

# ArtiSynth Installation Guide for MacOS

---

**John Lloyd and Sebastian Kazenbroot-Guppy**

August 20, 2014

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Prerequisites</b>	<b>4</b>
<b>3</b>	<b>Downloading a Prepacked Release</b>	<b>4</b>
3.1	Downloading and unpacking the zip file . . . . .	4
<b>4</b>	<b>Checking out from Subversion</b>	<b>5</b>
4.1	Checking out using Eclipse . . . . .	5
4.2	Checking out using the command line . . . . .	5
4.3	Downloading the libraries . . . . .	5
<b>5</b>	<b>Building ArtiSynth</b>	<b>5</b>
5.1	Building with Eclipse . . . . .	5
5.2	Building from the command line . . . . .	6
<b>6</b>	<b>Running ArtiSynth</b>	<b>6</b>
6.1	Running from the command line . . . . .	6
6.2	Running from the file browser . . . . .	6
6.3	Running using Eclipse . . . . .	6
6.4	Testing the Demo Models . . . . .	6
<b>7</b>	<b>Installing External Models and Packages</b>	<b>7</b>
7.1	Downloading . . . . .	7
7.2	Building . . . . .	7
7.3	Running . . . . .	7
7.3.1	Adding external classes using the Eclipse Classpath . . . . .	7
7.3.2	Adding external classes using EXTCLASSPATH . . . . .	7
7.3.3	Adding external classes using CLASSPATH . . . . .	8
<b>8</b>	<b>Updating ArtiSynth</b>	<b>8</b>
8.1	Library updates . . . . .	8
<b>9</b>	<b>The Eclipse IDE</b>	<b>8</b>
9.1	Obtaining Eclipse . . . . .	8
9.2	Installing a Subversion plug-in . . . . .	8
9.3	Importing an ArtiSynth project into Eclipse . . . . .	9
9.4	Importing an ArtiSynth project directly from Subversion . . . . .	9
9.5	Configuring environment variables . . . . .	11
9.5.1	Setting environment variables . . . . .	11
9.6	Adding projects to the build path . . . . .	11
9.7	Adding projects to the ArtiSynth launch configuration . . . . .	12
9.8	Preventing excessive resource copying . . . . .	12

---

<b>10 Additional Information</b>	<b>12</b>
10.1 Environment variables . . . . .	12
10.1.1 Example environment set up for <code>bash</code> . . . . .	13
10.1.2 Example environment setup for <code>csh</code> or <code>tcsh</code> . . . . .	13
10.2 ArtiSynth Libraries . . . . .	14
10.3 The <code>EXTCLASSPATH</code> File . . . . .	14

## 1 Introduction

This document describes how to install and run ArtiSynth on MacOS machines. There are two ways to obtain ArtiSynth: downloading a prepackaged release, or checking out the latest development version via Subversion. Downloading a prepackaged release is the easiest solution to simply try out some of the basic demo programs. Checking out the development version is recommended for developers who want to keep their codebase current.

The typical install sequence looks like this:

**Download** Download either a release (Section 3) or check out the development version (Section 4).

**Build** Compile the system (Section 5). This step is not needed for prepackaged releases.

**Run** Start ArtiSynth and run the demonstration models (Section 6).

Generally, users will also want to install and run external models and packages that have been created either by others or by themselves. This is discussed in (Section 7).

## 2 Prerequisites

To install ArtiSynth on MacOS , you will need:

- A 64 bit version of MacOS
- Java JDK 1.7
- Linux systems require GNU libc version 2.17 or higher
- A three button mouse is recommended for GUI interaction

Note that we have stopped supporting 32 bit systems, both because they are becoming obsolete, and because ArtiSynth applications often require more memory than they can provide.

For Java, the full Java development kit (JDK) is required, which comes with the Java compiler `javac`. The run time environment (JRE) will not be sufficient. However, there is no need for extra bundles such as JavaFX, NetBeans, or EE.

At the time of this writing, JDKs can be obtained free from Oracle at <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. We currently recommend either JDK 7 or 6. Note that JDKs are sometimes referred to by multiple names; for example, JDK 6 is sometimes referred to as JDK 1.6.

In this document, the location of the ArtiSynth installation directory will be denoted by `$ARTISYNTH_HOME`. That means that if ArtiSynth is installed in

```
/home/roger/artisynth_core
```

then `$ARTISYNTH_HOME/lib` denotes the directory

```
/home/roger/artisynth_core/lib
```

## 3 Downloading a Prepacked Release

### 3.1 Downloading and unpacking the zip file

To obtain one of the packaged distributions, go to [www.artisynth.org/downloads](http://www.artisynth.org/downloads) and select the distribution you want. Download it, and unzip it in an appropriate location on your computer.

Once ArtiSynth is downloaded and unpacked, it should be possible to run it immediately by executing the `artisynth` command located in `$ARTISYNTH_HOME/bin` (see Section 6.1).

---

## 4 Checking out from Subversion

The latest development version is available from Subversion. Once checked out, users can continue to update the codebase to keep it current (Section 8). Developers that we work with closely can also obtain, by mutual arrangement, write access to our Subversion repository, allowing them to also commit changes.

The ArtiSynth Subversion URL is

```
https://svn.artisynth.org/svn/artisynth_core/trunk
```

**Note:**

If you omit the trailing `/trunk` from the Subversion URL, then the checkout will contain the entire Subversion directory structure, including the subdirectories `trunk`, `branches`, and `tags`, which is generally not needed by most users. Also, if you do check out the entire structure, the resulting `trunk` directory should be specified as `ARTISYNTH_HOME`.

There are several ways to check out ArtiSynth using its Subversion URL:

### 4.1 Checking out using Eclipse

If you are planning to develop ArtiSynth models in Java, and if you are planning to do this with Eclipse (Section 9), then it might be easiest to do the Subversion checkout directly in Eclipse. Follow the instructions in Section 9.4, using the ArtiSynth Subversion URL (above) as the *Subversion\_url*.

### 4.2 Checking out using the command line

If your MacOS distribution has Subversion installed, then you can check out ArtiSynth using the following command:

```
> svn co https://svn.artisynth.org/svn/artisynth_core/trunk artisynth_core
```

This will check out ArtiSynth and place it in the specified directory (in this case `artisynth_core`).

### 4.3 Downloading the libraries

Because the `jar` files and native libraries used by ArtiSynth are large, they are not stored in the Subversion repository. Instead, they must be downloaded separately. This can be done using the command `updateArtisynthLibs`, located in `$ARTISYNTH_HOME/bin`. You can execute it from the command line like this:

```
> cd $ARTISYNTH_HOME
> bin/updateArtisynthLibs
```

## 5 Building ArtiSynth

If ArtiSynth has been obtained directly from Subversion, it will be necessary to build (compile) it. This is not necessary for prepackaged releases, which are pre-built, although one may want to rebuild a release anyway for development purposes.

### 5.1 Building with Eclipse

If your Subversion checkout has been done externally to Eclipse (i.e., not according to Section 4.1), then you need to first import ArtiSynth into Eclipse. Follow the instructions in Section 9.3.

Once ArtiSynth has been imported, you should be able to build it. If necessary, first open a Java perspective by choosing Window > Open Perspective > Java. The project `artisynth_core` (or whatever you might have named it) should appear in the Package Explorer window. To build the system, select the project in the Package Explorer window, and then choose Project > Build Project. Note that it may be necessary to deselect Build Automatically in order to enable Build Project.



Figure 1: The ArtiSynth play controls. From left to right: step size control, current simulation time, and the reset, play/pause, and single-step buttons.

## 5.2 Building from the command line

ArtiSynth can also be built by running a `make` command in the top level directory. Before doing this, you need to first set the environment variables `ARTISYNTH_HOME` and `CLASSPATH` as described in Section 10.1. ArtiSynth can then be built by executing

```
> cd $ARTISYNTH_HOME
> make
```

# 6 Running ArtiSynth

## 6.1 Running from the command line

The most direct way to start ArtiSynth is to run the command `$ARTISYNTH_HOME/bin/artisynth`:

```
> cd $ARTISYNTH_HOME
> bin/artisynth
```

## 6.2 Running from the file browser

You can also run ArtiSynth from a file browser by double clicking on

`$ARTISYNTH_HOME/bin/artisynth.command`

Note that `artisynth.command` is just a copy of `artisynth`; the `.command` suffix makes it recognizable to the MacOS GUI as a command.

## 6.3 Running using Eclipse

Once ArtiSynth has been imported into Eclipse (and built if necessary), it should contain a launch configuration called `ArtiSynth` that will allow ArtiSynth to be run.

Generally, it is first necessary to set environment variables in the launch configuration to allow ArtiSynth to find runtime configuration files and libraries. Instructions for setting these variables are contained in Section 9.5. If the same environment variables have already been set externally in MacOS (Section 10.1), then they do not need to be set in the launch configuration.

Once the environment variables are set, it should be possible to run ArtiSynth by choosing `Run > Run`.

## 6.4 Testing the Demo Models

Once ArtiSynth starts up, you can try out the various demonstration models. From the menu bar, `Select Models > Demos`. This will display a submenu of demonstration models. Choosing one will cause that model to be loaded and displayed in the viewer. Simulation of the model can then be started, paused, single-stepped, or reset using the play controls (Figure 1) located at the upper right of the ArtiSynth window frame.

Comprehensive information on exploring and interacting with models is given in the [ArtiSynth User Interface Guide](#).

## 7 Installing External Models and Packages

Typically, an ArtiSynth developer will want to use external models and packages that exist outside of `artisynth_core`. Some of these may be obtained from external sources. For example, `artisynth_models` is a collection of packages, currently hosted at [www.artisynth.org/models](http://www.artisynth.org/models), that provides a variety of publicly available anatomical models.

Installing external models and packages requires a sequence of operations similar to that for installing ArtiSynth itself:

1. Download
2. Build (if necessary)
3. Run

### 7.1 Downloading

Some model and package collections, such as `artisynth_models` mentioned above, may be available either as prepackaged distributions or as Subversion checkouts. Prepackaged distributions should be downloaded and unpacked into a desired location, while Subversion checkouts may be obtained as described in Section 4, using the appropriate Subversion URL in place of the ArtiSynth Subversion URL. Some collections require authorization for checkout. In those cases, it will be necessary to also provide Subversion with an account name and password (which you will have obtained from us by prior arrangement).

Some collections maintained by ArtiSynth may contain Eclipse project settings (in an `eclipseSettings.zip` file in their root directory), allowing them to be imported into Eclipse, either directly from Subversion (Section 9.4), or after being obtained separately (Section 9.3).

### 7.2 Building

Collections that are obtained from Subversion will need to be built (compiled).

Those imported into Eclipse can be built as described in Section 5.1. However, in order to compile properly, the `artisynth_core` project (and any other projects they depend on) were have to be added to their build path. Details on doing this are given in Section 9.6.

Alternatively, if the collection has a `Makefile` in its root directory, then it can be compiled from the command line by running `make` in the root directory. Before doing this, the top-level directory for the collection's `class` files must to be added to the `CLASSPATH` environment variable (Section 10.1). In collections maintained by ArtiSynth, this will be the directory `classes`, located directly under the collection root directory (e.g., `artisynth_models/classes`).

### 7.3 Running

External models are executed by running ArtiSynth itself (Section 6). However, in order to execute these models, ArtiSynth must be able to locate their associated classes. This can be arranged in three different ways:

#### 7.3.1 Adding external classes using the Eclipse Classpath

If you are running from Eclipse, then you can make the classes of external projects visible to ArtiSynth by adding the projects to the Classpath of your ArtiSynth launch configuration, as described in Section 9.7.

#### 7.3.2 Adding external classes using EXTCLASSPATH

Alternatively, you can make the classes of external projects visible to ArtiSynth by adding the path names of all their top-level class directories (or `jar` files, if relevant) to the file `$ARTISYNTH_HOME/EXTCLASSPATH` (described in Section 10.3).

For example, suppose the collection `artisynth_models` has been placed in `/projects/artisynth_models`. The top-level class directory for this collection is located in `artisynth_models/classes`, and so the following entry should be placed in the `EXTCLASSPATH` file:

```
/projects/artisynth_models/classes
```

---

### 7.3.3 Adding external classes using CLASSPATH

Finally, if you are running from the command line using the `artisynth` command, then you can make external classes visible by adding them to your `CLASSPATH` environment variable (see Section 10.1).

## 8 Updating ArtiSynth

One reason to use a checkout of the latest ArtiSynth development version is to be able to migrate recent changes into your code base. When a significant update occurs, a posting is made to the ArtiSynth update log, currently located at [www.artisynth.org/doc/html/updates/updates.html](http://www.artisynth.org/doc/html/updates/updates.html). Users may also be notified via the `artisynth-updates` email list.

Users working from Eclipse with a Subversion plug-in installed (Section 9.4) may update simply by selecting the project in the Package Explorer and selecting Team > Update from the context menu.

Updating may also be done from the command line using the `svn` command in the ArtiSynth installation directory :

```
cd $ARTISYNTH_HOME
svn update
```

### 8.1 Library updates

Occasionally, a software update will be accompanied by a change in the libraries located in `$ARTISYNTH_HOME/libs`. When this happens, it will be indicated on the ArtiSynth update log and appropriate instructions will be given. Sometimes, it will be necessary to explicitly update the libraries after doing the main update. This can be done by executing `updateArtisynthLibs` as described in Section 4.3.

## 9 The Eclipse IDE

Eclipse is an integrated development environment (IDE) commonly used for Java code development, and many ArtiSynth developers use it for both programming models and for running the system. This section describes how to load ArtiSynth projects into Eclipse, and how to configure it for running ArtiSynth. A general introduction to Eclipse is beyond the scope of this document, but there are many Eclipse resources available online.

### 9.1 Obtaining Eclipse

Eclipse can be obtained from [www.eclipse.org/downloads](http://www.eclipse.org/downloads). A good version to obtain (at the time of this writing) is Eclipse IDE for Java Developers.

### 9.2 Installing a Subversion plug-in

In order to work with Subversion from within Eclipse, either to check out ArtiSynth from the repository, or to update or commit changes, it is necessary to use a Subversion plug-in. First, check to see if your version of Eclipse contains an Subversion plug-in:

Open an import panel using File > Import..., and then look for SVN in the set of available import sources. If you don't see SVN listed, it will be necessary to install a plug-in.

We recommend the Eclipse-supported Subversive plug-in, but if this proves difficult for any reason, there are other options, such as Subclipse, currently obtainable from [subclipse.tigris.org](http://subclipse.tigris.org).

Instructions for installing Subversive can be obtained at [www.eclipse.org/subversive/installation-instructions.php](http://www.eclipse.org/subversive/installation-instructions.php).

One way to install Subversive is through the Eclipse Marketplace. If you have an older version of Eclipse that doesn't have Marketplace, you may be able to obtain it from [www.eclipse.org/mpc](http://www.eclipse.org/mpc). To access the Marketplace, click Help > Eclipse Marketplace. Once the available applications have been displayed, type `Subversive` into the Find box in the top-left corner of the Marketplace window. Navigate to the package labeled Subversive - SVN Team Provider and click



Install. On the Confirm Selected Features screen, ensure all boxes are checked and click the button labeled Confirm >. Restart Eclipse when prompted.

One more step is now necessary. Re-open Eclipse, and you should be prompted to choose an SVN connector in the start menu. SVN connectors interface Subversive to the SVN server, and are OS and server-specific. A recommended SVN Connector will be pre-selected for downloading; this is most likely the one you need.

If Eclipse did not prompt you to choose a connector when it restarted, you can install SVN connectors separately (thanks to bmaupin at Stackoverflow for this information):

1. Go to [www.polarion.com/products/svn/subversive/download.php](http://www.polarion.com/products/svn/subversive/download.php)
2. Under the latest Release, copy the Subversive SVN Connectors URL. The current URL for Eclipse 4.3 Kepler is <http://community.polarion.com/projects/subversive/download/eclipse/3.0/kepler-site>.
3. In Eclipse, go to Help > Install New Software... and click Add...
4. Copy the URL for the Subversive SVN Connectors into the Location box and click OK
5. Check Subversive SVN Connectors, click Next, and then follow the instructions to complete installation.

If in doubt about the connector you need, you can install multiple ones, and then adjust the one Subversive actually uses by going to Windows > Preferences, opening Team > SVN, and then opening the SVN Connector tab.

### 9.3 Importing an ArtiSynth project into Eclipse

An ArtiSynth project can be either ArtiSynth itself, or an associated project containing specific modeling applications. Let `$PROJECT_ROOT` denote the project root directory. For ArtiSynth itself, this will be `$ARTISYNTH_HOME`.

1. From **outside** Eclipse, install the Eclipse settings by unzipping `$PROJECT_ROOT/eclipseSettings.zip` into `$PROJECT_ROOT`. This will create the files `.project` and `.classpath`, along with the directory `.settings`, in `$PROJECT_ROOT`. For ArtiSynth itself, it will also create the file `ArtiSynth.launch` containing the launch configuration.

#### Attention MacOS users:

The default zip utility on MacOS will create a new sub-folder called `eclipseSettings` and will extract the files there. *You do not want this!!* Some of the files are then labelled as “hidden” by MacOS, which will prevent you from moving them to the correct place manually. Either extract the file directly to the `$ARTISYNTH_HOME` directory with a more standard application like 7-Zip (7zX for OSX), or use the `unzip` utility from the command-line. For the latter, open a terminal window, change to the ArtiSynth install directory, and enter the command

```
unzip eclipseSettings.zip
```

2. From within Eclipse, choose File > Import ....
3. In the Import window, select General > Existing Projects into Workspace and click Next.
4. In the field Select root directory, enter (or browse to) `$PROJECT_ROOT` and then click Finish.

If Eclipse complains that "No projects are found to import", that most likely means that `eclipseSettings.zip` was not properly unzipped into `$PROJECT_ROOT`.

### 9.4 Importing an ArtiSynth project directly from Subversion

If Eclipse has a Subversion plug-in installed (Section 9.2), you may import an ArtiSynth project by checking it out directly from the repository located by the project's `Subversion_URL`. For the core ArtiSynth distribution, this is

```
https://svn.artisynth.org/svn/artisynth_core/trunk
```

Other projects will have different URLs.

The following instructions assume the Subversive plug-in.

1. Choose File > Import from the main menu, select SVN > Project from SVN and click Next.
2. You now need to specify a repository location, as specified by a *Subversion\_URL*. If you've previously done an SVN checkout, a menu will appear allowing you to select a previously used URL. If one of these is sufficient, select it and click Next to go to Step 4. Otherwise, select Create a new repository location and click Next to enter a repository dialog. If no previous locations are known this dialog will appear automatically.
3. If you are specifying a new location in the repository dialog:
  - Under the General tab, enter the *Subversion\_URL* in the URL box. If you are just checking out the trunk of the repository (i.e., if your Subversion URL ends in /trunk), then you should omit the final /trunk since this is selectable in Step 4.
  - If you are checking out a repository that is not available for anonymous access, or if you need write access to the repository, enter your ArtiSynth User ID and Password (which you will have obtained from us separately) in the Authentication section of the dialog. You will probably want to check Save authentication as well.
  - Click Next.
4. In the Select Resource dialog, use the URL selector box to select the full URL to be used for the checkout. If you are just checking out the trunk of the repository, then choose Subversion\_URL/trunk which should be available as a selection.
5. Click Finish
6. In the Check Out As dialog, select Check out as a project with name specified, adjust the project name if desired, and click Next.
7. Specify the location for the check out. If you leave Use default workspace location selected, this will be workspace/project\_name, where workspace is the Eclipse workspace directory and project\_name is the project name selected in the previous step. Otherwise, you can specify an explicit checkout location (which does not have to be located in the Eclipse workspace). For ArtiSynth core checkouts, the project name is typically artisynth\_core and the checkout location will become the ArtiSynth install directory \$ARTISYNTH\_HOME.
8. Click Finish.
9. If necessary, open a Java perspective by choosing Window > Open Perspective > Java. The project should appear in the Package Explorer window.
10. From **outside** Eclipse, install the Eclipse settings by unzipping \$PROJECT\_ROOT/eclipseSettings.zip into \$PROJECT\_ROOT. This will overwrite the file .project, and create the file .classpath, along with the folder .settings, in \$PROJECT\_ROOT. For ArtiSynth itself, it will also create the file ArtiSynth.launch containing the launch configuration.

Note: if unzip queries about overwriting .project, answer [y]es.

#### Attention MacOS users:

The default zip utility on MacOS will create a new sub-folder called eclipseSettings and will extract the files there. *You do not want this!!* Some of the files are then labelled as “hidden” by MacOS, which will prevent you from moving them to the correct place manually. Either extract the file directly to the \$ARTISYNTH\_HOME directory with a more standard application like 7-Zip (7zX for OSX), or use the unzip utility from the command-line. For the latter, open a terminal window, change to the ArtiSynth install directory, enter the command

```
unzip eclipseSettings.zip
```

and respond with y when asked if it is OK to replace .project.

11. From **outside** Eclipse, download the required jar files and native libraries as described in Section 4.3.
12. Finally, load the new settings into the project by selecting the project in the Package Explore window and selecting Refresh from the context menu.

## 9.5 Configuring environment variables

To run ArtiSynth from Eclipse, it is generally necessary to set certain environment variables directly in your Eclipse launch configuration so that ArtiSynth can locate configuration files and native library support. Directions on setting the environment variables are given in Section 9.5.1. The required settings are:

- Set `ARTISYNTH_HOME` to the path of the ArtiSynth installation directory .
- Set `DYLD_LIBRARY_PATH` to `$ARTISYNTH_HOME/lib/MacOS64`
- Optionally, set `OMP_NUM_THREADS` (see Section 10.1).
- Optionally, set `ARTISYNTH_PATH` (see Section 10.1).

If any of the above variables have already been set externally in MacOS (Section 10.1), then they do not need to be set in the launch configuration.

**Note:** At present, eclipse does not expand environment variables. In all the variable settings described below, references to `$ARTISYNTH_HOME` should be expanded (manually) to the path of the ArtiSynth install directory .

### 9.5.1 Setting environment variables

To set environment variables within Eclipse:

1. Open a java perspective if necessary by choosing Window > Open Perspective > Java.
2. Select the ArtiSynth project in the Package Explorer form.
3. Choose Run > Run Configurations... to open the Run Configurations window.
4. In the left panel, under Java Application, select ArtiSynth.
5. In the right panel, select the environment tab.
6. To create a new environment variable, click the New button and enter the name and value in the dialog box.
7. When finished, make sure that Append environment to native environment is selected, and click Apply.

## 9.6 Adding projects to the build path

A project imported into Eclipse may depend on the packages and libraries found in other projects to compile properly. For example, ArtiSynth applications which are external to `artisynth_core` will nonetheless depend on `artisynth_core`. To ensure proper compilation, project dependencies should be added to each dependent project's build path.

1. Select the dependent project in the Package Explorer form.
2. Right click and choose Build Path > Configure Build Path...
3. In the right panel, select the Projects tab.
4. Click the Add button, select the project dependencies, and click OK
5. Click OK in the Java Build Path panel

## 9.7 Adding projects to the ArtiSynth launch configuration

The classes of external projects can be made visible to ArtiSynth by adding the projects to the Classpath of the ArtiSynth launch configuration.

1. From the main menu, choose Run > Run Configurations... to open a Run Configurations dialog.
2. In the left panel, under Java Application, select your ArtiSynth launch configuration (the default one is called ArtiSynth). This may already be selected when you open the panel.
3. In the right panel, select the Classpath tab.
4. In the Classpath: window, select User Entries, and then click the Add Projects button.
5. In the Project Selection dialog, select the external projects that you wish to add. Generally, the boxes Add exported entries ... and Add required projects ... can be unchecked. Click OK.
6. Close the Run Configurations dialog.

## 9.8 Preventing excessive resource copying

By default, ArtiSynth classes are built in a directory tree (\$PROJECT\_ROOT/classes) that is separate from the source tree (\$PROJECT\_ROOT/src), where \$PROJECT\_ROOT denotes the project root directory and is \$ARTISYNTH\_HOME for ArtiSynth itself. That means that Eclipse will try to copy all non-Java files and directories from the source tree into the build tree. For ArtiSynth, this is excessive, and results in many files being copied that don't need to be, since ArtiSynth looks for resources in the source tree anyway.

It is possible to inhibit most of this copying:

1. Choose Window > Preferences (or Eclipse > Preferences).
2. Select Java > Compiler > Building.
3. Open Output folder, and in the box entitled Filter resources, enter the string:

```
Makefile,*.l*,*.*,*.*?*,*.*??*,*.*???*,*.*????,???,????,?????
```

That should filter out the copying of most non-java files.

Or, to prevent copying any resource, simply enter:

```
*
```

## 10 Additional Information

### 10.1 Environment variables

This is a glossary of all the environment variables that are associated with building or running ArtiSynth. Often, the system can detect and set appropriate values for these automatically. In other cases, as noted in the above documentation, it may be necessary or desirable for the user to set them explicitly.

#### ARTISYNTH\_HOME

The path name of the ArtiSynth installation directory .

#### ARTISYNTH\_PATH

A list of directories , separated by colons ":", which ArtiSynth uses to search for configuration files such as .artisynthInit or .demoModels. A typical setting for ARTISYNTH\_PATH consists of the current directory (indicated by "."), the user's home directory , and the ArtiSynth installation directory . If ARTISYNTH\_PATH is not defined explicitly in the user's environment, ArtiSynth assumes an implicit path consisting of the directory sequence just described.

## CLASSPATH

A list of directories and/or jar files, separated by colons ":", which Java uses to locate its class files. This variable should be set to include `$ARTISYNTH_HOME/classes` and `$ARTISYNTH_HOME/lib/*` (the latter uses the wildcard `*` to specify all the jar files in `$ARTISYNTH_HOME/lib`).

## PATH

A list of directories, separated by colons ":", which the operating system uses to locate executable programs and applications. This should be set to include `$ARTISYNTH_HOME/bin`

## DYLD\_LIBRARY\_PATH

A list of directories, separated by colons ":", which the operating system searches in order to find shared libraries. Should be set to include `$ARTISYNTH_HOME/lib/MacOS64`.

## OMP\_NUM\_THREADS

Specifies the maximum number of processor cores that are available for multicore execution. Setting this variable to the maximum number of cores on your machine can significantly increase performance.

Note that settings for most of the above can be derived from the value of `ARTISYNTH_HOME`.

### 10.1.1 Example environment set up for bash

If you are using `bash` as your shell, then the environment can be configured by placing a block of commands similar to the following in one of your `bash` initialization files (typically `~/.bashrc`), located in your home directory:

```
# set ARTISYNTH_HOME to the appropriate location ...
setenv ARTISYNTH_HOME $HOME/artisynth_2_X
setenv ARTISYNTH_PATH .:"$HOME": "$ARTISYNTH_HOME"
setenv DYLD_LIBRARY_PATH $ARTISYNTH_HOME/lib/MacOS64:"$DYLD_LIBRARY_PATH"
setenv CLASSPATH "$ARTISYNTH_HOME/classes:$ARTISYNTH_HOME/lib/*:$CLASSPATH"
setenv PATH $ARTISYNTH_HOME/bin:"$PATH"
# Set to the number of cores on your machine:
setenv OMP_NUM_THREADS 2
```

Be sure to set `ARTISYNTH_HOME` to the proper location of your ArtiSynth installation directory.

These environment variables will be passed on to any program which you run from the shell (such as `artisynth` or `eclipse`). However, they will **not** be passed on to programs (such as `eclipse`) which you launch from the dock.

Alternatively, you can source the script `setup.bash`, located in the installation directory:

```
> source setup.bash
```

This will determine the system type automatically and set the environment variables accordingly, with `ARTISYNTH_HOME` set to the current directory from which the script is called (however, it *won't* set `OMP_NUM_THREADS`).

### 10.1.2 Example environment setup for csh or tcsh

If you are using `csh` or `tcsh` as your shell, then the environment can be configured by placing a block of commands similar to the following in your `.cshrc` file, located in your home directory:

```
# set ARTISYNTH_HOME to the appropriate location ...
setenv ARTISYNTH_HOME $HOME/artisynth_2_X
setenv ARTISYNTH_PATH .:"$HOME": "$ARTISYNTH_HOME"
setenv DYLD_LIBRARY_PATH $ARTISYNTH_HOME/lib/MacOS64:"$DYLD_LIBRARY_PATH"
setenv CLASSPATH "$ARTISYNTH_HOME/classes:$ARTISYNTH_HOME/lib/*:$CLASSPATH"
setenv PATH $ARTISYNTH_HOME/bin:"$PATH"
# Set to the number of cores on your machine:
setenv OMP_NUM_THREADS 2
```

These environment variables will be passed on to any program which you run from the shell (such as `artisynt` or `eclipse`). However, they will **not** be passed on to programs (such as `eclipse`) which you launch from the dock.

Alternatively, you can source the script `setup.csh`, located in the installation directory :

```
> source setup.csh
```

This will determine the system type automatically and set the environment variables accordingly, with `ARTISYNTH_HOME` set to the current directory from which the script is called (however, it *won't* set `OMP_NUM_THREADS`).

## 10.2 ArtiSynth Libraries

ArtiSynth uses a set of libraries located under `$ARTISYNTH_HOME/lib`. These include a number of `jar` files, plus native libraries located in architecture-specific sub-directories (`MacOS64` for MacOS systems).

As described in Section 4.3, these libraries need to be downloaded automatically if the system is obtained from the Subversion repository. The required libraries are listed in the file `$ARTISYNTH_HOME/lib/LIBRARIES`. This file is checked into the repository, so that the system can always determine what libraries are needed for a particular checkout version.

Occasionally the libraries are changed or upgraded. If you run ArtiSynth with the `-updateLibs` command line option, the program will ensure that not only are all the required libraries present, but that they also match the latest versions on the ArtiSynth server.

## 10.3 The EXTCLASSPATH File

In order to run an external model or package in ArtiSynth, all class paths (i.e., class directories or `jar` files) associated with those external classes must be made visible to ArtiSynth. One way to do this is to list these class paths as entries in the text file `EXTCLASSPATH`, located in `$ARTISYNTH_HOME`.

To add class paths to `EXTCLASSPATH`, open it using a standard text editor (such as `vim`, `gedit`, or `emacs`), and add each required path. For clarity, each path is typically added on a separate line. However, multiple paths can be added on the same line if they are separated by the path separator character used for that OS.

The syntax rules for `EXTCLASSPATH` are:

1. Class path entries on the same line should be separated by a path separator character (a semi-colon `;` for Windows and a colon `:` for MacOS and Linux).
2. The `#` character comments out all remaining characters to the end of line.
3. The `$` character can be used to expand environment variables.
4. Any spaces present **will** be included in the path name.

An example `EXTCLASSPATH` might look like this:

```
/research/artisynt_models/classes  
/research/models/special.jar  
$HOME/projects/crazy/classes
```