

# MASS C++ Developer's Guide

Updated December 11, 2015

## 1 SETUP

---

### 1.1 LOCATION AND PROJECT SETUP

MASS C++ can be found on the Dslab computer and the Linux Lab computers. Currently, the RedHat Linux machine Hercules is being used for development.

The master copy of MASS C++ can be found in the folder `~/MASS/c++`, as shown in figure 1.

```
[dslab@uw1-320-06:~$ cd MASS/c++/
[dslab@uw1-320-06:~/MASS/c++$ ls
appls          MASS_Cpp_Developer_Guide.pdf  source          work_source
docs          MassCpp.pdf                   ubuntu          work_ubuntu
libssh2.tar   redhat                        work_redhat
dslab@uw1-320-06:~/MASS/c++$ █
```

Figure 1 Location of MASS C++

The codebase is broken up into several folders. The `work_` folders are currently unused and present only for historical reasons. Development occurs with git branches based on the source directory. The `redhat` and `ubuntu` folders contain Makefiles specific to those systems. The `source` folder contains the release version of the MASS C++ library. Within the `appls` directory there are several sample MASS applications that can be used to test the MASS C++ library, those applications are: SugarScape, Wave2D, Heat2D, and Conway's Game of Life.

### 1.2 GETTING A COPY OF THE CODE FOR DEVELOPMENT

To get a local copy of MASS C++, git clone the repository on your machine from:

[https://bitbucket.org/mass\\_library\\_developers/mass\\_cpp\\_core](https://bitbucket.org/mass_library_developers/mass_cpp_core) (the repository is private for now)

or on the Linux lab machines copy the contents of:

```
/net/metis/home/dslab/MASS/c++/
```

## 2 MAKING CHANGES

---

### 2.1 MAKING THE CHANGE

If you are unfamiliar with Git, checkout the Git training files for more indepth discussion on how to use in in development. The training files can be found in the `~/Training/GitTraining` folder.

MASS C++ is using the Git Flow model for updates. To add a feature, branch off of the `develop` branch. You can then push your branch to origin to test it in Jenkins.

Here is the basic workflow in the command line, which should be very close to any GUI Git application's process.

```
git checkout -b <MY_BRANCH_NAME>

git add <CHANGED_FILES>

git commit

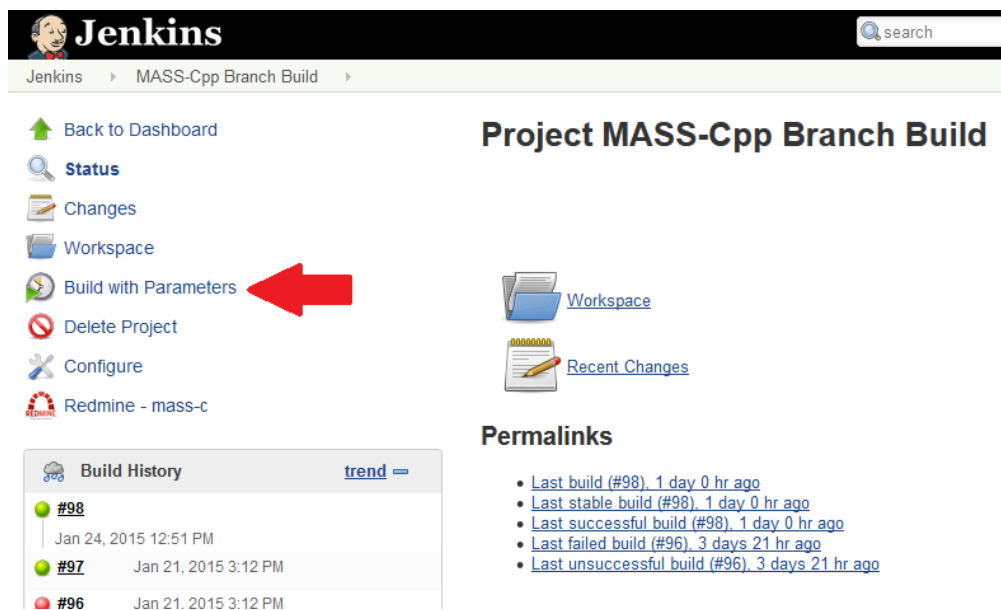
git push origin <MY_BRANCH_NAME>
```

## 2.2 BUILDING THE CHANGE

There are two ways to build your project.

### 2.2.1 Method 1: Jenkins

The easiest is to go to [Jenkins MASS-Cpp Branch Build](#). Select Build with Parameters. Then enter the name of your branch.



The screenshot shows the Jenkins web interface for the 'Project MASS-Cpp Branch Build'. The left sidebar contains navigation options: 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build with Parameters' (highlighted with a red arrow), 'Delete Project', 'Configure', and 'Redmine - mass-c'. The main content area displays 'Project MASS-Cpp Branch Build' with 'Workspace' and 'Recent Changes' links. Below this is a 'Permalinks' section with a list of links: 'Last build (#98), 1 day 0 hr ago', 'Last stable build (#98), 1 day 0 hr ago', 'Last successful build (#98), 1 day 0 hr ago', 'Last failed build (#96), 3 days 21 hr ago', and 'Last unsuccessful build (#96), 3 days 21 hr ago'. At the bottom left, a 'Build History' table shows three builds: #98 (green, Jan 24, 2015 12:51 PM), #97 (green, Jan 21, 2015 3:12 PM), and #96 (red, Jan 21, 2015 3:12 PM).

Figure 2 Building with Parameters

If the build is green, then it compiled successfully. If not, compilation has failed.

### 2.2.2 Method 2: Linux Lab

To build a particular branch, you will need to clone the repository into a folder so you can checkout the correct branch you need to build.

For the Linux Lab machines, you can do the following:

```
ssh dslab@uw1-320-lab.uwb.edu
```

```
mkdir <MY_NAME>

cd <MY_NAME>

cp -a /net/metis/home/dslab/MASS/c++/. /net/metis/home/dslab/<MY_NAME>
```

or clone your branch into your new directory.

You will now be able to checkout your branch. To build it, enter the ubuntu directory and type

```
make
```

This will build the MASS library.

**Warning:** This only builds your changes. Runtime errors will not be detected. Do not assume that you have not broken anything until you run your branch on one of the Linux machines!

**Note:** When building MASS C++, use the redhat makefile if building on Hercules, and use the Ubuntu makefile if building on the Linux lab machines. The Ubuntu machines are preferred for testing development.

## 2.3 EDITING THE MAKEFILE

To compile your MASS C++ library inside of your /source you need to edit the makefile inside of the Ubuntu directory or Redhat directory. The change you will need to make is on the SOURCE variable, the source needs to point to the directory that contains your MASS library.

If you keep the structure of the /net/metis/home/dslab/MASS/c++/ then your SOURCE should point to:

```
SOURCE=./source
```

If you want to compile and use the release version of MASS C++ library point your source to

```
SOURCE=/net/metis/home/dslab/MASS/c++/source
```

**Note:** When using the release MASS library you will need to point your applications to this library in your compile.sh file.

## 2.4 TESTING THE CHANGE

In addition to cloning the repository inside your MASS application directory, you should have each of the following files: symbolic link to mprocess, symbolic link to killMProcess.sh, machinefile.txt, compile.sh, and run.sh files; along with your MASS C++ application files.

### 2.4.1 Symbolic Links

First navigate to your MASS application directory and then use the following terminal command to create a symbolic links.

```
ln -s ~dslab/MASS/c++/Ubuntu/mprocess mprocess  
ln -s ~dslab/MASS/c++/Ubuntu/killMProcess.sh killMProcess.sh
```

**Note:** If your MASS application is hanging when you run it, please check that you have a good symbolic link to mprocess.

### 2.4.2 Mchinefile.txt

The machinefile.txt will tell your MASS application which other Linux lab machines will be child nodes and should be a plain text file that has the following format:

```
uw1-320-01  
uw1-320-02  
uw1-320-03  
uw1-320-04
```

If your machinefile.txt looks like the example given, then this means your parent node cannot be any of those machines; meaning you cannot start your MASS application from those machines. Also this means you will use a total of 5 machines while running your MASS application (this is passed in as argument 4 in your run.sh).

### 2.4.3 Compile.sh

The compile.sh script is used to compile all of your MASS application files.

Compile your main program as well as all your Agents/Places-derived classes. To compile your program that includes main( ), say main.cpp, type:

```
g++ -Wall main.cpp -I$MASS_DIR/source -L$MASS_DIR/Ubuntu -Imass -  
I$MASS_DIR/Ubuntu/ssh2/include -L$MASS_DIR/Ubuntu/ssh2/lib -Issh2 -o  
main
```

To compile your Agents/Places-derived class, say Land.cpp, type:

```
g++ -Wall Land.cpp -I$MASS_DIR/source -shared -fPIC -o Land
```

Note that you must compile all your Agents/Places-derived classes whose executable is dynamic-linked to mprocess whenever your main program invokes new Places( ) or new Agents( ).

Inside your compile.sh file you will also need to set up the following two shell variables:

```
export MASS_DIR=/net/metis/home/dslab/MASS/c++
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/net/metis/home/dslab/MASS/c++/Ubuntu
/ssh2/lib:/net/metis/home/dslab/MASS/c++/ubuntu
```

To use your MASS c++ library point MASS\_DIR to the directory that contains your MASS library directory.

#### 2.4.4 Run.sh

Your run script will run your executable file and must pass in the four arguments that MASS::init() needs, along with any specific arguments for your MASS application.

Here are the arguments for MASS::init():

```
arguments[0] //username
arguments[1] //password
arguments[2] //machinefile name
arguments[3] //port number
```

#### 2.4.5 MASS Sample Application

**To run MASS with the sample program, you will need to have setup a clone of the repository as shown in Method 2.**

In the ubuntu/samples folder, type

```
sh compile.sh
sh run.sh
```

This will run MASS with a test program so you can catch obvious runtime errors.

## 3 RELEASING A NEW VERSION

---

### 3.1 TAGGING IN GIT FOR RELEASE

### 3.2 COPY TO THE RELEASE DIRECTORY