# Heimdali Documentation

***Release 0.0.1***

**Dominique Bérézia, David Froger, Isabelle Herlin**

February 10, 2015

Contents

Heimdali is a set of commmand line tools to perform Image processing tools, based on ITK and HDF5

CHAPTER 1

# User guide

## 1.1 Installation

The recommanded way to install **Heimdali** is using the Conda package manager.

### 1.1.1 *Conda* installation

Download the Python *2.7* version of *miniconda* corresponding to your plateform from http://conda.pydata.org/miniconda.html

Execute the downloaded file: this will install the Conda package manager.

Add the *dfroger binstar* channel to your config:

```
conda config --add channels https://conda.binstar.org/dfroger
```

### 1.1.2 Heimdali installation

Install the *heimdali* package in a environment called *heim* or whather name you want:

```
conda create -n heim heimdali
```

Activate the environment:

```
source activate heim
```

Your are ready to use **Heimdali**.

### 1.1.3 Test Heimdali installation

Download some input data to test:

```
git clone https://github.com/dfroger/heimdali-data
```

Execute the *par* command:

```
par heimdali-data/imtest_z5_y4_x3_c2.h5
```

### 1.1.4 Heimdali update

You can update **Heimdali** when a new version is released:

```
conda update heimdali
```

You may also want to keep the currently installed **Heimdali** version, and install a new version in another *Conda* environment:

```
conda create -n heim0.1 heimdali==0.1.0
```

You can now switch between the two version of **Heimdali**:

```
source activate heim
```

or

```
source activate heim0.1
```

## 1.2 Format options

These are options describing the image format. There are used each time an image is created.

| option | description |
|---|---|
| -z N | Number of planes |
| -y N | Number of lines |
| -x N | Number of pixel per line |
| -v N | Number of value per pixel |
| -r | Floating point values |

## 1.3 par

**par** prints format parameters of images.

### 1.3.1 Synopsis

```
par  [--wr <output.txt>] [--x0] [--y0] [--z0] [-o] [-x] [-y] [-z] [--]
     [--version] [-h] <INPUT> ...
```

### 1.3.2 Description

**par** print on *stdout* in the file *outout.txt*, the format parameters of iamges given as arguments.

The *–wr* options can be given the special file names *stdout* and *stderr*. If the file name *output.txt* starts with >>, result are written at the end of the file.

If one or more options *–x0 –y0 –z0 -x -y -z* is given, **par** print the corresponding parameters, in the *Format options*. This allow to use **par** in command subsitution, as for example:

```
create image-copy.h5 `par -x -y image.h5` -r
```

If no options are given, all format parameters are printed for all images on argument.

## 1.4 h5toinr

**h5toinr** convert an *HDF5* image into *INRimage* image.

### 1.4.1 Synopsis

```
h5toinr  [--] [--version] [-h] <inputFilename> <outputFilename>
```

## 1.5 inrtoh5

**inrtoh5** convert an *INRimage* image into *HDF5* image.

### 1.5.1 Synopsis

```
inrtoh5  [--] [--version] [-h] <inputFilename> <outputFilename>
```

## 1.6 Artithmetic operations

Arithmetic operations between two images element by element.

### 1.6.1 Synopsis

```
ad image0-in image1-in [image-out]
so image0-in image1-in [image-out]
mu image0-in image1-in [image-out]
di image0-in image1-in [image-out]
min image0-in image1-in [image-out]
max image0-in image1-in [image-out]
```

### 1.6.2 Description

All of these commands perform an operation between *image0-in* and *image1-in* and write result to *image-out*. If argument *image0-in* or *image1-in* is equal to -, the command reads on standard input. If argument *image-out* is absent, the command writes to standard output.

| command | description |
|---|---|
| ad | Add two images |
| so | Substract two images |
| mu | Multiply two images |
| div | Divide two images |
| min | Compute minimum of two images |
| max | Compute maximum of two images |

All operation are performed on pixels element by element.

### 1.6.3 See also

See also *Local arithmetic operations*.

## 1.7 Local arithmetic operations

### 1.7.1 Synopsis

```
bi -n value [image-in] [image-out]
sc -n coeff [image-in] [image-out]
sd -n coeff [image-in] [image-out]
lo value [image-in] [image-out]
exp [image-in] [image-out]
ra [image-in] [image-out]
sba -n threshold [image-in] [image-out]
sha -n threshold [image-in] [image-out]
mb -n threshold [image-in] [image-out]
mh -n threshold [image-in] [image-out]
mo [image-in] [image-out]
car [image-in] [image-out]
vb -n threshold value [image-in] [image-out]
vh -n threshold value [image-in] [image-out]
```

### 1.7.2 Description

All of these commands perform an operation on *image-in* (which can be of any type), and write result in *image-out*.

*image-in* and *image-out* must have the same dimension.

If argument *image-in* is absent or egal to -, the command reads on standard input.

If argument *image-out* is absent, the command writes on standard output.

Values passed with *-n* are *float*.

| command | description |
|---|---|
| bi | Add *value* to each pixel. |
| sc | Multiply each pixel by *coeff*. |
| sd | Divied each pixel by *coeff*. |
| lo | Compute logarithm of each pixel. |
| exp | Compute exponential of each pixel. |
| ra | Compute square root of each pixel. |
| sba | Every pixel inferior or egual to *threshold* is replaced by *threshold*. |
| sha | Every pixel greater or egual to *threshold* is replaced by *threshold*. |
| mb | Every pixel inferior or egual to *threshold* is replaced by 1 and others by 0. |
| mh | Every pixel greater or egual to *threshold* is replaced by 1 and others by 0. |
| mo | Compute each pixel modulo |
| car | Compute each pixel square |
| vb | Every pixel inferior or egual to *threshold* is replaced by *value*. |
| vh | Every pixel greater or egual to *threshold* is replaced by *value*. |

CHAPTER 2

# Heimdali API

CHAPTER 3

# Community

CHAPTER 4

# Developer guide

## 4.1 Build Heimdali in development mode

Create a *conda* enviromnent named *heimdali-dev* containing all dependencies:

```
conda config --add channels http://conda.binstar.org/dfroger
conda create -n heimdali-dev h5unixpipe itk tclap cmake pip
```

Install lettuce:

```
source activate heimdali-dev
hash -r
pip install lettuce
```

Get Heimdali data files, and set *HEIMDALI_DATA_DIR*:

```
git clone https://github.com/dfroger/heimdali-data
export HEIMDALI_DATA_DIR=/path/to/heimdali-data
```

Build heidmali:

| variable | meaning |
|---|---|
| *CONDA_ENV_PATH* | For example, *~/miniconda/envs/heimdali* |
| *CMAKE_PREFIX_PATH* | Where *CMake* will search for dependent libraries |
| .. | Path to Heimdali main CMakeLists.txt |

```
mkdir build
cd build
source activate heimdali-dev
CONDA_ENV_PATH=$(conda info -e | grep '*' | tr -s ' ' | cut -d" " -f3)
cmake -DCMAKE_PREFIX_PATH=$CONDA_ENV_PATH ..
make
```

Add path to the built executables:

```
cd build
export PATH=$PWD/cmd:$PATH
```

Execute the functional tests:

```
cd tests
lettuce
```

CHAPTER 5

# Indices and tables

- *genindex*
- *modindex*
- *search*