NeuroManager User and Programmer's Guide – 0.985 Addendum ("OneCompile" Upgrade)

David B. Stockton

September 21, 2016

1 Introduction

This addendum is a supplement to the NeuroManager 0.961 User and Programmer's Guide and to the subsequent 0.980 Addendum. This Addendum describes the conversion of NeuroManager from doing multiple MATLAB compilations (one per SimMachine) to doing one single MATLAB compilation and distributing it to all SimMachines. The resulting code is slightly less flexible (since the previous version could handle multiple versions of MATLAB) but simpler, easier to troubleshoot, faster setup, and has the need for only one MATLAB compile-capable license.

Although NeuroManager handles other types of compilations, the only compilation we will refer to in this document is a MATLAB compilation of MATLAB *.m files to produce an executable that can run license-free using the MATLAB Compile Runtime. As a result, for the rest of this document we will reduce the term "MATLAB Compilation" to "compilation" only.

2 Current deficiencies

- The LINUX host modifications have not yet been introduced.
- The GPU examples are not yet updated.
- Stampede has gone to a two-factor authentication approach and we are in the process of working things out with them.

3 Summary of Changes and Additions

Here is a list of changes and additions that appear in the new version. Elaboration appears in subsequent sections.

1. The user now sets the simulator type just once, overtly.
2. MATLAB compilation is now done overtly, just once. This requires that all machines have the same MCR version as the MATLAB compiler used.
3. The user specifies the MATLAB compiler machine overtly.
4. The Xcompilation approach has been completely removed.
5. Addition of a machine no longer includes Simulator type in the parameter list.
6. Construction of the machine set no longer specifies the Simulator type in the parameter list.

7. The TestCommunications() method has been extended with compiler/MCR compatibility testing and is now called **verifyConfig()**.
8. All file uploads have been redone and remote file movements have been redone so that only one upload is required per machine, rather than per simulator.
9. The workflow stages associated with file uploads have been replaced by simpler, clearer ones called by the SimMachine constructor.
10. A new remote directory, ModelRepository, has been added at the Machine Base directory and is used as a source of unmodified model files.
11. The SimulationCommon remote directory has been eliminated.
12. A new remote directory, *model*, has been added at the Simulation Base and is used for a simulation's use for model file access and modification.
13. All Simulator files are replaced for each Simulation.
14. We have added some safety features for use with remote compilation and simulation to help eliminate the risk of unwanted file destruction.
15. We have changed the way that files for compilation and upload are discovered within the Simulator hierarchy.
16. All examples have been upgraded to reflect the new compilation and simulator class changes, including the XML examples.
17. Running the XML examples has been simplified by restructuring using a new Core file, ProcessNM-SessionMLFile.m.
18. The XML schema and stylesheet demonstration files have been upgraded to support cloud servers, wisps, and the one–compile approach.
19. The MATLAB status webpage has been enhanced to show the MATLAB Compilation phase.

4 Previous Design

The previous design of NeuroManager left it up to the first Simulator on a SimMachine to do the compilation for all the Simulators on a SimMachine. Each SimMachine used a separate MATLAB compilation. Simulator construction was done in two parts, the first part being the compilation and the second Simulator file construction and file distribution. For speed all Simulator compilations were done in parallel, either on the SimMachine itself or on a so-called "cross-compilation" machine (in case the SimMachine didn't have a compiler and license available). This approach allowed NeuroManager to use whatever compilers were available, and the output reports showed what which MATLAB version had been used to produce each simulation. The MCR version of a machine had to match the compiler in use on that machine, whether local compilation or cross-compilation.

This approach, however, turned out to be of dubious value, since 1) the user could install any MCR for free so there was no need to mix MCR versions, and 2) there is obvious advantage to using the same compiler version for all simulations. In addition, 3) multiple compilations increased setup time and complexity while the slowest compiler would slow down NM progress anyway, and most of all, 4) if cross-compilation were involved, as many separate licenses were required as the number of SimMachines being cross-compiled.

A single–compile system was desirable, which would upload all files to be compiled to the machine with the compiler, then distribute the compilation products to all SimMachines. Although this would require the MCRs to be identical on all machines, this would be an acceptable, even preferable necessity because it would allow simpler operation and clearer troubleshooting.

5 New Design

Our new design performs only a single compilation and expects the corresponding version of the MCR to be available on all SimMachines. It removes the tasks of pre-compilation, compilation, and post-compilation from the Simulator object and gives it to the NeuroManager object, which places the resulting products in specific places in the host's MachineScratch subdirectories. The Machine constructors upload all files to the remote Machine's SimulatorCommon and (new) ModelRepository directories. The Simulator constructors, then, simply copy from those directories rather than perform the compilation/upload themselves. Figure 1 shows the new movement and treatment of the various file types within the NeuroManager host/remote system. This approach to compilation and file movements is cleaner, faster, and more transparent.

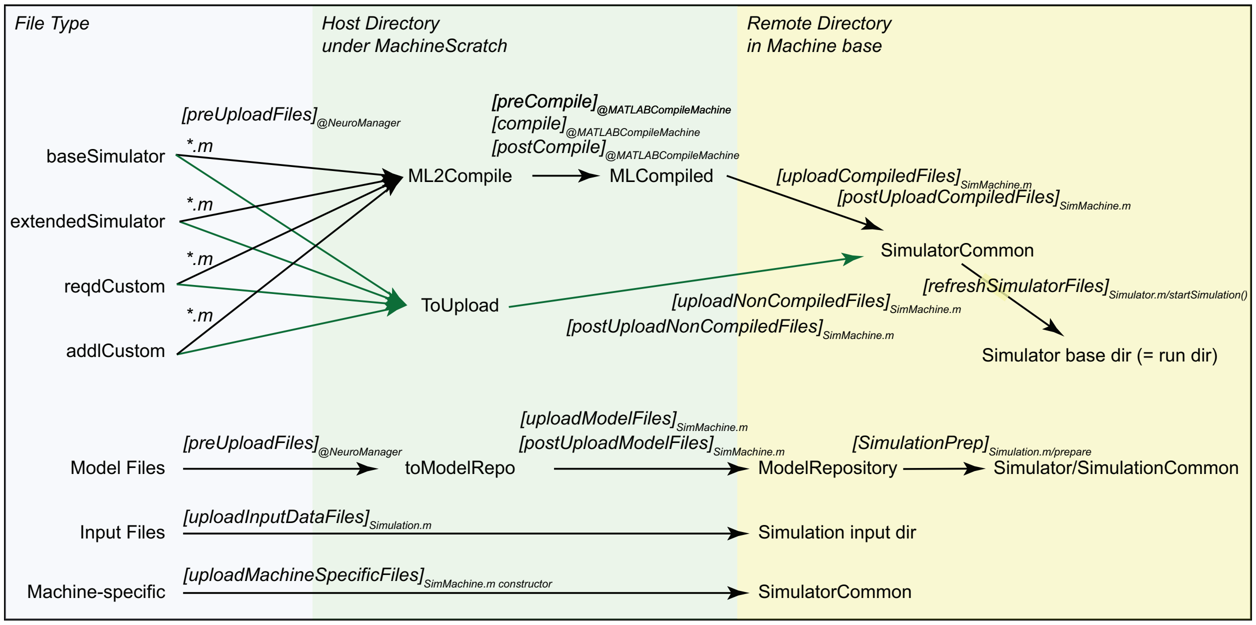
File Type
Host Directory under MachineScratch
Remote Directory in Machine base
baseSimulator
extendedSimulator
reqdCustom
addlCustom
*.m
*.m
*.m
*.m
[preUploadFiles]@NeuroManager
[preCompile]@MATLABCompileMachine
[compile]@MATLABCompileMachine
[postCompile]@MATLABCompileMachine
ML2Compile
MLCompiled
[uploadCompiledFiles]SimMachine.m
[postUploadCompiledFiles]SimMachine.m
SimulatorCommon
ToUpload
[uploadNonCompiledFiles]SimMachine.m
[postUploadNonCompiledFiles]SimMachine.m
[refreshSimulatorFiles]Simulator.m/startSimulation()
Simulator base dir (= run dir)
Model Files
[preUploadFiles]@NeuroManager
toModelRepo
[uploadModelFiles]SimMachine.m
[postUploadModelFiles]SimMachine.m
ModelRepository
[SimulationPrep]Simulation.m/prepare
Simulator/SimulationCommon
Input Files
[uploadInputDataFiles]Simulation.m
Simulation input dir
Machine-specific
[uploadMachineSpecificFiles]SimMachine.m constructor
SimulatorCommon

Figure 1: Upload File Movements. Files in the categories on the left (light blue background) are collected through class action and transferred into host directories (light green background) perhaps with synthesis or modifications, then transferred up to each remote resource (light yellow background) for use by the Simulators. Subscripts show the location of the operation in the code. Most items in square brackets are workflow stages. See the code for more detail.

In conjunction with those changes we removed the "MATLABCompileMachine" class from the Machine tree primary line, paving the way for porting NeuroManager into languages which do not have the MATLAB compilation requirement. See Figure 2 for the new Machine class tree.

In the process, we have renamed the various upload file categories and formed a clearer set of definitions of each (see Table 1). We have also distributed files more powerfully in the Simulator class hierarchy by using nested constructors so that now files can be picked up throughout the class inheritance line, thus allowing for better subclassing of user customization. The SimpleSpike01 example shows a simple example of this; since that class has unique contributions to the hoc and mod file lists, it adds them to its own private lists then extends the getHocFileList(), getModFileList(), and getModelFileList() methods to append the private lists to those of the superclass, SimNeuron. This approach allows the user to develop model trees that specify model files at any point in the tree.[1]

[1]Please refer to the code and to the User Guide for more details.

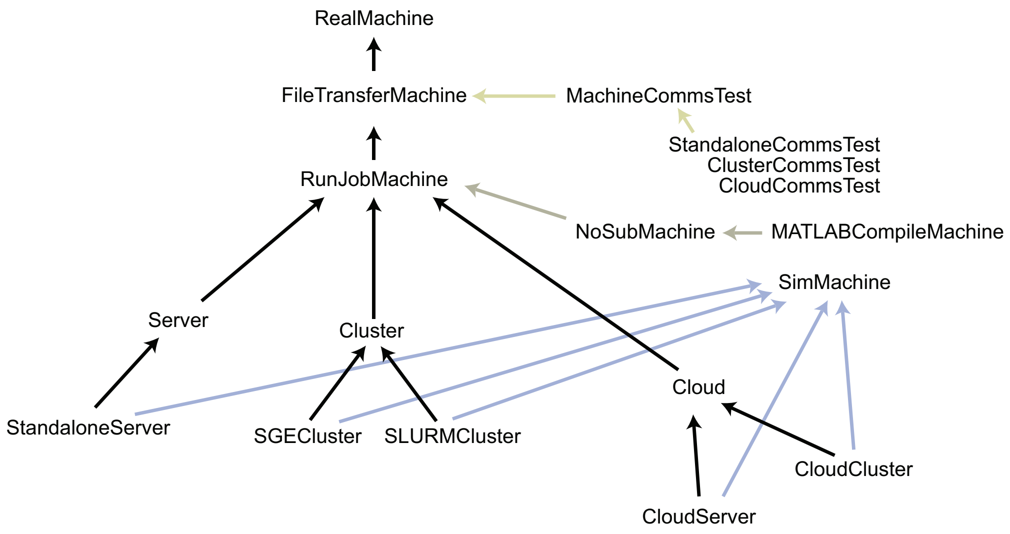

Figure 2: Machine Class Hierarchy. The MATLABCompileMachine class has been moved out of the main inheritance line to facilitate future porting.

Type	New name	Example file	List location	Description
stdUploadFiles	baseSimulatorFileList	runSimulation.m	Simulator.m	The basic files that all Simulators need to be compiled into the remote executable.
addlStdUploadFiles	extendedSimulatorFileList	createPythonNrnivsh.m	Simulator m–files in class Hierarchy	Simulator type specific tiles (both m-files and non-m-files) that all Simulators of that type must have uploaded.
custUploadFiles	reqdCustomFileList	UserSimulation.m	custom files location	Just userSimulation.m with possibility of additional files in future
addlCustUploadFiles	addlCustomFileList	PySim.py	custom files location	Simulator type-specific files
Model files		*.hoc, *.mod, *.dat, *.txt, *.mat, etc	model files location	Model files that are not m-files
Simulation input files		*.dat, *.txt, *.mat, etc	custom files location	Simulation–specific files that serve as data input to the simulation in question
Machine–specific files		MachineData.dat	constructed on-the-fly	Saves the machine config information in a file and uploads it so that remote code has access to machine–specific directories, etc

Table 1: File categories. “Type” indicates the old names for the category; “New name” the names used in this upgrade and beyond. “List location” indicates where the files are named (so that NeuroManager knows to work with them).

We have separated Simulator Type definition from machine set definition with a new NeuroManager class method called setSimulatorType(), and moved the compilation process out of the constructMachineSet() method. These changes speed the user through the setup process, since they allow the user to debug the compilation before any other work is done. Setting the SimulatorType also simplifies the addMachine... commands, which is useful since specifying the type in each command (as was the case previously) implied that each could have different type, which was never true.

A result of possibly dubious value is that the new flow also allows the user to completely redo the Simulator type, compile, and rebuild the Simulators, all within a single NeuroManager object lifetime (this feature has not yet been tested).

We also have instituted checks to ensure that the compiler version used matches the MCR specified by the user for each machine, another benefit of separating compilation from MachineSet construction. The compiler version is determined by directly invoking the compiler that the user specified in the info file named in the setMLCompileServer() method, also new. We have made this easier by adding a new entry in image descriptions called mcrVer, which is the numeric identifier for the MATLAB version (such as 8.1) found at `http://www.mathworks.com/products/compiler/mcr/`. The checks are done in the testCommunications.m method, and so we have renamed that method to verifyConfig(). The return value of verifyConfig can be tested to see if there was a problem with the verification, giving the user the freedom to remove automatically any Wisps that had already been constructed.

Finally, we have instituted some safety features associated with both MATLAB compilation and the use of the remote. We now require that the remote compilation directory as specified in the Info file be named "NMComp" and have no subdirectories. We require the remote work directory as specified in the Info file be named "NMWork". These requirements reduce the possibility that the compilation or work directories could be set incorrectly, resulting in the loss of data. We still however, suggest that the user use remote resources that are separate from the rest of his/her work in order to provide best possible safety.

6 Workflow changes

Due to the extensive changes in approaches to upload and compilation, the related workflow stages have also changed.

The **Pre Upload Files** workflow stage now has an additional NeuroManager aspect in addition to its machine and simulator aspects, which is used before any "official" machines or simulators are constructed, primarily for precompilation/preupload file splitting and relocation on the host.

The workflow stages **Upload Standard Files**, **Upload Custom Files**, **Post Standard Files Upload**, and **Post Custom Files Upload** have been replaced. Their work is accomplished in the machine constructors through simpler and more–aptly–named stages called **Upload Machine Specific Files**, **Upload Compiled Files**, **Post Upload Compiled Files**, **Upload NonCompiled Files**, and **Post Upload NonCompiled Files** (see SimMachine.m).

The **Pre Compile**, **Compile**, and **Post Compile** workflow stages are now associated with the NeuroManager–class–run compilation, not with all machines or simulators, so they are localized in one area and are not class–dependent as before.

The **Upload Model Files** and **Post Upload Model Files** workflow stages are now completely machine–based and called by a machine's constructor. Currently **Post Upload Model Files** is a no–op.

preRunModelProcPhaseH no longer moves model files into a Simulation's input directory. Instead, all model files are copied into the Simulation's *model* directory before the Simulation runs and all model operations are performed there. The Simulation can move model files when it runs if it needs to using this stage or the **P** and **D** stages. For example, hoc files are moved to *model* before this stage, but this stage moves the hoc files into the simulator base directory, which is where the NEURON SimCore runs. This approach facilitates the use of published models that assume the hoc files are in the run directory.

7 Conclusion

The new changes make NeuroManager easier to set up, use, and troubleshoot. We hope the new changes will encourage you to use the software. We should remind you that this is, in effect, demonstration code and is not industrial–strength production code. We simply do not have the resources to do that. In particular, our compiler/MCR, SimCore, and other compatiblity checks are not very robust (though still helpful). However, we anticipate that you will agree that the time spent configuring and learning NeuroManager will pay for itself very quickly.