified version which only you have.

The tool used to create a patch is called `diff` and the tool used to apply the patch is called `patch`. These tools are free software and standard in most UNIX-style operating systems.

7.1.1 Creating a patch

Example:

Let's say you have started from the DOLFIN release 0.3.10. You have a directory structure under `dolfin-0.3.10` where you have made some modifications to some files which you think could be useful to other DOLFIN users.

1. Clean up your modified directory structure to remove temporary and binary files which will be rebuilt anyway:

   ```
   $ make clean
   ```

2. Rename the directory to something else:

   ```
   $ mv dolfin-0.3.10 dolfin-0.3.10mod
   ```

3. Unpack the version of DOLFIN which you started from:

   ```
   $ tar xzvf dolfin-0.3.10.tgz
   ```

4. You should now have two DOLFIN directory structures in your current directory:

   ```
   $ ls
   dolfin-0.3.10
   dolfin-0.3.10mod
   ```

5. Use the `diff` tool to create the patch:

   ```
   $ diff -u --new-file --recursive dolfin-0.3.10 dolfin-0.3.10mod >
   mypatch.patch
   ```

   "`-u`" means "unified", a format in which to describe the patch.

   "`--new-file`" accepts the creation of new files in addition to modifications.

   "`--recursive`" means to recursively compare files through the directory structure.

6. The patch now exists as `mypatch.patch` and can be distributed to other people who have dolfin-0.3.10 to easily create your modified version. If the patch is large, compressing it with for example `gzip` is advisable.

A patch can also be created relative to a CVS version of DOLFIN. Make sure to notify receivers of the patch exactly which CVS version it is however (date and time of checkout should suffice), since the patch should be applied to the same version used to build it. Patching is possible against a different source, but conflicts can arise, and even though there are heuristics which take care of most simple cases, the heuristics can fail, and there can be conflicts which are theoretically impossible to resolve.

7.1.2 Applying a patch

Example:

Let's say the patch has been built relative to DOLFIN release 0.3.10.

1. Unpack the version of DOLFIN which the patch is built relative to:

   ```
   $ tar xzvf dolfin-0.3.10.tgz
   ```

2. Check that you have the patch `mypatch.patch` and the DOLFIN directory structure in the current directory.

   ```
   $ ls
   dolfin-0.3.10
   mypatch.patch
   ```

3. Enter the DOLFIN directory structure:

   ```
   $ cd dolfin-0.3.10
   ```

4. Apply the patch:

   ```
   $ patch -p1 < ../mypatch.patch
   ```

   "`-p1`" strips the leading directory from the filename references in the patch, to match the fact that we are applying the patch from inside the directory.

5. The modified version now exists as `dolfin-0.3.10`