# Installing and running ePTM v2

These instructions are designed to help you get started with setting up and running a simple instance of the **ePTM** v2. Throughout this tutorial, user inputs and filenames are in blue, inputs and filenames from the GitHub repo are in purple, and hepful hints or “Notes” are *italicized*.

Using this basic foundation, you can build more complex workflows. You will need to have a version of **DSM2** already installed on your machine somewhere. You will also need R 4.0 or above and python 3.6 or above. Please make sure you have the necessary R packages and python libraries already installed. It is essential to have a Python environment manager such as Conda to follow along with this tutorial. These dependencies can be obtained from the scripts in the repo. You should also install a text editor such as Notepad++ for all your scripting needs.

As you become more proficient, you can use the starter scripts herein to build more sophisticated workflows. With experience, you will also realize that the folder structure shown here is just one recommendation to get started. The possibilities are really endless. Good luck!!

## ePTM Git Repo cloning

This section will guide you on how to download the latest version of the **ePTM** from GitHub. To use this part of the tutorial, you will need to have created a GitHub account and installed Git on your local machine.

### Creating a local **ePTM** v2 repository

1. Create a folder for your work, and move to it using explorer, and then open a command prompt from within this folder by typing cmd in the address bar.
2. In the command prompt, type the following: git clone <https://github.com/cvclcm/ePTM_v2>. Enter your username and password if you are prompted to do so. This will download and clone a local instance of the Git repository on your computer.
3. On the command line, type cd ePTM\_v2. You are now in the local git repository’s main branch.

### Making alterations to your version

1. To make changes to your specific setup and not affect the main branch, and to ensure that keeping the main branch updated is straightforward, create a local branch by typing git branch your\_branch\_name. Then enter this branch by typing git checkout your\_branch\_name. You are now in this branch, where you can do whatever you want without affecting the code or the files in the main branch.
2. If you make any changes to your branch, and want to update them, you can commit your changes by typing git commit -am “Simple message describing changes”. If you want to keep your branch up-to-date after you pull into the main, checkout into your branch and type git merge main. If there are differences in the number of files, or within the files themselves between the two branches, you will get a merge conflict.

**PLEASE DO NOT CHECKOUT INTO THE MAIN BRANCH AND MERGE YOUR BRANCH. USE THE MAIN BRANCH SOLELY TO KEEP IT UP-TO-DATE WITH THE ONLIEN REPO.**

### Resolving merge conflicts and staying up-to-date

1. If you get a merge conflict, Git will show you what files are conflicting. For each file that Git reports as having a conflict, open it in a text editor and look for the following line: >>>>. Anything below this indicates the block of lines that Git could not automatically resolve, and you will have the resolve those changes manually by editing your text. Usually, Git is very good at resolving conflicts.

## Creating a folder structure for your work and generating relevant files

This section will guide you on how to create a simple folder structure for your project and populate it with all the configuration files you will need for a single **ePTM** v2 run.

*Note: Make sure you use forward slashes everywhere. For folder paths, don’t forget the ‘/’ at the end.*

### Creating a folder structure

1. First, create the following folder tree somewhere such that the absolute path contains no spaces:

Study> Inputs

Configs

Outputs

Scripts

In the Inputs folder, put the **DSM2** tide file (usually a large HDF5 file), and the **DSM2** and PTM configuration template files. In the Configs folder, put all the configuration files to be used by **ePTM** v2. The configuration template files, and all the configuration files for the latest calibration of the model are available in the configFiles folder in the Git repo. The Outputs folder is where the model outputs will be written into. The Scripts folder will contain all the scripts you write to pre-process data, setup the runs and post-process the results.

*Note: For all subsequent instructions in this section, ./ will indicate the absolute path to the parent directory immediately above the Study folder.*

1. Next, create a folder called Scripts somewhere, and put all the scripts from the scripts folder in the Git repo into it. This is where you will save all the scripts you use in this project.
2. Rename DSM2ConfigTemplate.inp as something else, say DSM2TestConfig.inp. Similarly, rename ePTMConfigTemplate.inp to say, ePTMTestConfig.inp.
3. In DSM2TestConfig.inp, make the following changes to the text alongside the keywords below:

DSM2INPUTDIR Change placeholder text to path where the **DSM2** tide file is

DSM2MODIFIER Name of your study

START\_DATE Date from which you want to run the model (e.g., 01JAN1992)

QUAL\_START\_DATE Add a day to the START\_DATE above (e.g., 02JAN1992)

START\_TIME Enter hours and minutes in military format (e.g., 1745)

END\_DATE Date until which you want to run the model

END\_TIME Enter hours and minutes in military format

HYDROTIDEFILE Path to **DSM2** tide file. In your case, this would be the path to the tidefile in the ./Study/Inputs folder.

*Note: There are many other variables in this file that are only applicable if you are running a* ***DSM2*** *simulation as well. Leave everything else untouched for now.*

*Note: In this configuration file, variables can be declared and used using the ${Var\_Name} syntax. As you become more proficient with configuring and running multiple* ***ePTM*** *runs on the fly, you can start experimenting with more sophisticated variable usage and relative paths.*

1. In the ./Study/Inputs folder, create a CSV with at least one column named time. This should be populated with the release schedule in dd/mm/yyyy hh:mm format. This is the list of times at which particles will be released into the model.

*Note: These release times cannot be earlier than or later than the model simulation time period.*

### Generating the configuration files

1. Make a copy of the ePTMCreateInputs.R script in the ./Study/Scripts folder.
2. In the ePTMCreateInputs.R script, make the following changes in the static data section:

dsm2ConfigDir Absolute path to the ./Study/Inputs folder in single quotes.

dsm2ConfigFile Name of the **DSM2** configuration file you would like to create.

channelPars Absolute path to the ePTMv2CalibValidParams812.csv file you copied from the local Git repo. This is the list of calibrated model parameter values in each **DSM2** water body

mainDir Absolute path to the ./Study/Configs folder.

inpDir Absolut path to the ./Study/Inputs folder.

modifier Name of the study (the same one you used for DSM2MODIFIER).

releaseRegime Name of the CSV you created in Step 4 above without the extension.

nRep Number of replicate particles you want to simulate at each release time.

releaseLocation Common name you can give for your release location. This can be a list.

releaseNode **DSM2** external node number where your release occurs. This can be a list.

checkpoints A list of **DSM2** external nodes where you want to track particle arrivals.

*Note: startDateTime and endDateTime timezones must be modified as needed depending on which system you are modeling. This is only applicable if you have created a* ***DSM2*** *grid for another system in a different timezone.*

1. In the ePTMCreateInputs.R script, in the “Writing to HDF5 files for junction types” section, make the following changes:

behaviorFile Replace the text within quotes on this line to name of the behavior parameters HDF5 file the program will create. Keep this file in the ./Study/Inputs folder.

In the h5write lines with “immortal,” “routing model” and “swim code,” the numbers in the first argument can be changed to toggle different model modes and features.

1. In the the ePTMCreateInputs.R script, in the “Writing ePTM configuration file” section, make the following changes:

startDateTime To spin up the model, **ePTM** v2 simulations will start 7 days prior to the first release, but no particles will be inserted until the first release. To change this spin-up time, modify the number inside - Days(X) on this line.

endDateTime The model will run for 10 days past the last release, To change the number of days, modify the number inside + Days(X) on this line.

ptmConfigFile Absolute path to the **ePTM** v2 configuration file from step 3 within single quotes

ptmTraceFile Absolute path of desired trace file with modifier \_trace.out at the end. This should be in the ./Study/Outputs directory.

ptmRunFile Absolute path of desired echoed runtime file with extension .pof. This should be in the ./Study/Outputs directory.

ptmEchoFile Absolute path of desired echo of the input file runtime file with modifier \_echo.inp at the end. This should be in the ./Study/Outputs directory.

ptmOutFile Absolute path to text model output with modifier \_out.txt at the end. We don’t use this, but there might be some useful information about the results in there. This should be in the ./Study/Outputs directory.

run\_start\_time and run\_end\_time can be modified by the user according to the military format HHMM.

Replace the text PATH\_TO\_DSM2\_CONFIG\_FILE within quotes with the absolute path to the **DSM2** configuration file you created in Step 3.

In the line, delay <- rep(0, nrow(releases)), change the number to some whole number to indicate the number of hours to delay each release by.

1. Run the ePTMCreateInputs.R script. The behavior input and the **ePTM** v2 configuration files will be created.

*Note: When you check the* ***ePTM*** *v2 configuration file, the PARTICLE\_INSERTION section will contain the list of releases that will drive the model.*

*Note: To include nonphysical fish passage barriers, you will have to create a section in the* ***ePTM*** *v2 configuration file with the following format:*

*PARTICLE\_FILTER*

*NAME NODE AT\_WB FILLIN FILE PATH*

*Name of DSM2 external DSM2 waterbody last constant x*

*nonphysical node where it into which fish are*

*barrier is placed barred from entering*

*END*

*Here, x is a decimal number from 0 to 1 indicating what fraction of fish will be barred from entry.*

*Note: To generate* ***ePTM*** *v2 animation, in the IO\_FILE section add the following line:*

*ptm anim out 15min [absolute path to animation binary file.bin]*

## Creating the link to DSM2

This section will guide you on how to link your **DSM2** install to your **ePTM** v2 install for various versions of **DSM2**. For various reasons, it is advisable to have different **DSM2** builds on your machine. For versions lower than 8.1.3, Java 1.8 is needed. For **DSM2** version 8.2, Java 1.6 is needed. So, it is also useful to have multiple JDKs installed on your computer. Assuming that you have two versions of **DSM2** and two versions of Java JDK:

1. On the windows run prompt, type sysdm.cpl to open the System properties window.
2. Click Advanced, then click Environmental Variables.
3. In the System variables section, create a variable for **DSM2** called **DSM2\_HOME**. For example, if you have **DSM2** v8.2 and **DSM2** v8.1.2, then you will create two system variables as:

DSM2\_HOME Absolute path until the bin folder in **DSM2** v8.2

1. Edit the Path variable, and add the following to the path by separating each variable by a semicolon: %DSM2\_HOME %\bin;%DSM2\_HOME %\vista\bin.
2. In the System variables section, create a new variable each for each JDK version, called say JAVA\_HOME\_ver, and enter as its attribute, the absolute path until the bin folder of that JDK.
3. In the path variable, remove any reference to the java JDK, and instead add %JAVA\_HOME\_ver%\bin.
4. Note down the names and paths of the system variables you created.

*Note: that each time you use a different DSM2 version, you have to modify the DSM2\_HOME system variable.*

## Working with the source code

This section will guide you on how to work with the **ePTM** v2 source code directly using the Eclipse IDE, and create executable JAR files for later use. For this, you will need to install the 32-bit Java JDK 1.8 (if you are using **DSM2** v. 8.1.3 or earlier) and 1.6 (if you are using **DSM2** v. 8.2). You will also need to install a version of the Eclipse IDE that supports 32-bit Java. Note that for working with the source code, it is advisable to be in your\_branch\_name on your local Git repository.

*Note: Make sure you use forward slashes everywhere. For folder paths, don’t put a ‘/’ at the end.*

*Note: You will typically work with the source code only if you are making modifications to the code or are debugging. For everything else, use the prebuilt JAR file that came with the Git repo.*

### Installing Java 32-bit compiler and adding it to Eclipse

1. In Eclipse, select Window, then select Preferences, the select Java and go to the Installed JREs tab. Depending on the version of Eclipse, this could be hiding within one of the drop-down menus.
2. Click Add, and then select Standard VM. Then click Next. Click the Directory button next to the JRE home address bar and select the folder in which the Java JDK bin directory is located. Usually, this will be in the Program Files (x86) folder somewhere.

*Note: You will have to select the JDK version that is appropriate to the* ***DSM2*** *version you are using.*

1. Since the underlying **DSM2**-PTM upon which **ePTM** v2 is built uses older versions of Java, you have select Java under Preferences, and set the Compiler Compliance Level to 1.8 or 1.6 depending on your **DSM2** build.
2. Click the Finish button and Apply and Close. You now have linked the Java runtime environment with the Eclipse IDE, and can run code directly from the IDE, compile it and export the JAR file.
3. The Eclipse IDE environment has three main components you will interact with: the Package Explorer pane, the Scripting environment and the Console. The Package explorer pane is where you will see your entire project file structure. The Scripting environment is where you will make code changes. The Console is where you will see your output. Refer to Eclipse documentation for other resources.

### Creating a new project from your local branch

1. From the File menu, select New, and then select Java Project.
2. Give your project a name in the Project Name text box.
3. In the window that opens, uncheck “Use default location” and specify the folder into which the **ePTM** v2 repo was cloned. This will be the folder named ./ePTM\_v2/ptm\_SWFSC in your case.
4. In the JRE section, check Use a Project Specific JRE and select the 32-bit Java version that you already added according to the previous section.
5. Depending on your version of Eclipse, you will either have a section below the Working sets section that has the Java settings. If this is the case, leave everything as is and click Finish. If you don’t see this section, click Next. A Java settings window will pop up. Leave everything as is and click Finish.
6. You should now see that a new project has been populated on the Package Explorer pane. Note that the project will appear as Project\_Name [ePTM\_v2 your\_branch\_name], indicating that the project is using the build on your local Git branch.

### Adding libraries to the project

1. Right click on the project in the Package Explorer pane, click Build Path, and then click Configure Build Path…
2. A Java Build Path window will open. Navigate to the Libraries tab, then select Classpath and click Add JARS…

*Note: Do not worry if you do not see Classpath. Sometimes, this is hidden depending on the version of the JDK. Simply click Add JARS… and continue.*

1. A JAR Selection window will open. Here, navigate down to ePTM\_v2/ptm\_SWFSC/lib, select all the .jar files there and click OK.
2. Back in the Java Build Path window, click Apply and Close. Now, all the libraries required to run **ePTM** v2 are linked with your project.

### Creating a run configuration to run the model

1. In the main menu, select Run, and then select Run configurations… Now in the Run Configurations window, click on Java Application and then click on the New launch configuration icon, which is the blank page with a plus.
2. Enter a name for your build on the top. The Main tab’s Project will be populated by the project name you used earlier. In the Main class text box, type DWR.DMS.PTM.MainPTM. This is the primary class which calls all other classes in **ePTM** v2.
3. In the Arguments tab, you will enter the arguments each on a new line to the **ePTM** 2 code as follows in the Program arguments text box:

Absolute path to the ePTM configuration file you created earlier

Absolute path to the behavior HDF5 file you created earlier

Absolute path to the ./Study/Outputs folder to where you want to write the ePTM output HDF5 file

Absolute path to where the ./Study/Config folder is

Absolute path to where the ./Study/Config folder is

ChansOrientsBends812.csv

Junctions812.csv

LatVel.csv

LatEps.csv

LatdEp.csv

Latd2Ep.csv

VerEps.csv

VerdEp.csv

These are the arguments to the model call.

In the VM arguments text box, enter -Xss5M -Xms512M -Xmx1024M -Xverify:none

These are the Java compiler flags, and they may vary depending on your system architecture. Check Oracle’s documentation for additional details on these flags.

Select the Other radio button in the Working directory section and enter the absolute path to ./Study/Inputs

1. In the JRE section, select alternate JRE and select your JDK version.
2. Leave the Classpath and Source tabs as they are.
3. In the Environment tab, click new and select the **DSM2** system variable you created earlier. Then select new, and create a variable called Path with attribute as the absolute path to ./your DSM2 directory/bin folder.
4. Leave the Common and Prototype tabs as they are.
5. If you followed all the steps correctly, then click Show command line, you should see something like this:

C:\Program Files (x86)\Java\jre1.8.0\_333\bin\javaw.exe -Xss5M -Xms512M -Xmx1024M -Xverify:none -Dfile.encoding=Cp1252 -classpath "D:\Projects\ePTMTutorial\ePTM\ePTM\_v2\ptm\_SWFSC;D:\Projects\ePTMTutorial\ePTM\ePTM\_v2\ptm\_SWFSC\lib\COM.jar;D:\Projects\ePTMTutorial\ePTM\ePTM\_v2\ptm\_SWFSC\lib\edu.jar;D:\Projects\ePTMTutorial\ePTM\ePTM\_v2\ptm\_SWFSC\lib\opencsv-3.0.jar;D:\Projects\ePTMTutorial\ePTM\ePTM\_v2\ptm\_SWFSC\lib\pj20120620.jar;D:\Projects\ePTMTutorial\ePTM\ePTM\_v2\ptm\_SWFSC\lib\sis-jhdf5-batteries\_included.jar;D:\Projects\ePTMTutorial\ePTM\ePTM\_v2\ptm\_SWFSC\lib\xml.jar" DWR.DMS.PTM.MainPTM D:/Projects/ePTMTutorial/Study/Inputs/ePTMTestConfigNew.inp D:/Projects/ePTMTutorial/Study/Inputs/behaviorTest.h5 D:/Projects/ePTMTutorial/Study/Outputs/ePTMTest.h5 D:/Projects/ePTMTutorial/Study/Configs D:/Projects/ePTMTutorial/Study/Configs ChansOrientsBends812.csv Junctions812.csv LatVel.csv LatEps.csv LatdEp.csv Latd2Ep.csv VerEps.csv VerdEp.csv

*Note: It might be useful for future reference to save the entries in the run configuration somewhere.*

### Running the model

1. In the main menu, click Run, and then click Run in the dropdown menu. If everything went well, you should see the model run progress in the console.
2. If there was any problem, then you will see some warnings or errors with a paper trail in the console:

Typically, issues in your project build will throw up flags even before you get to the point where you can run the code.

Issues related to the Java code of the ePTM will appear as warnings or errors with class and line number locations.

Issues related to the underlying folder structure, **DSM2** model or Fortran will usually manifest as Java exceptions.

### Creating a Jar File

1. If you ran the code successfully, this means you are ready to build your own JAR. Congrats!
2. In the main menu, open File, and click Export.
3. In the Export window, click Java and then select Runnable JAR file from the dropdown menu.
4. In the Runnable JAR file window, select your launch configuration that you created above. Create an export destination filename in the ./Study/Outputs folder.
5. In the Library handling section, select Package required libraries into generated JAR and click Finish.
6. You should see a new JAR file in your ./Study/Outputs folder.

Note: The JAR file you have just created is using the version of **DSM2** you linked it to, and the JDK version it was built with.

## Running the model using the prebuilt JAR file

This section will guide you on how to run an **ePTM** v2 instance using the prebuilt JAR file. Note that the JAR file that you get with the ePTM Git repo uses **DSM2** v8.1.2 or 8.0.6 and needs JDK 1.8 to work. You must have already installed Anaconda and created a Python 3.x environment.

1. From the scripts folder in the cloned Git repo, copy the runEPTMBasic.py file to the ./Study/Scripts folder.
2. In this file, modify the ALL CAPS text within the parenthesis to point to the correct locations of the specified files and the ./Study/Configs folder.
3. Open a Conda terminal in the folder where you copied the runEPTMBasic.py file.
4. Navigate to the folder where the runEPTMBasic.py file has been copied into.
5. Change to the Python 3.x environment by typing conda activate your\_environment\_name.
6. Enter python runEPTMBasic.py. If everything was correct until now, you should see **ePTM** v2 start to run. Congrats!

*Note: The procedure to run* ***ePTM*** *using the JAR you created above is exactly the same.*

## Animating the ePTM output

This section will guide you on how to generate a trajectory history HDF5 file from the animation binary file you would have created if you used the animation output as well.

1. From the scripts folder, copy the runAnimEPTM.py file into the ./Study/Scripts folder.
2. In the runAnimEPTM.py file, change PATH\_TO\_ANIMATE\_EPTM\_JAR to the absolute path to the animateEPTM.jar file.
3. In the runAnimEPTM.py file, change PATH\_TO\_FOLDER\_CONTAINING\_ANIMATION\_OUTPUT to the absolute path to the animation binary file. Don’t use / at the end here.
4. In the runAnimEPTM.py file, change NAME\_OF\_ANIMATION\_OUTPUT\_BINARY\_FILE to the name of the animation binary file.
5. Open a Conda terminal in the folder where you copied the runAnimEPTM.py file.
6. Navigate to the folder where the runAnimEPTM.py file has been copied into.
7. Change to the Python 3.x environment by typing conda activate your\_environment\_name.
8. Enter python runAnimEPTM.py. If everything was correct until now, it will take a while and nothing will appear on screen, but at the end, you will have a HDF5 in your output directory containing the particle histories. Congrats!