

---

# **CHAI Documentation**

*Release 2.1.1*

**CHAI Developers**

**May 14, 2020**



<b>1</b>	<b>Getting Started</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	Basic Usage . . . . .	4
<b>2</b>	<b>Tutorial</b>	<b>5</b>
<b>3</b>	<b>Configuring CHAI</b>	<b>7</b>
<b>4</b>	<b>Code Documentation</b>	<b>9</b>
<b>5</b>	<b>Contribution Guide</b>	<b>11</b>
5.1	Forking CHAI . . . . .	11
5.2	Developing a New Feature . . . . .	11
5.3	Developing a Bug Fix . . . . .	12
5.4	Creating a Pull Request . . . . .	12
5.5	Tests . . . . .	12
<b>6</b>	<b>Developer Guide</b>	<b>13</b>



CHAI is a C++ library providing an array object that can be used transparently in multiple memory spaces. Data is automatically migrated based on copy-construction, allowing for correct data access regardless of location. CHAI can be used standalone, but is best when paired with the RAJA library, which has built-in CHAI integration that takes care of everything.

- If you want to get and install CHAI, take a look at our getting started guide.
- If you are looking for documentation about a particular CHAI function, see the code documentation.
- Want to contribute? Take a look at our developer and contribution guides.

Any questions? Contact [chai-dev@llnl.gov](mailto:chai-dev@llnl.gov)



This page provides information on how to quickly get up and running with CHAI.

## 1.1 Installation

CHAI is hosted on GitHub [here](#). To clone the repo into your local working space, type:

```
$ git clone --recursive git@github.com:LLNL/CHAI.git
```

The `--recursive` argument is required to ensure that the *BLT* submodule is also checked out. *BLT* is the build system we use for CHAI.

### 1.1.1 Building CHAI

CHAI uses CMake and BLT to handle builds. Make sure that you have a modern compiler loaded and the configuration is as simple as:

```
$ mkdir build && cd build  
$ cmake -DCUDA_TOOLKIT_ROOT_DIR=/path/to/cuda ../
```

By default, CHAI will attempt to build with CUDA. CMake will provide output about which compiler is being used, and what version of CUDA was detected. Once CMake has completed, CHAI can be built with Make:

```
$ make
```

For more advanced configuration, see *Configuring CHAI*.

## 1.2 Basic Usage

Let's take a quick tour through CHAI's most important features. A complete listing you can compile is included at the bottom of the page. First, let's create a new `ManagedArray` object. This is the interface through which you will want to access data:

```
chai::ManagedArray<double> a(100);
```

This creates a `ManagedArray` storing elements of type `double`, with 100 elements allocated in the CPU memory.

Next, let's assign some data to this array. We'll use CHAI's `forall` helper function for this, since it interacts with the `ArrayManager` for us to ensure the data is in the appropriate `ExecutionSpace`:

```
forall(sequential(), 0, 100, [=] (int i) {  
    a[i] = 3.14 * i;  
});
```

CHAI's `ArrayManager` can copy this array to another `ExecutionSpace` transparently. Let's use the GPU to double the contents of this array:

```
forall(cuda(), 0, 100, [=] __device__ (int i) {  
    a[i] = 2.0 * a[i];  
});
```

We can access the array again on the CPU, and the `ArrayManager` will handle copying the modified data back:

```
forall(sequential(), 0, 100, [=] (int i) {  
    std::cout << "a[" << i << "] = " << a[i] << std::endl;  
});
```



## CHAPTER 2

---

### Tutorial

---

The file `src/examples/example.cpp` contains a brief program that shows how CHAI can be used. Let's walk through this example, line-by-line:



---

## Configuring CHAI

---

In addition to the normal options provided by CMake, CHAI uses some additional configuration arguments to control optional features and behavior. Each argument is a boolean option, and can be turned on or off:

`-DENABLE_CUDA=Off`

Here is a summary of the configuration options, their default value, and meaning:

Variable	De- fault	Meaning
<code>ENABLE_CUDA</code>	Off	Enable CUDA support.
<code>ENABLE_HIP</code>	Off	Enable HIP support.
<code>EN- ABLE_GPU_SIMULATION_MODE</code>	Off	Simulates GPU execution.
<code>ENABLE_UM</code>	Off	Enable support for CUDA Unified Memory.
<code>ENABLE_CNEM</code>	Off	Enable cnmem for GPU allocations
<code>EN- ABLE_IMPLICIT_CONVERSIONS</code>	On	Enable implicit conversions between ManagedArray and raw pointers
<code>DISABLE_RM</code>	Off	Disable the ArrayManager and make ManagedArray a thin wrapper around a pointer.
<code>ENABLE_TESTING</code>	On	Build test executables.
<code>ENABLE_BENCHMARKS</code>	On	Build benchmark programs.

These arguments are explained in more detail below:

- `ENABLE_CUDA` This option enables support for GPUs using CUDA. If CHAI is built without CUDA, HIP, or `GPU_SIMULATION_MODE` support, then only the `CPU` execution space is available for use.
- `ENABLE_HIP` This option enables support for GPUs using HIP. If CHAI is built without CUDA, HIP, or `GPU_SIMULATION_MODE` support, then only the `CPU` execution space is available for use.
- `ENABLE_GPU_SIMULATION_MODE` This option simulates GPU support by enabling the GPU execution space, backed by a `HOST` umpire allocator. If CHAI is built without CUDA, HIP, or `GPU_SIMULATION_MODE` support, then only the `CPU` execution space is available for use.

- `ENABLE_UM` This option enables support for Unified Memory as an optional execution space. When a `ManagedArray` is allocated in the `UM` space, CHAI will not manually copy data. Data movement in this case is handled by the CUDA driver and runtime.
- `ENABLE_CNEM` This option enables the use of the `cnmem` library for GPU allocations. The `cnmem` library provides a pool mechanism to reduce the overhead of allocating memory on the GPU.

When `ENABLE_CNMEM` is set to `On`, you must tell CMake where to find the `cnmem` library. This can be done by setting the `cnem_DIR` variable, for example:

- `ENABLE_IMPLICIT_CONVERSIONS` This option will allow implicit casting between an object of type `ManagedArray<T>` and the corresponding raw pointer type `T*`. This option is disabled by default, and should be used with caution.
- `DISABLE_RM` This option will remove all usage of the `ArrayManager` class and let the `ManagedArray` objects function as thin wrappers around a raw pointer. This option can be used with CPU-only allocations, or with CUDA Unified Memory.
- `ENABLE_TESTING` This option controls whether or not test executables will be built.
- `ENABLE_BENCHMARKS` This option will build the benchmark programs used to test `ManagedArray` performance.

## CHAPTER 4

---

### Code Documentation

---



This document is intended for developers who want to add new features or bugfixes to CHAI. It assumes you have some familiarity with git and GitHub. It will discuss what a good pull request (PR) looks like, and the tests that your PR must pass before it can be merged into CHAI.

## 5.1 Forking CHAI

If you aren't a CHAI developer at LLNL, then you won't have permission to push new branches to the repository. First, you should create a [fork](#). This will create a copy of the CHAI repository that you own, and will ensure you can push your changes up to GitHub and create pull requests.

## 5.2 Developing a New Feature

New features should be based on the `develop` branch. When you want to create a new feature, first ensure you have an up-to-date copy of the `develop` branch:

```
$ git checkout develop
$ git pull origin develop
```

You can now create a new branch to develop your feature on:

```
$ git checkout -b feature/<name-of-feature>
```

Proceed to develop your feature on this branch, and add tests that will exercise your new code. If you are creating new methods or classes, please add Doxygen documentation.

Once your feature is complete and your tests are passing, you can push your branch to GitHub and create a PR.

## 5.3 Developing a Bug Fix

First, check if the change you want to make has been fixed in `develop`. If so, we suggest you either start using the `develop` branch, or temporarily apply the fix to whichever version of CHAI you are using.

Assuming there is an unsolved bug, first make sure you have an up-to-date copy of the `develop` branch:

```
$ git checkout develop
$ git pull origin develop
```

Then create a new branch for your bugfix:

```
$ git checkout -b bugfix/<name-of-bug>
```

First, add a test that reproduces the bug you have found. Then develop your bugfix as normal, and ensure to make `test` to check your changes actually fix the bug.

Once you are finished, you can push your branch to GitHub, then create a PR.

## 5.4 Creating a Pull Request

You can create a new PR [here](#). GitHub has a good [guide](#) to PR basics if you want some more information. Ensure that your PR base is the `develop` branch of CHAI.

Add a descriptive title explaining the bug you fixed or the feature you have added, and put a longer description of the changes you have made in the comment box.

Once your PR has been created, it will be run through our automated tests and also be reviewed by RAJA team members. Providing the branch passes both the tests and reviews, it will be merged into RAJA.

## 5.5 Tests

CHAI's tests are all in the `test` directory, and your PR must pass all these tests before it is merged. If you are adding a new feature, add new tests.



## CHAPTER 6

---

### Developer Guide

---

Coming soon!