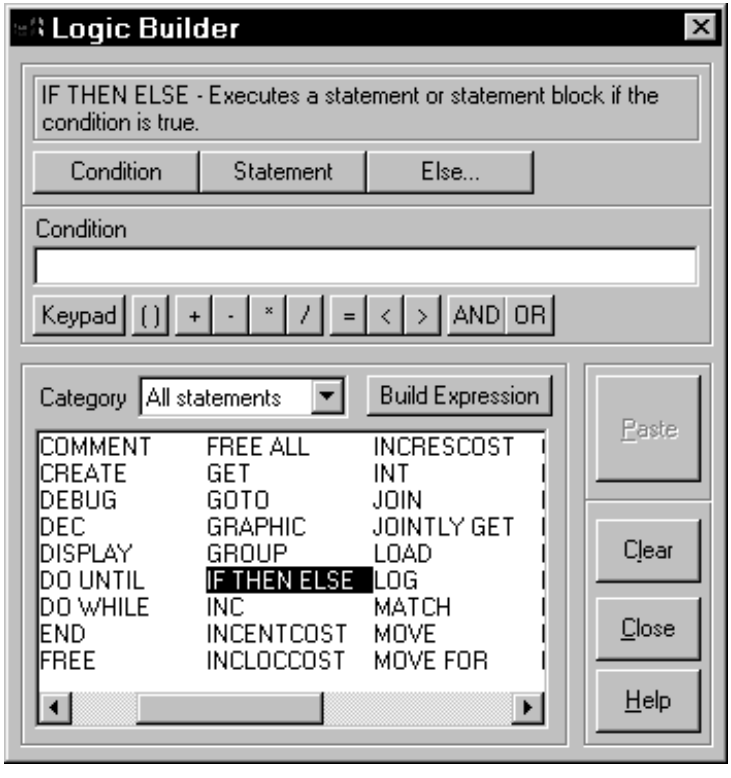
Chapter 9 Building The Logic

CHAPTER CONTENTS

9.1 Logic Builder

The Logic Builder in ProModel provides a quick and powerful way to create valid statements and expressions in logic windows or fields. It takes you through the process of creating statements or expressions, as well as providing point-and-click access to every element defined in your model. The Logic Builder knows the syntax of every statement and function, and allows you to define logic simply by filling in the blanks.



How To **Access the Logic Builder:**

- Click the right mouse button in the logic window or expression edit field. Or click the **Build** button on the logic window's toolbar.

9.1.1 Using the Logic Builder

When the Logic Builder is opened from a logic window, it remains on the screen until you click the Close button or close the logic window or table from which it was invoked. This allows you to enter multiple statements in a logic window and even move around to other logic windows without having to constantly close and re-open the Logic Builder. The Logic Builder closes automatically when pasting to an expression field.

You can move to another logic window or field while the Logic Builder is still up by right-clicking in that field or logic window. The Logic Builder is then reset with only valid statements and elements for that field or window, and will paste the logic you build into that field or window.

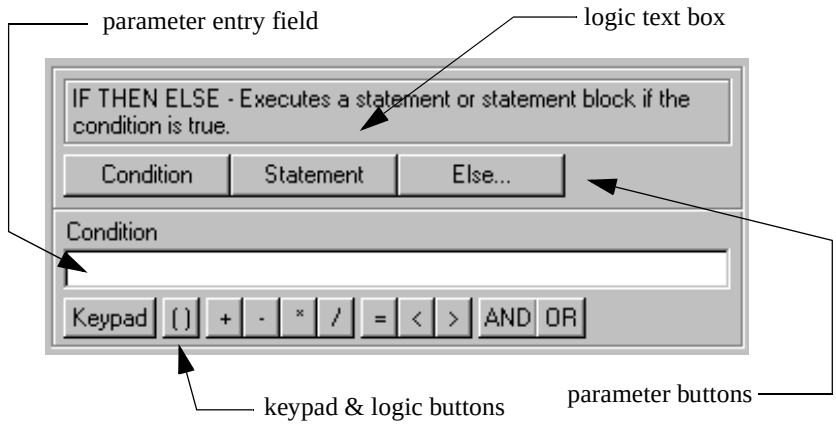
How To **Build a statement or expression:**

1. Right click the mouse in an expression field or logic window to open the Logic Builder or click on the **Build** button in a logic window.
2. Select the statement or expression you want to use from the list box. When opened from a logic window, you have the option to click on the **Build Expression** button to create only an expression.
3. Enter the parameters for the statement or expression. These may be expressions using model elements and/or functions or other statements. Parameters may be edited or entered manually in the Parameter Entry field.
4. Paste the results into the logic field or window by clicking the Paste button.

Logic Builder Components

When invoking the Logic Builder from a logic window, you have the option of building either statements or expressions. Different buttons and lists appear in the Logic Builder as you use the Logic Builder's options depending upon whether you are selecting a statement or building an expression. The Logic Builder shown at the beginning of this section shows a statement being selected for building.

At the top of the Logic Builder is a display (logic text box) of the statement or expression you are building exactly as it will appear after it is pasted into the logic window. A brief description of the selected statement or function is displayed in the logic text box. This description is replaced with the statement or function syntax when you type a parameter, click a parameter or logic button, or double-click on the statement name. Other components of the Logic Builder are as follows:



Parameter buttons Below the logic text box are one or more buttons to control which parameter to enter for the statement or expression. Parameters can be expressions, statements, or functions. These buttons only appear when parameters may be required by the statement, and may change when you select a different statement. The name of the currently selected button appears immediately below the row of buttons and indicates whether or not the parameter is optional.

Parameter entry field This edit field allows you to enter or edit the current parameter. This only appears when parameters may be required by the statement. As soon as something is entered in this field, the Logic Builder switches to build mode to allow selection of functions or elements of the model.

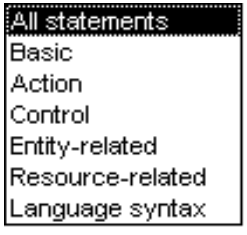
Keypad button Click on this button to display the numeric keypad for entering numbers in the parameter entry field without using the keyboard.

Logic buttons These buttons can be used to insert logical operators and other punctuation in the parameter entry field above. When you click the button, the operator is inserted at the cursor position in the field. A button appears only when the currently selected parameter can use that particular logical operator.

- **Logical & String Operators:**

- **Time Unit Operators:**

Category This combo box allows you to select which type of statements appear in the statement selection list below it. You can select all or a specific type.



Build Expression button This button allows you to create only an expression. It displays the logic elements list (see below) for you to create the expression. An expression consists of a combination of numbers, model elements, and/or functions, but does not include statements.

Statement selection list Choose which statement you wish to use from this list. Only valid statements are displayed for the logic window or field you are using.

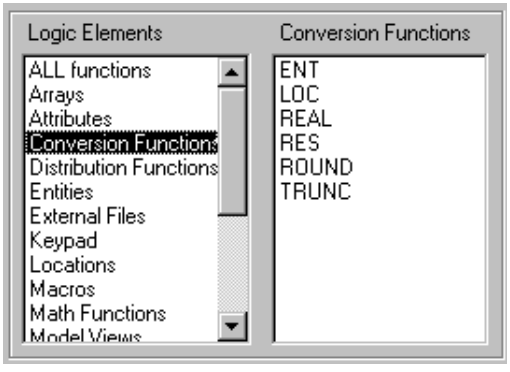
Paste button This button pastes the text of the logic text box into your logic window or field. It is only available once the minimum requirements of the statement or expression have been completed. The Paste button closes the dialog if you are pasting to a field.

Clear button This button clears whatever you have done since the last paste and allows you to start over.

Close button Closes the Logic Builder without pasting the current logic text box.

Help button Launches the context sensitive help system.

Logic Elements When editing an expression in the parameter entry field, the Statement selection list is replaced by Logic Elements. The box on the left lists logic and model element types. The box on the right lists individual selections from the logic or model element type selected.



Return and Cancel Buttons When editing the parameters of a function or nested statement, two additional buttons appear to the right of the parameter edit box: Return and Cancel.

- **Return button** This button (available only when required parameters have been entered) returns to the previous parameter entry field and pastes the function or statement at the last cursor position.

- **Cancel button** Aborts editing of the function or nested statement and returns to editing the previous statement or function.

Selecting Statements

The first thing to do in creating a statement with the Logic Builder is to select the desired statement from the statements list box. You can restrict the list of statements to choose from using the Statement Type combo box above the statement list. If you are just starting to use ProModel, you may want to select the Basic Statements type to list only the most commonly used statements.

To select a statement, left-click on the statement name in the list box. The statement name appears in the logic text box along with a brief description. The parameter buttons also appear just below the logic text box.

Before you begin editing the parameters of the statement, you can select a different statement. However, once you begin defining a parameter, you must click the Clear button to abandon that statement and select another.

Editing Statement Parameters

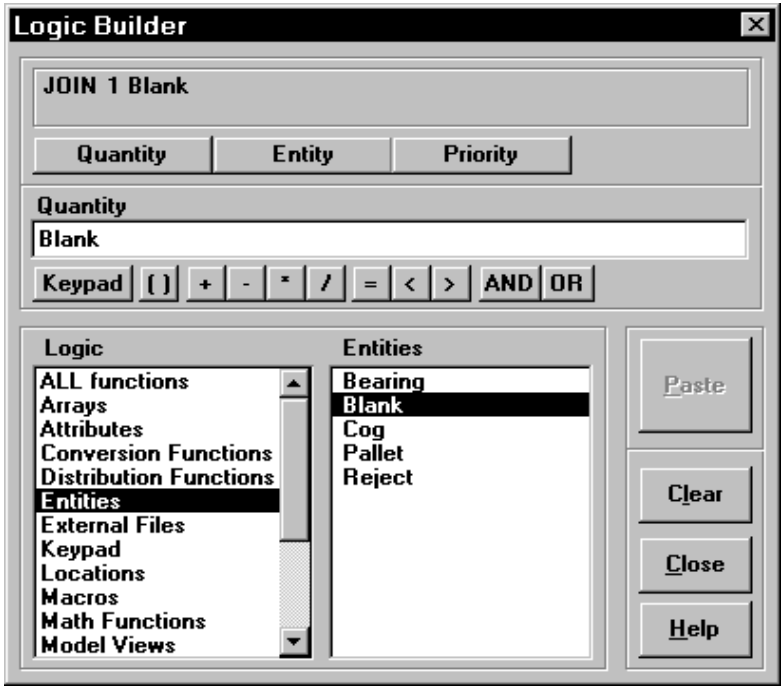
Building a statement is simply a matter of filling in the parameters. To enter a parameter, click the appropriate parameter button below the logic text box. The parameter name is displayed above the parameter entry field. Whatever you type in this field or select from the Logic Elements list replaces the parameter name in the logic text box. The parameter name reappears when the entry field is cleared. The names of optional parameters are not displayed in the logic text box.

An optional shortcut to begin editing the statement's first parameter is to double click on the statement name in the statement list box.

Selecting Logic Elements

The Logic Elements list box, containing model elements and functions, appears with a selection list box to its right. A number pad element is included in the list, which can also be accessed using the Keypad button.

When you click on an item in the Logic Elements list, the model elements or functions related to that item are listed in the selection box on the right. For example, when you click on Entities, the selection list is entitled Entities and it contains all the entities defined in the model.



How To **Place a model element in the parameter entry field:**

1. Left click on the desired element type in the Logic Elements list box. The elements for that type will be placed in the selection list box.
2. Left click on the desired element to paste it into the parameter entry field at its current cursor position. Note that the element is highlighted in the parameter field; clicking on another element will replace the highlighted element.

How To **Place a function in the parameter entry field:**

1. Left click on the desired function type or on All functions in the Logic Elements list box. The functions for that type will be placed in the selection list box.
2. Left click on the desired function to paste it into the parameter entry field at its current cursor position. The Logic Builder jumps into build mode for that function's parameter(s). Note that two new buttons are placed to the right of the parameter edit field: Return and Cancel.

3. To fill in the function's parameter(s), repeat steps one and two. If you do not want to use this function, abandon it by clicking on the Cancel button.
4. Once the function's parameters are complete, click on the Return button. The parameters you just completed will be pasted into the function's parameter entry field, and the completed function with its parameters is pasted into the original statement's parameter entry field.

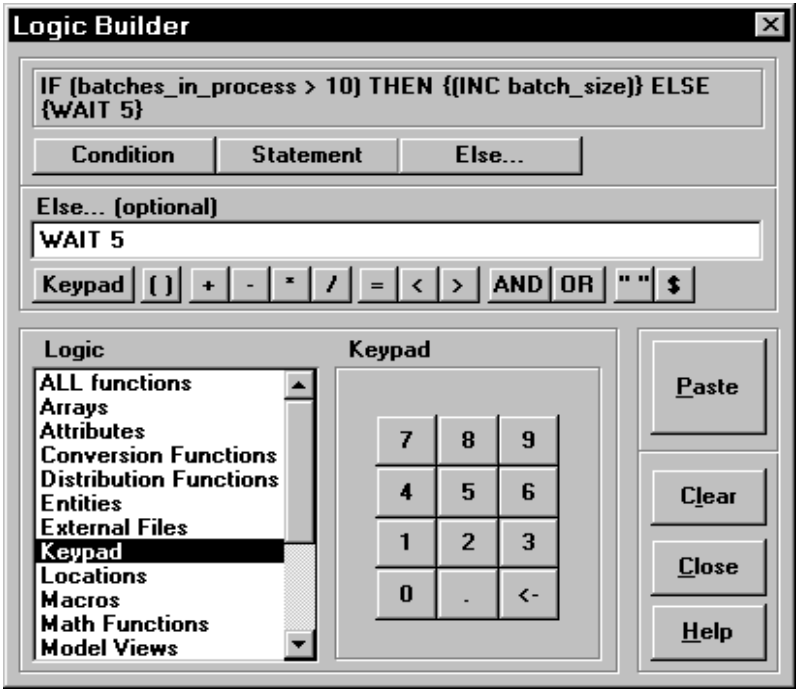
Nested Functions & Statements

When you select a function as the parameter for your statement, you must also define that function's parameter(s). In defining the function's parameter(s), you may use another function which will also require defined parameters. This second function is called a nested function. In addition, a function may be nested within a nested function. Functions can be nested as many levels as you like. In this way, the logic builder helps you create complex expressions that would be difficult to enter manually.

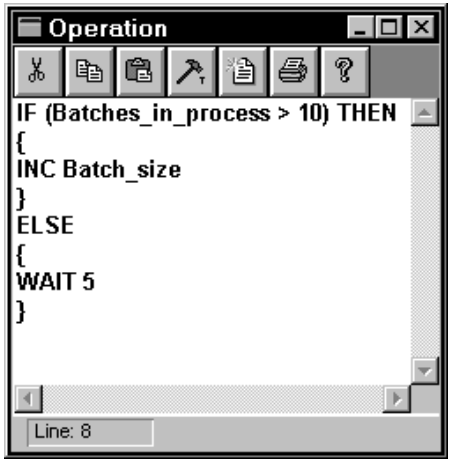
Control statements such as IF...THEN and WHILE...DO contain parameters that are themselves statements. These are called nested statements. For example, an IF...THEN...ELSE statement might look something like this:

```
If (Variable1 > 10) Then
{
    < Statement1 >
    < Statement2 >
}
Else
{
    < Statement3 >
    < Statement4 >
}
```

One or more statements may appear in the block between the curly braces. The Logic Builder allows you to define the first statement of the block. To add other statements to the block, place the cursor in the logic edit window where you want the next statement and use the Logic Builder to write the statement and paste it in. For example, in the following Logic Builder window, an IF...THEN...ELSE statement is being built.



The ELSE statement is built with the parameter button labeled Else and the statement following THEN is built using the button labeled Statement. When you click on the parameter button labeled Statement, the statement list box is displayed where you can select from the list of valid logic statements. Click on paste in the above example and you get:



The parameters of nested statements may be model elements or functions, and within those functions you may have nested functions. This allows you to easily build complex control statements without worrying about syntax and placement of nested statements and functions.

Creating Expressions or Pasting Logic/Model Elements Only

In addition to creating statements, the Logic Builder can also be used to create just an expression or to simply paste a particular element such as a variable or resource name. You may not need to create a complete statement, or the field may not accept statements. Pressing the Build Expression button allows you to build and paste expressions, including individual logic or model element names, into your logic window or field.

The expression being built or element being selected is displayed at the top of the window in the logic text box. Use the parameter edit field to build the expression. You can use model elements and/or functions in your expression. When you are finished, click the Paste button to place the expression or selected logic/model element in the logic window or edit field.

9.2 Operation Logic

Operation logic defines what happens to an entity when it enters a location. Operation logic is optional, but typically contains at least a WAIT statement for the amount of time the entity should spend at the location. For modeling purposes, the exact nature of the operation (machining, inspection, etc.) is irrelevant. What is essential is to know what happens in terms of the time consumed, the resources used, and any other logic that impacts system performance. For operations requiring more than a time and resource designation, detailed logic may need to be defined using IF...THEN or action statements.

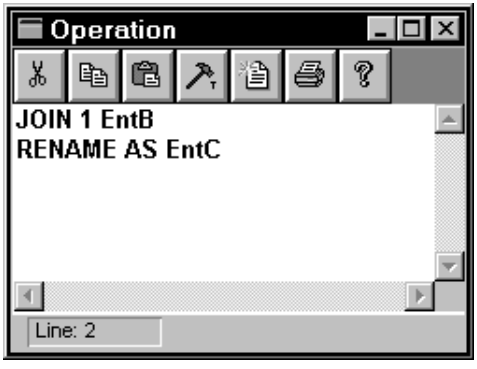
Special operation statements are provided to define the activities that are to occur. By using operation logic, any of the following activities can be defined:

- Detain an entity for a specified length of time while an activity (fabrication, inspection, cleaning) is performed.
- Detain an entity until one or more resources are obtained.
- Detain an entity until one or more additional entities are joined to it.
- Detach one or more entities from the current entity.
- Consolidate one or more entities into a group.
- Separate an entity into two or more entities.
- Detain an entity until a particular system condition is reached.
- Destroy an entity.
- Create one or more new entities.
- Execute a block of logic (assignment of variables, etc.)
- Signal the start of some other action in the system (e.g., place an order).
- Make some decision about further routing (a diverter on a conveyor or a decision point where parts may continue processing or be placed in storage).

Statements can be typed directly into the operation field, or inside a larger logic window after double-clicking in the field or clicking on the Operation button. Alternatively, the Logic Builder can help build logic and is accessed by clicking the right mouse button inside the operation field or logic window. All statements, functions, and distributions available in the operation field are discussed in detail, including examples, in the *ProModel Reference Guide*.

See example on next page

Each entity processes the operation statements defined for it at a particular location, independent of other operations performed on other entities at the same location. The following example presents the operation logic for an entity joining an entity, EntB, and renaming the entity as EntC.



9.3 Preemption Process Logic

With Preemption Process Logic, ProModel makes it possible to control preemption rather than being limited to program defaults and preemption priority levels and values. This feature pertains only to the preemption of entities using a location. It does not include preemption of downtimes.

Normally, if an entity or downtime attempts to preempt an entity that is using a location, the location is immediately preempted (assuming it is preemptable). Once the location finishes the preempting activity, the original preempted entity regains possession of the location and resumes processing where it left off.

To override this default preemption procedure, in the case of entities that are preempted by another entity or a downtime, ProModel allows a Preemption Process Record to be defined which postpones the actual preemption of the location until after the current entity explicitly releases it.

The Preemption Process Record, allows you to control if and when the location is given up. In the case of a preemptive request for a location, the preemption process record allows the entity to be routed elsewhere if desired. If a preemption process is defined, the actual preemption does not occur until the preemption process explicitly frees the location.

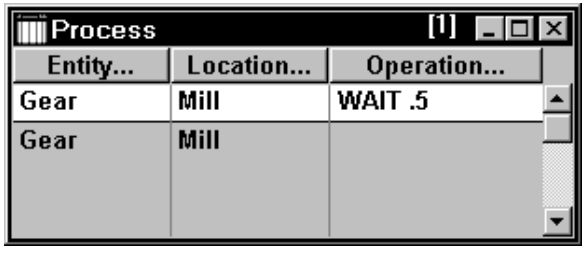
After the location is given up, the entity may (1) elect to use an alternative location to complete the process, or (2) seek to regain access to the same location to complete the process.

While an entity is executing a preemption process, it cannot be preempted by any other entity or downtime.

How To

Create a Preemption Process Record:

1. Using the same entity and location names, create a process record somewhere following the process record where the preemption may occur.



2. Click on the Entity button to display the Entities dialog shown here.

3. Check the Preemption Process option box and click OK.

4. Click on the Operation button to enter the preemption logic. You can use any valid operation logic including delays. It is recommended that you enter a comment as the first line in the logic indicating that this is a preemption process. This will make the record easily identifiable as a preemption process.

Note When a preemption occurs, the entity looks forward and then from the beginning of the process list trying to find a preemption process that matches the same entity and location (a process with ALL as the entity name will match any entity). If a match is found, the preemption process gets executed. Otherwise the default preemption occurs. Only the first preemption process encountered will be executed in the event that multiple preemption processes are defined with the same entity and location names.

Possible Effects of Delayed Preemption

Several circumstances can be created through the use of preemption process records. An entity delaying a preemption, for example, may find at the conclusion of the delay that the preemption is no longer required, or that it faces an even higher preemption priority.

In cases where an entity has multiple locations or resources from which it is choosing, a preemptive request for any one of them is not necessarily a commitment to select that particular location. If any of the alternative locations becomes available, the entity will select it and withdraw the preemption request.

Functions for Defining Logic in a Preemption Process

The Preemption Process Logic feature includes the following functions for use in the preemption logic.

PREEMPTOR() Identifies whether a downtime or entity is making the preemptive request. It returns a 0 if the preemptor is a downtime, otherwise it returns the index number of the preempting entity.

TIMELEFT() Returns the amount of time remaining if the preemption occurred during a WAIT or USE statement. It returns a time value in default time units (real). If multiple entities are preempted from a location, it returns the longest remaining time for all of the entities.

Note The values returned by these functions must be checked before any processing delay occurs since they are updated whenever a preemption takes place. If the values must be referred to later, they should be assigned to the entity's attribute or to a local variable.

Preemption Process Example

In this example, a Gear may be preempted in its occupancy of the Lathe. This preemption may be because of either a preempting entity or downtime. Before the actual preemption takes place, the operation time for the Gear is interrupted and the Gear immediately begins processing the operation logic defined in the preemption process.

In the preemption process, the remaining operation time is stored in Attr1. The Gear routes to Lathe_Backup where it finishes processing. Because the backup lathe is not as efficient as the other Lathe, it takes 50% longer to process the Gear on Lathe_Backup. Therefore, we multiply the time left to process the Gear by a factor of 1.5.

Process Table

Entity	Location	Operation (min)
Gear	Lathe	WAIT 10 min
Gear	Lathe	Attr1=TIMELEFT()
Gear	Lathe_Backup	WAIT Attr1*1.5

Routing Table

Blk	Output	Destination	Rule	Move Logic
1	Gear	Mill	FIRST 1	
1	Gear	Lathe_Backup	FIRST 1	
1	Gear	Mill	FIRST 1	

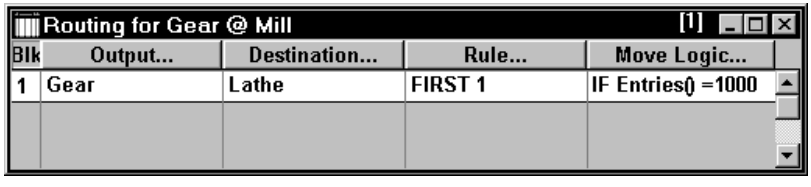
9.4 Routing Move Logic

To accommodate the use of multiple resources and cranes for entity movement, the Move Logic window allows you to define the method of movement as well as any other logic to be executed upon movement.

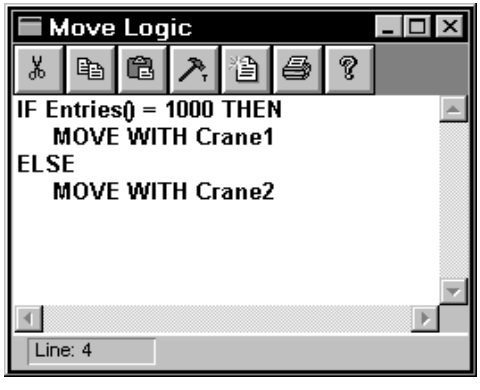
Once the route condition or rule has been satisfied for allowing an entity to route to a particular location, the move logic is immediately executed. The entity does not actually leave the current location until a move related statement (MOVE FOR, MOVE ON, or MOVE WITH) is executed or until the move logic is completed, whichever happens first. This allows the entity to get one or more resources, wait additional time, or wait until a condition is satisfied before actually leaving the location.

Any statements encountered in the move logic after a move related statement are executed after the move is complete, but before the entity actually enters the next location. This is often useful for freeing multiple resources used to transport the entity.

When you access the Processing module in the Build menu, the Routing edit table appears with the Move Logic button in the right hand column as shown here.



When you click the Move Logic the following Move Logic window appears.



This window allows you to manually edit the logic or click on the Build button to use the Logic Builder. It also provides other convenient buttons for editing and printing the move logic.

9.4.1 Move-Related Statements

Admissible statements in the Move Logic window include the new move-related statements listed here. A brief description of how each statement functions in ProModel follows the list. See *Statements & Functions* on page 111 of the *ProModel Reference Guide* for complete syntax, description, and examples of these statements.

```
MOVE FOR <time>
MOVE WITH<res1>{,{p1}{,{p2}{,{p3}}}} {FOR <time>} {THEN FREE}
MOVE ON <path network>
```

MOVE FOR *<time>* Used to specify the amount of time required to move the entity. If the <time> is zero, events for other entities occurring at the same simulation time will be processed before any additional logic is processed for the current entity.

MOVE WITH This statement is used to move an entity using a designated resource, such as a crane or forklift. With the OR operator, you can designate alternative resources for making the move. In this case, the statement captures the first available alternative resource designated in the expression and makes the move. If one of the resources is already owned by the entity, that resource will be used. Please note that you cannot use the AND operator to capture (and move with) more than one resource with this statement. To move an entity with multiple resources, you must use a GET statement to capture the additional resources.

This statement also allows you to set the priority (p1) for accessing the designated resource. If the resource is already owned by the entity, this priority is ignored. And if the resource is a crane, you can set the two additional priorities (p2 for moving to pick up the entity and p3 while moving to deliver the entity) for the crane in situations where the crane might have to compete with another crane for the same bridge space while moving to pick up or deliver the entity.

If the resource is static, a time (FOR <time>) may be specified for the move. The resource used to make the move is only freed if the THEN FREE option is used.

MOVE ON *<path network>* Use this statement to move an entity along a path network. Entities cannot travel on crane networks.

Note Entering no move related statement in the Move Logic window causes entities to move immediately to the next location when the move logic is completed and begin executing the operation logic for that location.

9.4.2 Related Logic Statements

In addition to these routing-specific statements, all statements and functions allowed in exit logic may be used in the Move Logic window. Note, however, that the LOCATION() function returns the *current* location when executed *before* the MOVE statement and returns the *destination* location when executed *after* the MOVE statement. Additional statements permitted before a move related statement include: WAIT, WAIT UNTIL, GET, JOINTLY GET, and USE.

With the use of cranes, three statements accept up to two priorities separated by commas: GET, JOINTLY GET, and USE. The second priority is the priority of the crane if competing with one or more cranes for the same bridge space while moving to the point of use.

See *Statements & Functions* on page 111 of the *ProModel Reference Guide* for more information.

9.4.3 Statement Processing

Statements executed before a MOVE related statement are processed after the entity claims the next location but before the entity actually departs from the current location.

Statements executed after a MOVE related statement are processed after the move has been completed but before the entity enters the location. If there is no move logic, the entity continues processing until it encounters an implicit WAIT. However, if “MOVE FOR 0” is placed in the move logic, the event list is broken and other processes scheduled to occur at the same time are executed. Once these processes are executed, the entity enters the destination location.

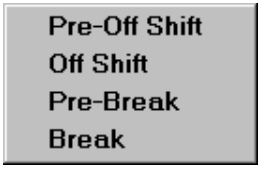
This processing sequence allows you to GET or JOINTLY GET one or more resources before the move and optionally FREE one or more resources at the end of the move. Please note that if a move related statement is not encountered in the logic, an implied MOVE will be assumed and executed at the end of the logic.

9.5 Shift & Break Logic

Shift & Break Logic

Shift and break logic are optional and are defined in four distinct logic windows. Each logic window is executed in a specific sequence throughout the simulation run. You can define logic that controls how resources and locations go off duty or off line and what happens once they are off-line.

To define shift or break logic, click on the Logic button in shift assignments to display a submenu of four shift events for which logic may be defined. Selecting an event from the submenu displays a standard logic window. You can enter separate logic for each of these four events to be executed when the event occurs. See *Sequence of Events* below.



You may want to use the Logic Builder to help you enter the logic. Just click on the Build button in the logic window.

Pre-Off Shift or **Pre-Break Logic** Executed whenever the location or resource is scheduled to go off shift or on break. This occurs before the location or resource is checked for availability, so it is executed regardless of availability. This logic may be used to check certain conditions before actually taking the resource or location off line. The logic is executed for each resource and location listed as members for this shift assignment record. This allows some members to be taken off line while others may be forced to wait. (Pre-off shift and pre-break logic may be referred to in this manual as pre-start logic when speaking of either one.)

Off Shift & **Break Logic** Logic executed at the instant the location or resource actually goes off line.

How To

Sequence of Events

1. When a location or resource is scheduled to go off line due to a break or the end of a shift, the pre-start logic for that particular location or resource is executed.
2. After executing the pre-start logic, which may contain conditional (WAIT UNTIL) or time (WAIT) delays, the location or resource is taken off line, assuming it is either available or the priority is high enough for preemption.

3. At the instant the location or resource is taken off line, the Off-Shift or Break logic is executed.
4. After executing this logic, the location or resource waits until the time defined in the shift file expires before going back on line.

Note If the off-shift and break nodes are not specified in the Resource Specs dialog, the resource will stay at the current node. If no resources or locations are assigned to a shift, the shift is ignored.

Functions And Statements

ProModel uses five functions and statements specifically for off-shift and break logic: SKIP, PRIORITY, DTLEFT(), FORLOCATION(), FORRESOURCE(), and RESOURCE(). Following is a brief description of each. For more details, see *Statements & Functions* on page 111 of the *ProModel Reference Guide*.

SKIP If used in pre-start logic, it causes the off-shift or break time (including any off-shift or break logic) to be skipped so that the location or resource never goes off line. If used in the off-shift or break logic, it causes the off-line time defined in the shift file to be skipped. This allows you to specify a WAIT statement for the off-line time and then SKIP the off-line time defined in the shift file.

PRIORITY This statement provides an alternative way to specify off-shift or break priorities. It also allows the priority to be changed after some time being off-shift or on break. If the priority is changed to a value lower than the current value, the system will check if any preemption may occur at that time. This statement is not allowed in pre-off shift or pre-break logic.

DTLEFT() This function returns the remaining off-shift time based on when the location or resource is scheduled to go back on shift as defined in the shift file. It may be used in off-shift and break logic to adjust the actual time that the location or resource is off-line.

FORLOCATION() This function returns TRUE if the member for which the shift or break logic being executed is a location. This may be followed by a test using the LOCATION() function to determine the precise location.

FORRESOURCE() This function returns TRUE if the member for which the shift or break logic being executed is a resource. The RESOURCE() function may then be used to determine the precise resource if multiple resources are listed as members.

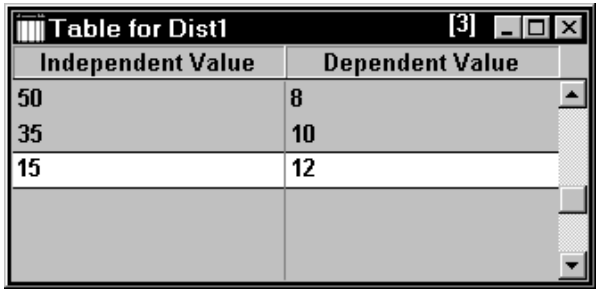
RESOURCE() This returns the name-index number of the resource currently processing the off-shift or break logic.

In addition to these functions, the LOCATION() and DTDELAY() functions are particularly useful when defining off-shift and break logic.

9.6 Arrival Logic

Arrival Logic allows you to perform certain logic as an entity enters the system. It is used primarily for assigning initial entity attribute values. Suppose you process three different types of rods at a manufacturing plant. Each rod has a different length of 8, 10, or 12 inches. Fifty percent of the rods are 8 inches long, 35% are 10 inches long, and 15% are 12 inches long.

The rods have different processing times depending on the length. To differentiate between the different types of rods, we assign an entity attribute called Length to the rods. We define a discrete, non-cumulative user distribution called Dist1 with the following information:



The arrivals logic for the entity called Rod is as follows: