METAMODEL MANAGER

User's Manual – Version 1.1

Chicago Zoological Society

Inspiring Conservation Leadership

METAMODEL MANAGER

Manual written by:
Becky E. Raboy, Robert C. Lacy, Taylor Callicrate, and Caroline M. Lees

Software written by:
JP Pollak and Robert C. Lacy

Development and distribution of the *METAMODEL MANAGER* software and this accompanying manual was supported by the Species Conservation Toolkit Initiative and a Research Coordination Network grant (DEB-1146198) from the US National Science Foundation.
Thanks are extended to the many colleagues within the IUCN SSC Conservation Planning Specialist Group, who helped in the conceptualization and development of the program. A special thanks to Phil Miller, Philip Nyhus, and Sara Zeigler, in addition to the developers and authors of this manual, for hours of brainstorming, head-scratching, and testing of the program.

Cover design by: Stephanie Raboy

Bacteria photo credit (upper right photo on cover): photo modified from Flickr contributor kaibara87 (http://www.flickr.com/photos/kaibara/2234750993/) according to the creative commons.

Wolf tracks photo credit (lower left photo on cover): NPS Photo – wolf images #16190.

Citation of this manual:
Raboy, B.E., Lacy, R.C., Callicrate, T., & C.M. Lees. 2018. *METAMODEL MANAGER. User's Manual. Version 1.1.* Chicago Zoological Society, Brookfield, Illinois, USA.

Citation of the software:
Pollak, J.P., & R.C. Lacy. 2020. *METAMODEL MANAGER. Version 1.0.6.* Chicago Zoological Society, Brookfield, Illinois, USA.

Chicago Zoological Society
Inspiring Conservation Leadership

CONSERVATION BREEDING SPECIALIST GROUP
Planning a Future for Wildlife

Contents

The Species Conservation Toolkit Initiative (SCTI)

A partnership to ensure that innovations needed for species risk assessment, evaluating conservation actions, and managing populations are developed quickly, available globally, and used effectively.

The past 30 years have seen major advances in the use of computer modeling to address the growing complexity of species conservation issues. The Species Conservation Toolkit currently includes the programs *VORTEX*, *OUTBREAK*, *PMX*, and *METAMODEL MANAGER*, which collectively have supported conservation decision-making for thousands of projects around the world.

Under the guidance of expert users, the toolkit allows conservation decision-makers and practitioners to assess risks to wildlife, evaluate conservation options, and guide active population management with greater efficiency and realism. It is essential that this toolkit be maintained and expanded to meet new needs and respond to new opportunities. To this end, a partnership of organizations has formed to foster lasting support for this suite of tools, ensuring its continued evolution, global distribution, and ongoing user-support by technical experts.

Thanks to Chicago Zoological Society, Species360, Smithsonian Conservation Biology Institute, Association of Zoos and Aquariums, European Association of Zoos and Aquaria, Wildlife Reserves Singapore, Auckland Zoo, Copenhagen Zoo, Cincinnati Zoo, Chester Zoo, Zoological Society of London, SOS Rhino, San Diego Zoo Global, Texas State Aquarium, Oceans Initiative, Raincoast Conservation Foundation, Living Desert, Saint Louis Zoo, Woodland Park Zoo, and Seattle Aquarium for helping to keep these conservation tools (and species!) alive and evolving.

Dr. Robert Lacy (Chicago Zoological Society)
Dr. Jonathan Ballou (Smithsonian Conservation Biology Institute)
Dr. Onnie Byers (IUCN SSC Conservation Breeding Specialist Group)
Dr. Taylor Callicrate (Species Conservation Toolkit Initiative)

Chapter 1. Introduction

The history and philosophy of the metamodel approach

The dynamics of wildlife populations are highly complex due to the diversity of influences and interactions that drive the ecological systems within which they reside. Relevant factors can include: species life history attributes; inter-species interactions; dispersal and social behaviors; genetics; human-induced stresses such as novel pathogens, invasive species, and the collapse of inter-connected communities; and shifts in the global environment.

Analytical tools for population viability analysis (PVA) (e.g., *VORTEX*) allow for consideration of deterministic and stochastic threats. However, these are typically limited to demographic and environmental stochasticity, genetic decay, and habitat loss. The inclusion of other factors is often possible, but requires a sometimes highly simplified characterization of both the relevant interactions and their likely variability of impact.

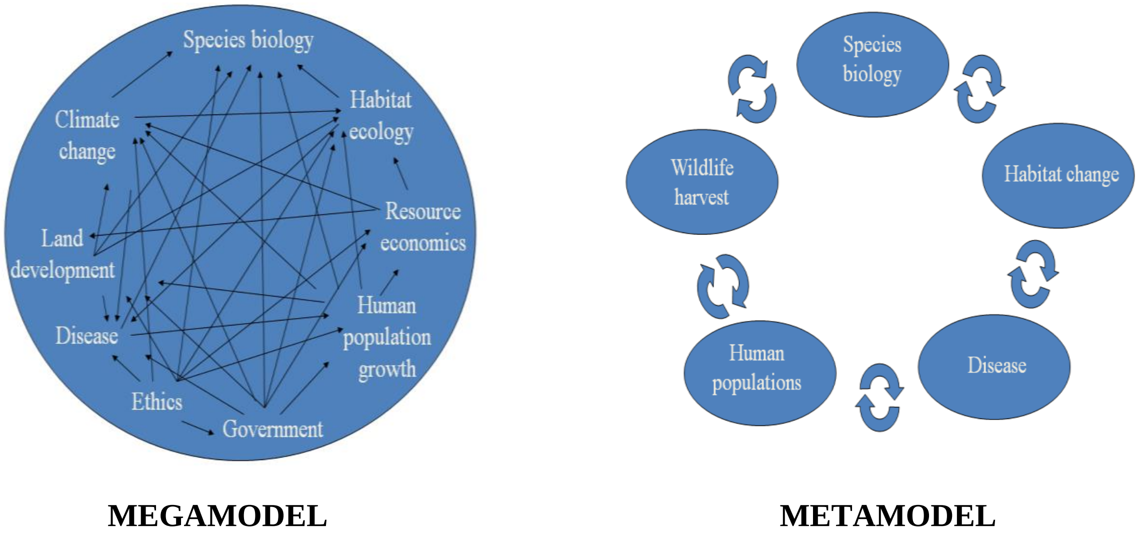


Figure 1. Conceptual comparison of a megamodel and a metamodel.

The value of extending PVA analyses to include a multi-disciplinary and multi-dimensional treatment of relevant threats is recognized by many. Advocates of this approach have targeted modeling and process tools as primary vehicles for study. Specifically, researchers have developed a "metamodeling" approach, through which discipline-specific models, each representing a component of an overall system, are linked (i.e., the outputs of one model become the inputs to another) to reveal emergent properties of multi-dimensional interactions. In the right circumstances, the approach of using a suite of inter-linked "sub-models" (a metamodel), can help by:

1) overcoming challenges related to the use of a set of independent models that lack dynamic interactions and which, therefore, underestimate or ignore important synergies or cumulative effects.
2) providing an alternative to the difficulty, resources, and model compromises potentially required in the development of a single, massively complex "megamodel" (see Figure 1.)

For example, a metamodel that combines a wildlife population sub-model, a habitat change sub-model, and an epidemiological sub-model of infectious disease can be used to examine how changes in habitat connectivity can alter dispersal, thereby altering disease spread and source-sink dynamics within a metapopulation. It would be difficult to assess the dynamics of a system like this from any single perspective; the population model would be likely to predict that habitat fragmentation would destabilize the system, whereas the disease model would predict that fragmentation would protect local populations from infection. The balance of these trade-offs in even this simple example would be difficult to predict without a metamodel (see Chapter 3.)

A metamodeling approach requires a central facilitator program to control the sharing of information between models. *METAMODEL MANAGER* has been developed to fulfil this role.

Users are encouraged to read Lacy *et al*., 2013 (http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0084211) for a more comprehensive treatment of the features, advantages, and uses of metamodels. Indeed, it probably is not wise to proceed any further in this manual or with your use of *METAMODEL MANAGER* until you have read that background paper!

What is *METAMODEL MANAGER*?

METAMODEL MANAGER is a software interface which controls the way in which a suite of "sub-models" interact as they simulate the dynamics of a wildlife population in the context of its many influences (see Figure 2). Each sub-model describes the properties and dynamics of a specific component of the system of interest. The advantage of this approach is in enabling the user to harness the unique capabilities of each sub-model program while exploring the implications of interactions between them, without the need to construct new and potentially cumbersome "megamodels" (see Figure 1).

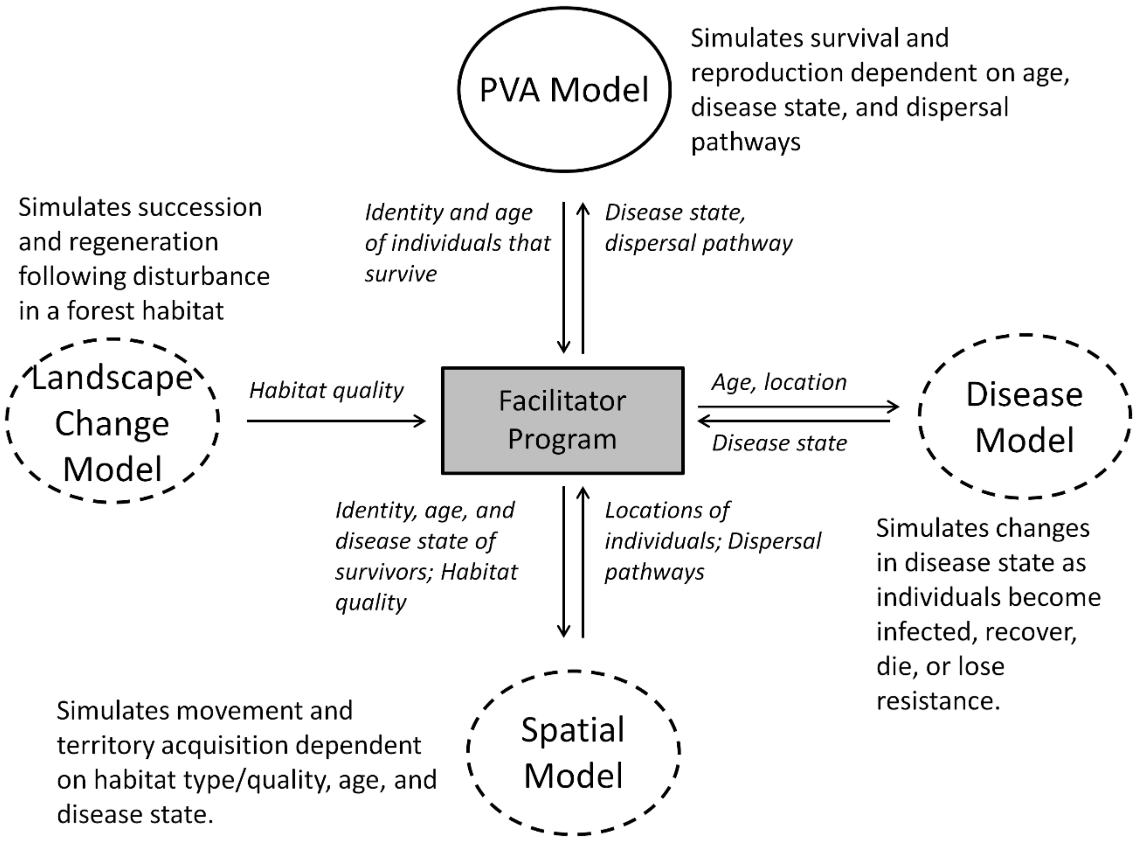

PVA Model
Simulates survival and reproduction dependent on age, disease state, and dispersal pathways
Simulates succession and regeneration following disturbance in a forest habitat
Identity and age of individuals that survive
Disease state, dispersal pathway
Landscape Change Model
Habitat quality
Facilitator Program
Age, location
Disease state
Disease Model
Identity, age, and disease state of survivors; Habitat quality
Locations of individuals; Dispersal pathways
Simulates changes in disease state as individuals become infected, recover, die, or lose resistance.
Simulates movement and territory acquisition dependent on habitat type/quality, age, and disease state.
Spatial Model

Figure 2. Illustration of the role of the facilitator program *METAMODEL MANAGER*, in directing the flow of information among sub-models within a multi-disciplinary metamodel (from Lacy *et al.*, 2013).

In essence, *METAMODEL MANAGER* acts as the facilitator of a directed conversation between the sub-model programs:

- it gives each sub-model program a turn to "speak", that is, it controls the order in which the various sub-model programs operate within the wider simulation;
- it passes information back and forth between sub-models as "open data" which can be modified by each before being passed on;
- it determines whether data are private or shared, that is, it can choose to accept or not accept data modified by particular sub-model programs and can choose to hide data from or share data with others;
- it functions as a translator, transforming data outputs from one sub-model into the appropriate input for another;
- it does not analyze or present results; this is left to the sub-model programs.

Each metamodel requires at least one "System Model" (such as *VORTEX*) to establish a representation of a wildlife population of interest. To this is added one or more "Modifier Programs" designed to tweak or otherwise act on the system created (see Figure 2).

Like some of the programs it works with, *METAMODEL MANAGER*'s function is to serve as a predictive modeling tool to determine likely population viability outcomes for a given management scenario, or to compare outcomes across multiple, alternative scenarios. The program runs iteratively, and its results are generated in the form of means or probabilities within the sub-model programs. As for other PVA modeling tools, results can assist the transparent development of conservation goals and can help direct or prioritize conservation action.

METAMODEL MANAGER can be used for analysis by individuals or in the context of an inter-disciplinary conservation planning workshop. For examples of its use see Lacy *et al*., 2013; Shoemaker *et al*., 2014; Wells *et al*., 2015.

CAUTION: Program users and those interpreting results may expect that as model complexity increases, the overall predictive accuracy should increase. However, the opposite could also occur, particularly where uncertainty surrounding the added complexity is high. Sensitivity testing can be used to indicate which variables are key to population outcomes and therefore, where in the program accuracy is most important. Sensitivity analyses can be carried out within sub-models or at higher levels (by adding and subtracting sub-models). Arguably, the success of the *METAMODEL MANAGER* program for conservation biology is not in the accuracy of the results generated *per se*, but in how well the results are interpreted, keeping in mind its aforementioned limitations.

When to use *METAMODEL MANAGER*

In many instances a simple PVA model, properly constructed and interpreted, will provide sufficient direction for effective conservation action. Adding complexity will not always be desirable; it may increase inaccuracy and will usually increase time and costs without necessarily changing the outcome. However, for reasons described above there will also be instances where increased model complexity will lead to greater insights and, as a result, better guidance on appropriate intervention.

Metamodels are most likely to be of value where two systems are tightly coupled (e.g. a predator and its primary or only prey species), or where there are known feedback loops which could cause an otherwise obscured escalation of vital rates. The following steps (using *VORTEX* as the potential System Level Model Program) may be useful in assessing whether to pursue a metamodel approach:

1) Set up a PVA model for the focal species using the best available information.
2) Sensitivity test thoroughly to identify key parameters affecting demographic performance. Check sensitivity to both variability and extremes.
3) Where either high variability and/or extremes in these key parameter values are shown to be influential AND where variability or extreme values are expected to result from the dynamics of other complex systems for which models could feasibly be constructed, consider a metamodel approach.

About the manual

The purpose of this manual is to introduce to potential users the principles of the metamodeling approach and the role of *METAMODEL MANAGER*, and to provide sufficient guidance to allow first-time users to build, run, and interpret a simple metamodel. The manual does not explain how to set up scenarios in the sub-models brought together by the *METAMODEL MANAGER* interface. It is assumed that those wishing to use this manual will already have experience with one or more of the programs for which *METAMODEL MANAGER* was developed (e.g., *VORTEX* or *OUTBREAK*). Those without this experience should first consult the software manuals and associated resources for these programs (available at https://scti.tools/downloads/)

Please send bug reports, suggestions and any requests for additional information to help@scti.tools. We cannot guarantee that all comments and requests will be answered, but responses will be provided when possible. We do, however, value all feedback.

Chapter 2. Quick start guide

It is strongly recommended that you read the complete Manual before attempting to use *METAMODEL MANAGER* for the first time. This section is designed to provide a memory prompt and check-list for those who have some prior experience with *METAMODEL MANAGER* and for those who have read the Manual, completed the Tutorial, and are ready to begin building their own metamodel.

Step 1. Sketch a 2-dimensional representation of your metamodel.
Identify the proposed sub-models, clarify what they will be expected to do, the information they will need to share, and the sequence through which that information will be passed from one sub-model to the next (see Box 1).

Step 2. Build and test your sub-models.
See the relevant software manuals for information (e.g. for *VORTEX* and *OUTBREAK* go to https://scti.tools/downloads/). Set up Global, Population, and Individual State Variables to allow data sharing among components of the metamodel (see Appendix II)

Step 3. Set up your file structure.
Create a new subfolder for your metamodel project within your MMMProjects folder. Copy all sub-model input files to this location, renaming them for clarity (see Folder and file management, Chapter 4)

Step 4. Open *METAMODEL MANAGER* and start a project.
Double-click the *METAMODEL MANAGER* icon on your desktop. From the opening screen select **Begin a New Project.**

Step 5. Select components for inclusion in your metamodel (see Chapter 5):

- **Choose a System Level Model program:** from the "Setting System Level and Modifier Programs" screen, underneath the heading "Population/System Models" and from the column headed "Model", select a System Model from the list (e.g. *VORTEX* or *OUTBREAK*).
- **Choose your System Level Model project:** from the column headed "Project File" click on the "Browse" button and select the relevant scenario from the appropriate System Level Model project file, which will be in the location you created for it within the MMMProjects folder.
- **Add Modifier or System Level Models:** using the buttons running along the bottom of the screen, add or remove System Level or Modifier Models, clearly identifying for each the relevant project file and scenario. Continue until all required programs are included.
- **Check shared variables**: on the next screen, "DataSet Definition", check that shared variables are as you intended before clicking "Next".

Step 6. Order Model Steps.
Use this screen to sequence and synchronize sub-model activity.

- Use **Up** and **Down** to arrange sub-models in the required sequence of operation.
- Use ± **Group** to group sub-models that should pass data back-and-forth for a specified number of cycles before passing it to another sub-model.
- Use ±**Break** to impose breaks in the simulation process during which the user may see, change, or save the interim results.
- Use ±**Input** to import values to override those generated in the modeling process.
- Use ±**Output** to create a file of all data on the population and its individuals, each time step.
- Use ±**Evaluator** to transform state variables describing the system, populations, or individuals.

Edit the column headed **Time Steps** to synchronize sub-model program cycles (e.g. by default *OUTBREAK* completes 365 cycles during the course of a year; *VORTEX* completes 1 – this should be reflected in the **Time Steps** column)

Use the **Total Years** field to set the number of years over which the simulation will run (default is 10 years). Use the **Iterations** field to set the number of times the metamodel simulation will be repeated. Hit **Next** to move to the next screen.

Step 7. Simulation options (see Chapter 7).
Choose the variables you would like to see plotted and how you would like them summarized, or accept the defaults (i.e., population size(s) plotted in "real time"). Click **OK** and **Save** your project before continuing.

Step 8. Simulate.
Click **Simulate.** Once the simulation is complete, use the buttons at the base of the screen to **Print Graph** or to **Save Graph and Data.**

Step 9. View results.
Detailed results about population dynamics can be accessed by opening the relevant System Level Model program (e.g., *VORTEX*) and accessing the relevant program file saved within the MMMProjects folder.

Chapter 3. Case study

The case study described in this chapter is drawn from Lacy *et al*., 2013. It illustrates the ability of a metamodel to reveal ecological dynamics that would be difficult to predict from individual models. In the hypothetical case presented, an infectious disease is introduced into a metapopulation consisting of 10 subpopulations with total abundance N = 250. Annual dispersal rate between subpopulations is varied (0%, 1%, 2%, 4%, and 10%), to explore the effect of metapopulation connectivity on viability in the presence of infectious disease.

In the metamodel, population viability was predicted within the PVA software *VORTEX*, and disease processes were simulated within the epidemiological model *OUTBREAK*. These sub-models were linked through *METAMODEL MANAGER*. Population viability was simulated under three model structures:

A. **Without disease**. In this scenario the metapopulation was simulated in *VORTEX* alone and included the full range of stochastic elements provided for in the program: inbreeding depression; individual variation in female breeding rates; annual environmental variation in mortality rates; random variation in sex-ratio; and demographic stochasticity (variation due to binomial sampling from constant reproductive and survival rates).
B. **Disease driven**. In this scenario, disease processes were included using *OUTBREAK* as a modifier model, and their effects emphasized by reducing the stochastic elements in *VORTEX* to only variation in sex-ratio and demographic stochasticity (which cannot be disabled).
C. **Disease plus all stochastic factors**. In this disease-inclusive metamodel, all *VORTEX* stochastic elements were included.

The results of these trials are illustrated in Figure 3.

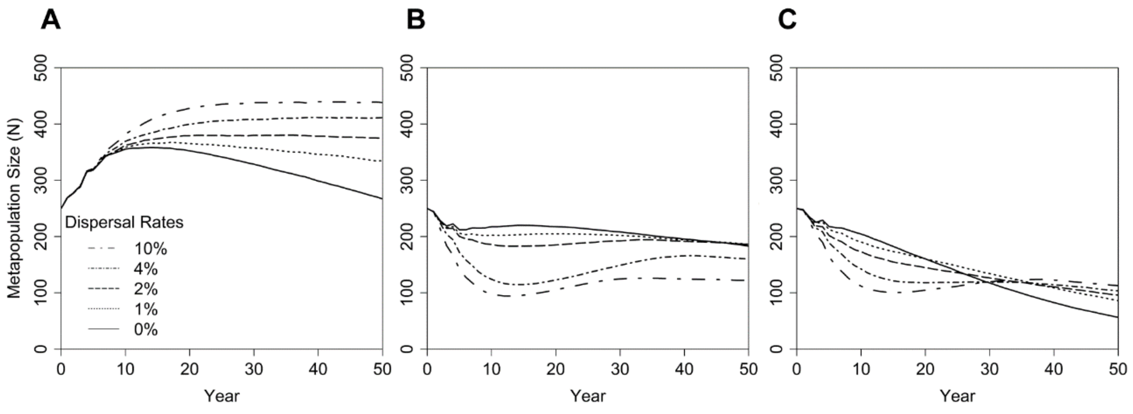


Figure 3. Comparison of three model structures: A. Without disease; B. Disease-driven; C. Disease plus all stochastic factors; from Lacy *et al*., 2013

In (A), higher rates of dispersal increase growth and stability of the metapopulation because stochastic effects in local subpopulations are dampened. When disease is introduced but stochasticity dampened, as in (B), higher rates of dispersal depress population size because of the faster spread of disease. Finally, when stochasticity, disease and dispersal are considered in (C), higher dispersal initially reduces population size because of the faster spread of disease. In later years, disease is largely eliminated from the system, and higher rates of dispersal stabilize the population against stochastic fluctuations. During a few years in the middle of the simulation, disease and stochastic processes are equally important, and intermediate rates of dispersal lead to the highest population size.

Differences in the impacts of population connectivity on metapopulation dynamics in the absence (A) or presence (B, C) of disease illustrates the opposing forces acting on sub-divided populations: stochastic processes threaten small, isolated populations and are countered by dispersal while infectious disease transmission threatens inter-connected populations and is blocked by isolation. Both processes impact many real populations, but the balance between these forces would be difficult to predict from the application of standard PVA methods for population modeling, from standard epidemiological models of disease, or even from the examination of both models independently, as illustrated above.

Overall, the balance between positive and negative effects of population fragmentation was dependent on the temporal scale over which metapopulation dynamics were projected as well as the specific population and disease parameters applied in the metamodel. Changes to these could change metamodel predictions regarding the optimal range of dispersal rates. Metamodels will perhaps be most useful for investigations of the compound threats facing specific populations in specific circumstances precisely because outcomes are determined by complex interactions that are sometimes synergistic and sometimes countervailing.

For further details of this study users are directed to Lacy *et al*., 2013. The *VORTEX* input file that includes all demographic rates, the *OUTBREAK* disease specification file, and the *METAMODEL MANAGER* control files used in this example are placed into a Samples subfolder during installation of *METAMODEL MANAGER*.

Chapter 4. Installation

System requirements

METAMODEL MANAGER is compiled to run on any version of the MS Windows (Microsoft Corp. Redmond, Washington, USA) operating system (although compatibility with all current and future versions of Windows is not guaranteed).

METAMODEL MANAGER does not itself require much computer memory or processor speed to run. It can run on either a 32-bit or 64-bit processor. However, often the models linked through it will work best (or only) if the computer has ample RAM and a fast processor. (We suggest a minimum of 4 GB of RAM and an i5 or newer processor. You may need more RAM depending on the complexity of your model).

METAMODEL MANAGER is copyrighted, so you cannot legally re-use or adapt the program code within your own software projects. However, the source code is freely available for examination, so that programmers of models to be linked via *METAMODEL MANAGER* can see exactly how it accesses linked models and shares data among them. The code is placed into a SourceCode subfolder of the program folder where *METAMODEL MANAGER* is installed. However, little attempt has been made to make the source code easily readable by others (the emphasis in coding was instead on creating a powerful and highly flexible program), and some third-party software tools would be required to recompile *METAMODEL MANAGER*. The program was written in the C# language, compiled with MS Visual Studio (Microsoft Corp. Redmond, Washington, USA), and uses some user interface controls from ComponentOne Studio WinForms (Grape City Inc., Pittsburgh, Pennsylvania, USA).

Installation

METAMODEL MANAGER is available for free download online at https://scti.tools/downloads/. The installation (MMMInstallation.msi) automatically configures necessary folders, includes the program libraries necessary to run *VORTEX* and *OUTBREAK* models within *METAMODEL MANAGER*, and includes the *SPATIAL* and *PATHHISTORY* programs.

The installation automatically creates a program short-cut on your desktop.

It is also highly recommended that you download and install the latest versions of the full *VORTEX* and *OUTBREAK* programs (also available at https://scti.tools/downloads/). These programs will be needed in order to create *VORTEX* and *OUTBREAK* projects for running within a metamodel.

CAUTION: *METAMODEL MANAGER* keeps its own copies of the *VORTEX* and *OUTBREAK* program files and so will use whichever versions were available when it was compiled. Therefore, when installing a newer version of *VORTEX* or *OUTBREAK* for use in a metamodel, reinstall *METAMODEL MANAGER* at the same time, so that it will use the most recent versions of these programs.

Folder and file management

METAMODEL MANAGER facilitates interactions between sub-models built in different programs. In addition to the files generated by these programs under the direction of *METAMODEL MANAGER*, each of these programs (e.g. *VORTEX*, *OUTBREAK*) is capable of generating and storing its own output when run outside the context of a metamodel. A large number of files will be created during the building, testing, running, and refining of both the sub-models and the meta-model. Disciplined management and storage of these files is challenging but critical to effective use of *METAMODEL MANAGER*. To avoid hours of frustration and loss of data, first time users are encouraged to use the following approach to folder and file management:

1) Use the installation defaults.
- Install *METAMODEL MANAGER* in the default Program Files (x86) folder.
- Use the separate C:\MMMProjects folder created during installation for storing *METAMODEL MANAGER* projects.
- Keep the project files for your individual models - e.g. *OUTBREAK*, *SPATIAL*, and *VORTEX* - in the folders created by the *METAMODEL MANAGER* installation for that purpose (i.e. C:\SpatialProjects, C:\OutbreakProjects, and C:\Vortex10Projects). Prior installations of these programs will also have created folders with these names. *METAMODEL MANAGER* installation will not remove any files from these pre-existing folders.

2) Maintain a clear and consistent separation between metamodel and sub-model outputs.
- Prior to running a metamodel, build and test each sub-model within the relevant program framework (e.g. build and test *VORTEX* models within *VORTEX*). Create a sub-folder within the relevant project folder (C:\Vortex10Projects\Prairie_Dog) to store all scenarios relevant to the sub-model. In *VORTEX* and *OUTBREAK*, all outputs from simulations run during the building and testing of that sub-model will be stored within that sub-folder (C:\Vortex10Projects\Prairie_Dog\VOutput).
- Create a sub-folder within C:\MMMProjects for the new metamodel (e.g. C:\MMMProjects\PD_BFF for a project with models of prairie dogs and ferrets).
- Create folders within the metamodel sub-folder for each of the sub-models. Give each folder an informative name to distinguish the sub-models from each other and from the overall metamodel (e.g. C:\MMMProjects\PD_BFF\V_PDog for <u>V</u>ortex <u>P</u>rairie <u>D</u>og sub-model and C:\MMMProjects\PD_BFF\V_BFF for <u>V</u>ortex <u>B</u>lack-<u>F</u>ooted <u>F</u>erret sub-model, both within the <u>P</u>rairie <u>D</u>og – <u>B</u>lack-<u>F</u>ooted <u>F</u>erret MMM project folder).

- Once a sub-model is ready for incorporation into the metamodel, copy the relevant program file (e.g. Prairie_Dog.xml from C:\VortexProjects\Prairie_Dog) and situate it within the sub-model folder you prepared for it in the previous step, renaming the file to reflect its incorporation into the metamodel (e.g. C:\MMMProjects\PD_BFF\V_PDog\ MMM_PDog.xml).
- Outputs for the Prairie Dog component of the metamodel will be stored in C:\MMMProjects\PD_BFF\V_PDog\VOutput and will be viewable by opening the C:\MMMProjects\PD_BFF\V_PDog\ MMM_PDog.xml project file from within the *VORTEX* program.

Should further changes to the Prairie Dog model be needed, they should be made to the original file (stored within the C:\VortexProjects folder) and this file then copied across to the C:\MMMProjects folder and renamed.

CAUTION: it is possible to keep a single sub-model file within, for example, the C:\VortexProjects folder, and direct *METAMODEL MANAGER* to locate the sub-model there. However, any outputs from metamodel simulations may then be inadvertently over-written by opening up and running the sub-model as a standalone simulation from within *VORTEX*. Similarly, it would be possible to maintain two copies of the sub-model simultaneously, one within *VORTEX* and one within *METAMODEL MANAGER*. However, this requires that any changes made are made twice – once in each copy. This is an easy protocol to forget, giving rise to confusion about which version (if either) carries all of the relevant changes. Disciplined annotation of your *VORTEX* projects and informative project and folder names should help reduce problems.

Processing speed

The speed with which *METAMODEL MANAGER* runs will depend on the nature and complexity of the sub-models and the programs running them. To understand the time-limiting steps in the metamodel it is recommended that test models are run initially with fewer iterations, for shorter periods (or fewer time-steps), and with fewer metamodel components.

Factors to bear in mind when the metamodel is running slowly:

- Program tasks that act on every individual, every day or time-step of the simulation will slow things down.
- *MMMACRO* and *EVALUATOR* work that operates on individuals is slow.
- *MMMACRO* is slow in general.
- *OUTBREAK* is slow compared to *VORTEX*; it takes much longer to run a single "year" of the simulation because it is doing daily checks on individuals.
- Initializing and Closing take time if there are many tasks to perform.

Asking *METAMODEL MANAGER* to show plot lines only at the end of the simulation will speed up iterations.

Chapter 5. Building a metamodel: what's involved?

The following section explains the basic components of *METAMODEL MANAGER*-facilitated metamodels, and describes the decisions that must be made and correctly implemented to ensure that these components perform and communicate with each other, in the way intended by the user.

METAMODEL MANAGER components

METAMODEL MANAGER builds metamodels from the following three component types:

- **System Level Models** initialize or define the individuals in the starting population(s), assigning them (as a minimum) an identity, an age, and a gender. A System Level program will often manage population demographics (births, deaths, gender designation, ageing), though this is not a requirement. For multi-species metamodels (e.g. predator – prey), a separate System-level program may be included for each species. *VORTEX* is the System-level program most commonly used with *METAMODEL MANAGER*.

- **Modifier programs or sub-models** may modify characteristics of the system, population(s), or individuals. For example, the epidemiological simulation program *OUTBREAK* may be used to modify the disease status of individuals according to their modeled susceptibility and exposure to a particular pathogen, during a simulated year in the life of the population. Those changes in disease status (e.g., whether individuals have become infected or not) can then be reported back to the System Level program (e.g., *VORTEX*), to direct changes in individual likelihood of dying or breeding. Note that *OUTBREAK* can also be used as a System Level program (to initialize or define the individuals in the starting population), and it is able to manage the demographics of the population as well as simulating disease.

- **Utility programs** perform data transformations outside the system and modifier models. These transformations can function as: 1) translators among the data formats used by different system or modifier models; 2) simple modifier models that apply transitions to an individual's characteristics during a simulation; or 3) a means of acquiring summary statistics from the metamodel. *EVALUATOR* and *MMMACRO* are the Utility programs that travel with, and are most commonly used with, *METAMODEL MANAGER*.

Table 1 provides details of programs that have been adapted for use with *METAMODEL MANAGER*. For all other programs additional preparatory work will be needed to enable interaction. For further information on this see Appendices VI.

Table 1. Programs pre-adapted for compatibility with *METAMODEL MANAGER*

Program	*Type*	*Developer*	*Web address for download*
VORTEX	System- level	R. Lacy and J.P. Pollak	https://scti.tools/downloads/
OUTBREAK	System level or modifier	J.P. Pollak, R. Lacy, and others	https://scti.tools/downloads/
SPATIAL	Modifier	J.P. Pollak	https://scti.tools/downloads/
EVALUATOR	Utility	R. Lacy	Included in the MMM installation
MMMACRO	Utility	R. Lacy	Included in the MMM installation
PATH HISTORY	Utility	J.P. Pollak	Included in the MMM installation

Specifying and managing the interaction between *METAMODEL MANAGER* components

Once System-level and Modifier programs have been determined it will be necessary to describe to *METAMODEL MANAGER* the way in which these component programs will communicate with each other: which one will speak first and for how long; how much of the information provided will be available to the next and subsequent speakers; will the information provided require interpretation or translation by a third party before being handed over? These decisions are communicated through the following steps:

1) Organizing the sequence of operation
Within a *METAMODEL MANAGER*-facilitated metamodel, the component programs do not run simultaneously; rather they run sequentially according to a user-defined order. The order in which sub-models perform their tasks may have consequences for metamodel outcomes, so should be carefully thought through and tested before finalizing a decision.

2) Synchronizing program cycles
Program cycles or time-steps can vary. For example, in *VORTEX*, the default time-step is one year (365 days). That is, once *VORTEX* has performed each of its functions on a population, a year is considered to have passed. In *OUTBREAK* the default time-step is one day. So, for example, where the intention is to emulate one year in the life of a population under a disease threat, using *VORTEX* to manage demography and *OUTBREAK* to manage and manipulate disease characteristics, *OUTBREAK* will need to run 365 times for each *VORTEX* run. Where multiple component programs are involved, synchronizing time-steps can become complex.

3) Managing the exchange of information

METAMODEL MANAGER allows the user to identify one or more potentially influential input parameters within a model, and to have their magnitude and variability over time determined by a second model or series of models. For example, the disease state of an individual (e.g. infected or not infected), obtained from an epidemiological modeling program such as *OUTBREAK*, can be used to influence its likelihood of reproduction within a population viability analysis model such as *VORTEX*. Preparatory work to understand and facilitate carefully targeted inter-program information exchange is critical to ensuring that the metamodel behaves as intended by the user. State variables are the means through which information is shared and these may operate at three levels:

- Global State Variables (GSVs) – properties of the whole system
- Population State Variables (PSVs) - properties of a population
- Individual State Variables (ISVs) – properties of individuals, including the *VORTEX* standard variables of Name, Age, Sex, and Alive.

Any sub-model can create new variables at any level and these can then be available to other sub-models. Names for GSVs, PSVs, and ISVs need to be the same within all of the models using them; otherwise *METAMODEL MANAGER* will treat them as independent properties of the system, population, or individual. New variables can be created while the program is running (e.g. at year 50 it is possible to introduce a new variable). Further information on the use of state variables in programs pre-adapted for use with *METAMODEL MANAGER* is provided in Appendix II of this manual. Information on establishing and managing state variables in *VORTEX* is provided in the *VORTEX* manual (available at https://scti.tools/downloads/).

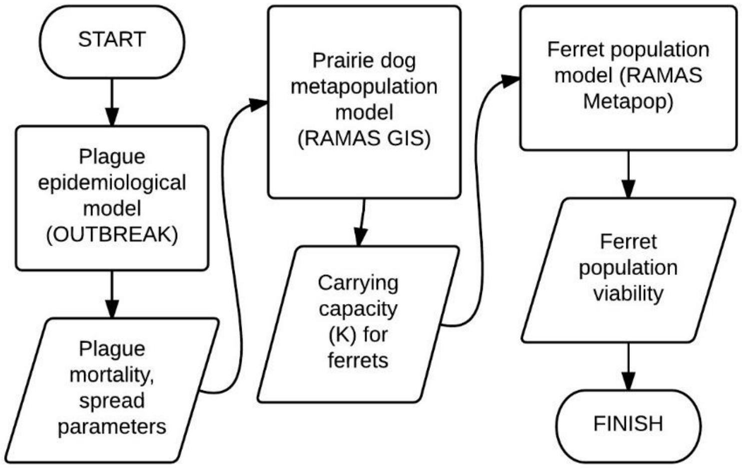
The coding for the linkage of shared data between metamodel components can be done directly within the independent specification files (i.e. by creating variables with the same names within each separate model), through a data translator (*EVALUATOR*) that is provided within *METAMODEL MANAGER* (i.e. by converting values of a variable used by one model to values of a different variable used by another model), or by using the utility program *MMMACRO* (installed with MMM) that allows more complex, multi-step macros or scripts for manipulating data.

EVALUATOR and *MMMACRO* provide the user platforms to manipulate variables passing through *METAMODEL MANAGER* to and from other programs. *EVALUATOR* provides the user with a way to write one-line transformations of variables describing individuals or the population. *MMMACRO* provides a simple way for a user to write, "on the fly", short BASIC-like programs (often sequences of transformations, each handled by *EVALUATOR*) to process the shared variables in *METAMODEL MANAGER*. Using *MMMACRO* is not required to establish linkage in cases where the variables to be shared have the same name and the same units or scale, and can therefore be directly shared without translation or transformation. In addition, *EVALUATOR* functions and even *MMMACRO*s can be embedded within a *VORTEX* project to transform variables being passed to or from *VORTEX*.

Box 1. Sketching a metamodel

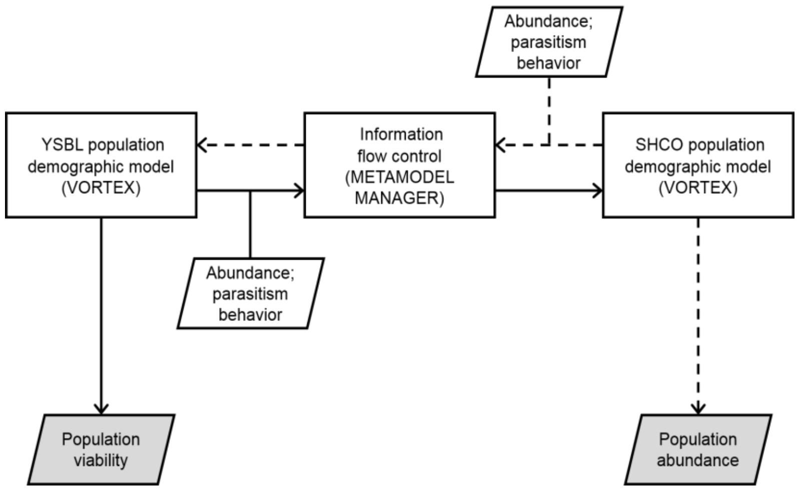
Creating a two-dimensional representation of the metamodel is an important first step in identifying what components are required and how they should interact and share data. Two examples are shown below:

Example 1. From Shoemaker *et al*., 2014: a metamodel built to explore the effects of prey metapopulation structure on the viability of black-footed ferrets in plague-impacted landscapes.


START
Plague epidemiological model (OUTBREAK)
Plague mortality, spread parameters
Prairie dog metapopulation model (RAMAS GIS)
Carrying capacity (K) for ferrets
Ferret population model (RAMAS Metapop)
Ferret population viability
FINISH

Schematic flowchart of the metamodel used in the study. Rectangular nodes represent processes (software in parentheses), and parallelogram nodes represent data output, which may be used as input for subsequent processes (indicated by arrows).

Example 2. From Miller *et al*., 2016: a metamodel built to explore the demographic interactions between the parasitic Shiny Cowbird and its host, the endangered Yellow-shouldered Blackbird.


Abundance; parasitism behavior
YSBL population demographic model (VORTEX)
Information flow control (METAMODEL MANAGER)
SHCO population demographic model (VORTEX)
Abundance; parasitism behavior
Population viability
Population abundance

Schematic flowchart of Yellow-shouldered Blackbird (YSBL)–Shiny Cowbird (SHCO) metamodel structure used for the analysis. Rectangles represent software components, and parallelograms identify model input/output variables from one implementation of *VORTEX* used to modify algorithms used by the other. Dashed lines link structural components pertaining to the Shiny Cowbird simulation. Gray parallelograms indicate final model output metrics used to evaluate model performance.

The next chapter steps through screen by screen to show how these metamodel elements are specified and brought together using the *METAMODEL MANAGER* program interface.

CAUTION: as indicated in Box 1, it is possible to link a population-based model (like *RAMAS*) to an individual-based model (like *OUTBREAK*), but there are potential pitfalls. For example, when data on sex-ratio and population size are made available from *RAMAS*, *METAMODEL MANAGER* creates a list of individuals with the required characteristics for use by *OUTBREAK*. However, when *OUTBREAK* hands the population back to *RAMAS*, the individual data are collapsed once again into population-level statistics. As a result, each time population data are passed between *RAMAS* and *OUTBREAK* a different set of individuals is created, making it impossible to follow a specific individual over time. When choosing the System-level model, users will need to consider if they will need to be able to track specific individuals, that is, to have individuals travel with accumulated characteristics across time-steps.

Chapter 6. Data input

This chapter takes the user through each *METAMODEL MANAGER* screen, explaining the features and choices available and their implications for metamodel construction.

MMM opening screen

Begin a New Project. Click on this option to set up a project for the first time.
Open an Existing Project. Click on this option to browse for and select a project file that you have worked with previously.
Manage Loaded Modeling Applications. This is primarily for use by program developers and not for users of existing programs. For information on this feature, see Appendix VI.
About. Click on this to view details about the MMM version in use and its correct citation.
Quit. Click on this to leave the program.

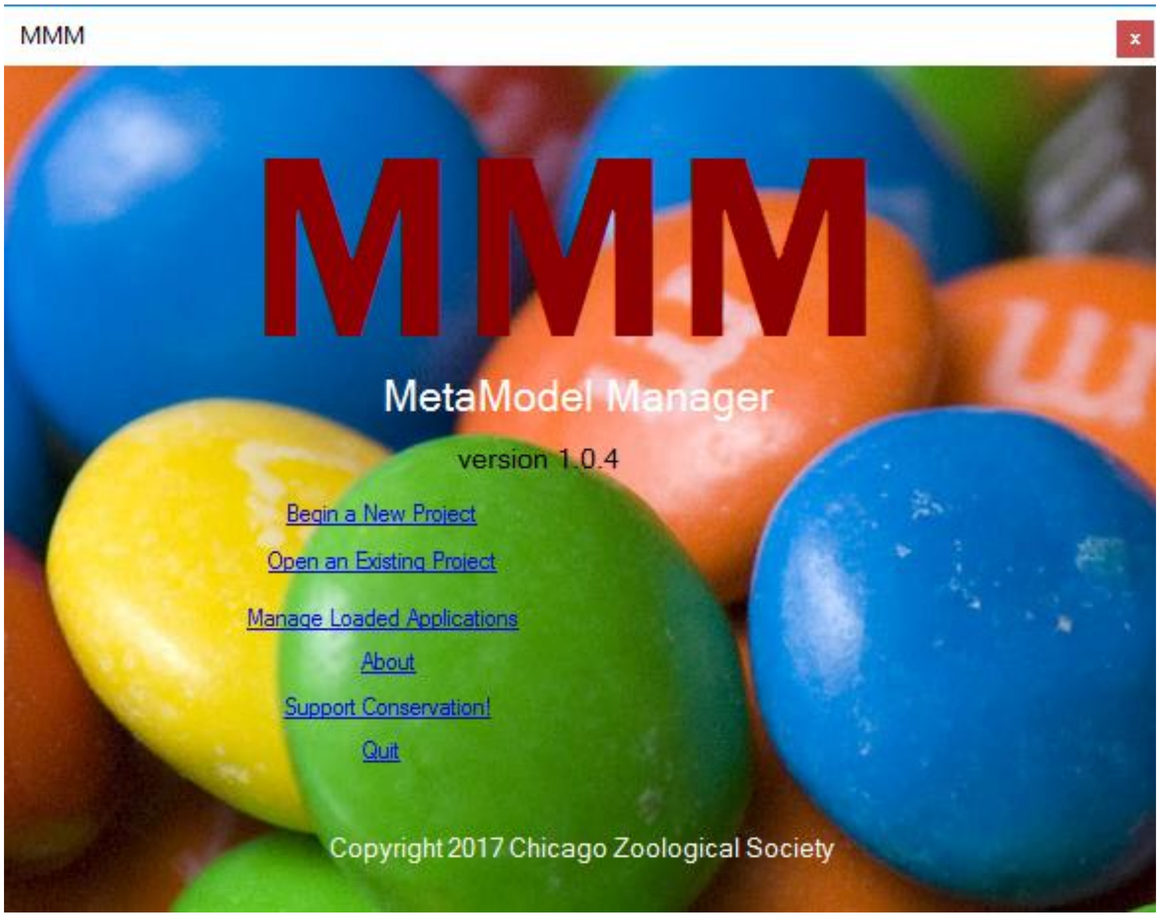
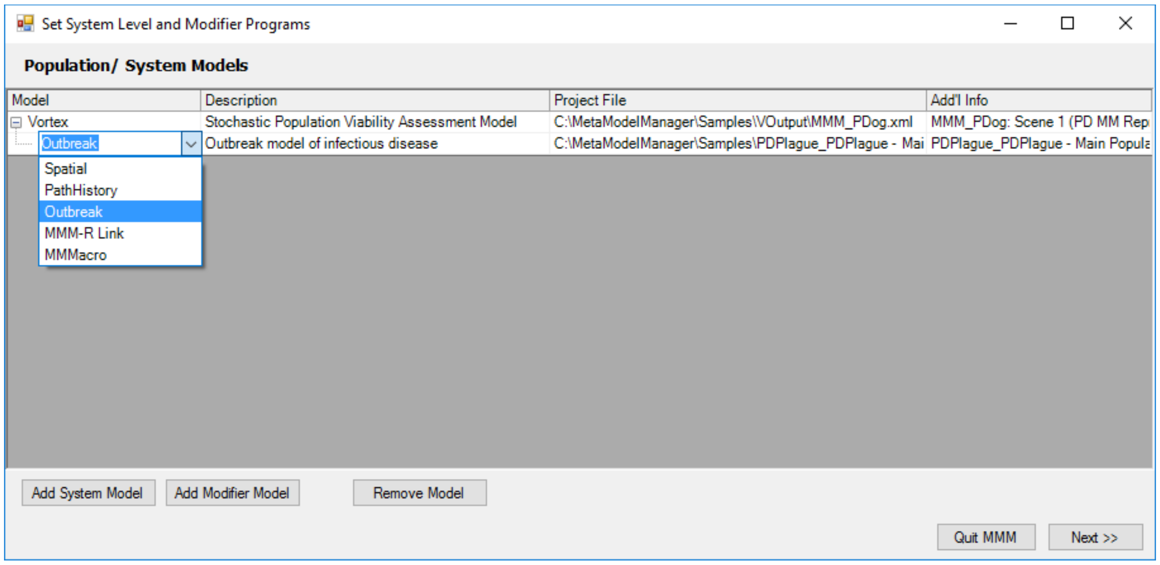


Figure 4. *METAMODEL MANAGER* **opening screen**

Set System Level and Modifier Programs screen



Model	Description	Project File	Add'l Info
Vortex	Stochastic Population Viability Assessment Model	C:\MetaModelManager\Samples\VOutput\MMM_PDog.xml	MMM_PDog: Scene 1 (PD MM Rep
Outbreak	Outbreak model of infectious disease	C:\MetaModelManager\Samples\PDPlague_PDPlague - Mai	PDPlague_PDPlague - Main Popula

Figure 5. Set System Level and Modifier Programs screen

Choosing to begin a new project will take you to the "Set System Level and Modifier Programs" screen, where you will be asked to specify the System Level and Modifier program components of your metamodel. You are first prompted to choose a "Population/System Model" from the drop-down menu (see Figure 5). Once the selection is made a browser box will open allowing selection of the relevant project file or scenario for that System Model. For example, a single *VORTEX* *.xml project file may include multiple scenarios. Only one of these at a time can be included in the metamodel.

Add Modifier Model. Click on this to add a Modifier Model to the metamodel. Select the relevant program from the drop-down menu provided. You may add more than one. Note that it is possible to run a metamodel without any Modifier Models.

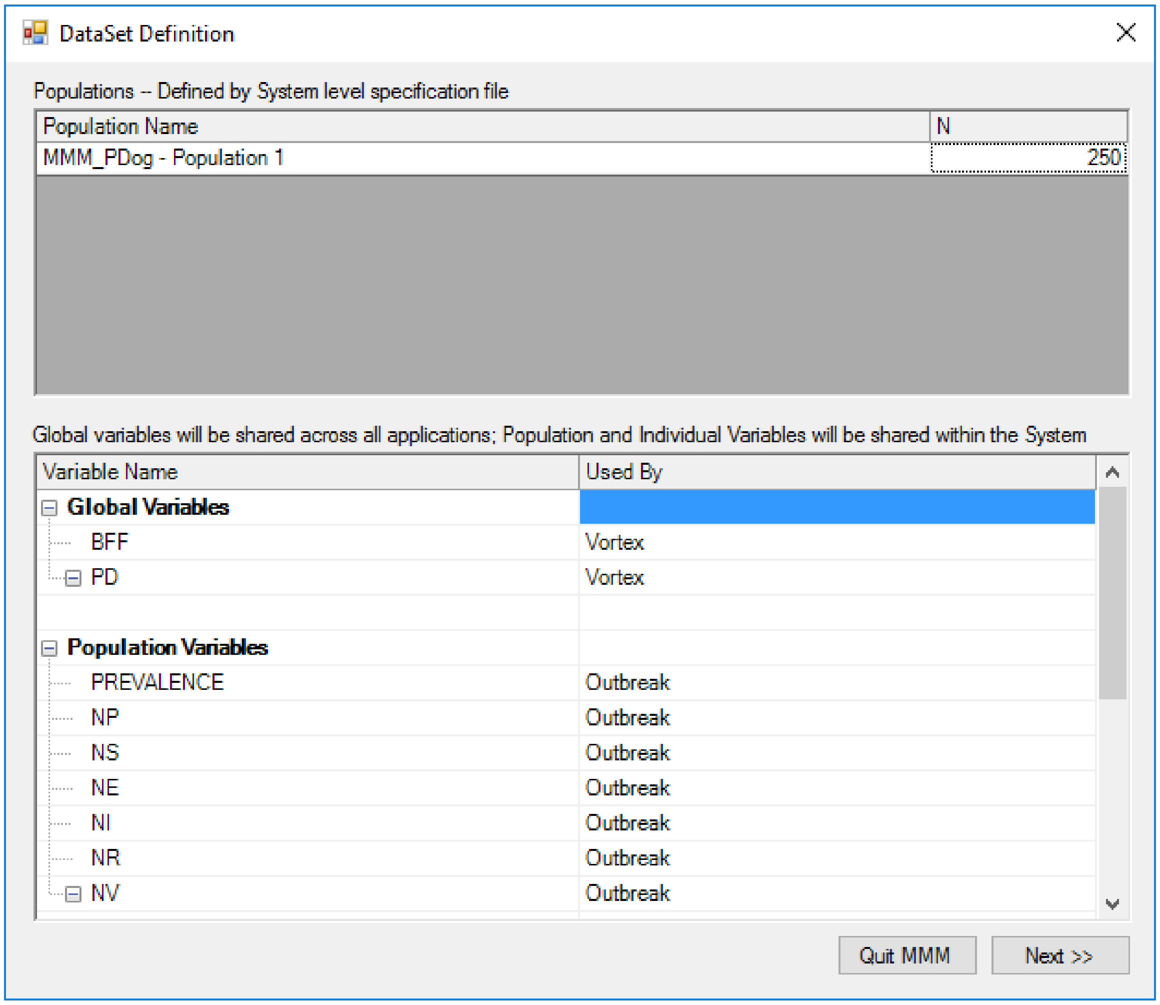
Add System Model. Additional System Level Models may be added using this button. For example, the user may include two *VORTEX* models, one characterizing a predator, the other its prey.

Remove Model. Click on this to exclude the highlighted program from the metamodel.

Once the metamodel components have been specified the user can proceed to the next page by clicking "Next >>"

Note. If you choose "Open an Existing Project" from the MMM opening screen, *METAMODEL MANAGER* will import the System Level and Modifier Models included in the selected file and you will be advanced automatically to the "Dataset Definition" screen.

DataSet Definition screen


DataSet Definition

Populations -- Defined by System level specification file

Population Name	N
MMM_PDog - Population 1	250

Global variables will be shared across all applications; Population and Individual Variables will be shared within the System

Variable Name	Used By
Global Variables	
BFF	Vortex
PD	Vortex
Population Variables	
PREVALENCE	Outbreak
NP	Outbreak
NS	Outbreak
NE	Outbreak
NI	Outbreak
NR	Outbreak
NV	Outbreak

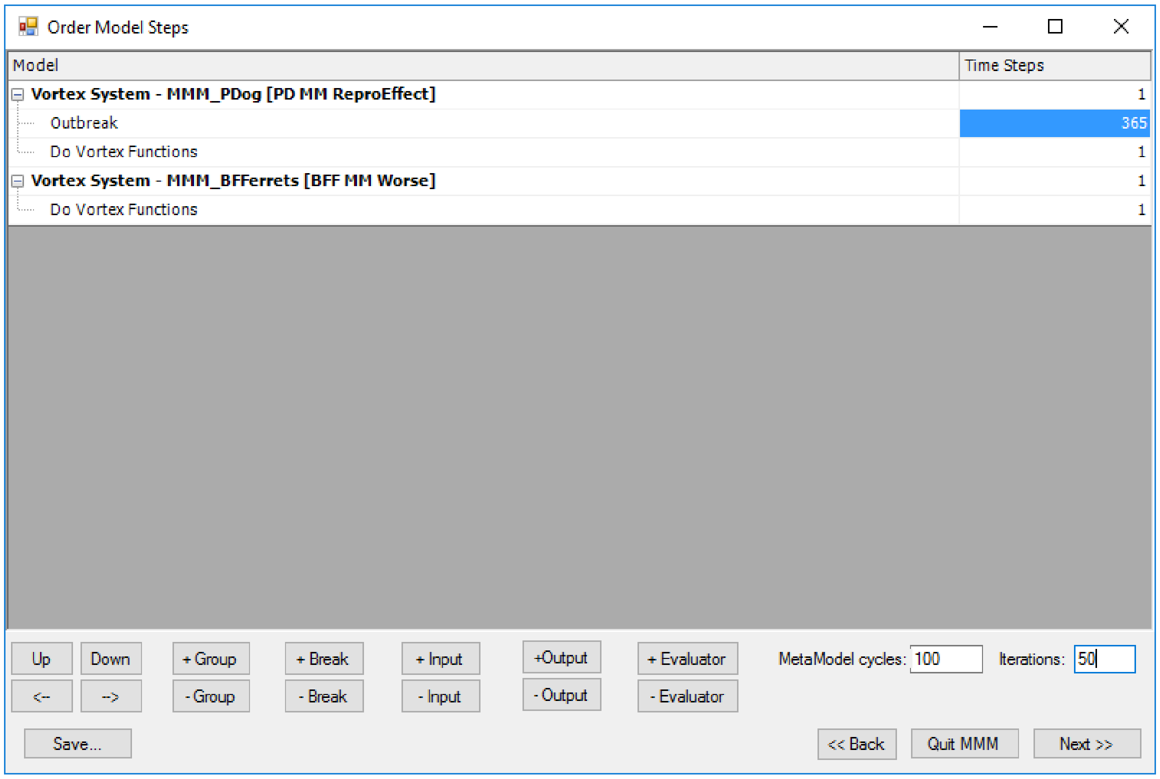
Figure 6. DataSet Definition screen

The DataSet Definition screen (Figure 6) displays what *METAMODEL MANAGER* knows about the populations, initial population sizes, and global, population, and individual state variables being used in your metamodel. You do not make any changes to the information on this screen; it is displayed just to let you confirm that your metamodel is being configured as you intended.

Populations – Defined by System Level Model specification files. This lists all populations referenced in all System Level Models included in the metamodel. If the System Level Model is *VORTEX*, the .xml project title is shown. Initial population sizes, imported from the System Level Model files, are shown on the right. This can be useful in confirming correct file selection.

Variables. Beneath the population information is a list of all Global, Population, and Individual State Variables available to be shared between programs in the metamodel. All of the programs included in the metamodel are potentially able to create, modify, or just read any or all of these variables. For a variable to appear on this list it must have been declared to *METAMODEL MANAGER* by at least one of the linked programs during its “initialization” step. The ‘Used By” column indicates which of the linked programs have “declared” which variables. For more information on configuring loaded applications, including “initialization”, see Appendix VI. Note that you won’t need to do anything to configure program initializations if you’re using the pre-loaded applications (*VORTEX*, *OUTBREAK*, etc.).

Order Model Steps screen


Order Model Steps

Model	Time Steps
Vortex System - MMM_PDog [PD MM ReproEffect]	1
Outbreak	365
Do Vortex Functions	1
Vortex System - MMM_BFFerrets [BFF MM Worse]	1
Do Vortex Functions	1

MetaModel cycles: 100
Iterations: 50

Figure 7. Order Model Steps screen

On this page you can control the order in which *METAMODEL MANAGER* runs each metamodel component, and also specify the number of time-steps or cycles each program will step through, before passing data to or sharing it with the next program in the sequence.

This is an important step because different programs may operate by default on different time-scales. The default for *VORTEX* is a time-step of 1 year, that is, once *VORTEX* has completed all of its standard operations on all of the individuals within a population, a year is considered to have passed. Individuals then age by one year, and the cycle begins again. For *OUTBREAK*, which considers finer scale events that can influence the shifting disease status of individuals throughout each year, each cycle or time-step is usually considered to take a day, such that typically, within each *VORTEX* time-step there will be 365 *OUTBREAK* time-steps. *METAMODEL MANAGER* is blind to these differences unless they are clearly articulated on this screen (see Box 2 for further illustration).

The screen shows two columns:

Model: lists the components of the metamodel in a tree format, with System Level programs (in bold) as the main branches and any accompanying Modifier programs nested beneath, as sub-branches.

Time-steps: shows the number of time-steps of that model completed before data are passed to the next program on the list.

Note. In the example shown, *VORTEX* is listed twice; first at the top (as Vortex System) and also in third place (as Do Vortex Functions). This is because in this metamodel it has two separate roles. First, *VORTEX* will create or "initialize" a population of the required size, for other programs to act on (this is indicated by the System Level listing in **BOLD)**. *VORTEX* will then pass information to *SPATIAL*, which will complete 365 daily time-steps before returning information to *VORTEX* to allow it to complete its second task – the annual sequence of demographic processes (Do Vortex Functions).

The required number of time-steps is typed directly into the cells provided. All other information entered into this screen is manipulated using the buttons and cells described below:

Up, Down - allows the user to move selected programs up or down to indicate the order in which model components will run.

+Group, -Group - allows the user to organize a subset of programs to run among themselves for a specified number of time steps, before passing information to a program outside the group. See Box 2 for further information.

+Break, -Break - This allows the user to insert a break into the simulation process in order to see, change, or save the interim results. Adding a "Break" requires the user to specify:

- whether the "Break" years are multiples of 1 (i.e. whether the *Break* occurs annually) or of a larger number (i.e. the "Breaks" occur once every several years) or
- whether the "Break" is triggered by a rule that relates to a Global State Variable.

Up, Down arrows are used to position "Breaks", the first of which is always at year 0. Once the simulation has begun and the program has paused, the user must select one of the following:

Show Population - generates a table displaying the entire population with individual characteristics [e.g. Index, Name, Sex, Age, Alive, Dam, Sire (for *VORTEX*) and DiseaseState, StatePermanent, DaysInState, IsVaccinated, Tenure (for *OUTBREAK*)]. During a "Break", the user can change any value in the table and can undo those changes by clicking on the "Reset Changes" button beneath the table. The button next this, "Write to File…" triggers a browser window in which the user can name and save this table as a text file.

Stop Simulation - stops the simulation and returns the user to the "Simulate" screen.

OK - resumes the simulation.

Note. The "Break" and "Show Population" features can be used to include user-directed, reactive management of the population. That is, the user can become an additional "sub-model" in the system. In the simplest case this can be done by setting up a metamodel that includes only a single *VORTEX* model, with *METAMODEL MANAGER* used to provide the extra *Break* utility that allows user-manipulation of the population during simulations. For example, changes can be made to the *Show Population* file such as killing or removing individuals with specific characteristics at each *Break*. *Breaks* can be set to occur only when certain conditions are met that correspond to triggers for management intervention (e.g. the population sinks below a target threshold). Users can experiment in this way to see whether, for example, it makes a difference to intervene with management more than once every 10 years.

+Input, -Input - allows the user to import values to override those generated in the modeling process. This might be relevant in a situation where the user wants more control over the impact of temporally occurring events on population characteristics. Input is applied using a text file.

The text file must be set up according to the following rules:

1. Any rows in the file that begin with "#" will be ignored, so rows relating to headers and additional information should carry this prefix.
2. Each row of the file represents a new year.
3. Within each row, the variables that you are changing must be separated by a semi-colon. Order matters!
4. If the variable is an array, separate the elements of the array with a comma (though see the CAUTION below for advice on using arrays).

The following are examples of simple input files. The first represents a file which sets two variables for 5 years:

```
#First example, 2 variables for 5yrs
0.1; 0.9
0.2; 0.8
0.3; 0.7
0.4; 0.6
0.5; 0.5
```

The second sets an array with 4 elements, for 5 years:

```
#Second example, 1 array 4 elements for 5yrs
0.1, 0.2, 0.3, 0.4
0.2, 0.3, 0.4, 0.5
0.3, 0.4, 0.5, 0.6
0.4, 0.5, 0.6, 0.7
0.5, 0.6, 0.7, 0.8
```

And finally, the third sets a regular variable and then an array of 4 elements, for 5 years:

```
#Third example, 1 normal variable, 1 array 4 elements for 5yrs
0.9; 0.1, 0.2, 0.3, 0.4
0.8; 0.2, 0.3, 0.4, 0.5
0.7; 0.3, 0.4, 0.5, 0.6
0.6; 0.4, 0.5, 0.6, 0.7
0.5; 0.5, 0.6, 0.7, 0.8
```

CAUTION: to date, using +*Input* to read arrays is complicated, not well-tested, and probably best left to the advanced user.

The *Input Step Setup* window, triggered when "+Input" is selected, will allow the user to choose the required input text file. The user will also be required to specify which variables will be input. If more than one variable is involved, order matters. The order must follow the order of variables in the input text file. The "Up" and "Down" buttons can be used to regulate the order.

+Output, -Output **-** allows the user to output the population data stored in *METAMODEL MANAGER* (as distinct from the data output specific to sub-models). Clicking "+Output" will create a file of all data on the population and its individuals, for every time step. This can be a very useful way to see exactly what is happening in your simulation. Clicking "+Output" triggers a browser window in which the user can specify a name and location for this file, and can choose whether to append to the file name the iteration number and year of the simulation.

+Evaluator, -Evaluator - provides to *METAMODEL MANAGER* (and other software) the ability to evaluate user-defined functions, to which are passed sets of parameter values. *MMMACRO* is an extension of *EVALUATOR*, allowing evaluation of short programs ("methods" or macros) that consist of assignments of variables using *EVALUATOR* functions, within simple (BASIC language-like) program flow. Both *EVALUATOR* and *MMMACRO* can be used to transform State Variables that describe the system, populations, or individuals. See Evaluator.doc (included in the MMM installation) for more information.

MetaModel Cycles - is the length of time (number of complete metamodel cycles) you want the simulation to run for. The default is 10.

Iterations - is the number of times you want *METAMODEL MANAGER* to repeat the simulation. You should probably start with just one iteration to see if the metamodel is working. Then run many more iterations to see the range of outcomes that might occur.

When you select "Next", you will be given an option to save the project. It is not necessary to save your project in order to continue with the simulation, but it is prudent to do so.

Box 2. Example of grouping

The metamodel in this example includes a single population, created by *VORTEX*. Within this population, the annual fertility of females that have been infected with a particular pathogen is expected to be zero. Status of infection with this pathogen is tracked daily, for all individuals in the population, through *OUTBREAK*. Likelihood of infection is increased in those individuals spending time in a specific area of habitat which is known to be favored by the primary vector of the pathogen. Time spent in this area of habitat is tracked daily, by *SPATIAL*. To simulate a year in the life of this population requires the following (simplified) steps:

VORTEX creates a population.
    OUTBREAK assigns DAY 1disease status for each individual;
    SPATIAL determines DAY 1 movement of individuals among habitat patches;
    OUTBREAK calculates DAY 2 disease status using information on contact with high risk habitat, as reported by *SPATIAL*;
    SPATIAL determines DAY 2 movement of individuals among habitat patches;
    Cycle repeats to completion of DAY 365
VORTEX completes annual demographic calculations with female fertility rates modified to account for infection status, as reported by *OUTBREAK*.

The correct grouping for this within *METAMODEL MANAGER* would look like this:

```
Vortex              1
     Group        365
          Outbreak  1
          Spatial   1
```

That is, the Group comprising alternating operations of *OUTBREAK* and *SPATIAL* will run 365 times for each *VORTEX* run.

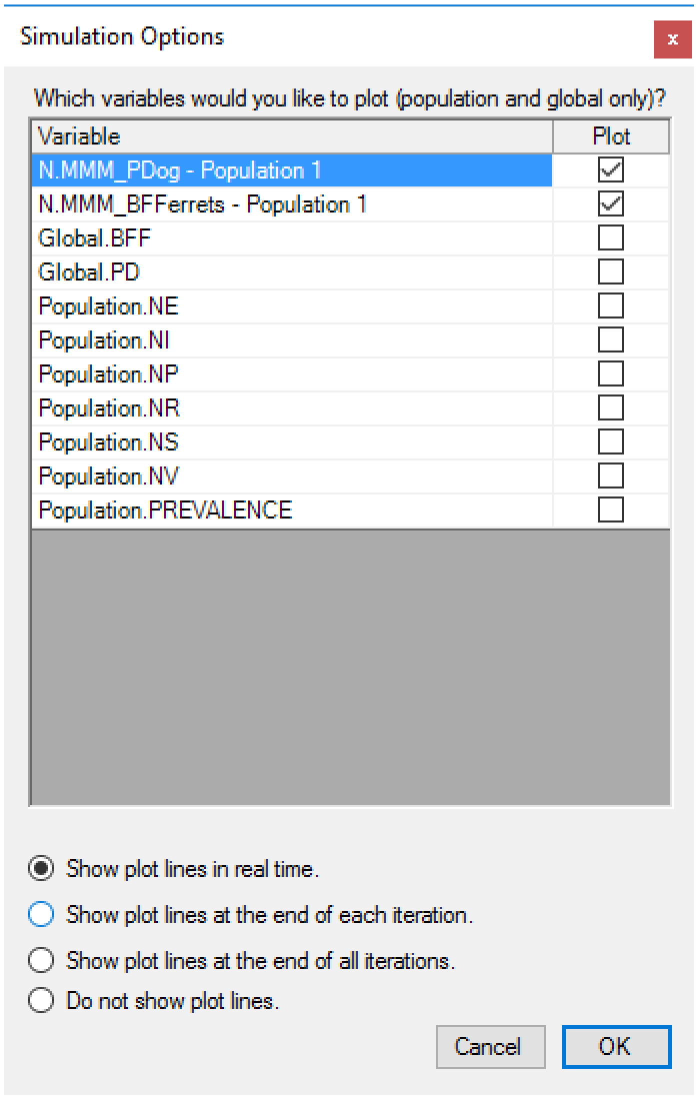
Without such grouping, the model might look like this:

```
Vortex              1
     Outbreak     365
     Spatial      365
```

In this case *OUTBREAK* would run for 1 year (365 time-steps) followed by a year of *SPATIAL* (365 time-steps) before the Year 1 demography update in *VORTEX*. As a result, *OUTBREAK* calculations of disease status would not be informed on a daily basis by time spent in high risk habitat and would not be informed by this at all until the second year of the simulation. The unintended consequence of this would be that the Year 1 *VORTEX* calculations for female fertility would remain unaffected by habitat use.

Chapter 7. Running the simulation and viewing output

Simulation Options screen


Simulation Options

Which variables would you like to plot (population and global only)?

Variable	Plot
N.MMM_PDog - Population 1	✓
N.MMM_BFFerrets - Population 1	✓
Global.BFF	☐
Global.PD	☐
Population.NE	☐
Population.NI	☐
Population.NP	☐
Population.NR	☐
Population.NS	☐
Population.NV	☐
Population.PREVALENCE	☐

[✓ Show plot lines in real time.]
Show plot lines at the end of each iteration.
Show plot lines at the end of all iterations.
Do not show plot lines.

Figure 8. Simulation Options screen

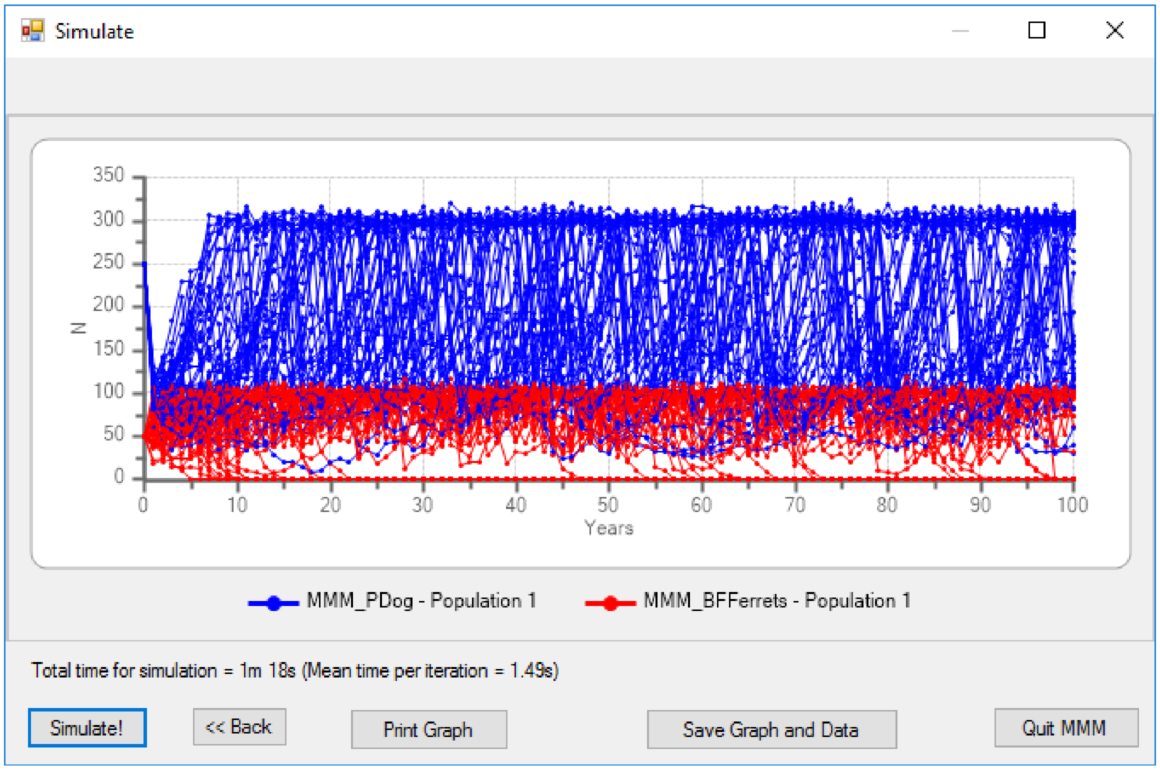
This screen allows the user to select which variables are plotted during or after the simulation. You can add any global or population state variables to the graph by checking the relevant box. The default is to plot the number of individuals (N) for each population, however in some instances you may wish to turn this off, as the scale needed to show N may make it impossible to see other variables (e.g. those measured on a scale of 0-1).

Note. One potential use of this is to display relative population sizes (on a scale of 0 to 1) in situations when different species have very different sizes and otherwise it would be hard to see the trends in N for the rare species on the same graph as the abundant species. To do this requires creating a GSV in each *VORTEX* (or other System Level Model) project (e.g., GS1=N/K) and then showing these GSVs rather than N on the graph.

On the lower part of this screen you are asked to specify whether you want to see the graph updated each year (i.e., in "real time"), after each iteration, after all iterations are completed, or not at all. The graphing is slow, so if you are running many iterations (in which case the graph will slowly turn into a solid blob of color that usually cannot be read), it is faster to show the graph at the end of each iteration, and even faster to run it without showing the graph at all.

Clicking "OK" takes you to the "Simulate" screen.

Simulate screen



Total time for simulation = 1m 18s (Mean time per iteration = 1.49s)

Figure 9. Simulate screen

Simulate! - enables *METAMODEL MANAGER* to begin running the metamodel simulation. A bar across the bottom indicates progress and on completion, a small box saying "Complete!" will appear.

After the run is complete you can edit the graph by double-clicking on it to open a graphics editor. You can also print or save the graph (in various formats) and the data (as an Excel file).

After each simulation, you can either repeat it by clicking "Simulate!" again, close the screen to return to the previous screen(s), change options and then re-run, or press "Quit MMM".

Quit MMM - provides a way to exit the program after the simulation is complete.

Unless the user specifies an "Output" step in the "Order Model Steps" screen, *METAMODEL MANAGER* itself won't produce any output files. Instead, output is viewed through the sub-model programs (e.g. *VORTEX, OUTBREAK, SPATIAL*). To see output, quit *METAMODEL MANAGER* and open up the relevant sub-model program. For further details on viewing output, refer to the user manuals for the individual programs.

Chapter 8. Tutorial

This chapter steps the user through an example of the use of *METAMODEL MANAGER* to model black-footed ferrets and their prairie dog prey, in the presence of sylvatic plague. The data files for this example are included in the installation of *METAMODEL MANAGER*, located in the C:\MMMProjects\Samples folder along with other sample projects. First time users are encouraged to complete this tutorial to familiarize themselves with the program.

Black-footed ferrets are one of the most endangered species in the United States. They have a specialized diet, with prairie dogs making up over 90% of the food ingested. Prairie dogs are subject to the sylvatic plague, which can reduce population size dramatically.

CAUTION: please note that in many cases, parameters are based on fictional information. The example presented below is for educational purposes only.

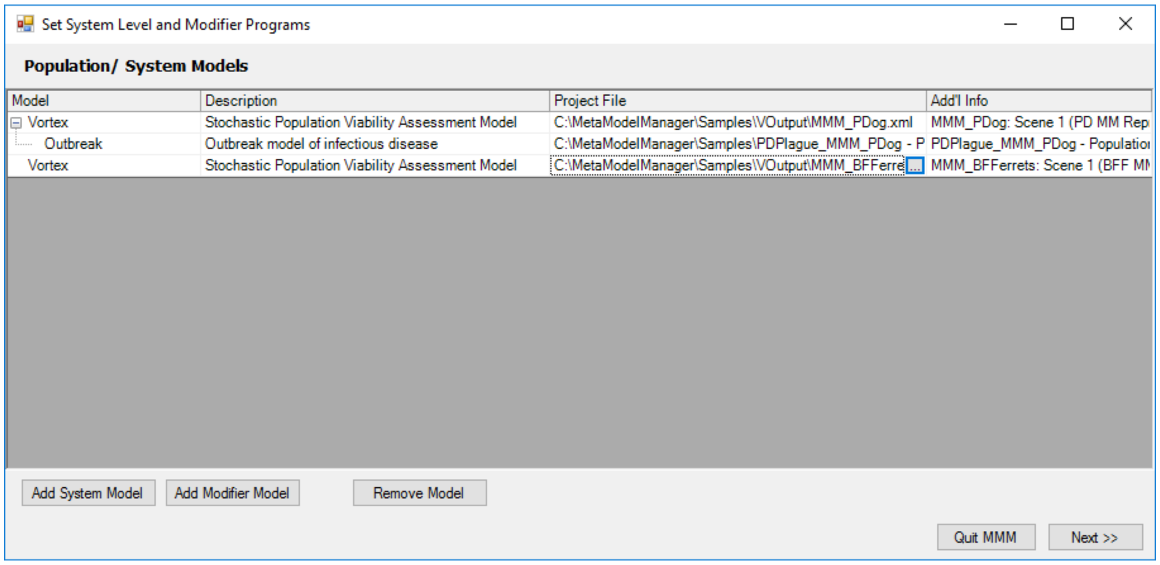
MMM opening screen

1) Select "Begin a New Project"

Setting System Level and Modifier Programs screen

1) Select the program, project and scenario that will "initialize" or determine the starting characteristics of the Prairie Dog population (and will later perform demographic functions on that population):
    a. From the drop-down list in the first column of the "Population/ System Models" table, select "VORTEX".
    b. Open the browser window next to "Choose a VORTEX project file" and select "MMM_PDog.xml". If it is not immediately available to you in the browser window, navigate to it in the C:\MMMProjects\Samples folder.
    c. Select the "PD MM ReproEffect" scenario from the "Choose" window.
2) Select the *OUTBREAK* model that will modify the disease status of individuals within the Prairie Dog population:
    a. Select the "Add a Modifier Model" button, and then choose "OUTBREAK" from the drop-down list.
    b. Open the browser window next to "Choose an OUTBREAK project file" and select "PDPlague.xml". If it is not immediately available to you in the browser window, navigate to it in the C:\MMMProjects\Samples folder.
    c. Select "Scenario1" from the "Choose" window.

3) Select the program, project and scenario that will "initialize" or determine the starting characteristics of the Black-footed Ferret population (and will later perform all demographic functions on that population):
    a. Select "Add a System Model", select "VORTEX" from the "Model" drop-down list.
    b. Select "MMM_BFFerrets.xml" from the "Choose a VORTEX project file" browser window.
    c. Select the "BFF MM Worse" scenario from the "Choose" window.
4) There are no other metamodel components. Select "Next" to move to the next screen.

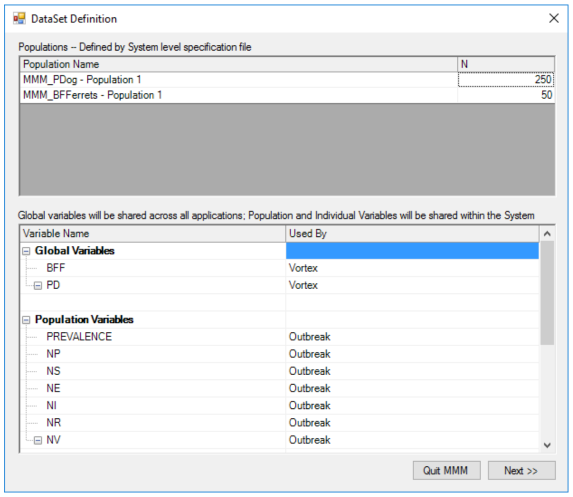

Set System Level and Modifier Programs
Population/ System Models

Model	Description	Project File	Add'l Info
Vortex	Stochastic Population Viability Assessment Model	C:\MetaModelManager\Samples\VOutput\MMM_PDog.xml	MMM_PDog: Scene 1 (PD MM Rep
Outbreak	Outbreak model of infectious disease	C:\MetaModelManager\Samples\PDPlague_MMM_PDog - P	PDPlague_MMM_PDog - Populatio
Vortex	Stochastic Population Viability Assessment Model	C:\MetaModelManager\Samples\VOutput\MMM_BFFerre	MMM_BFFerrets: Scene 1 (BFF M

Figure 10. Setting System Level and Modifier Programs - Tutorial

DataSet Definition screen

1) Check that the initial Prairie Dog population carries 250 individuals, and that the initial Black-footed Ferret population carries 50.
2) Check that you can see: two Global State Variables (GSVs), seven Population State Variables (PSVs), and ten Individual State Variables (ISVs). Then select "Next".

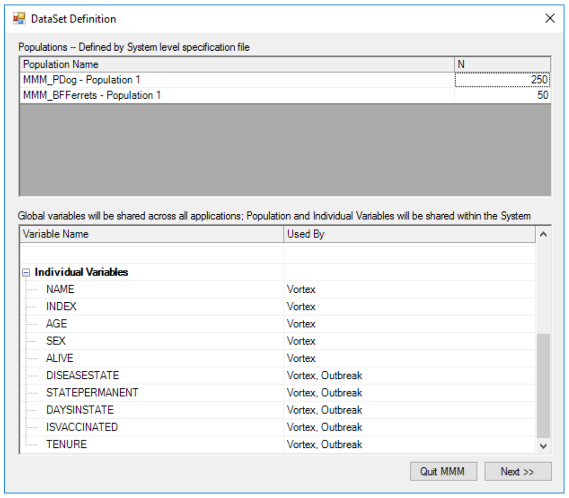


Populations -- Defined by System level specification file

Population Name	N
MMM_PDog - Population 1	250
MMM_BFFerrets - Population 1	50

Global variables will be shared across all applications; Population and Individual Variables will be shared within the System

Variable Name	Used By
Global Variables	
BFF	Vortex
PD	Vortex
Population Variables	
PREVALENCE	Outbreak
NP	Outbreak
NS	Outbreak
NE	Outbreak
NI	Outbreak
NR	Outbreak
NV	Outbreak



Populations -- Defined by System level specification file

Population Name	N
MMM_PDog - Population 1	250
MMM_BFFerrets - Population 1	50

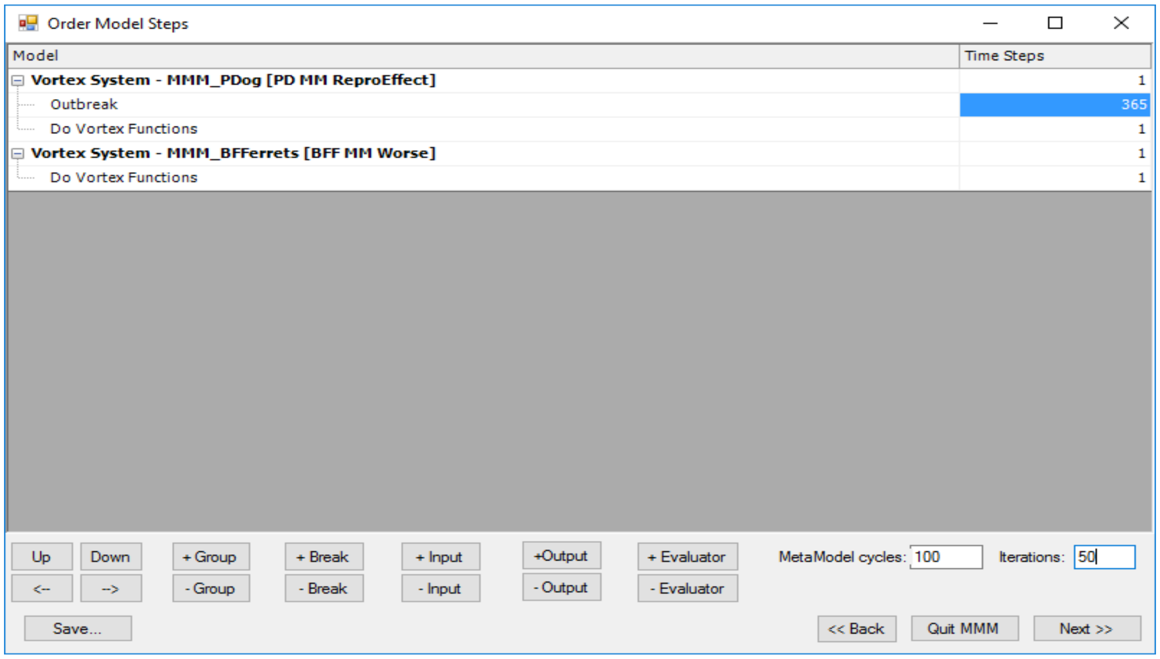
Global variables will be shared across all applications; Population and Individual Variables will be shared within the System

Variable Name	Used By
Individual Variables	
NAME	Vortex
INDEX	Vortex
AGE	Vortex
SEX	Vortex
ALIVE	Vortex
DISEASESTATE	Vortex, Outbreak
STATEPERMANENT	Vortex, Outbreak
DAYSINSTATE	Vortex, Outbreak
ISVACCINATED	Vortex, Outbreak
TENURE	Vortex, Outbreak

Figure 11. DataSet Definition - Tutorial

Order Model Steps screen

1) *OUTBREAK* runs on a daily cycle and so will need to run 365 cycles for every one (annual) cycle of *VORTEX*. Edit the *OUTBREAK* "Time Steps" to read 365.
2) Change "Total Years" from 10 to 100, and Iterations to 50 (note that you will want to run more iterations for scientific analyses but this will take longer).
3) Select "Save" and when the "Save as" window appears, navigate to the C:\MMMProjects\Samples folder and save the project with the name "MyFirstTutorial".
4) Select "Next".



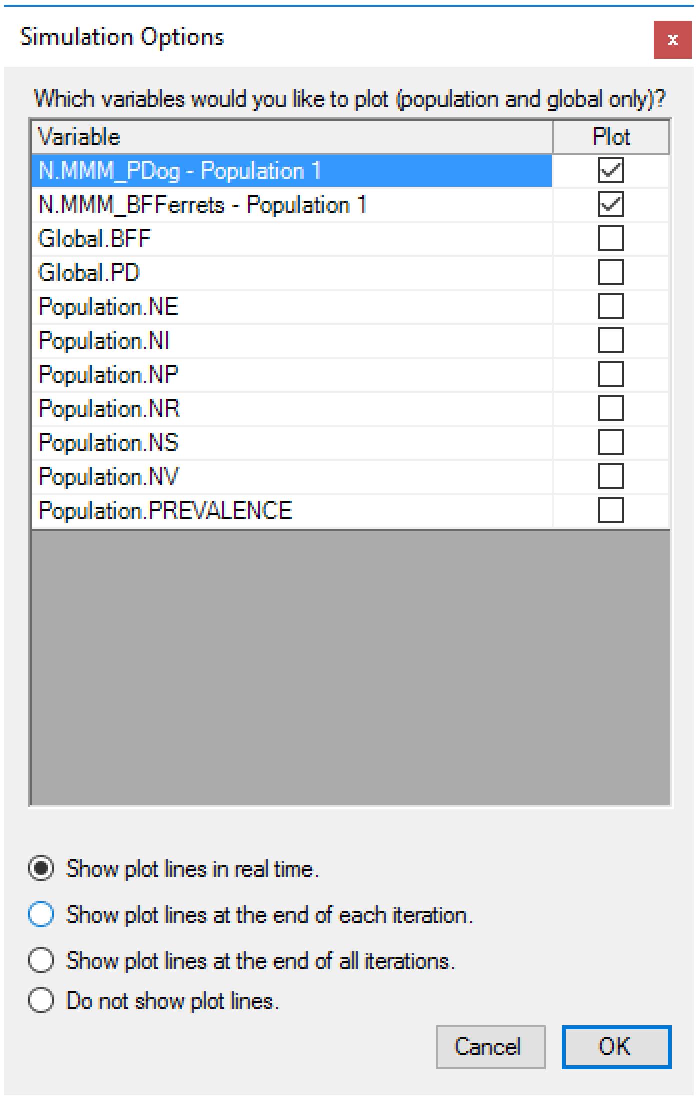
Model	Time Steps
Vortex System - MMM_PDog [PD MM ReproEffect]	1
Outbreak	365
Do Vortex Functions	1
Vortex System - MMM_BFFerrets [BFF MM Worse]	1
Do Vortex Functions	1

MetaModel cycles: 100 Iterations: 50

Figure 12. Order Model Steps - Tutorial

Simulation Options screen

1) Choose which variables to plot and how you would like to see them displayed. The default of plotting population size (N) for each of the modelled populations is already selected. Leave this in place and select "Show plot lines in real time". Select "OK".


Simulation Options

Which variables would you like to plot (population and global only)?

Variable	Plot
N.MMM_PDog - Population 1	✓
N.MMM_BFFerrets - Population 1	✓
Global.BFF	☐
Global.PD	☐
Population.NE	☐
Population.NI	☐
Population.NP	☐
Population.NR	☐
Population.NS	☐
Population.NV	☐
Population.PREVALENCE	☐

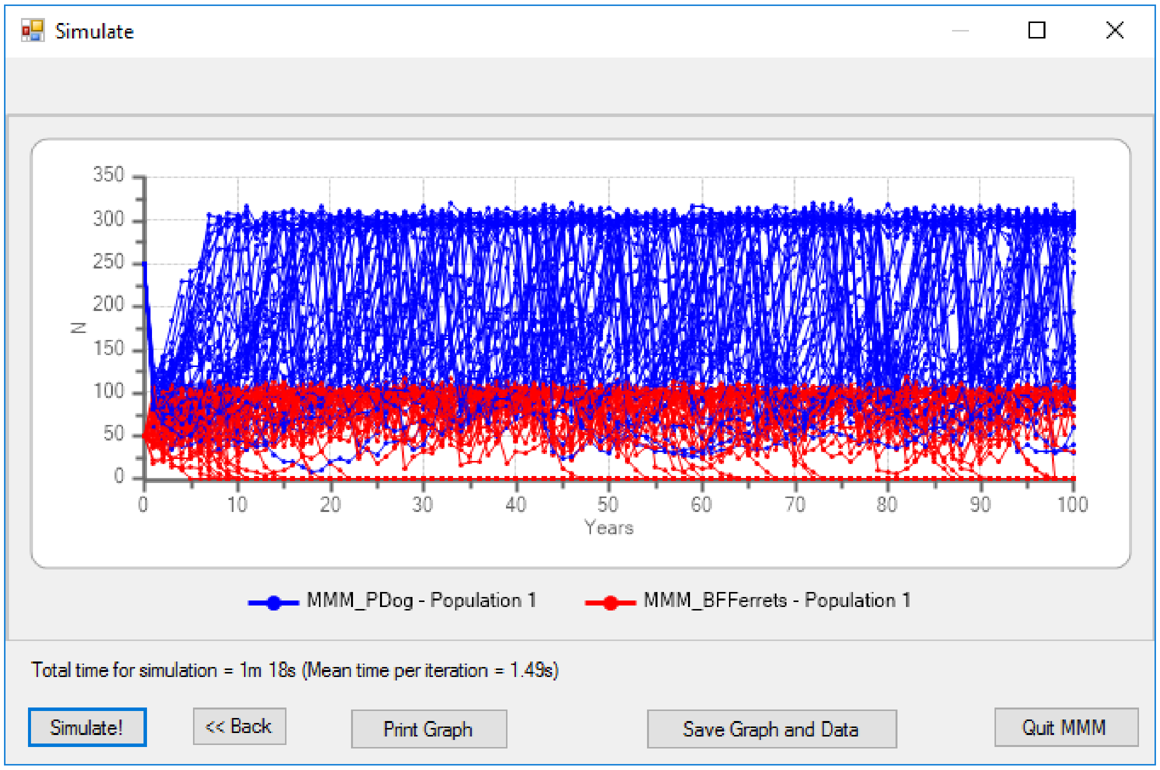
● Show plot lines in real time.
○ Show plot lines at the end of each iteration.
○ Show plot lines at the end of all iterations.
○ Do not show plot lines.

Figure 13. Simulation Options - Tutorial

Simulate screen

1) Select "Simulate!". The simulation will start.
2) When the processing is finished, you will see a "Complete!" box. Select "OK" to remove it and return to your graph.

Note. Initially, the screen may not show the graph correctly as in some cases your graphics card is not able to keep up with the simulation processing. Also, if you click elsewhere to move control to a different window, the simulation screen may cease updating and display a message indicating that *VORTEX* is "not responding". Do not worry. The simulations are continuing to work away in the background. The final graph will be displayed correctly when the simulation is finished.



Total time for simulation = 1m 18s (Mean time per iteration = 1.49s)

Figure 14. Order Model Steps - Tutorial

Congratulations! You have completed your first metamodel analysis of a dynamic, two-species system in which the prey species is subject to epidemics of disease. The final figure generated should show considerable fluctuations in Prairie Dog population size (as a result of disease), sometimes leading to the decline or local extinction of Black-footed ferrets. The modelled Prairie Dog populations rarely go extinct despite the large fluctuations.

Where are my results?

Results are viewed through the linked programs (in this case in *VORTEX* and *OUTBREAK*). The process for viewing output within linked programs will depend on where and how each program stores results. Information on this is available in the relevant manuals.

VORTEX stores results in subfolders called VOutput, which are housed within the relevant project folders. To view results for MyFirstTutorial, which is housed within the "Samples" project folder, open *VORTEX*, navigate to C:\MMMProjects\Samples and select either MMM_BFFerrets or MMM_PDog for Black-footed Ferret and Prairie Dog results respectively.

When *OUTBREAK* is run within *METAMODEL MANAGER* it creates a copy of the project file, adding to it information about the linked System Model and population. For example, the copy of *OUTBREAK* file "PDPlague" might be named "PDPlague _MMM_PDog - Population 1", and its output stored in an associated subfolder "PDPlague_MMM_PDog - Population 1_Results". This ensures that multiple System Models or multiple populations in a metamodel that call on the same *OUTBREAK* project and scenario will not overwrite each other's output.

To prevent *VORTEX* over-writing results when multiple metamodels call on the same *VORTEX* project and scenario, it is necessary to put copies of all of the relevant project files into separate folders, before running the various metamodels.

Using breakpoints within *METAMODEL MANAGER*

To learn more about the options available in *METAMODEL MANAGER*, select "Quit MMM", and re-open the program:

Opening Screen

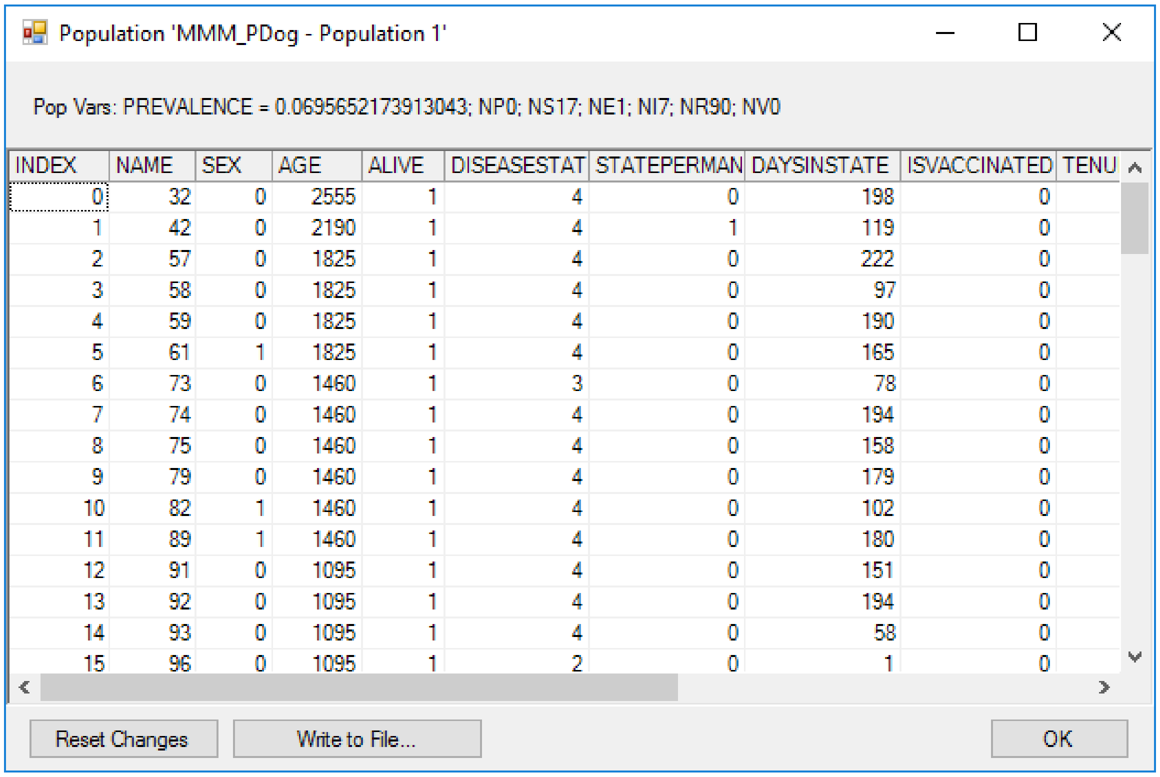
1) Select "Open an Existing Project" from the opening screen.
2) Select "MyFirstTutorial" from the C:\MMMProjects\Samples project folder.

Order Model Steps

1) Create a break at the end of the metamodel cycle by selecting "+ Break" and, using the "Up" and "Down" buttons, making sure it is the last item on the list of programs.
2) In the "New Break" window that appears, change "Year is a multiple of" from 1 to 10 (i.e. change from an annual break to a break every 10 years) and select "OK".
3) Change "Total Years" to 50, and iterations to 1, so the metamodel will run faster, and select "Next". Accept the current setting on the "Simulation Options" screen.

Simulate Screen

1) When "Simulate!" is selected a "Paused: Iteration" screen will appear showing population statistics at year 0.
2) With "MMM_PDog – Population 1" highlighted, select "Show Population". Population details at year (or time step) 0 will appear for Prairie Dogs, including their *OUTBREAK* disease state characteristics.
3) With "MMM_BFFerrets – Population 1" highlighted, select "Show Population". Population statistics at time step 0 will appear for Black-footed Ferrets. There is no disease state information to show because *VORTEX* was run alone for this population without *OUTBREAK* as a modifier.



Pop Vars: PREVALENCE = 0.0695652173913043; NP0; NS17; NE1; NI7; NR90; NV0

INDEX	NAME	SEX	AGE	ALIVE	DISEASESTAT	STATEPERMAN	DAYSINSTATE	ISVACCINATED	TENU
0	32	0	2555	1	4	0	198	0	
1	42	0	2190	1	4	1	119	0	
2	57	0	1825	1	4	0	222	0	
3	58	0	1825	1	4	0	97	0	
4	59	0	1825	1	4	0	190	0	
5	61	1	1825	1	4	0	165	0	
6	73	0	1460	1	3	0	78	0	
7	74	0	1460	1	4	0	194	0	
8	75	0	1460	1	4	0	158	0	
9	79	0	1460	1	4	0	179	0	
10	82	1	1460	1	4	0	102	0	
11	89	1	1460	1	4	0	180	0	
12	91	0	1095	1	4	0	151	0	
13	92	0	1095	1	4	0	194	0	
14	93	0	1095	1	4	0	58	0	
15	96	0	1095	1	2	0	1	0	

Figure 15. Prairie dog population at Break0 - Tutorial

4) Save the statistics of Black-footed Ferrets to a file by selecting "Write to File" and assigning the filename "MyFirstTutorial_BFFBreak0".
5) Select "OK" on the "Population MMM_BFFerrets – Population 1" screen and then again on the "Paused: Iteration" screen, to resume the simulation, which will pause again at year 10. Write the Black-footed Ferret data to file again, assigning the filename MyFirstTutorial_BFFBreak10. Repeat this through the first iteration.
6) You can now compare variables at specific breakpoints. You might do this to better understand the patterns you see in your overall MMM results.
7) If you want to use the break to manipulate the population, you can do so on the list of individual variables displayed when you "Show Population". To test this, go ahead and kill a few prairie dogs (changing Alive code to 0), or infect them (changing DiseaseState to 3), or cure them (change Disease State from 3 to 4).

Chapter 9. Trouble shooting

Problem: Models don't seem to be influencing each other in the way you expect. Results don't look different from when you run models independently.

Solution: Check to make sure your models are sharing variables. Are the variables you intend to share named identically in the interacting models? Do you need to add an *EVALUATOR* or *MMMACRO* to translate the scale or units of the variable from one model to the proper scale of the variable in another model?

You can try running the model one iteration at a time and/or adding breakpoints to look at the values of specific variables. This can help you identify if the value of a variable in Model 2 is as expected, given the value of a variable in Model 1.

Be aware, however, that just because your results are unexpected does not mean that they are wrong. It is possible that a complex system does behave differently than you would have guessed. That is why we run metamodels!

Problem: MMM results can't be found.

Solution: Results will be stored independently for each model, and to view them, you'll have to use the program for each model. For example, let's say your metamodel includes a *VORTEX* system model and you've saved your *VORTEX* *.xml project file within a folder you created for your metamodel. Your file structure might be something like this:
C:\Users\You\MMMProjects\YourProject\YourVortexModel\VOutput
You will need to use *VORTEX* to open the *.xml *VORTEX* project file that is in the YourVortexModel folder.

Problem: I can't save my MMM project file.

Solution: You might not have write access to the directory where MMM is trying to save the project. Try saving in a folder, like 'My Documents'.

Problem: After running some metamodel simulations, going back a few screens to change settings, and then running more simulations, *METAMODEL MANAGER* crashed.

Solution: When moving back through screens to change metamodel settings, *METAMODEL MANAGER* can sometimes get confused. (It happens to the best of us.) When running different metamodels, it is safest to exit *METAMODEL MANAGER* after completing one analysis, and then re-start the program to load and run your next metamodel.

Problem: Sometimes, *METAMODEL MANAGER* won't let me back up to change settings on earlier screens.

Solution: Some settings in *METAMODEL MANAGER* require the program to do a lot of work to configure your metamodel. Consequently, the program won't let you go back to change some of the more fundamental aspects of the metamodel that you are building. If you need to start over with basic settings for your project, close *METAMODEL MANAGER* and start again.

Problem: I want to simulate some changes in the system that cannot be modeled in *VORTEX* or *OUTBREAK* or any of the sub-models that seem to be available (e.g., maybe changes in social structure, or habitat characteristics, or climate fluctuations).

Solution: Link in a different program written by you or someone else as a Modifier model (see Appendix VI), or create a small *MMMACRO* to represent the processes (see Appendix V), or link an R script to *METAMODEL MANAGER* (see Appendix V).

Problem: The reviewer of my manuscript (or my academic advisor, or my boss, or just me) wants to see the algorithms used by *METAMODEL MANAGER* so that I can confirm that it is doing what I want.

Solution: *METAMODEL MANAGER* does not have much in the way of algorithms, at least not ones that carry out computations. The computational work is done by the sub-models (*VORTEX*, *OUTBREAK*, *RAMAS*, *MMMACRO*, etc.), while *METAMODEL MANAGER* controls the sequence in which the sub-models access the shared data and the flow of data among the sub-models. Go to the documentation for the linked programs to learn how they work. If you really do need to see the source code for *METAMODEL MANAGER* (or *MMMACRO* or *OUTBREAK*), it is available for inspection in the SourceCode folder within the *METAMODEL MANAGER* program folder (usually in C:\Program Files (x86)\Species Conservation Toolkit Initiative\MetaModelManager).

Problem: It is really hard to create a complex metamodel and be certain that it is modeling my case properly. Where can I get help?

Solution: Tap into the expertise or curiosity of your friends. Often asking a colleague to look over your project leads to him or her seeing something that you overlooked. (Or just confirming that your model does look good.)

You can also join the MetaModels Listserv (https://scti.tools/help-support/#Community) to share problems and ideas with other users. At this time, there are not yet many users on this list, but we expect that as more people use *METAMODEL MANAGER*, the list will become a very useful network.

If you need further help from an experienced and skilled user of *METAMODEL MANAGER*, either the Species Conservation Toolkit Initiative (email: help@scti.tools) or the IUCN SSC Conservation Planning Specialist Group (email: office@cpsg.org) might be able to put you in contact with an expert user who could help (most likely for reasonable compensation).

Question: The values for many of the input parameters in my models are very uncertain. How can I do sensitivity tests to determine if the possible alternative values would significantly influence my results?

Solution: *METAMODEL MANAGER* does not (yet) have any built-in tools for sensitivity testing. There are two approaches that you can use to test the effect of alternative values for model input. First, you can create multiple scenarios, each as a separate metamodel, with alternative parameter values (or even model structure). *METAMODEL MANAGER* can be run from a command line, with the name of the project file specified as a parameter passed to MeMoMa.exe, making it possible to use a shell program to create and run many metamodel scenarios without requiring manual specification and starting of each one. Analyzing the results across such metamodel scenarios will then likely need to be done within the interfaces of the separate sub-models, or in external statistical packages used to compile and analyze results across scenarios.

Alternatively, within the sub-models of a metamodel, or via the utility programs *EVALUATOR* or *MMMACRO*, you can often sample the values for uncertain variables at each iteration. For example, uncertainty in a dispersal rate might be estimated with a normal distribution. By sampling from the normal distribution the rate to be used in each iteration, the results across iterations will represent the range of possible outcomes, and influence of the uncertain dispersal rate on outcomes can be quantified and tested with appropriate statistical analysis. The details of how best to do such sensitivity testing are much beyond the scope of this manual, and depend on the sub-models you are running, the variables that are uncertain, and the nature (distribution) of that uncertainty.

Appendix I. Literature cited and other useful references

Bradshaw, C.J.A., McMahon, C.R., Miller, P.S., Lacy, R.C., Watts, M.J., Verant, M.L., Pollak, J.P., Fordham, D.A., Prowse, T.A.A. and B.W. Brook (2011). Novel coupling of individual-based epidemiological and demographic models predicts realistic dynamics of tuberculosis in alien buffalo. Journal of Applied Ecology 49: 268–277

Lacy, R.C., Miller, P.S., Nyhus, P.J., Pollak, J.P., Raboy, B.E., and Zeigler, S.L. (2013) Metamodels for Transdisciplinary Analysis of Wildlife Population Dynamics. PLoS ONE 8(12)

Miller, P.S., and R.C. Lacy. 2003. Metamodels as a tool for risk assessment. Pp. 333-351 in: F.R. Westley and P.S. Miller, eds. Experiments in Consilience: Integrating Social and Scientific Responses to Save Endangered Species. Island Press, Washington, DC.

Miller P.S., Lacy R.C., Medina-Miranda, R.., López-Ortiz, R., Díaz-Soltero, H. (2016). Confronting the invasive species crisis with metamodel analysis: An explicit, two-species demographic assessment of an endangered bird and its brood parasite in Puerto Rico. Biological Conservation 196:124–132

Nyhus, P. J., Lacy, R.C., Westley, F.R., Miller, P.S., Vredenburg, H., Paquet, P., and J.P. Pollak (2007). Tackling biocomplexity with meta-models for species risk assessment. Ecology and Society 12(1): 31

Prowse, T.A.A., Johnson, C.N., Lacy, R.C., Bradshaw, C.J.A., Pollak, J.P., Watts, M.J. and B.W. Brook (2013). No need for disease: testing extinction hypotheses for the thylacine using multi-species metamodels. Journal of Animal Ecology 2013 (82): 355–364

Wells, K., Brook, B.W., Lacy, R.C., Mutze, G.J., Peacock, D.E., Sinclair, R.G., Schwensow, N., Cassey, P., O'Hara, R.B. and D.A. Fordham (2015). Timing and severity of immunizing diseases in rabbits is controlled by seasonal matching of host and pathogen dynamics. J. R. Soc. Interface 12:20141184

Shoemaker, K.T., Lacy, R.C., Verant, M.L., Brook, B.W., Livieri, T.M., Miller, P.S., D.A. Fordham, D.A. and H.R. Akcakaya (2014). Effects of prey metapopulation structure on the viability of black-footed ferrets in plague.-impacted landscapes: a metamodelling approach. Journal of Applied Ecology 2014 (51): 735–745

Appendix II. Understanding the use of State Variables to model biological links

METAMODEL MANAGER shares data among sub-models via Global (system), Population, and Individual State Variables, and the use of these variables in an "open-data" framework is at the core of metamodels. The variables must be defined by one or more of the sub-models, and then accessed by other sub-models as the metamodel simulation is running. The values of the state variables might be changed by just one or possibly several of the sub-models. For example, both *VORTEX* and *OUTBREAK* might cause some individuals to die (possibly from predation and disease), but presumably only the System-level model would set each individual's sex (unless the species can change sex through the lifetime) and would increment age. Importantly, the shared variables are then used by the sub-models as inputs into their processes. For example, the age and sex specified by *VORTEX* might alter the probability of becoming infected by a disease in *OUTBREAK*, and the disease status emerging from *OUTBREAK* might affect reproduction or dispersal in *VORTEX*.

VORTEX offers the infrastructure to set up user-defined state variables in order to provide extra information regarding an individual, population, or metapopulation. Defining and using these variables is key to creating successful linkages between programs in order to express how the parameters of one population may be influenced by outcomes from other programs. *VORTEX* has the same three levels of state variables that are handled in *METAMODEL MANAGER*: global (GSV), population (PSV), and individual (ISV). These state variables must be numerical, and can change over time. (See the *VORTEX* manual for more information on how it creates and uses state variables.)

Other modeling programs may also provide the means for users to set up state variables, or may have pre-defined state variables that are shared through *METAMODEL MANAGER*, or may read (from *METAMODEL MANAGER*) the variables created by other linked programs. For example, *OUTBREAK* always creates a PSV called "Prevalence", and ISVs called "DiseaseState", "DaysInState", "StatePermanent", and "IsVaccinated". In addition, *OUTBREAK* can use any other population or individual state variables created by other programs and handed to it by *METAMODEL MANAGER*, if those variables are also defined within *OUTBREAK*.

Any program linked to MMM can access the state variables defined by other linked programs, or can create new variables and add them to the data set that is shared with all linked programs. This allows each linked program to read and optionally change the population and individual data created by other programs.

Just as writing your own model to be linked in to others via MMM is not something that can be explained quickly in this manual, understanding how to make use of the data that flow among linked programs is also not something that will be immediately obvious to a new user. Later versions of this manual will provide more detailed explanations and examples, but for now an example is very briefly described below.

Example: Linking *VORTEX-VORTEX*
In a two-species two-*VORTEX* metamodel, information from each species that needs to inform the model of the other species would be coded into GSVs such as in the case of BFF-PD demonstrated in the earlier chapter. The following is a more detailed description of how to set up that linkage.

1) Creating the Operational Link:
A key to creating a link to share data between two *VORTEX* models is to define the same (or at least overlapping) set of global or population state variables (GSVs and PSVs) that will be shared by both programs but, in most cases, modified only by one of the two models. In this example, GS1 is set to be the number of ferrets and GS2 is set to be the number of prairie dogs.

For BFFs in their own BFF sub-model, the number of individuals (GS1) is set each year to "N", the number of individuals generated by the *VORTEX* model of the ferrets. For BFFs in the PD sub-model, the BFF population size (GS1) is first set to 50 and then assigned to remain to be GS1, which means that the PD model will not change this GS1 variable but will read the number of ferrets for use in specifying impacts (predation!) on the prairie dogs. It would usually be meaningless for the PD *VORTEX* model to change the GS1 that is the number of ferrets, because that N is an emergent property of the BFF individual-based simulation.

GS2, the number of PDs, is set in the PD sub-model each year to "N", the number of individuals generated by its model. In the BFF model, the number of PDs (GS2) is first set to 250 and then assigned to stay as GS2, which serves to read the information from the PD sub-model but not try to change the number of prairie dogs from within the ferret sub-model.

Global State Vars	Label	Trophic Level			
		PD (Prey)		**BFF(Predator)**	
		Initializ. fn	Transition fn	Initializ. fn	Transition fn
GS1	BFF	50	=GS1	=N	=N
GS2	PD	=N	=N	250	=GS2

It is important to keep in mind that *METAMODEL MANAGER* knows that a state variable is to be shared by two components only if it has the same label in both. Thus, if the PD sub-model changes a GSV labeled "PD", then any GSV labeled "PD" in the BFF sub-model will be changed synchronously. The "GS1", "GS2", etc. designations used in *VORTEX* are meaningless to *METAMODEL MANAGER*. Thus, the variable BFF might be GS1 in the PD model, but GS2 in the BFF model, but it will still be shared across the sub-models via *METAMODEL MANAGER*. This also means that you need to be careful with the labels of shared variables. If one sub-model uses the label "BFF-N" and the other uses the label

"BFFN", they will be treated as independent, unlinked variables (even if both are GS1 in the *VORTEX* projects). You can give GS1 the label "GS1" in both models, and then they will link properly, but this would be a bad practice because nothing about the variable name indicates what data it contains.

2) Creating the Ecological Link:
The ultimate ecological link between the two *VORTEX* models occur when the GSVs are later used in equations to parameterize the demographics of each of the two populations. In our example, the mortality of PDs in the PD sub-model for females from age 0 to 1 equals 45+((15*BFF)/N). The fewer PDs present in the population relative to BFFs, the more young PD mortality goes up. In the BFF sub-model, first year mortality is specified as 46.6+((20*N)/(1+PD)), so that a higher ratio of predators to prey reduces the survival of young ferrets.

Links between sub-models may also be made through using ISVs, whereby specific information is tied to specific individuals in the population. In the BFF-PD-Plague metamodel, IS1 tracks the DiseaseState of each prairie dog. DiseaseState is determined within the *OUTBREAK* model of plague, and transferred to the *VORTEX* sub-model by MMM. *VORTEX* then uses this IS1 variable to modify the reproductive rate of prairie dogs, with a function that specifies that currently sick (DiseaseState 'I': IS1=3) females cannot reproduce. *VORTEX* ignores the disease status of each individual when setting mortality rates, because the death of prairie dogs from plague occurs within the *OUTBREAK* model (with 50% mortality of diseased PDs).

We do not want to imply that these few simple relationships describe accurately the functional response of predator to prey and the reverse; we just inserted some simple functions to show how such relationships can be specified and modeled within a 2-species metamodel, with disease impacting one of the species. A more complete and realistic metamodel for ferrets and prairie dogs might include reciprocal effects on adult survival of both species, effects on reproduction of ferrets, and possibly other functions defining the interactions between the species.

Appendix III. Using *VORTEX* in *METAMODEL MANAGER*

It is not possible to use a metamodel that links a *VORTEX* model of population dynamics to models of other processes or to *VORTEX* models of other species unless you first are fully familiar with the use of *VORTEX*. This includes knowing how to use state variables to define and track global, population, and individual properties and then use the GSVs, PSVs, and ISVs as parameters within functions that define demographic rates or other input parameters. Therefore, if you are not experienced at using *VORTEX*, STOP here, and go learn *VORTEX* first.

When you install *METAMODEL MANAGER*, it will install in the same program folder the *VORTEX* simulation library (vortex10lib.dll) that is required to run *VORTEX* as a component of a metamodel. You should also download and install the full *VORTEX* program (downloaded from https://scti.tools/downloads/) so that you have the user interface for building *VORTEX* projects and displaying results.

To create a metamodel that includes a *VORTEX* component, you should start by creating the *VORTEX* project that describes the population dynamics. Initially, focus on the basic demography of the species, and don't worry (yet) about state variables needed for the metamodel and the ecological linkages to properties of other models. Test this initial *VORTEX* model carefully to confirm that the demographic model is performing as expected. It might also be a good idea at this stage to run sensitivity tests to determine which uncertain input variables have the largest effects on population trajectories. Once you link the *VORTEX* model with other sub-models in a metamodel, uncertainties compound (or even explode!), so it is wise to first determine for which *VORTEX* parameters it will be important to carry the uncertainty through to the metamodel.

Next, add the state variables that will be needed to link the *VORTEX* model to other models. The values of the state variables that will be accessing information coming from other models can be set simply to a plausible default value for the initialization function and a transition function that leaves the variable unchanged. Pay careful attention to what labels you assign to these state variables. Although functions within the *VORTEX* model can refer to a state variable either by its label or by GS1, GS2, PS1, IS1, etc., *METAMODEL MANAGER* will use the labels to share the state variable with other models. There are a few individual properties that are automatically made available from *VORTEX* to *METAMODEL MANAGER*, so they do not need to be assigned to Individual State Variables in *VORTEX*. These are: NAME (=ID in *VORTEX*), INDEX, SEX, AGE (in years), and ALIVE. Any other individual properties (such as genotypes or mates) need to be assigned to an ISV if they are to be made accessible to other linked programs. Note that if another component model in a metamodel changes a state variable in *VORTEX*, the state variable itself is changed but the trait that was assigned to the state variable will not be. For example, if an ISV that is assigned the mtDNA haplotype ("=MT") is changed by another program, that does not then change the mtDNA of the individual in *VORTEX*. (I.e., the assignment of the haplotype to the ISV is unidirectional; a change in the ISV does not then change the haplotype). Similarly, if an ISV (with perhaps label MySex) is set in *VORTEX* to be the sex ("=S"), a change in that ISV by *METAMODEL MANAGER* does not change the sex of the individual in *VORTEX*. However, SEX is automatically defined as a shared individual variable in *METAMODEL MANAGER*, so if another component of the metamodel changes SEX, that *will* change the sex of the individual in *VORTEX*.

Note also that the case of a variable is ignored in *VORTEX* (e.g., S and s are the same in *VORTEX* functions, and a GSV labeled Habitat will be the same as one labeled HABITAT) – as *VORTEX* internally converts all functions to upper case. However, for some component models linked through *METAMODEL MANAGER*, and in *METAMODEL MANAGER* itself, variable labels are case sensitive. Thus, a state variable called HABITAT in *VORTEX* will not be seen as the same as one called Habitat in another linked program.

After the state variables needed for linking models are created in *VORTEX*, add to your *VORTEX* scenario the functions that are needed to make these state variables influence demographic rates. Then, with default values of the state variables that will be obtained from other models when the metamodel is run, test the stand-alone *VORTEX* model to be sure that the population projections are plausible under these conditions.

Now, go build the other components of your metamodel, build the metamodel in *METAMODEL MANAGER*, and run your metamodel!

After the metamodel has run, you can view the results in the Text Output and Tables & Graphs sections of the *VORTEX* interface by opening the *VORTEX* project file that was used in the metamodel. For examining the results across multiple species that were run in separate *VORTEX* models linked via *METAMODEL MANAGER*, it is often useful to make sure that the output metrics you wish to see for all the species are defined within Global State Variables that are shared. In that way, you can access them all from the output tables saved from one *VORTEX* model, rather than needing to consolidate results across models.

One more caution about constructing metamodels that include *VORTEX* simulations of populations: think carefully about when variables are updated in the sequence of the *VORTEX* year and the metamodel time step. For example, in the prairie dog – ferret example described earlier, the GSV BFF will be updated in the ferret model each time that GSVs are updated, but the internal variable N (to which BFF is set in each update) will be updated between each event within the *VORTEX* annual cycle. From the perspective of the PD sub-model, the variable BFF has the value that it was last assigned in the BFF sub-model when that *VORTEX* component had control. This might or might not be the same as the last value of N in the BFF sub-model, depending on when in the annual sequence of *VORTEX* events the UpdateVars occurs in the BFF sub-model.

Appendix IV. Using *OUTBREAK* in *METAMODEL MANAGER*

OUTBREAK can be run as either a System-level model (creating the initial population and doing its own demography) or as a Modifier model (changing only the disease state of individuals whose demography is managed by *VORTEX* or some other System model). When *OUTBREAK* is used as a Modifier model in a metamodel, its demography is disabled – that is, the values on its Initial Population and Demography input tabs are ignored, because they are replaced by the demography provided by the System-level model.

The demographic simulation in *OUTBREAK* (when it is used as the System-level model) differs in several ways from the one in *VORTEX* – primarily in that *OUTBREAK* models events on a daily basis so that demographic events are interlaced with disease processes, and *OUTBREAK* does not include many aspects of stochasticity (e.g., environmental variation, catastrophes, genetic drift) that are modeled in *VORTEX*.

OUTBREAK now provides the option (as of version 2.5) for the user to create Population or Individual state variables, and any PSVs and ISVs created in *OUTBREAK* will be made accessible to *METAMODEL MANAGER*. In addition, *OUTBREAK* always creates a PSV called "Prevalence", and ISVs called "DiseaseState", "DaysInState", "StatePermanent", and "IsVaccinated". Within functions defining rates in *OUTBREAK*, these variables should be referenced by the labels PREV, Z, DSTATE, CHRONIC, and V, respectively. The four ISVs describing the disease condition of individuals can be changed by other models in a metamodel. *OUTBREAK* does not provide the ability to create Global state variables, because *OUTBREAK* is always (for now) a single population model. However, any GSV available in *METAMODEL MANAGER* can be linked to a similarly labeled PSV in *OUTBREAK*, allowing *OUTBREAK* to access and change the GSVs of *METAMODEL MANAGER* or other linked models.

OUTBREAK can use any GSVs, PSVs, or ISVs created by other programs and handed to it by *METAMODEL MANAGER*, if those state variables are also declared within the *OUTBREAK* project. Thus, for example, if a *VORTEX* scenario and the *OUTBREAK* scenario each create an ISV called BodyCondition, within the linked *OUTBREAK* the probability of disease transmission (or any other input variable) can be specified to be a function of BodyCondition. Similarly, with an ISV set to be the genotype at one or more loci, disease susceptibility or recovery rates can be made to be dependent on the genotypes that are simulated in *VORTEX*.

OUTBREAK does provide a utility to simulate movements of individuals on the landscape, because the distance between individuals can be an important determinant of disease transmission. The spatial model in *OUTBREAK* is relatively basic, with the user needing to specify the functions that define movement probabilities on an X-Y grid at each daily time step, but this is a capability that is not included in *VORTEX*. Therefore, in a linked *VORTEX-OUTBREAK* metamodel, *OUTBREAK* can be used to simulate individual movements, and *VORTEX* can use the X-Y location of each individual as a determinant of demographic rates or events. For example, the criteria for acceptable mates in *VORTEX* can be that a potential mate must be within a certain Euclidean distance (*VORTEX* variable DIST) to be accepted. Or the probability of mortality can be specified to be higher in some regions of the X-Y space.

When using *OUTBREAK* or *VORTEX* in a metamodel with spatial movement, the locations of individuals should be obtained with variables XCOORD and YCOORD, rather than X and Y. This is because X and Y have alternative meanings as built-in variables in *VORTEX* (X = number of females, Y = year). Thus, the ISVs in *VORTEX* that will be used for spatial locations and linked to the *OUTBREAK* simulation should be labeled XCOORD and YCOORD, and any functions of location in *VORTEX* or *OUTBREAK* should use these labels.

As with *VORTEX*, after the metamodel has run, you can view the tables and graphs of *OUTBREAK* results (the summaries of disease dynamics) by opening the *OUTBREAK* project in the full *OUTBREAK* interface.

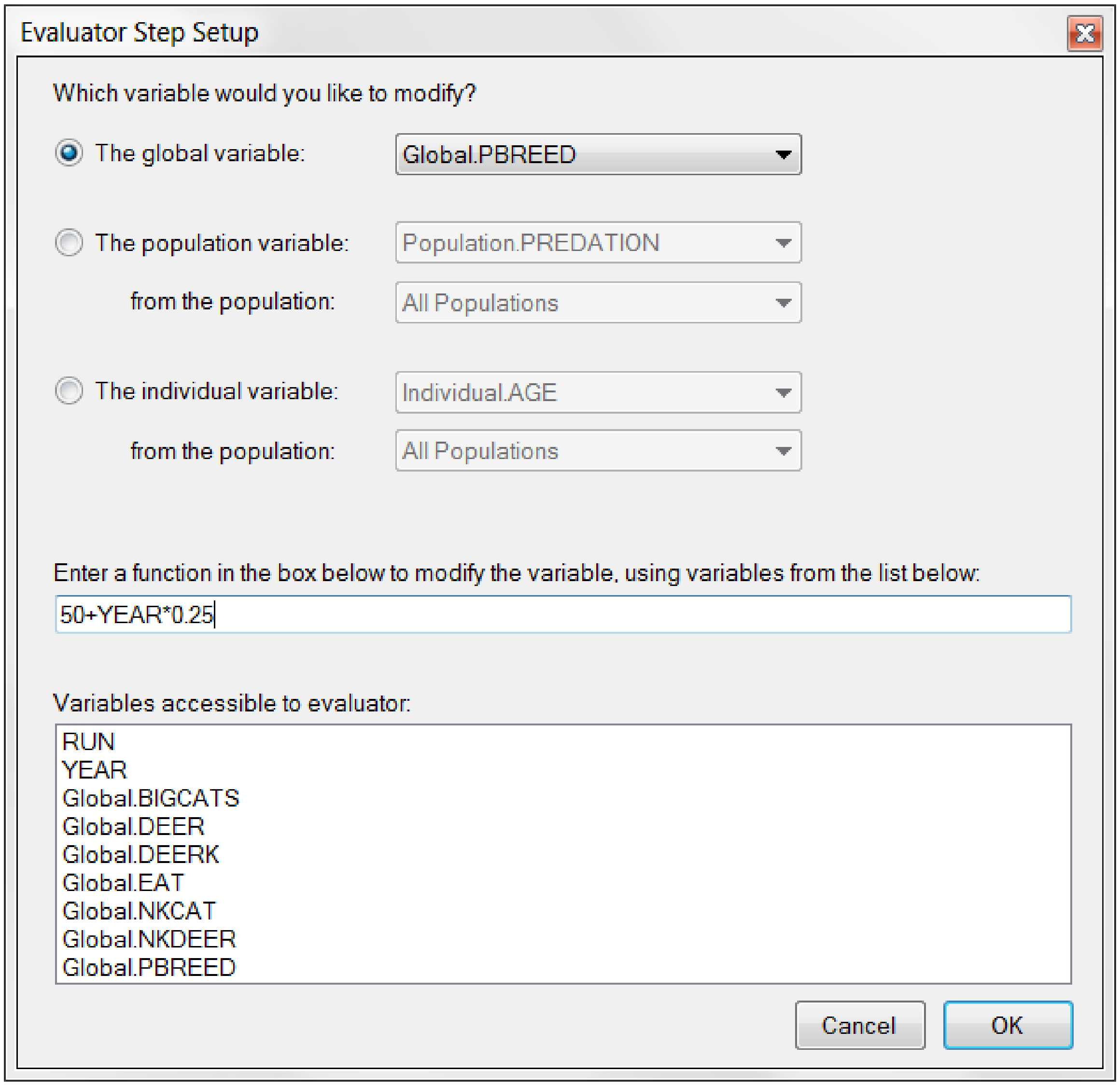
Appendix V. Using *EVALUATOR* and *MMMACRO* in metamodels

EVALUATOR was developed as a utility program to provide to modeling software the ability to evaluate user-defined functions, to which are passed the sets of parameter values. *EVALUATOR* is included in the *VORTEX* population simulation software and *METAMODEL MANAGER* to provide these programs with the ability to evaluate functions. *Macro* is an extension of *EVALUATOR*, allowing evaluation of short programs ("methods", "scripts", or macros) that consist of assignments of variables using *EVALUATOR* functions, within simple (Basic language-like) program flow. *Macro* has been embedded within *VORTEX* (as *VMACRO*) and *METAMODEL MANAGER* (as *MMMACRO*). *EVALUATOR* and *Macro* have also been packaged into a separate MS Windows dll library so that other such programs can make use of these tools. Both *EVALUATOR* and *Macro* also now support calling R scripts as a way to determine the values of variables with more complex algorithms (as long as they can be coded in R).

EVALUATOR is written in C# and compiled with MS Visual Studio.Net for use on any Windows-based computer (including Macs running Windows emulators or compatible OSes). The source code for *EVALUATOR* is included in the MMM installation. Technical information for programmers about the use of the *EVALUATOR* library in other programs is provided in the Evaluator.doc file that it included in the MMM installation.

Within *METAMODEL MANAGER*, *EVALUATOR* and *MMMACRO* can be inserted into the sequence of sub-model steps and used to transform any state variables, by evaluating functions of other state variables and the metamodel values for RUN (which iteration is currently running) and YEAR (the current metamodel cycle). In this way, you can translate variables from the form used by one sub-model to a form needed by another, or you can write what are essentially simple programs to derive or update state variables. For example, you could use an *EVALUATOR* or an *MMMACRO* to generate a pattern of habitat change over time, and then use this variable within sub-models to modify survival, dispersal, or other rates. (If one of your sub-models is *VORTEX*, you could almost always embed the work of *EVALUATOR* or *MMMACRO* to modify variables within the *VORTEX* model, as it has the same ability as *METAMODEL MANAGER* to do such transformations. However, the metamodel structure might be clearer if processing of transformations that involve data from other, and possible several, sub-models is kept within the *METAMODEL MANAGER* project rather than carried out within one of the *VORTEX* sub-models.)

When you add an *EVALUATOR* to your metamodel on the Order Model Steps screen (using the +Evaluator button), an *EVALUATOR* Step Setup window will open up. (See Figure 15.) In this window, you first specify which GSV, PSV, or ISV you want to modify with the *EVALUATOR*. If you pick a PSV or ISV, you need also to specify for which population (or all populations) you want to calculate updated values. If you select an ISV, the *EVALUATOR* will apply the transformation to all individuals in the population (although it is possible to specify within the function that the transformation applies to only some individuals, based on their ID, sex, age, or any ISVs).


Evaluator Step Setup

Which variable would you like to modify?

[✓ The global variable:] Global.PBREED

The population variable: Population.PREDATION

from the population: All Populations

The individual variable: Individual.AGE

from the population: All Populations

Enter a function in the box below to modify the variable, using variables from the list below:

50+YEAR*0.25

Variables accessible to evaluator:

RUN
YEAR
Global.BIGCATS
Global.DEER
Global.DEERK
Global.EAT
Global.NKCAT
Global.NKDEER
Global.PBREED

Figure 16. Evaluator Step Setup window.

The list at the bottom of the window shows what variables are available for use in the function. For transformations of a GSV, only RUN, YEAR, and GSVs can be used in the function (because it would be ambiguous which population was being referenced if a PSV was used). For functions to transform a PSV, you can use RUN, YEAR, GSVs, and PSVs. For *EVALUATOR*s that transform an ISV, any GSV, PSV, or ISV can be used.

<u>Operators and built-in functions available</u>

EVALUATOR includes an extensive list of operators and mathematical functions for use in defining the transformations of state variables. Most of these operators are listed below, and more information about them is included as an Appendix to the *VORTEX* manual.

Operators, listed in order of precedence (evaluated first at the top), with left-to-right evaluation for sets of operators on the same line. Parentheses (any of '(', '[', or '{', and the matching ')', ']', or '}') can be used to force the order of evaluation of operators.

^	(exponentiation, e.g., 5^3 = 125
-	(negation, e.g., 5+(-3) = 2
!	(logical NOT)
*, /, %	(multiplication, divide, and modulus [remainder from integer division])
+, -	
<, >, <=, >=	
=, !=	("==" will be made equivalent to "=" and to EQUALS, and "#" is equivalent to "!=")

AND, OR, NAND, NOR ("&" and "&&" and "AND" are all equivalent; "|" and "||" and "OR" are all equivalent; "NAND" can be coded as "$" and "NOR" can be coded as "~".)

IF(A;B;C) B if A is true; C if A is false

Other functions available for use (with the standard meanings, if not explained) are:

<u>Mathematical functions</u>

MAX(A;B)
MIN(A;B)
POW(A;B) [=A^B]
EXP(A) [=E^A]
SQRT(A) [=SQR(A)]
MOD(A;B) [=A%B], the remainder after integer division. E.g., 8%3=2
ABS(A), absolute value
LN(A) [=LOGE(A)=LOG(A)]
LOG10(A)
CEIL(A), smallest integer value equal to or greater than A
FLOOR(A), largest integer equal to or less than A
ROUND(A), nearest integer value, rounds down for even numbers + 0.5, rounds up for odd numbers + 0.5.
PROUND(A), probabilistic rounding, rounds up with probability equal to the non-integer part of the number. E.g., PROUND(5.75) = 6 in 75% of the cases; and 5 in 25% of the time.

Random number generators

RAND, a random number between 0.0 to 1.0.
SRAND(S), a seeded RAND, such that for a given integer seed, S, the random value is always the same
NRAND, returns a random (0,1) normal
SNRAND(S), a seeded NRAND
UNIFORM(A;B), returns a random number drawn from the interval A to B.
SUNIFORM(A;B;S), a seeded UNIFORM
IUNIFORM(A;B), returns a random integer drawn from the interval A to B.
SIUNIFORM(A;B;S), a seeded IUNIFORM
POISSON(A), returns a random number from a Poisson distribution with mean A.
SPOISSON(A;S), a seeded POISSON
BINOMIALN(A;B), returns the number of successes sampled from a binomial distribution with parameters mean = A and sample size n = B.
BINOMIALP(A;B), returns the proportion of successes sampled from a binomial distribution with parameters mean = A and sample size n = B.
SBINOMIALN(A;B;S), a seeded BINOMIALN
SBINOMIALN(A;B;S), a seeded BINOMIALP
BETA(A;B), returns a random number sampled from a beta distribution with shape parameters A and B.
SBETA(A;B;S), a seeded BETA
BETAM(A;B), returns a random number sampled from a beta distribution with parameters mean = A and SD = B.
SBETAM(A;B;S), a seeded BETAM
GAMMA(A;B), returns a random number sampled from a gamma distribution with shape parameters A and B.
SGAMMA(A;B;S), a seeded GAMMA
GAMMAM(A;B), returns a random number sampled from a gamma distribution with parameters mean = A and SD = B.
SGAMMAM(A;B;S), a seeded GAMMAM
DECAY(A), returns an integer sampled from the distribution representing exponential decay at a constant rate; returns the number of timesteps until transition in a process with probability A each timestep. If A <= 0, then the function returns 0 (not infinity).
SDECAY(A;S), a seeded DECAY.

Trigometric functions

SIN(A), sine of A, with A in radians (as is the case for all trigometric functions)
COS(A), cosine
TAN(A), tangent
ASIN(A), arcsine, in radians
ACOS(A), arccosine
ATAN(A), arctangent
SINH(A), hyperbolic sine
COSH(A), hyperbolic cosine

TANH(A), hyperbolic tangent
DEGREES(A), convert radians to degrees
RADIANS(A), convert degrees to radians

Operations on lists

LOW(A;B;C;…), lowest value in the list
HIGH(A;B;C;…), largest value in the list
MEDIAN(A;B;C;..), median of a list
MEAN(A;B;C;…), mean value of a list
SUM(A;B;C;…), sum of a list, i.e., A+B+C+…
PRODUCT(A;B;C;…), product of a list, i.e., A*B*C*…
LOOKUP(X;A;B;C;D;…), value in the X position of the subsequent list, A;B;C;D;…
    e.g., LOOKUP(3;B;C;D;E;F) = D
CHOOSE(A;B;C;…), a randomly chosen value from the list
SCHOOSE(S;A;B;C;D;…), a seeded randomly chosen value from the list, with seed S
FIND(X;A;B;C;D;…), the location (1-based index) of value X in the list A;B;C; … . Returns 0 if not found.
COUNT(A;B;C;…), the number of values in a list
RETURNLIST (N;A;B;C;D;…), places the list values (A;B;C; …) into the PPS(N) array (replacing all prior values) and returns the number of values in the list
PERCENTILE(P;A;B;C;D;…), the value in list A;B;C;D; … that is at the P percentile; i.e., P percent of the values fall at or below the result.
    E.g., PERCENTILE(60;1;2;3;4;5) = 3; PERCENTILE(65;1;2;3;4;5) = 4
PRANK(X;A;B;C;D;…), the percentile (rank) of value X in the list A;B;C;D; interpolated for values of X not in the list.
    E.g., PRANK(3;1;2;3;4;5) = 50; PRANK(3.2;1;2;3;4;5) = 54
TIMESERIES(A;B;C;…), the list item that is at location given by the current value of Y (year), returns the last value if year > number of elements in the list.
    Note that the value of a TIMESERIES can be obtained through a special use of the doEval method (see above), such that doEval(x) returns the x item in the sequence.

Operations to create or modify lists
    Note that these functions can’t stand alone in an *EVALUATOR*, because they do not return a single value. Instead, they must be used within a function that operates on a list – see above. For example, an *EVALUATOR* of “=SORT(1;3;5)” is not valid, but “=FIND(3;SORT(1;3;5;4;2))” is valid and returns the answer 3.

FILE(filename), creates a list from the values in a text file
LIST(A), creates a list from the values from the PPS(A) values
SEQUENCE(A;B;C), creates a list with values from A to B (inclusive), by increments of C
SORT(A;B;C;…), sorts the list from low to high
SORTREV(A;B;C;…), sorts the list from high to low
REVERSE(A;B;C;…), reverses the order of the list

SUBSET(A;B;C;D;E;…), creates a list from C; D; E; …, starting with the A position and ending at the B position
SAMPLE(N;A;B;C;…), creates a new list of N items randomly chosen from A; B; C; …
SSAMPLE(S;N;A;B;C;…), creates a new list of N items randomly chosen with seed S
APPEND(X;A;B;C;…), appends X to the end of the list
PREPEND(X;A;B;C;…), inserts X at the beginning of the list
INSERT(X;N;A;B;C;…), inserts the value X at the (1-based) position N
REMOVE(X;A;B;C;…), removes first item equal to X from the list
REMOVEALL(X;A;B;C;…), removes all items equal to X from the list
REMOVEAT(N;A;B;C;…), removes the item at the (1-based) position N from the list

A few notes about format:

Variable names and operation names are case-insensitive. (They are always converted to upper case within *EVALUATOR*.)

TRUE = 1 and FALSE = 0 in the code, and the numeric values can be used interchangeably with TRUE and FALSE.

'F' [in single quotes to signify a character, not the variable F] can be used for FEMALE and is coded as (and can be replaced by) 0.
'M' [in single quotes to signify a character, not the variable M] can be used for MALE and is coded as (and can be replaced by) 1.
'E' [in single quotes to signify a character, not the variable E] will be translated into the base of the natural logarithms.
PI is translated into 3.1415…

Numeric format: The program will assume that floating point numbers are formatted according to the Region settings on the computer. E.g., in the USA, 31/10 = 3.1; whereas in most of Europe, 31/10 = 3,1.

The comma (',') should not be used as a delimiter to separate parameters, because of its meaning as a decimal delimiter in most regions. Instead the semi-colon is used to separate parameters being passed to functions. Thus, the correct format is, e.g.,
MAX(A;B), *not* MAX(A,B)
POW(C;10), *not* POW(C,10)

Parentheses enclosing sets of parameters or defining order of operations can be interchangeably designated by '(', '[', or '{', and ')', ']', or '}'. Although the choice of brackets or braces makes no difference to the program, it can be useful to alternate among them to help with the readability of functions. For example, the following are equivalent, but the second one might be easier to read.
```
SRAND((A+B)*C)+5^(SQRT((D+E)*LOG(ABS(F)))+G)
SRAND[(A+B)*C]+5^{SQRT[(D+E)*LOG(ABS[F])]+G}
```

Using *MMMACRO*s in *METAMODEL MANAGER*

MMMACRO is an extension of *EVALUATOR*, allowing evaluation of short programs ("methods", "scripts", or macros) that consist of assignments of variables using *EVALUATOR* functions, within simple (Basic language-like) program flow.

To add an *MMMACRO* as a sub-model to your metamodel, on the Set System Level and Modifier Programs screen, you click on "Add Modifier Model" and then chose *MMMACRO* from the dropdown list of available models. You then specify the Project File that contains your *MMMACRO*.

The *MMMACRO* file can contain a macro that modifies GSVs, one that modifies PSVs, and one that modifies ISVs, or any subset of the three types. The macro for modifying GSVs will be run once when the *MMMACRO* step is processed in the metamodel, the macro for modifying PSVs will be evaluated for each population, and the individual macro will be evaluated for each individual. Each of the three macros must specify which state variables will be "shared" with the metamodel, optionally what values to assign to the shared variables when the *MMMACRO* is invoked, and a series of steps that can include assignment functions and program flow statements.

An example of a simple *MMMACRO* file is given below. In it, the GSV labeled AREA is decreased by 5 every fifth year; the PSV K is set to 1.5 * AREA, and MORT is reduced by 1 every year; and the ISV REPRO is an exponentially declining function of INBR.

```
SharedGlobalVars=AREA
Vals=100
* Any line starting with a '*' is a comment and is ignored by MMMACRO
# Comments can also be started with '#'
RATE=-5
AREA=AREA+(YEAR%5=0)*RATE
RETURN
SharedPopVars=K;MORT
Vals=150
K=AREA*1.5
RATE=-1
MORT=MORT+RATE*YEAR
RETURN
SharedIndVars=INBR;REPRO
Vals=0;75
REPROINIT=75
B=3.14
REPRO=REPROINIT*EXP(-B*INBR)
RETURN
```

The "Vals=" lines in a *MMMACRO* are optional, and they specify what values should be assigned to the shared state variables at the start of each iteration (with the order of the values corresponding to the order of the shared variable names). If the "Vals=" is omitted for a shared variable, *MMMACRO* the first instance of the macro (like all others during the simulation) will start with whatever value the state variable had from assignment in the other sub-models. Note that at least one sub-model or *MMMACRO* must assign the initial value to each state variable. (Actually, if no component of the metamodel assigns an initial value to a state variable, it will start at -999.) If *MMMACRO* does assign an initial value to a variable, that will over-write whatever initial values were assigned by the System level model or Modifier models that run prior to the *MMMACRO* in the metamodel steps. The RETURN statements at the end of each macro are optional, but it is a good practice to use them.

It is important to understand the distinction between Shared variables (those that can be modified from the outside, and can be read by outside programs) and Local (or private) variables (those that are used only within that macro). Note that private variables, even with the same name, do not carry across from Global, to Population, and to Individual macros.

Global shared variables automatically are carried down to population and individual macros within the same *MMMACRO*. Similarly, Population shared variables automatically available to an individual macro that follows the population macro.

The following example is a more complex macro that illustrates some of the program flow statements available in *MMMACRO*s.

```
SharedGlobalVars=GS1;Joe
Vals=5;7
A=10+RAND
PS1=6
* Note that PS1 is not a shared variable in this macro,
* so PS1 is treated as internal to the macro and
* unrelated to a PS1 that might exist in the metamodel
B=A*2
C=2
if (RAND<0.5)
C=3
endif
X=1
do
X=X+2
PS1=PS1+X
while (X<5)
GS1=A
Return
```

Note that an *MMMACRO* can use variables (such as A, B, C, X, and PS1 in the above example) that are internal to the macro and not "shared", and therefore will not be saved and returned to MMM. In addition, it is not required that shared variables be initialized with an initial Values list, if they are initialized within the *MMMACRO* itself, or by another sub-model.

***MMMACRO* program flow statements available**

The following means of controlling program flow through the *MMMACRO* are available (with examples given for each):

IF/ELSE/END – a block of commands that execute if the condition is true. The optional use of an ELSE allows specification of an alternative block of commands that execute if the condition is false.

```
A=1
IF(A%2=1)
B=A*2
ELSE
B=A*3
END
RETURN
```

Note that IF is also used as an in-line *EVALUATOR* function for assignment of values to variables. The use of IF as a *MMMACRO* control command vs an operation within an equation is determined by the context. For example, the following *MMMACRO* has the same result as the previous one.

```
A=1
B=IF(A%2=1;A*2;A*3)
RETURN
```

RETURN (synonymous with QUIT and EXIT) – closes a *MMMACRO*, and can be used to terminate execution of further steps. Optional at the end of the *MMMACRO*.

```
A=10
IF(A%2=1)
RETURN
END
B=A*3
RETURN
```

DO/WHILE – a loop that iterates until the closing WHILE condition is false.

```
A=1
DO
A=A+2
WHILE (A<10)
RETURN
```

DOWHILE/END – a loop that iterates as long as the condition specified in the opening DOWHILE is true.

```
A=1
DOWHILE(A<10)
A=A+2
END
RETURN
```

CONTINUE (synonymous with LOOP) – returns to the top of the DO/WHILE or DOWHILE loop

```
A=1
B=2
DOWHILE(A<10)
A=A+2
IF(A<B)
CONTINUE
END
A=SIN(C)
END
RETURN
```

Note that each END statement applies to the most nearest IF or DOWHILE statement above it that was not already matched by a previous END statement. ENDIF and ENDWHILE can be used to make it clearer to which block an END refers (but the ENDIF and ENDWHILE do not override the syntax rule that every END always applies to the previous non-ENDed block).

BREAK – goes to the statement following the innermost current DO/WHILE or DOWHILE loop

GOTO – goes to the line number specified

```
A=RAND
IF(A>0.5)
GOTO 6
END
A=RAND
B=A*5
RETURN
```

Note that line numbers are not entered, but the GOTO command sends control to the step that occurs at the specified line of the macro.

LIST(n)=… -- Assigns the list defined (with functions such as SEQUENCE or FILE) on the right-hand side to a list labeled LIST(1), or LIST(2), etc. This list can then be used within functions in further steps of the macro.

MAKELIST(A;B;C; …) – Makes a list from values A;B;C; … This is provided to facilitate the use of LIST(n) = MAKELIST(A;B;C;…). [Simply LIST(n) = (A;B;C;…) will not work because the right-hand side of the equation is not a well-formed expression.] MAKELIST has no usefulness in other contexts, because the simple list (A;B;C;…) can be used as the parameters passed to other functions that call for a list.

Testing *MMMACRO*:
MMMacroEntry.exe is a little interface program that allows you to build and test an *MMMACRO*. *MMMACROENTRY* can read an *MMMACRO* file or facilitate building and editing a file in a simple text grid.

New *MMMACRO*, VMacro, and *EVALUATOR* commands for using R scripts

Functions ("*EVALUATOR*"s) and Macros (*VMACRO* or *MMMACRO*) in *VORTEX* and *MMM* can call R scripts to process data and provide results that can be used in input or as output descriptors.

When any R script or command is run within a function of macro in *VORTEX* and *MMM*, an instance of the R engine will be started if one wasn't already started by the program. The same R engine remains in use (i.e., it is not closed and then reopened) until the program (*VORTEX*, *EVALUATOR*, *MMM*, etc.) closes. This means that a script can be run to do some calculations, and then the working environment and all of its data remain accessible to further R scripts.

Methods available in *EVALUATOR* functions and *MMMACRO*s for running R scripts:

RMACRO(rscriptfile) -- runs the R script in the file. Note that normally this is where you would put a command to load an R library to be used for later commands. In here, you can also put other start-up commands that might be needed. Rscriptfile must be passed as a string in quotes, e.g., RMACRO("C:/WorkingFolder/myRfile.txt"). The user would normally need to run such a script when a scenario starts running, in order to load the necessary R libraries.

REVAL(command) – evaluates the R command and returns a double (if the R command returns anything). The R command must be a quoted string, e.g., REVAL("newdata<-data.frame(A=3,B=4,C=5)"), in order to avoid confusion between characters in the R command and delimiters and operators in the function.

REVALGS(command) – creates a data.frame called RDATA with all the current GSVs and the variables A to Z, and then evaluates the R command. The data.frame labels the variables by their labels, not by GS1, GS2, etc. (unless those are also used as the labels).

Labels will be given in upper case when assigned to RDATA, so reference to them in the R command must also use upper case. Returns whatever the R command returns.

REVALPS(command) – creates a data.frame (RDATA) with the current GSVs, PSVs, and the variables A to Z, and then evaluates the R command.

REVALIS(command) – creates a data.frame (RDATA) with the current GSvars, PSvars, ISVs, and the variables A to Z, and then evaluates the R command.

REVALVARS(command) – creates a data.frame (RDATA) with the current variables A to Z (plus any further VAR1, VAR2, etc.) and then evaluates the R command.

REVALVALS(command; x1; x2; x3; …) – creates a data.frame (RDATA) with the values x1, x2, x3, etc. and then evaluates the R command. The data.frame labels for these variables will be VAR1, VAR2, VAR3, etc.

When writing the R commands above, the user will need to be cognizant that the data.frame created by the above commands is always called RDATA, although other data.frames can be created with REVAL or RMACRO. For example, after evaluating an REVALGS (which will create a data.frame with the GSvars), the statement REVAL("GSDATA<-RDATA") could be used to save those data for later use together with PSVs in an REVALPS.

A few more things to be aware of …

R commands are case sensitive, whereas *EVALUATOR* commands are not. E.g., reval("GSDATA<-RDATA") is the same as REVAL("GSDATA<-RDATA"), but these are not the same as REVAL("gsdata<-rdata").

R commands are passed as quoted strings to REVAL, e.g., as REVAL("GSDATA<-RDATA"), *not* as REVAL(GSDATA<-RDATA), and this means that the R command can contain symbols that would normally cause problems for *EVALUATOR* (e.g., might be interpreted as operators or delimiters) if they were outside of a quoted string. (E.g., to *EVALUATOR*, GSDATA<-RDATA, without quotes, would be a test of whether a variable called GSDATA was less than the negative of a variable called RDATA.) When an R command contains elements that are in quotes, you must use single quotes rather than double quotes within the R command, so that the (single!) quotes within the R command do not get confused with the (double!) quotes that encase the R command.

Appendix VI. Template for open data exchange via *METAMODEL MANAGER*

We hope that in addition to the demographic, animal movement, and disease spread program suite built by contributors to *METAMODEL MANAGER*, you will find opportunities to build your own programs in conjunction with MMM. If you choose to do so, your program must be developed in a way that is compatible with the MMM. You can use the information and template described below to help set up the coding for your own programs so that they are compatible with the *METAMODEL MANAGER* and can share system, population, and individual data with other linked programs. We won't pretend that those who are new to programming will be able to write computer code for a model that is then linked to other models via *METAMODEL MANAGER*. You need to be experienced in a computer language such as C#, Visual Basic, Delphi, or Java to be able to write the functions necessary to have programs developed in these languages access data in MMM. (However, even if you have no expertise programming in such languages, you can write simple macros or scripts – as you might in Excel or for a statistical package – to carry out relatively simple data manipulations using the *MMMACRO* or *EVALUATOR* utilities that are included with MMM. *MMMACRO* handles all the data exchange with MMM for you, so you do not need to know how it or other programs interface with MMM.) *METAMODEL MANAGER* is written in C#, and you will find the source code in the SourceCode subfolder of the program folder where *METAMODEL MANAGER* is installed.

To add your own System or Modifier models to the toolbox available to *METAMODEL MANAGER*, you will need to create a .NET compatible dynamically linked library (dll) with functions that provide the interface through which to access the Global, Population, and Individual data contained within class MData. The standard function calls that *METAMODEL MANAGER* uses to communicate with any linked model are Initialize(), Simulate(), optionally a Year0Sim(), and Close(), although these three functions can be given different names if the synonymous names are specified to *METAMODEL MANAGER* in the model specifications user interface. Initialize() carries out any work that is needed to start a sub-model running. This often includes reading input values from a project file, notifying MMM what state variables will be used by the model, some memory management, initializing arrays for output data, and creating output files. For System level models, Initialize() is also where the initial creation of the starting populations will often occur. Simulate() contains the work that is done each time that MMM hands control of the metamodel to the sub-model. This will normally require first accessing the current values of state variables, populations, and individuals from MMM, performing a time step of the sub-model simulation, and then passing updated values of state variables and individuals back to MMM. Year0Sim() is optionally available as a method that is called before the first cycle through the simulation. This could include any further initialization of the sub-model that might require that all sub-models have previously been initialized and have added their state variables to the metamodel. For example, a System sub-model might require that Modifier models have defined their state variables before the System model creates the initial populations of individuals. The Close() method completes any work that is necessary when the simulation has finished. This might include final calculations of summary output, closing of output files, and freeing memory. It can also be an empty method if all work to finish calculations, close files, and free memory is done after each time step (or done when the sub-model runs the Simulate() method for the last time step of the simulation). The parameters that are passed to Initialize(), Year0Sim(), and Simulate() are determined by MMM at run time from the function definitions in the dll. See the code files for examples.

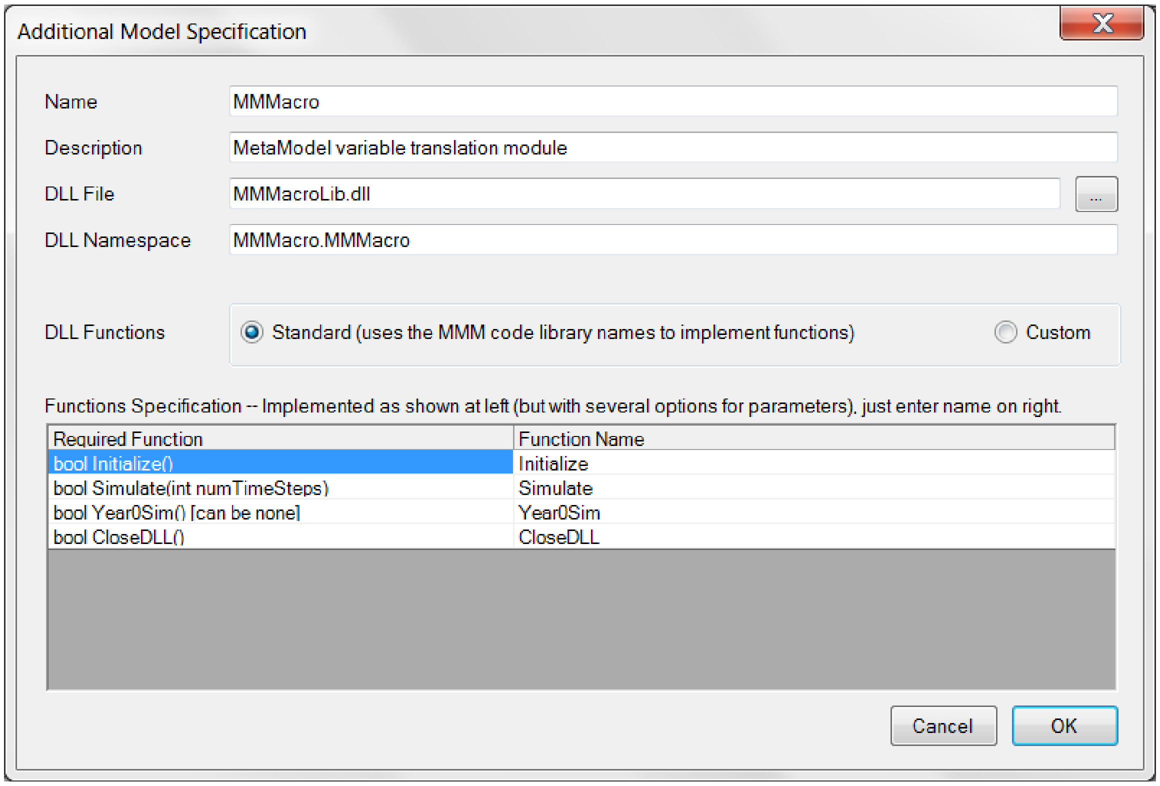
The structure of the metamodel data in MData is defined in the small library MMMCore, and usually the sub-model to be linked to MMM will need to use the MMMCore.dll library. The C# code for the MMMCore.dll is provided in the *METAMODEL MANAGER* installation file.

Another small code file – MMLinkTemplate.cs – is included to provide a template for the three key functions for passing data between *METAMODEL MANAGER* and a user-provided model. In addition, the code for the *MMMACRO* modifier program which provides a means to modify variables in MData via macros or scripts is provided to show a more complete and complex example.

Your program will appear on the program list on your computer if you set it up to do so. It will only be resident on your computer. This is done via the Manage Loaded Apps option on the opening screen of MMM. When you Manage Loaded Apps, you have two options to add a new program to the list of sub-models available to your metamodels. You can load the sub-model's program specifications from a small xml file. The following is the xml file used by MMMacro:

```
<MAppSpecs>
  <Name>MMMacro</Name>
  <Description>Metamodel variable translation module</Description>
  <DLL>MMMacroLib.dll</DLL>
  <NameSpace>MMMacro.MMMacro</NameSpace>
  <Variables>
  </Variables>
  <Functions>
    <Standard>True</Standard>
  </Functions>
</MAppSpecs>
```

It is easier, however, to use the 'Add' button to add the specifications from the user interface, in which case MMM will create the xml specification file for you. Figure 17 is the window that opens to allow you to specify the library methods that will link your model to MMM, with the specification for MMMacro shown. You can use the standard names for Initialize(), Simulate(), Year0Sim(), and Close() in your program, or you can specify different names for these functions in this screen.



Name: MMMacro

Description: MetaModel variable translation module

DLL File: MMMacroLib.dll

DLL Namespace: MMMacro.MMMacro

DLL Functions: [✓ Standard (uses the MMM code library names to implement functions)] Custom

Functions Specification -- Implemented as shown at left (but with several options for parameters), just enter name on right

Required Function	Function Name
bool Initialize()	Initialize
bool Simulate(int numTimeSteps)	Simulate
bool Year0Sim() [can be none]	Year0Sim
bool CloseDLL()	CloseDLL

Figure 17. Additional Model Specification.

After you have added the specifications for your model to MMM, your model will show up on the dropdown lists of System or Modifier models available to you when you create a new metamodel. Other users who want to use your model in their metamodels will need to repeat the above steps, or just get the specification xml that you created and load it into their *METAMODEL MANAGER* by Manage Loaded Apps. On each computer, any xml specification file created by MMM and the lists of your added apps (in files called apps.l and sysapps.l) will be located in the user's local AppData folder (usually C:/Users/[user]/Local/AppData/Species Conservation Toolkit Initiative/MeMoMa). If the lists of loaded apps gets corrupted (as can happen if you start to add another app, but don't complete the process), go into the AddData folder, edit or delete the sysapps.l and apps.l files, and then if necessary add your models again to your MMM.

If you think your program may have broader utility to the conservation community, then we encourage you to make your program available for others to use. (Provide them with the dll, the xml specification file, documentation, and examples!) The metamodeling tools website (https://scti.tools) will provide downloads for relevant software that developers are willing to distribute freely, and will provide links to other sites for programs that charge a fee or are otherwise restricted in their use.