L A B

8 Model Verification and Validation

Dew knot trussed yore spell chequer two fined awl yore mistakes.
—Brendan Hills

In this lab we describe the verification and validation phases in the development and analysis of simulation models. In Section L8.1 we describe an inspection and rework model. In Section L8.2 we show how to verify the model by tracing the events in it. Section L8.3 shows how to debug a model. The ProModel logic, basic, and advanced debugger options are also discussed.

L8.1 Verification of an Inspection and Rework Model

Problem Statement

Bombay Clothing Mill is a textile manufacturer, importer, and reseller. Boxes of garments arrive (three per hour, exponentially distributed) from an overseas supplier to the Bombay Clothing mill warehouse in Bombay for identification and labeling, inspection, and packaging (Figure L8.1). The identification and labeling take U(10,2), inspection takes N(5,1), and packaging takes N(4,0.25) minutes. At times the labeling machine malfunctions. The labeling mistakes are detected during inspection. Sixty percent of the garments are labeled properly and go on to the packaging area; however, 40 percent of the garments are sent back for rework (relabeling). Garments sent back for rework have a higher priority over regular garments at the labeling machine. Movement between all areas takes one minute. Simulate for 100 hours.

Priorities determine which waiting entity or downtime is permitted to access a location or resource when the location or resource becomes available. Priorities may be any value between 0 and 999 with higher values having higher priority. For simple prioritizing, you should use priorities from 0 to 99.

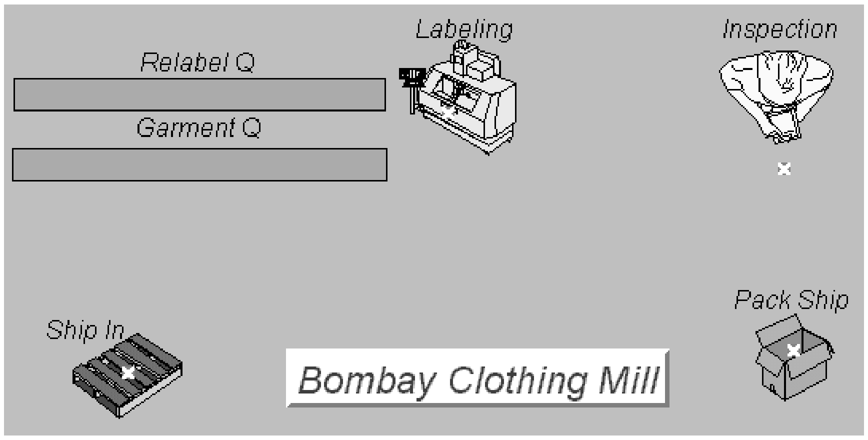
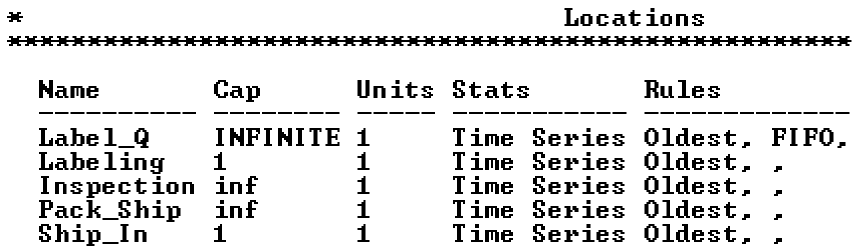


FIGURE L8.1
Layout of the Bombay Clothing Mill.

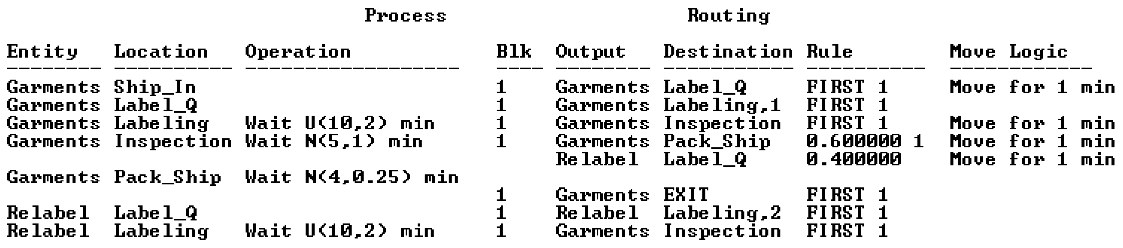


```
*                                         Locations
*******************************************************

  Name        Cap       Units Stats       Rules
  ----------- --------- ----- ----------- -------------
  Label_Q     INFINITE  1     Time Series Oldest, FIFO,
  Labeling    1         1     Time Series Oldest, ,
  Inspection  inf       1     Time Series Oldest, ,
  Pack_Ship   inf       1     Time Series Oldest, ,
  Ship_In     1         1     Time Series Oldest, ,
```

FIGURE L8.2
Locations at the Bombay Clothing Mill warehouse.

FIGURE L8.3
Process and routing tables at the Bombay Clothing Mill.



```
                                Process                              Routing

Entity   Location   Operation             Blk  Output   Destination Rule       Move Logic
-------- ---------- ------------------    ---- -------- ----------- ---------- ------------
Garments Ship_In                          1    Garments Label_Q     FIRST 1    Move for 1 min
Garments Label_Q                          1    Garments Labeling,1  FIRST 1
Garments Labeling   Wait U(10,2) min      1    Garments Inspection  FIRST 1    Move for 1 min
Garments Inspection Wait N(5,1) min       1    Garments Pack_Ship   0.600000 1 Move for 1 min
                                               Relabel  Label_Q     0.400000   Move for 1 min
Garments Pack_Ship  Wait N(4,0.25) min
                                          1    Garments EXIT        FIRST 1
Relabel  Label_Q                          1    Relabel  Labeling,2  FIRST 1
Relabel  Labeling   Wait U(10,2) min      1    Garments Inspection  FIRST 1
```

Five locations (Ship_In, Label_Q, Relabel_Q, Labeling, Inspection, and Pack_Ship) are defined as shown in Figure L8.2. Two entities (Garments and Relabel) are defined next. The processes and routing logic are defined as shown in Figure L8.3. Note that the garments sent back for relabeling have a higher priority of 2 to access the labeling machine location as opposed to a lower priority of 1 for incoming garments.

L8.2 Verification by Tracing the Simulation Model

Now we will run the model developed in the previous section with the Trace feature on. Trace will list events as they happen during a simulation. A *trace* is a list of events that occur during a simulation. For example, a line in the trace could read, "EntA arrives at Loc1, Downtime for Res1 begins." A trace listing also displays assignments, such as variable and array element assignments. A trace listing of a simulation run may be viewed in several ways and is generated in one of two modes, Step or Continuous.

- *Off:* Select this option to discontinue a trace.
- *Step:* Choose this option to step through the trace listing one event at a time. Each time you click the left mouse button, the simulation will advance one event. Clicking and holding the right mouse button while in this mode generates a continuous trace.
- *Continuous:* Choose this option to write the trace continuously to the output device selected from the Trace Output submenu. This is useful when you do not know exactly where to begin or end the trace. Clicking and holding the right mouse button causes the trace to stop until the right mouse button is released.

Run the simulation model. Choose Trace from the Options menu. You can choose either the step mode or the continuous mode for running Trace (Figure L8.4). Choose the step mode for now. The trace output can be sent either to the Window or to a file (Figure L8.5). Let's choose to send the output to the Window.

The trace prompts are displayed at the bottom of the screen as shown in Figure L8.6. Follow the trace prompts and verify that the model is doing what it is supposed to be doing. Make sure through visualization and the trace messages that the rework items are indeed given higher priority than first-time items. Also, from the garment and relabel queue contents shown in Figure L8.7, notice that the relabel queue is quite small, whereas the garment queue grows quite big at times (as a result of its lower priority for the labeling operation).

FIGURE L8.4

Trace modes in ProModel.

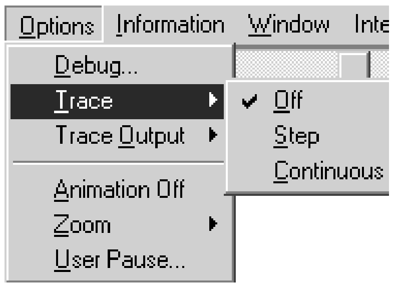


FIGURE L8.5

Trace output options in ProModel.

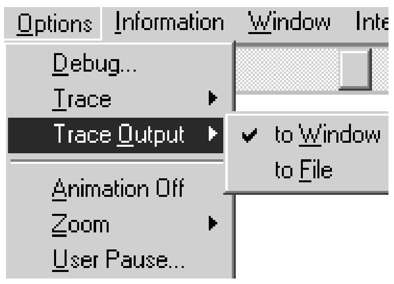
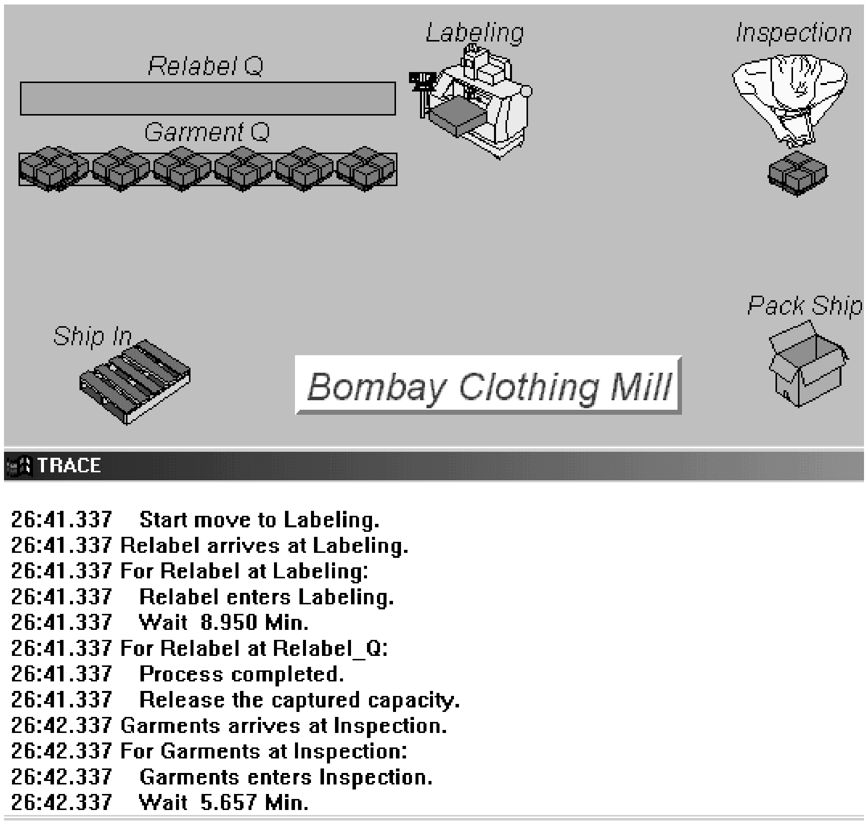


FIGURE L8.6

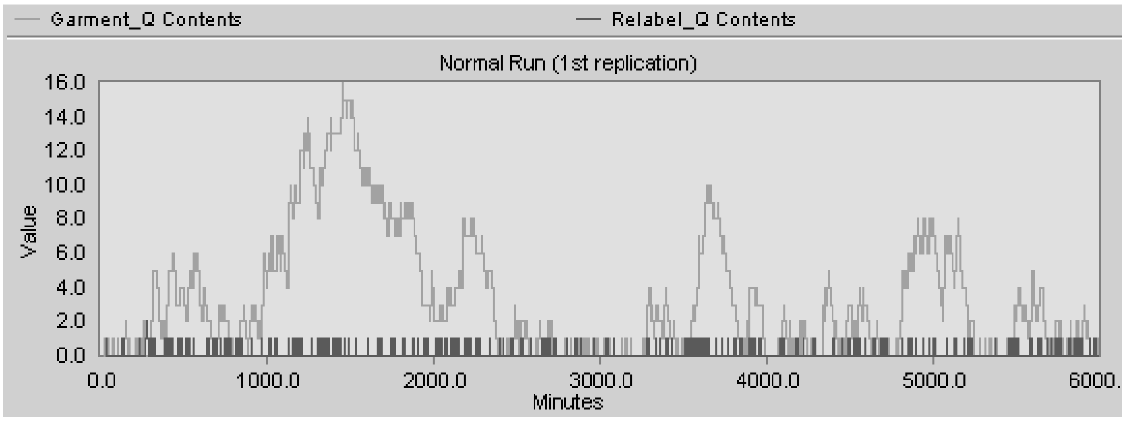
Tracing the simulation model of the Bombay Clothing Mill warehouse.



```
26:41.337    Start move to Labeling.
26:41.337 Relabel arrives at Labeling.
26:41.337 For Relabel at Labeling:
26:41.337    Relabel enters Labeling.
26:41.337    Wait  8.950 Min.
26:41.337 For Relabel at Relabel_Q:
26:41.337    Process completed.
26:41.337    Release the captured capacity.
26:42.337 Garments arrives at Inspection.
26:42.337 For Garments at Inspection:
26:42.337    Garments enters Inspection.
26:42.337    Wait  5.657 Min.
```

FIGURE L8.7

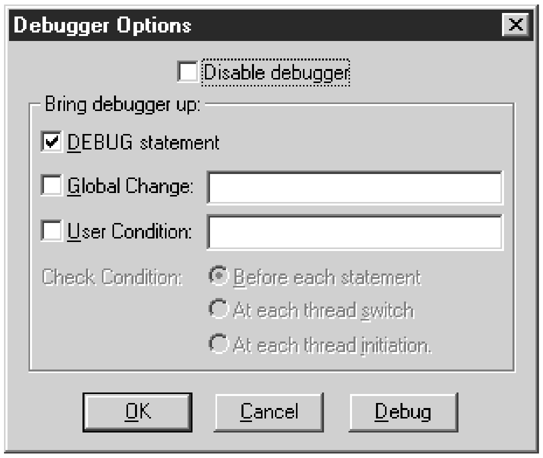
Plots of garment and relabel queue contents.



L8.3 Debugging the Simulation Model

The debugger (Figure L8.8) is a convenient, efficient way to test or follow the processing of any logic in your model. The debugger is used to step through the logic one statement at a time and examine variables and attributes while a model is running. All local variables, current entity attributes, and so forth are displayed, and the user has full control over what logic gets tracked, even down to a specific entity at a specific location (Figure L8.9).

FIGURE L8.8
Debugger options menu.


Debugger Options

[ ] Disable debugger

Bring debugger up:

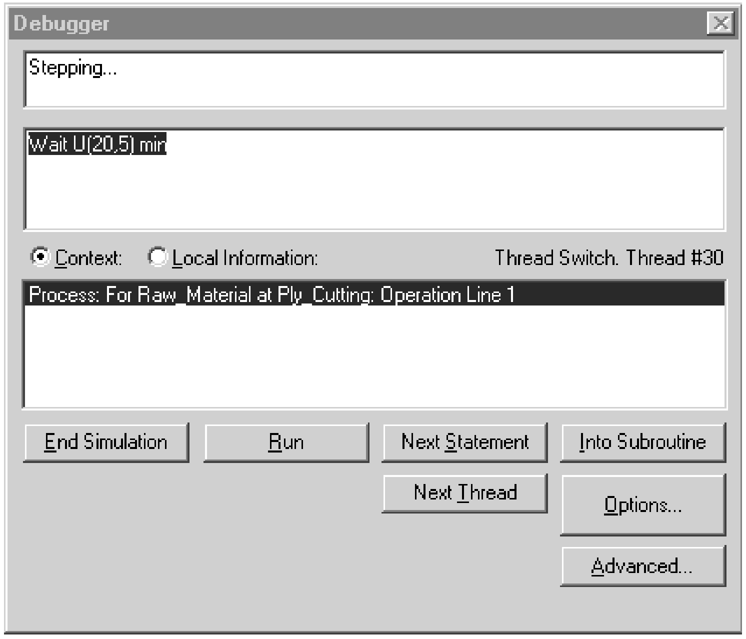
[✓] DEBUG statement

[ ] Global Change:

[ ] User Condition:

Check Condition: [● Before each statement] ○ At each thread switch ○ At each thread initiation.

FIGURE L8.9
The ProModel Debugger menu.


Debugger

Stepping...

Wait U(20,5) min

[● Context:] ○ Local Information: Thread Switch. Thread #30

Process: For Raw_Material at Ply_Cutting: Operation Line 1

The user can launch the debugger using a DEBUG statement within the model code or from the Options menu during run time. The system state can be monitored to see exactly when and why things occur. Combined with the Trace window, which shows the events that are being scheduled and executed, the debugger enables a modeler to track down logic errors or model bugs.

L8.3.1 Debugging ProModel Logic

Before we discuss the details of the debugger, it is important to understand the following terms:

- *Statement:* A statement in ProModel performs some operation or takes some action—for example, JOIN, WAIT, USE, MOVE, and so on (refer to the ProModel Reference Guide for more information on all ProModel statements).
- *Logic:* Logic is the complete set of statements defined for a particular process record, downtime event, initialization or termination of the simulation, and so forth.
- *Thread:* A thread is a specific execution of any logic. A thread is initiated whenever logic needs to be executed. This can be an entity running through operation logic, the initialization logic, a resource running some node logic, downtime logic, or any other logic. Note that the same logic may be running in several threads at the same time. For example, three entities of the same type being processed simultaneously at the same multicapacity location would constitute three threads.

Even though several threads can execute the same logic at the same time in the simulation, the simulation processor can process them only one at a time. So there is really only one current thread, while all other threads are suspended (either scheduled for some future simulation time or waiting to be executed after the current thread at the same simulation instant).

Every time logic is executed, a thread that is assigned a unique number executes it. THREADNUM returns the number of the thread that called the function. This function can be used with the IF-THEN and DEBUG statements to bring up the debugger at a certain process.

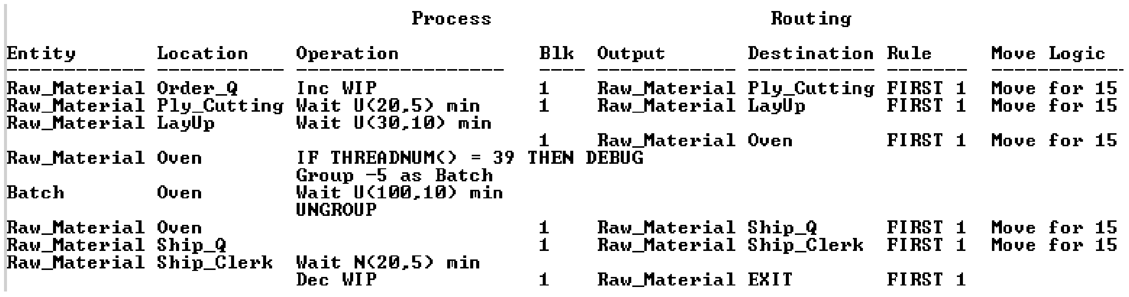
For example, for ElSegundo Composites, from Lab 7, Section L7.6.1, suppose the GROUP quantity has been typed as –5 (typo). A debug statement `IF THREADNUM () = 39 THEN DEBUG` has been added to the operation logic at the Oven location that causes the error, as shown in Figure L8.10. The simulation will run until the proper process and then bring up the debugger (Figure L8.11). The debugger can then be used to step through the process to find the particular statement causing the error.

L8.3.2 Basic Debugger Options

The debugger can be used in two modes: Basic and Advanced. The Basic Debugger appears initially, with the option of using the Advanced Debugger. The Basic

FIGURE L8.10

An example of a DEBUG *statement in the processing logic.*

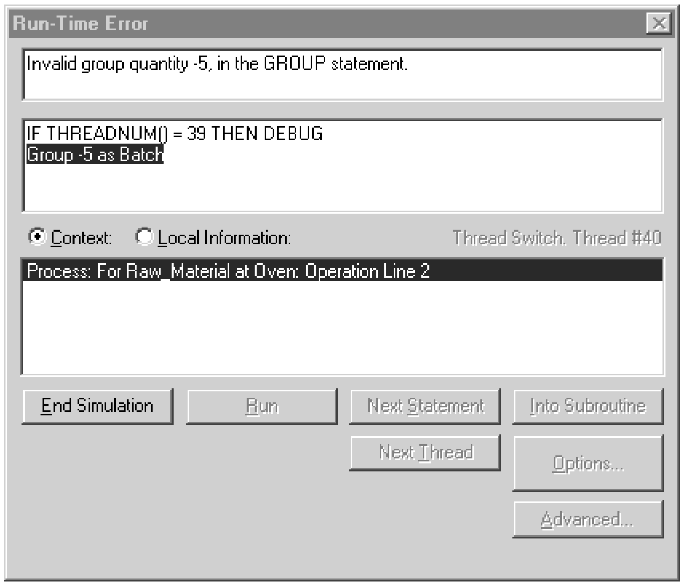


```
                                         Process                               Routing

Entity       Location    Operation            Blk  Output       Destination Rule     Move Logic
------------ ----------- -------------------- ---- ------------ ----------- -------  ------------
Raw_Material Order_Q     Inc WIP              1    Raw_Material Ply_Cutting FIRST 1  Move for 15
Raw_Material Ply_Cutting Wait U(20,5) min     1    Raw_Material LayUp       FIRST 1  Move for 15
Raw_Material LayUp       Wait U(30,10) min
                                              1    Raw_Material Oven        FIRST 1  Move for 15
Raw_Material Oven        IF THREADNUM() = 39 THEN DEBUG
                         Group -5 as Batch
Batch        Oven        Wait U(100,10) min
                         UNGROUP
Raw_Material Oven                             1    Raw_Material Ship_Q      FIRST 1  Move for 15
Raw_Material Ship_Q                           1    Raw_Material Ship_Clerk  FIRST 1  Move for 15
Raw_Material Ship_Clerk  Wait N(20,5) min
                         Dec WIP              1    Raw_Material EXIT        FIRST 1
```

FIGURE L8.11

The Debugger window.



Invalid group quantity -5, in the GROUP statement.

IF THREADNUM() = 39 THEN DEBUG
Group -5 as Batch

Thread Switch. Thread #40

Process: For Raw_Material at Oven: Operation Line 2

Debugger (Figure L8.11) has the following options:

- *Error display box:* Displays the error message or reason why the debugger is displayed, such as the user condition becoming true.
- *Logic display box:* Displays the statements of the logic being executed.
- *Information box:* Displays either the context of the logic or local information.
- *Context:* Displays the module, operation, and line number in which the debugger stopped.
- *Local information:* Displays local variables and entity attributes with nonzero values for the thread in the information box.
- *End Simulation:* Choose this option to terminate simulation. This will prompt for whether or not you would like to collect statistics.

- *Run:* Continues the simulation, but still checks the debugger options selected in the Debugger Options dialog box.
- *Next Statement:* Jumps to the next statement in the current logic. Note that if the last statement executed suspends the thread (for example, if the entity is waiting to capture a resource), another thread that also meets the debugger conditions may be displayed as the next statement.
- *Next Thread:* Brings up the debugger at the next thread that is initiated or resumed.
- *Into Subroutine:* Steps to the first statement in the next subroutine executed by this thread. Again, if the last statement executed suspends the thread, another thread that also meets debugger conditions may be displayed first. If no subroutine is found in the current thread, a message is displayed in the Error Display box.
- *Options:* Brings up the Debugger Options dialog box. You may also bring up this dialog box from the Simulation menu.
- *Advanced:* Changes the debugger to Advanced mode.

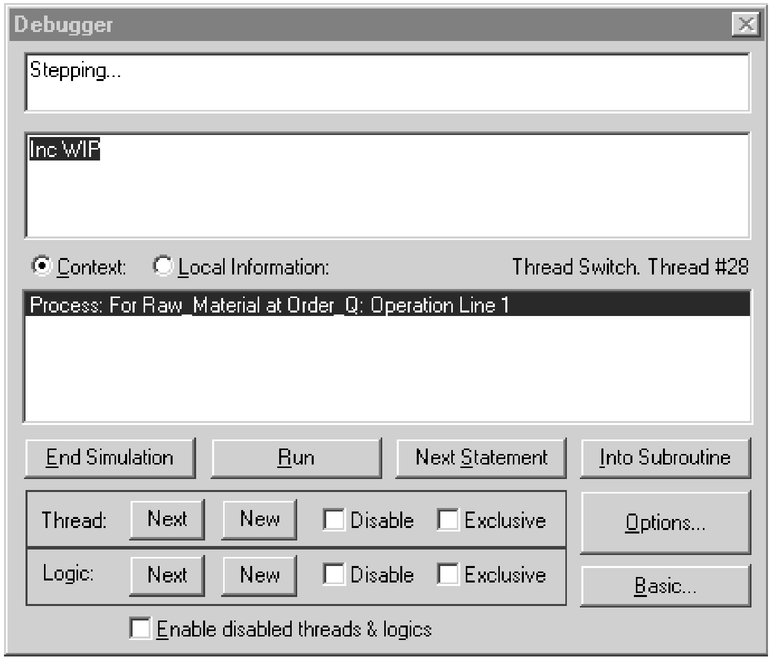
L8.3.3 Advanced Debugger Options

The Advanced Debugger (Figure L8.12) contains all the options in the Basic Debugger plus a few advanced features:

- *Next (Thread):* Jumps to the next thread that is initiated or resumed. This button has the same functionality as the Next Thread button in the Basic Debugger.
- *New (Thread):* Jumps to the next thread that is initiated.
- *Disable (Thread):* Temporarily disables the debugger for the current thread.
- *Exclusive (Thread):* The Debugger displays the statements executed within the current thread only. When the thread terminates, the Exclusive setting is removed.
- *Next (Logic):* Jumps to the next initiated or resumed thread that is not executing the same logic as the current thread.
- *New (Logic):* Jumps over any resumed threads to the next initiated thread that is not executing the same logic as the current thread. This will automatically jump to a new thread.
- *Disable (Logic):* Temporarily disables the debugger for all threads executing the current logic.
- *Exclusive (Logic):* The debugger displays only the statements executed in any thread that are an instance of the current logic.
- *Enable disabled threads and logics:* Enables the threads and logics that were disabled previously.

For more information and detailed examples of how to use the various debugger options, consult the ProModel Users Guide.

FIGURE L8.12
The ProModel Advanced Debugger options.


Debugger

Stepping...

Inc WIP

Context: Local Information: Thread Switch. Thread #28

Process: For Raw_Material at Order_Q: Operation Line 1

End Simulation
Run
Next Statement
Into Subroutine
Thread: Next New Disable Exclusive
Logic: Next New Disable Exclusive
Options...
Basic...
Enable disabled threads & logics

L8.4 Exercises

1. For the example in Section L8.1, insert a DEBUG statement when a garment is sent back for rework. Verify that the simulation model is actually sending back garments for rework to the location named Label_Q.
2. For the example in Section L7.1 (Pomona Electronics), trace the model to verify that the circuit boards of type B are following the routing given in Table L7.1.
3. For the example in Section L7.5 (Poly Casting Inc.), run the simulation model and launch the debugger from the Options menu. Turn on the Local Information in the Basic Debugger. Verify the values of the variables WIP and PROD_QTY.
4. For the example in Section L7.3 (Bank of India), trace the model to verify that successive customers are in fact being served by the three tellers in turn.
5. For the example in Section L7.7.2 (Shipping Boxes Unlimited), trace the model to verify that the full boxes are in fact being loaded on empty pallets at the Inspector location and are being unloaded at the Shipping location.