

ArtiSynth Installation Guide for Windows

John Lloyd, Sebastian Kazenbroot-Guppy, and Antonio Sánchez

Last updated: January, 2020

Contents

1	Introduction	4
2	Prerequisites	4
3	Installing Java	4
3.1	Making the JDK visible to your system	4
4	Installing a Precompiled Release	5
5	Installing from Github	5
5.1	Cloning the repository	6
5.1.1	Clone using Git for Windows	6
5.1.2	Cloning using Cygwin	6
5.1.3	Cloning using Eclipse	6
5.2	Downloading the libraries	6
6	Compiling ArtiSynth	6
6.1	Compiling with Eclipse	6
6.2	Compiling with Cygwin	7
7	Running ArtiSynth	7
7.1	Running using artisynth.bat	7
7.2	Running using Cygwin	7
7.3	Running using Eclipse	8
7.4	Command line arguments	8
7.5	Loading and Running Models	8
8	Installing artisynth_models and Other External Packages	8
8.1	Downloading	9
8.2	Compiling	9
8.2.1	Compiling with Eclipse	9
8.2.2	Compiling from the command line	9
8.3	Making Classes Visible to ArtiSynth	9
8.3.1	Using the Eclipse Classpath	10
8.3.2	Using the EXTCLASSPATH file	10
8.3.3	Using CLASSPATH environment variable	10
9	Updating ArtiSynth and Other Packages	10
9.1	Library updates	10

10 The Eclipse IDE	11
10.1 Installing Eclipse	11
10.2 Configuring Eclipse for ArtiSynth	11
10.3 Importing ArtiSynth projects into Eclipse	11
10.3.1 Importing external projects	11
10.3.2 Importing projects from a remote Git repository	12
10.3.3 Cloning a project from a remote Git repository	13
10.3.4 Importing from a Subversion repository	14
10.3.5 Installing project files	15
10.4 Configuring environment variables	15
10.4.1 Setting environment variables	15
10.5 Command line and JVM arguments	15
10.5.1 Setting command line and JVM arguments	16
10.6 Adding projects to the build path	16
10.7 Adding projects to the ArtiSynth launch configuration	17
10.8 Installing a Subversion plug-in	17
10.9 Preventing excessive resource copying	18
11 Additional Information	18
11.1 Adding Directories to the System Path	18
11.1.1 Windows 10	18
11.1.2 Windows 8 and earlier	18
11.2 Git for Windows	19
11.3 The TortoiseSVN Client	19
11.4 Cygwin	19
11.5 Environment variables	20
11.5.1 Setting environment variables	20
11.5.2 Typical environment settings	21
11.5.3 Cygwin environment settings	21
11.6 ArtiSynth Libraries	21
11.7 The EXTCLASSPATH File	22
11.8 Quick Git Summary	22
11.9 Quick Subversion Summary	23

1 Introduction

This document describes how to install and run ArtiSynth on 64-bit Windows machines. There are two ways to obtain ArtiSynth: installing a precompiled release, or installing the latest development version from Github. Installing a precompiled release is the easiest way to try out some of the basic demo programs. Installing the development version is recommended for developers who want to keep their codebase current and be able to install new features and bug fixes.

Generally, users will also want to install and run external models and packages that have been created either by others or by themselves. In particular, the package collection `artisynt_models` contains an open source set of models primarily related to head and neck anatomy. Installation of this and other packages is discussed in Section 8.

2 Prerequisites

To install ArtiSynth on Windows, you will need:

- A 64 bit version of Windows
- Java 8 (see Section 3)
- A three-button mouse is recommended for GUI interaction

We have stopped supporting 32 bit systems, for ease of maintenance and because ArtiSynth applications often require more memory than 32 bit systems can provide.

In this document, the location of the ArtiSynth installation folder will be denoted by `<ARTISYNTH_HOME>`. For example if ArtiSynth is installed in

```
C:\people\roger\artisynt_core
```

then `<ARTISYNTH_HOME>` denotes this folder and `<ARTISYNTH_HOME>\lib` denotes the sub-folder

```
C:\people\roger\artisynt_core\lib
```

3 Installing Java

By default, ArtiSynth is compiled to be compliant with Java 8. While it may be possible to run ArtiSynth under later Java versions, there have been reports of compatibility problems and warnings involving the Java OpenGL (JOGL) interface. Therefore we recommend using Java 8 until these issues are resolved.

ArtiSynth requires that you have a full Java development kit (JDK) installed, which comes with a Java compiler. A simple run time environment (JRE) will not be sufficient. We recommend Java SE Development Kit 8uXXX (where XXX is the latest revision number), which can be obtained from Oracle (registration required) at

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

3.1 Making the JDK visible to your system

After the JDK has been installed, it is important to ensure that it is visible to your system and that it supersedes any other Java installations. One test for this is to open a CMD window and run the command

```
> javac -version
```

The output should match the version of the installed JDK. If it does not, or if the command `javac` is not found, then one fix is to add the folder `<JDK_DIR>\bin` to your system Path, as described in Section 11.1, where `<JDK_DIR>` is the JDK installation folder. On Windows, `<JDK_DIR>` is likely to be located under `Program Files\Java`. In particular, `<JDK_DIR>\bin` should be added *ahead* of any other Java installations that might be specified on the Path. To see the current contents of the Path, open a CMD window and run the command

```
> echo %PATH%
```

It may not be necessary to add `<JDK_DIR>\bin` to your system Path if you are intending to compile and run ArtiSynth exclusively within an integrated development environment (IDE), such as Eclipse, since you should be able to specify the JDK directly within the IDE (as described in Section 10.2).

4 Installing a Precompiled Release

Installing one of the precompiled releases is the easiest way to obtain ArtiSynth for running demo programs or some existing models. To do this, go to www.artisynth.org/downloads, download the distribution you want, and unzip it in an appropriate location on your computer.

On Windows, it is recommended that ArtiSynth be installed in a location for which none of the folder names contain spaces (i.e., `Program Files`). This can be important for some legacy ArtiSynth programs to run correctly.

Once ArtiSynth is downloaded and unzipped, it should be possible to run it immediately by executing the `artisynth.bat` file located in `<ARTISYNTH_HOME>\bin` (see Section 7.1). More details on running ArtiSynth and its demo models are given in Section 7.

If you modify any of the demonstration models, or add models of your own, it will be necessary to compile the changes. Compilation is discussed in Section 6.

5 Installing from Github

If you wish to obtain updates and bug fixes, we recommend installing the current development version from Github, which is a web-based repository service based on the source control management system Git. A very brief summary of Git is given in Section 11.8.

The latest ArtiSynth development version is available from Github at the URL

```
https://github.com/artisynth/artisynth_core.git
```

Users who install from Github can continue to update their codebase to keep it current (Section 9). In some cases, developers we work with closely can also obtain, by mutual arrangement, write access to our Github repository, allowing them to also commit changes.

Users who have a Github account combined with SSH keys may instead wish to clone using the SSH URL

```
git@github.com:artisynth/artisynth_core.git
```

For users with repository write access, this will allow them to perform subsequent `push` operations without having to enter a username and password.

Installing from Github entails the following steps:

- Clone (i.e., checkout) the ArtiSynth repository (Section 5.1).
- Download the Java and native libraries (Section 5.2).
- Compile the codebase (Section 6).

It should then be possible to run ArtiSynth and its demo models as described in Section 7.

5.1 Cloning the repository

There are several ways to clone ArtiSynth from Github.

5.1.1 Clone using Git for Windows

Install Git for Windows (Section 11.2), and then from either Git Bash or the CMD console, enter the command

```
git clone https://github.com/artisynth/artisynth_core.git [<dir>]
```

The argument `<dir>` is optional and gives the name of the folder into which the repository and working copy should be extracted; if omitted, the folder will be named `artisynth_core`.

5.1.2 Cloning using Cygwin

If you have Cygwin (Section 11.4) installed, along with the `git` package, then you can check out ArtiSynth within a Cygwin shell window using the same `git clone` command described in Section 5.1.1.

5.1.3 Cloning using Eclipse

If you are planning to develop ArtiSynth models in Java, and if you are planning to do this with the Eclipse IDE (Section 10), then it might be easier to do the Git clone directly in Eclipse. Follow the instructions in Section 10.3.2, using the URL `https://github.com/artisynth/artisynth_core.git` described above.

5.2 Downloading the libraries

Because the `jar` files and native libraries used by ArtiSynth are large, they are not stored in the Github repository. Instead, they must be downloaded separately. This can be done using the command `updateArtisynthLibs.bat`, located in `<ARTISYNTH_HOME>\bin`. You can execute it by double-clicking on it in a file-browser, or by opening a command console (CMD), navigating to `<ARTISYNTH_HOME>`, and entering the command

```
% bin\updateArtisynthLibs
```

If you have Cygwin (Section 11.4) installed, the same command is available as `updateArtisynthLibs`, also located in `<ARTISYNTH_HOME>\bin`.

6 Compiling ArtiSynth

Versions of ArtiSynth obtained from Github need to be compiled before they can be run. Precompiled releases do not need to be compiled in order to run the demonstration models, but will need to be compiled if models are modified or new ones are added.

Java compilation and code development is typically done using an integrated development environment (IDE), although it is possible (particularly on Linux and MacOS) to use external text editors and command line tools. This section describes how to compile ArtiSynth using either the Eclipse IDE, or command line tools available in the Cygwin Unix emulator. For more information on Eclipse and how to obtain it, see Section 10.

6.1 Compiling with Eclipse

Before using Eclipse to work with ArtiSynth projects, you should follow some of the basic configuration steps described in Section 10.2.

If your Github clone has been done outside of Eclipse (i.e., not according to Section 5.1.3), then you need to first import ArtiSynth into Eclipse. Follow the instructions in Section 10.3.1, using `<ARTISYNTH_HOME>` as the top-level project folder `<PROJECT_DIR>`.

Once ArtiSynth has been imported, you should be able to compile 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 compile 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.

6.2 Compiling with Cygwin

If you have Cygwin (Section 11.4) installed, ArtiSynth can also be built from a Cygwin terminal by running a `make` command in the top level folder. Before doing this, you need to first set the environment variables `ARTISYNTH_HOME` and `CLASSPATH` as described in Sections 11.5 and 11.5.3. ArtiSynth can then be built by executing

```
> cd <ARTISYNTH_HOME>
> make
```

7 Running ArtiSynth

7.1 Running using artisynth.bat

The most direct way to start ArtiSynth is to run the batch file `artisynth.bat`, located in `<ARTISYNTH_HOME>\bin`. This can be done by double-clicking on it in a file browser.

You can also create a shortcut to this batch file (by right clicking on it and selecting Create Shortcut), and then placing this shortcut in either the START menu or on the Desktop. However, the batch file itself must remain in `<ARTISYNTH_HOME>\bin`.

Alternatively, you can open a command console (CMD), navigate to `<ARTISYNTH_HOME>`, and enter the command

```
% bin\artisynth
```

It is recommended to place `<ARTISYNTH_HOME>\bin` in your PATH environment variable (Section 11.5), so that the console command simplifies to

```
% artisynth
```

regardless of the current folder.

Note:

If the `bin` directory for your JDK installation is not in your systems's Path, the `java` command may not be visible and this will cause `artisynth.bat` to fail. See 3.1 for a description of how to fix this.

7.2 Running using Cygwin

If you have Cygwin (Section 11.4) installed, then you can run ArtiSynth directly with the `artisynth` command located in `<ARTISYNTH_HOME>/bin`:

```
> cd <ARTISYNTH_HOME>
> bin/artisynth
```

If `<ARTISYNTH_HOME>/bin` has been placed in your PATH variable (Section 11.5), then it is sufficient to do

```
> artisynth
```

`artisynth` can be called with a number of options; to see these, run

```
> artisynth -help
```

7.3 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 by choosing `Run > Run`.

In some cases, one may wish to adjust environment variables, command line arguments, or Java JVM arguments to affect how ArtiSynth behaves. Instructions for doing so are contained in Sections [10.4](#) and [10.5](#).

7.4 Command line arguments

The `artisynth` command accepts command line arguments, a full list of which can be seen by running `artisynth` with the `-help` option:

```
% artisynth -help
```

Descriptions of these options appear in various places within the ArtiSynth documentation. For example, one commonly used option is `-model <modelName>`, which instructs ArtiSynth to preload a model associated with a given class name:

```
% artisynth -model artisynth.demos.mech.SpringMeshDemo
```

When running under Eclipse, command line arguments can be set in the launch configuration, as described in Section [10.5](#).

7.5 Loading and Running Models

Once ArtiSynth starts up, you can use it to load and run models. General instructions on how to load and run models are given in the section “Loading and Simulating Models” of the [ArtiSynth User Interface Guide](#).

By default, ArtiSynth comes with a number of demonstration models, which can be loaded and run as follows:

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 main ArtiSynth window.

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



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

8 Installing `artisynth_models` and Other External 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 an open source collection of anatomical models, focused primarily on the head and neck region (see www.artisynth.org/models).

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

1. Downloading (either a precompiled version or repository checkout; Section [8.1](#)).
2. Compiling (if necessary; Section [8.2](#)).
3. Making classes visible to ArtiSynth (Section [8.3](#)).

8.1 Downloading

Model and package collections are usually available from either Git or Subversion repositories.

For `artisynt_models`:

- A precompiled version is available from www.artisynt.org/models. To ensure compatibility, this should only be used in combination with the matching precompiled ArtiSynth version. For example, `artisynt_models_3.5.zip` should only be used with `artisynt_core_3.5.zip`.
- The current development version is available from Github at

`https://github.com/artisynt/artisynt_models.git`

To ensure compatibility, this should only be used in combination with the most recent development version of ArtiSynth.

Precompiled collections can simply be downloaded and unpacked into a desired location. Those available from Git or Subversion may be obtained as described in Sections 11.8 or 11.9. For convenience, we recommend placing each collection in a folder adjacent to `<ARTISYNTH_HOME>`.

Repository-based collections that contain Eclipse project files (such as `artisynt_models`) can be imported directly into Eclipse as described in Section 10.3.2 (Git) or Section 10.3.4 (Subversion). Other collections that contain the project files bundled inside an `eclipseSettings.zip` file may be imported as described in Section 10.3.3 (Git) or Section 10.3.4 (Subversion).

8.2 Compiling

Collections that are obtained from Git or Subversion will need to be compiled. Compilation will also be necessary if models in the collection are modified or if new ones are added.

8.2.1 Compiling with Eclipse

If compilation is being done with Eclipse, the collection will need to be imported into Eclipse as a project (if this has not already been done) by following the instructions in Section 10.3.1. Compilation can then be performed as described in Section 6.1.

Important: for collection projects to compile properly in Eclipse, the `artisynt_core` project (and any other projects they depend on) will have to be added to their build path. The default Eclipse settings supplied with some projects may already contain the required build path dependencies. For example, the settings for `artisynt_models` contain the required reference to `artisynt_core`. In other cases, it may be necessary to add projects to the build path explicitly, as described in 10.6.

8.2.2 Compiling from the command line

If the collection has a `Makefile` in its root folder, then it can be compiled from a Cygwin terminal by running `make` in the root folder, as described in Section 6.2. Before doing this, the top-level folder for the collection's `class` files must be added to the `CLASSPATH` environment variable (Sections 11.5 and 11.5.3). In collections maintained by ArtiSynth, this will be the folder `classes`, located directly under the collection root folder (e.g., `artisynt_models\classes`).

8.3 Making Classes Visible to ArtiSynth

Models in an external collection are executed by running ArtiSynth itself (Section 7). However, the classes associated with these models must be made visible to ArtiSynth. This can be arranged in several different ways:

8.3.1 Using the Eclipse Classpath

If you are running from Eclipse, then you can make the classes of a collection visible to ArtiSynth by adding its associated Eclipse project to the Classpath of your ArtiSynth launch configuration, as described in Section 10.7.

8.3.2 Using the EXTCLASSPATH file

Alternatively, you can make the classes of external projects visible to ArtiSynth by adding the path names of all their top-level class folders (or jar files, if relevant) to the file <ARTISYNTH_HOME>\EXTCLASSPATH (described in Section 11.7).

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

```
C:\projects\artisynth_models\classes
```

8.3.3 Using CLASSPATH environment variable

Finally, if you are running using `artisynth.bat` (or `artisynth` in Cygwin), then you can make external classes visible by adding them to your CLASSPATH environment variable (see Sections 11.5 and 11.5.3).

9 Updating ArtiSynth and Other Packages

One reason to use a clone 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. Users may also be notified via the `artisynth-updates` email list.

Users working from Eclipse may update simply by selecting the project in the Package Explorer and selecting **Team > Pull** from the context menu.

Updating may also be done by issuing the

```
> git pull
```

command from within <ARTISYNTH_HOME>, using either Git for Windows (Section 11.2), the Cygwin shell (Section 11.4), or another Git application.

Other Git-based packages may be updated similarly.

For Subversion-based packages, updating is done by issuing an update request (Section 11.9). This can be done from within Eclipse (if a Subversion plugin has been installed; Section 10.8) by selecting the project in the Package Explorer and selecting **Team > Update** (or **Team > Update to Head**) from the context menu. Subversion updates can also be done using either TortoiseSVN (Section 11.3), or from the Cygwin command line by calling

```
> svn update
```

from inside the top-level project folder.

9.1 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 5.2.

10 The Eclipse IDE

Eclipse is an integrated development environment (IDE) commonly used for Java code development, and many ArtiSynth developers use it for both developing models in Java 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.

10.1 Installing Eclipse

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

The Eclipse instructions described below are based on the “Neon” distribution, but should be largely similar for later versions.

10.2 Configuring Eclipse for ArtiSynth

There are a few things one should do to configure Eclipse for ArtiSynth related projects:

- *Open a Java perspective*, by choosing Window > Open Perspective > Java. This will place Eclipse in a state suitable for editing Java projects. Individual projects can be viewed and navigated through the Package Explorer window.
- *Check that Java is ArtiSynth compatible*: The Java version used by Eclipse needs to be set to one that is ArtiSynth compatible (which is at present Java 8). To verify that this is the case, choose Window > Preferences > Java > Installed JREs and verify that a compatible Java is installed and selected. If an appropriate JRE is not installed, it can be added by clicking the Add... button on the right of the panel, and then finding the Java 8 distribution on your system.
- *Configure building to prevent excess resource copying*: The steps required to do this are described in Section 10.9, and will speed up project compilation by preventing unneeded files from being copied into the `classes` folder.

10.3 Importing ArtiSynth projects into Eclipse

ArtiSynth projects include the core distribution (`artisynth_core`), the open source models collection `artisynth_models` (which contains human anatomy models), as well as other model and code collections maintained by the ArtiSynth team and other users.

There are several ways to import ArtiSynth projects into Eclipse. If the project has already been downloaded or checked out from a repository, then it can be imported as an external project (Section 10.3.1). Otherwise, Eclipse itself may sometimes be used to checkout the project directly from either Git or Subversion (Sections 10.3.2, 10.3.3, and 10.3.4).

10.3.1 Importing external projects

Let `<PROJECT_DIR>` denote the top-level project folder. For the core distribution `artisynth_core`, this will also be `<ARTISYNTH_HOME>`.

First check the following:

- `<PROJECT_DIR>` should contain the Eclipse project files (including `.project`). If it does not, it should contain a file `eclipseSettings.zip`, which should be unzipped directly into `<PROJECT_DIR>` (not into a sub-folder), as described in Section 10.3.5, so that `.project` and `.classpath` then appear there.
- If the project is the ArtiSynth core distribution (i.e., `artisynth_core`), and was obtained from Github, then make sure you have downloaded the required `jar` files and native libraries as described in Section 5.2.

Then import the project into Eclipse as follows:

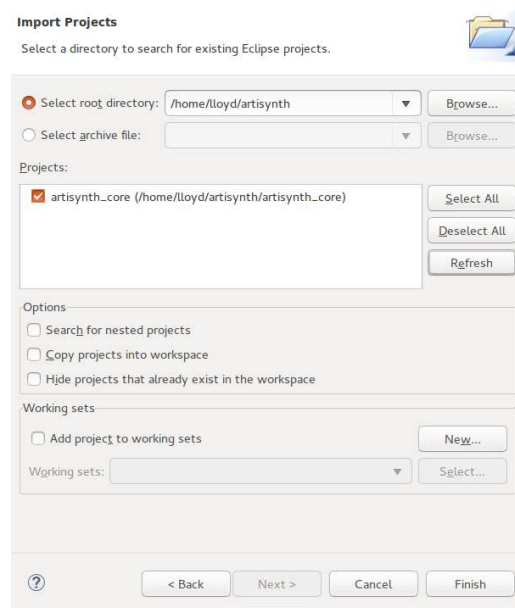


Figure 2: Eclipse Import Projects dialog.

1. From within Eclipse, choose File > Import
2. An Import dialog will appear. Select General > Existing Projects into Workspace and click Next.
3. An Import Projects dialog will appear. In the field Select root directory, enter (or browse to) the *parent* folder of <PROJECT_DIR>. The project itself should now appear in the Projects box (Figure 2). (If other projects are contained in the parent folder, these will appear as well.) Make sure that the desired project is selected and then click Finish.

If Eclipse complains that "No projects are found to import", or does not otherwise show the project as available for import, then most likely the <PROJECT_DIR> folder does not contain a .project file. In the case of repositories that keep Eclipse project files bundled in an eclipseSettings.zip file, this usually means that the contents of that file were not properly unzipped into <PROJECT_DIR> (Section 10.3.5).

10.3.2 Importing projects from a remote Git repository

Some source repositories contain Eclipse project files in their repositories, and so can be imported directly into Eclipse using the repository's URL. The Eclipse project associated with ArtiSynth is called `artisynth_core`, while the models project is called `artisynth_models`.

To import these directly:

1. If necessary, open a Java perspective by choosing Window > Open Perspective > Java.
2. Choose File > Import... > Git > Projects from Git from the main menu.
3. A Select Repository Source dialog will appear. Choose Clone URI and click Next.
4. A Source Git Repository dialog will appear (Figure 3, left). Under Location, enter the project URI in the URL field. The default URL for ArtiSynth is `https://github.com/artisynth/artisynth_core.git`. In some cases, such as when using an SSH URL, or accessing a project with restricted access, it may also be necessary to provide a user name and password in the Authentication fields. (This will be your Gitlab or Github account information for repositories stored on those sites.) After entering the required information, click Next.
5. A Branch Selection dialog may appear. If it does, make sure only the master branch is selected, and then click Next.

6. A Local Destination dialog will appear (Figure 3, right). In the Directory field, enter the path of the local folder, which will contain both the cloned repository and the working copy. For ArtiSynth itself, this will also be the ArtiSynth home folder (<ARTISYNTH_HOME>). After entering the folder information, click Next.
7. A Select a wizard ... dialog will appear. Select Import existing Eclipse projects and click Next.
8. An Import Projects dialog will appear. Make sure project you wish to import is selected and click Finish.
9. In the case of ArtiSynth (i.e., `artisynth_core`), from *outside* Eclipse, download the Java and native libraries, as described in Section 5.2. Then refresh the project from within Eclipse (by selecting it in the Package Explorer window and choosing Refresh from the context menu). The project should now be able to compile.

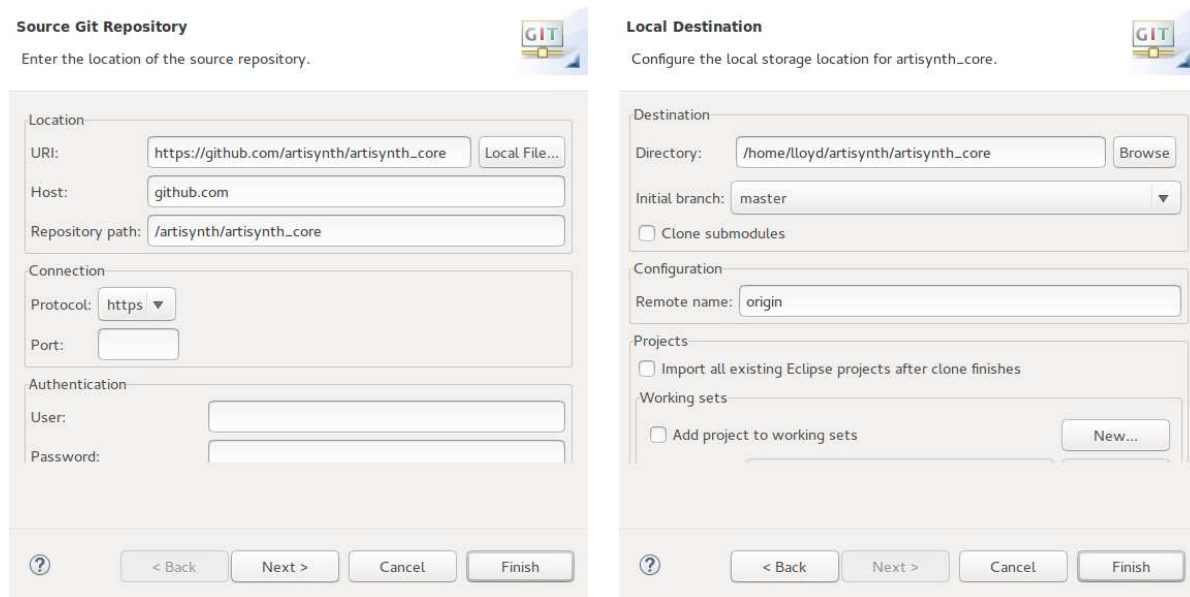


Figure 3: Eclipse dialogs for importing a Git repository.

10.3.3 Cloning a project from a remote Git repository

Some source repositories (such as `artisynth_research`) do not directly contain Eclipse project files in their repositories. Instead, the project files are contained inside an `eclipseSettings.zip` file that must be extracted into the project root folder. This is to prevent undesired local changes to the project settings from being propagated to all users.

In this case, we proceed as follows:

1. Choose Window > Show View > Other ... > Git > Git Repositories from the main menu to open a Git Repositories view window.
2. Within the Git Repositories window, choose the button (or pull down menu item) that says Clone a repository.
3. A Source Git Repository dialog will appear (Figure 3, left). Enter the URL for the repository. Also, if the repository has read access restrictions, it will generally be necessary to specify a user name and password in the Authentication fields. (This will be your Gitlab or Github account information for repositories stored on those sites.) After entering the required information, click Next.
4. A Branch Selection dialog may appear. Usually you want to select only the master branch, and then click Next.
5. A Local Destination dialog will appear (Figure 3, right). In the Directory field, enter the path of the local folder, which will contain both the cloned repository and the working copy. After entering the folder information, click Finish.
6. Finally, from *outside* Eclipse, locate the file `eclipseSettings.zip` in the project's top folder, and then unzip this file directly into that folder (not into a sub-folder), so that `.project` and `.classpath` appear in the top folder. For details, see Section 10.3.5.

The project can now be imported into Eclipse by following the steps in Section 10.3.1, using the project's parent folder as the "root directory".

10.3.4 Importing from a Subversion repository

If Eclipse has a Subversion plug-in installed (Section 10.8), you may import an ArtiSynth project by checking it out directly from the repository located by the project's *Subversion_URL*. For the project `artisynth_projects`, this is

```
https://svn.artisynth.org/svn/artisynth_models/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 the appropriate user name and password in the Authentication section of the dialog. (If the SVN repository is hosted by us, we will have given you this name and password.) 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 folder 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 folder `<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, check to see if the file `eclipseSettings.zip` exists in the project's top folder. If it does, unzip this file directly into the top folder (not into a sub-folder), so that `.project` and `.classpath` appear there. For details, see Section 10.3.5.
 11. Finally, load the new settings into the project by selecting the project in the Package Explorer window and selecting Refresh from the context menu.
-

10.3.5 Installing project files

Some project repositories contain their eclipse project files bundled in the zip file `eclipseSettings.zip`, instead of keeping them under direct repository control. This is to prevent unwanted local configuration changes from being propagated back into the repository. The project files need to be unzipped directly into the project folder (not into a sub-folder) to enable the project to be loaded into Eclipse.

Let `<PROJECT_DIR>` denote the top-level project folder. On Windows, project files can be extracted from within the file browser. Double click on `eclipseSettings.zip` and extract the files into `<PROJECT_DIR>`. This will create the files `.project` and `.classpath`, along with the folder `.settings`, in `<PROJECT_DIR>`. In the case of `artisynt_core`, it will also create the file `ArtiSynth.launch` containing the default launch configuration.

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

10.4 Configuring environment variables

While it is generally *not* necessary to set environment variables in Eclipse, it may be useful to do this on occasion to control certain aspects of ArtiSynth's operation. Directions on setting the environment variables are given in Section 10.4.1, and descriptions of the variables themselves may be found in Section 11.5.

Some variables that are commonly set within Eclipse include:

- `ARTISYNTH_HOME`: If set, this should be set to `<ARTISYNTH_HOME>`. Normally ArtiSynth is able to infer its own location internally, so it is generally unnecessary to set this variable explicitly.
- `OMP_NUM_THREADS`: Specifies the maximum number of processor cores available for multicore execution.
- `ARTISYNTH_PATH`: A list of folders, separated by semi-colons ";", which ArtiSynth uses to search for configuration files. See Section 11.5.

If any of the above variables have already been set externally in Windows (Sections 11.5.1 and 11.5.2), such that they are visible to Eclipse at start-up, then they do not need to be set in the launch configuration.

10.4.1 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 the launch configuration (the default is named 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.

10.5 Command line and JVM arguments

As described in Section 7.4, the `artisynt` command accepts command line arguments. To invoke these when running from Eclipse, it is necessary to set the desired arguments in the launch configuration, as described below.

Sometimes it may also be necessary to set JVM arguments, which control the Java virtual machine running ArtiSynth. An example of such an argument is `-Xmx`, which can be used to increase the maximum amount of memory available to the application. For example, `-Xmx6g` sets the maximum amount of memory to 6 gigabytes.

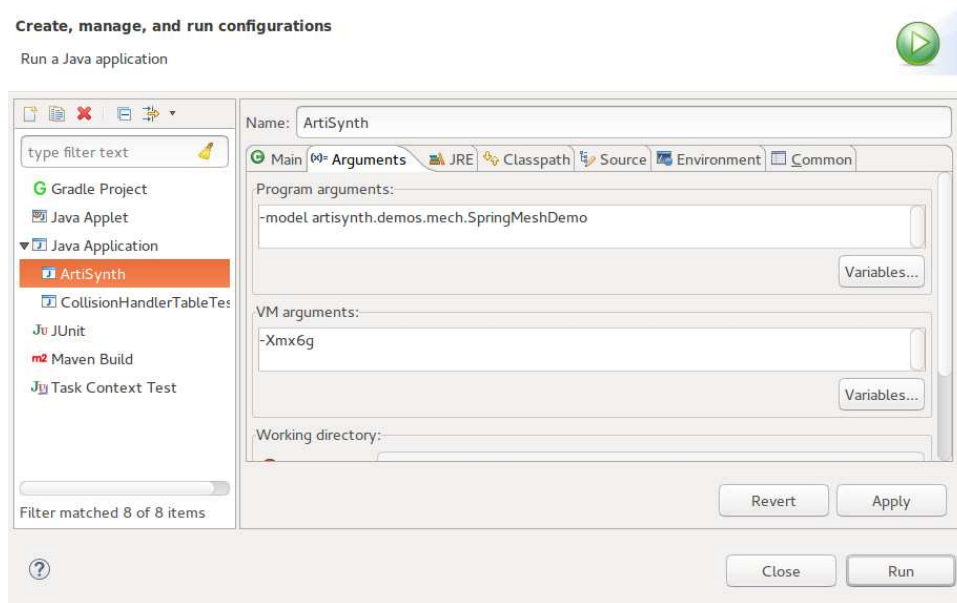


Figure 4: Setting command line and JVM arguments for a run configuration.

10.5.1 Setting command line and JVM arguments

To set command line arguments for your Eclipse application:

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 the launch configuration (the default is named ArtiSynth).
5. In the right panel, select the Arguments tab.
6. Program arguments (which are passed directly to ArtiSynth) should be specified in the Program arguments box. JVM arguments should be specified in the VM arguments box. See Figure 4.
7. When finished, click Close.

10.6 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

10.7 Adding projects to the ArtiSynth launch configuration

The classes of external projects can be made visible to ArtiSynth by adding the projects themselves 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.

10.8 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.

Instructions for installing Subversive can be obtained at 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. 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
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.

10.9 Preventing excessive resource copying

By default, ArtiSynth classes are built in a directory tree (`<PROJECT_DIR>\classes`) that is separate from the source tree (`<PROJECT_DIR>\src`), where `<PROJECT_DIR>` denotes the project root folder and is `<ARTISYNTH_HOME>` for ArtiSynth itself. That means that Eclipse will try to copy all non-Java files and folders 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 single character '*'.

11 Additional Information

11.1 Adding Directories to the System Path

The system "Path" is a list of directories which the system searches in order to find executables. Adding a directory to the path allows executables contained in that directory to be called directly from a command window such as CMD.

11.1.1 Windows 10

1. Open the Start search, enter "env", and choose "Edit the system environment variables".
2. Click on Environment Variables.
3. Under User variables (the top window), click on Path and click Edit. If Path does not exist, click New.
4. In the Edit environment variable dialog, click New and enter the full path name for each directory you wish to add.
5. Close each dialog by clicking OK.

11.1.2 Windows 8 and earlier

1. Right-click My Computer, and then click Properties.
2. Click the Advanced tab.
3. Click Environment variables.
4. In the top User variables window, click on Path and then Edit. If Path does not exist, click New.
5. In the edit window, add the full path name for each new directory, separated by semi-colons '; '.
6. Close each dialog by clicking OK.

For example, if ArtiSynth is installed at `C:\artisynth\artisynth_core` and the desired JDK is at `C:\Program Files\Java\jdk1.8.0_221`, then we can add the bin directories for both by setting the User path to

```
C:\artisynth\artisynth_core\bin;C:\Program Files\Java\jdk1.8.0_221\bin
```

Most most command windows and applications need to be restarted in order to get them to notice changes to the PATH.

11.2 Git for Windows

A version of Git for Windows can be installed from git-scm.com/downloads.

It installs a version of Git Bash for that can be used for entering Git commands. While going through the install steps, there is an option to select between:

1. Use Git from Git Bash only
2. Use Git from the Windows Command Prompt
3. Use Git and optional Unix tools from the Windows Command Prompt

Option 2 (Git from CMD) is often fine for most users, and allows them to run all git commands directly from a command console CMD. There are other options for setting the default editor, the default console to open when starting Git Bash, etc., but the defaults should work well in most cases.

11.3 The TortoiseSVN Client

As mentioned above, some ArtiSynth models may be distributed via Subversion (SVN), and access to these will require an SVN client program. A popular SVN client for Windows is TortoiseSVN, which can be acquired from tortoisesvn.net. On that website, navigate to Downloads and then select the package appropriate for your operating system (which presumably will be 64-bits).

Once downloaded, run the MSI installer and follow the instructions in the installer. When the download completes, right-clicking in any file browser in Windows should now present you with SVN version control options, analogous to the command line instructions discussed in other parts of the documentation. We will briefly discuss the three most common operations:

Checking out a repository To check out an SVN repository with TortoiseSVN, right-click in any file exploration window and select SVN Checkout... from the context menu. The current folder will be the default folder for the Checkout directory field, and the *Subversion_URL* can be selected in the URL of repository field. It is recommended that other fields remain as their default values.

Updating a working copy While inside a working copy of a repository in a file browser window, updating that repository is as simple as right-clicking and selecting SVN Update.

Committing changes Users which have write-access to a Subversion repository may commit changes that they have made. Right-click in a file browser window inside a working copy of a repository and select Commit.... Write a commit message and then select OK.

11.4 Cygwin

Cygwin provides Windows users with a Linux-like, shell-based command environment. It provides useful tools and enables ArtiSynth users to employ all the script-based commands in `<ARTISYNTH_HOME>\bin`, as well as various Makefile commands.

Cygwin can be downloaded from www.cygwin.com. Run the executable that was just downloaded (setup.exe) to begin installation. After selecting a download source, an install directory, a package directory, an internet connection, and a download site, the installer will display a list of packages which the user can select for download. Normally, a default set of packages is already selected for installation. It is advisable to install these packages to ensure that Cygwin retains its basic capabilities. It is also recommended that you select the following additional packages:

- archive
- git, svn, make (located under devel)
- python (located under interpreters).
- openssh (located under net)

If you are planning to compile C/C++ code, you may also want to install the various gcc and gdb packages (located under devel).

11.5 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 folder.

ARTISYNTH_PATH

A list of folders, separated by semi-colons ";", which ArtiSynth uses to search for configuration files such as `.artisynthInit` or `.demoModels`. A typical setting for `ARTISYNTH_PATH` consists of the current folder (indicated by `"."`), the user's home folder, and the ArtiSynth installation folder. If `ARTISYNTH_PATH` is not defined explicitly in the user's environment, ArtiSynth assumes an implicit path consisting of the folder sequence just described.

CLASSPATH

A list of folders and/or `jar` files, separated by semi-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 folders, separated by semi-colons ";", which the operating system uses to locate executable programs and applications. Placing `<ARTISYNTH_HOME>\bin` in your `PATH` (as described in [Section 11.1](#)) will allow you to run `artisynth` and related commands directly from a command window.

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`.

11.5.1 Setting environment variables

On Windows, a user can view, set, or change environment variables via the following steps:

Windows 10:

1. Open the Start search, enter "env", and choose "Edit the system environment variables".
2. Click on Environment Variables.
3. Choose one of the following options:
 - Click New to add a new variable name and value.
 - Click an existing variable, and then Edit to change its name or value.
 - Click an existing variable, and then Delete to remove it.
4. Close each dialog by clicking OK.

Windows 8 and earlier:

1. Right-click My Computer, and then click Properties.
 2. Click the Advanced tab.
 3. Click Environment variables.
 4. Choose one of the following options:
 - Click New to add a new variable name and value.
-

- Click an existing variable, and then **Edit** to change its name or value.
- Click an existing variable, and then **Delete** to remove it.

5. Close each dialog by clicking **OK**.

Variable settings can reference other environment variables, by surrounding them with percent signs, as in `%VARIABLE_NAME%`. For example, suppose you already have an environment variable `HOME` that gives the location of your home folder, and your ArtiSynth distribution is located in `packages\artisynth_core` relative to your home folder. Then the environment variable `ARTISYNTH_HOME` can be specified as

```
%HOME%\packages\artisynth_core
```

11.5.2 Typical environment settings

Typical settings for the environment variables described above might look like this:

```
ARTISYNTH_HOME c:\users\joe\artisynth_core
ARTISYNTH_PATH .;c:\users\joe;%ARTISYNTH_HOME%
CLASSPATH %ARTISYNTH_HOME%\classes;%ARTISYNTH_HOME%\lib\*
PATH %ARTISYNTH_HOME%\bin;%PATH%
OMP_NUM_THREADS 2
```

11.5.3 Cygwin environment settings

When running a Cygwin terminal, it is possible to set environment variables in the startup script for the terminal's shell. Assuming that this shell is `bash`, then the environment settings described in 11.5.2 can be set by inserting the following into one of `bash`'s initialization files (typically `~/.bashrc`):

```
# set AH to the location of the ArtiSynth install folder
AH=$HOME/artisynth_core
export PATH=$AH/bin:$PATH

# Use Windows path style for ARTISYNTH_HOME, ARTISYNTH_PATH, and CLASSPATH:
export ARTISYNTH_HOME=`cygpath -w $AH`
export ARTISYNTH_PATH=".;`cygpath -w $HOME`; $ARTISYNTH_HOME"
export CLASSPATH="$ARTISYNTH_HOME\classes; $ARTISYNTH_HOME\lib\*;"$CLASSPATH"
export OMP_NUM_THREADS=2
```

Note that even on Cygwin, the environment variables `ARTISYNTH_HOME`, `ARTISYNTH_PATH`, and `CLASSPATH` should be set using Windows path conventions. That is because they may not necessarily be invoked in a Unix-like context.

11.6 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-folders (Windows64 for 64-bit Windows systems).

As described in Section 5.2, these libraries need to be downloaded automatically if the system is obtained from the Github 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.

11.7 The EXTCLASSPATH File

In order to run an external model or package in ArtiSynth, all class paths (i.e., class folders 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 plain text editor (such as Notepad) 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:

```
C:\research\artisynth_models\classes
C:\research\models\special.jar
$HOME\projects\crazy\classes
```

11.8 Quick Git Summary

Git is a distributed source control management (SCM) system that is widely used in the software industry. A full discussion of Git is beyond the scope of this document, but a large literature is available online. Generally, when you *clone* a Git repository, you create a local copy of that repository on your machine, along with a checked out working folder containing the most recent version of the code (which is referred to as the `HEAD`).

Unlike client/server SCMs, Git is distributed, with users maintaining their own private copies of a repository. This allows a great deal of flexibility in usage, but also adds an extra “layer” to the workflow: when you “checkout” from a repository or “commit” to it, you do so with respect to your own *local* copy of that repository, *not* the original (*origin*) repository from which you performed the original clone. The process of merging in changes from the origin to the local repository is known as “pulling”, while committing changes from the local repository back to the origin is known as “pushing”.

There is also another layer of interaction when you commit changes to the local repository: you first *add* them to a staging area (also known as the “index”), and then commit them using the `commit` command.

A very simple workflow for a typical ArtiSynth user is summarized below. The actions are described in command-line form, but the same commands can generally be issued through Eclipse or other interfaces. First, clone the most recent version of the ArtiSynth repository on Github:

```
git clone https://github.com/artisynth/artisynth_core.git [<dir>]
```

This will create a local copy of the Github repository, along with a checked out “working copy”, in the folder specified by `<dir>`, or in `artisynth_core` if `<dir>` is omitted. The repository itself will be located in a sub-folder called `.git`.

Other Git repositories can be cloned in a similar manner. If the repository has read access restrictions, then when performing a checkout it may also be necessary to specify a user name for which the repository has granted read access. This is typically done by embedding the user name in the URL, as in (for example) `https://user@host.xz/path/to/repo.git`.

Later, to fetch the latest updates from the Github repository and merge them into your working copy, then from within the working copy folder you can do

```
git pull
```

If you make changes to some files in your working copy and wish to commit these to your local repository, you first *add* (or remove them) from the staging area using commands such as:

```
git add <fileName>    # add a new (or modified) file
git add *              # add all files
git rm <fileName>     # remove a file
```

and then commit them to your local repository using

```
git commit -m "commit message"
```

Note that you can also add modified files and commit them using the single command

```
git commit -m -a "commit message"
```

To see the current status of the files in your working copy and the staging area, use the command

```
git status
```

and to see the commit history for particular files or folders, use

```
git log [ <filename> ... ]
```

Finally, to push your changes back to the Github repository (assuming you have permission do so), you would do so using the command

```
git push origin master
```

Note that the above commands all have various options not mentioned. There are also numerous topics that haven't been discussed, including the creation and merging of branches, but there are many useful online resources that describe these in detail. Some current references include

```
https://git-scm.com/docs
http://rogerdudler.github.io/git-guide
```

11.9 Quick Subversion Summary

Subversion is a client/server source control management (SCM) system that is widely used in the software industry. A full discussion of Subversion is beyond the scope of this document, but a large literature is available online.

Subversion allows you to *check out* a codebase from a (often remote) repository into a local *working copy*, *update* recent changes from the repository into the working copy, and (if one has the appropriate permissions) *commit* local changes back into to repository.

A Subversion *client* application is used to access both Subversion repositories and local working copies. The remainder of this discussion will assume use of the command-line client *svn*, although other clients are available, including TortoiseSVN for Windows (Section 11.3) and the Subversion plug-ins for Eclipse (Section 10.8).

Some ArtiSynth models collections and code extensions are distributed through Subversion, including the *artisynth_projects* package used by some collaborators. A very simple workflow involving one of these is summarized below.

First, check out the most recent version from the repository, using the repository's URL. For example, the URL for *artisynth_projects* is https://svn.artisynth.org/svn/artisynth_projects, and the associated checkout command is

```
svn checkout https://svn.artisynth.org/svn/artisynth_projects/trunk [<dir>]
```

This will create a local working copy of the "trunk" branch of *artisynth_projects* in the folder specified by *<dir>*, or in *artisynth_projects* if *<dir>* is omitted. Local repository information is stored in a sub-folder called *.svn*.

If the SVN repository has read access restrictions (which *artisynth_projects* actually does), then when performing a checkout it may also be necessary to specify a user name or email address for which the repository has granted read access. This may be done with the *-username* option. The user will also typically be prompted for an access password.

Note:

If you omit the trailing `/trunk` from the Subversion URL, then the checkout will contain the entire Subversion folder structure, including the subdirectories `trunk`, `branches`, and `tags`, which is generally not needed by most users.

Later, to fetch the latest updates from the repository and merge them into your working copy, from within the local folder, would you simply do

```
svn update
```

If you make changes to some files in your working copy and wish to commit these back to the repository (assuming you have the necessary permissions), then you can issue the command

```
svn commit -m "commit message"
```

To add or remove files from the repository, one may use the commands

```
svn add <fileName> ...      # add files
svn delete <fileName> ...    # delete files
```

prior to performing the commit.

To see the current status of the files in your working copy, use the command

```
svn status
```

and to see the commit history for particular files or folders, use

```
svn log [ <filename> ... ]
```

Note that the above commands all have various options not mentioned. There are also numerous topics that haven't been discussed, including the creation and merging of branches, but there are many useful online resources that describe these in detail. The most comprehensive is probably the [Subversion "Redbook"](#).