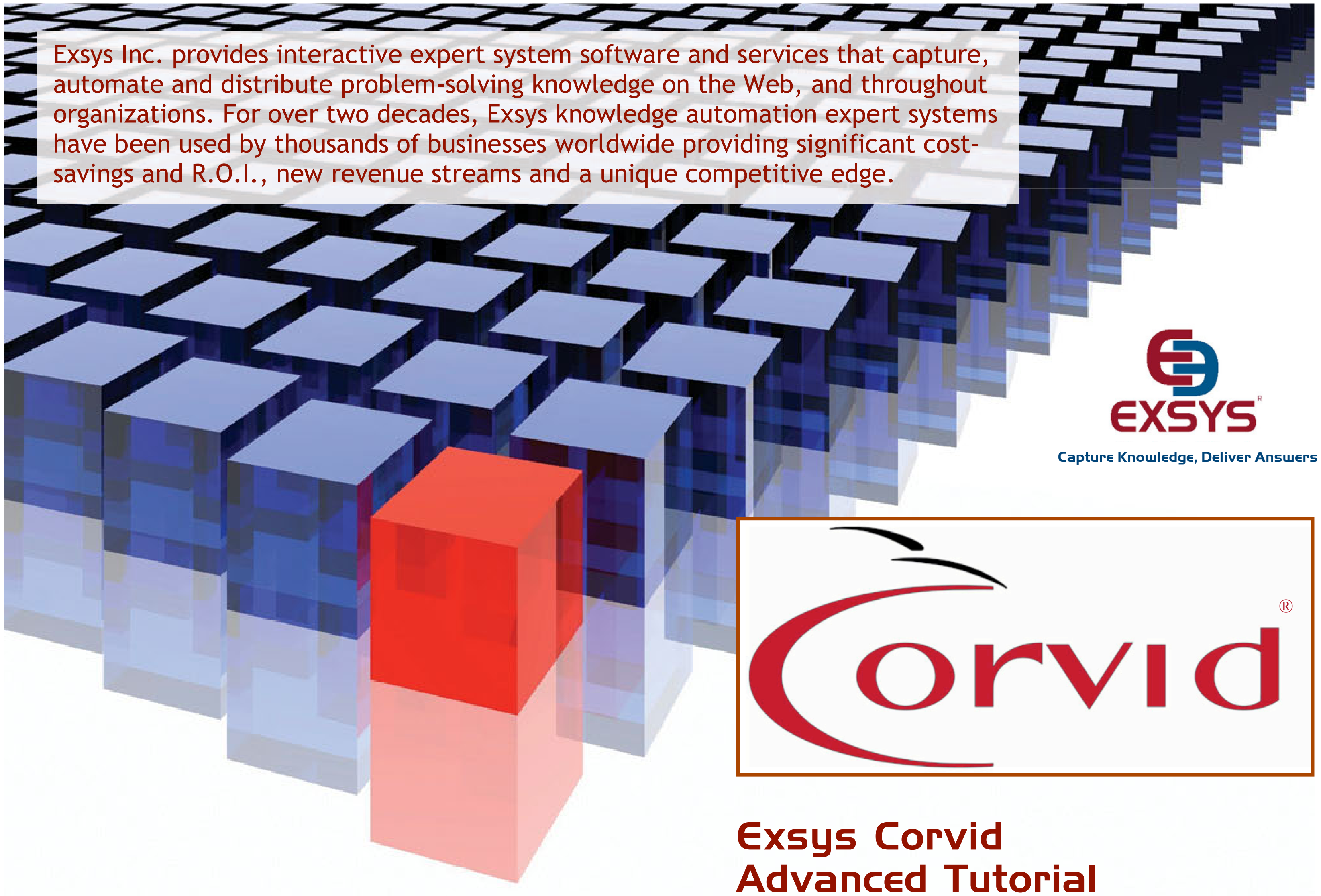


Exsys Inc. provides interactive expert system software and services that capture, automate and distribute problem-solving knowledge on the Web, and throughout organizations. For over two decades, Exsys knowledge automation expert systems have been used by thousands of businesses worldwide providing significant cost-savings and R.O.I., new revenue streams and a unique competitive edge.

EXSYS®

Capture Knowledge, Deliver Answers

Corvid®

Exsys Corvid Advanced Tutorial

Explore powerful, easy-to-learn Knowledge Automation Expert System development.

Exploring Exsys Corvid®
Advanced Tutorial

Before starting this tutorial it is highly recommended that you review the Corvid QuickStart Guide and Tutorials. It provides a more detailed explanation of the development environment fundamentals, knowledge automation basics, system approaches and concepts.

Exsys Corvid is easy to learn and use. It is also a very powerful expert system development tool capable of handling a wide range of problems. Consequently, it has many options and controls that are used for advanced systems and special situations. Many of these advanced features are not typically needed for most systems. This guide will utilize the common controls that are used to build most systems, and help you develop a more complex system than those in the **Corvid Quick Start Guide Tutorials**.

Selecting an Appropriate Problem

Exsys Corvid is designed to help you describe the logical steps in a decision-making process in a way that allows the knowledge to be delivered to others as if they were interacting with a human expert. This is actually very similar to the way you would explain how to solve a problem to a person.

The first step in building a Corvid application is to select a problem that can be broken into logical steps or pieces. As you become more experienced with Corvid, you will learn advanced ways to approach complex problems, but for your first system select a simple problem that can be described in a few dozen rules.

The problem should be able to be solved using logic that can be explained to a person using statements in the form: "IF …. THEN…", or "SINCE …. I KNEW THAT……". The "IF", or "SINCE", part may involve several conditions that are combined. For example:

IF it is raining THEN wear a raincoat.

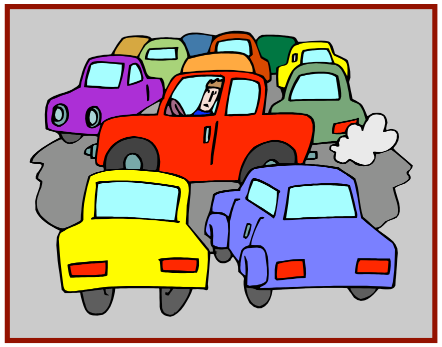
IF the car will not start AND the gas gage is on empty THEN the car is out of gas.

SINCE the light was on I KNEW the computer was getting power.

SINCE it is Saturday AND it is sunny I KNEW Fred was probably playing golf.

Describing a problem in this way puts it in the correct form for input into Corvid. Notice that the last example used "probably". As you will see, Corvid systems have many ways to handle probability. One of the best uses of Corvid is for solving problems that involve selecting the "best" solution when there are multiple possible solutions.

Sample Problem – What's the Best Way to Drive to Work?



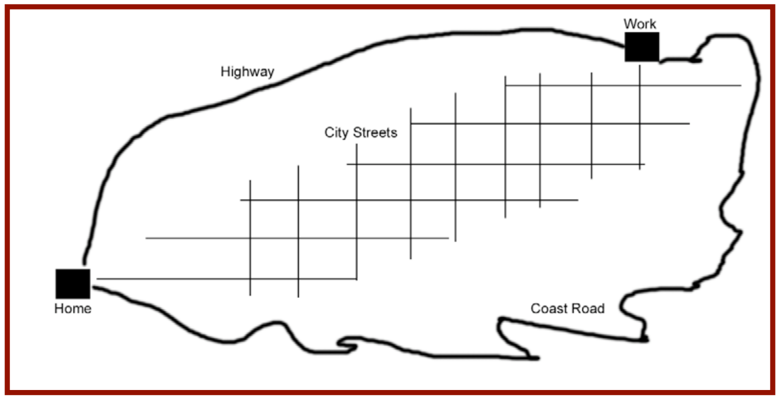
In this tutorial, you will build a small Corvid system to make a decision. The sample problem is a simple one, but will still demonstrate many of the fundamental features of Corvid, especially Confidence Variables.

You will decide how to drive from home to work. There are 3 routes to consider:

1. **The Highway** - This is the fastest way to go, as long as there are no traffic problems. If there is extra heavy traffic, it is very slow. It also does not have any gas stations, and you do not want to run out of gas.

2. **The City Streets** - This is a slow, but reliable way to go. There are plenty of gas stations that are always open. Even if there is a traffic problem, there are plenty of alternate roads, so it is a very reliable way to go.

3. **The Coastal Road** - This is a scenic road along the coast. It is a very pretty and nice drive in good weather, but difficult in bad weather or at night. It takes about the same amount time as going on the city streets. It has no gas stations. When practical, it is the preferred way to go.



Putting it in IF/THEN Rules

To start, state the problem-solving steps in IF / THEN form. These should be simple statements about different aspects of the decision. They do not need to be combined. The Corvid's Inference Engine will automatically combine the various factors into an overall system.

1. The only route that has gas stations is the City Streets, and it takes 2 gallons of gas by any of the routes.

> **IF there is less than 2 gallons of gas**
> **THEN City Streets are the only possibility**

2. If driving conditions are bad, the Coast Road is difficult and should not be taken.

> **IF driving conditions are bad**
> **THEN do not take the Coast Road**

3. The highway is the fastest way to go, and if you are in a hurry, this is the best route. However, if there is a traffic problem, it becomes the slowest way.

> **IF you need to get to work quickly**
> **AND there are <u>no</u> traffic problems on the highway**
> **THEN take the highway**

> **IF you need to get to work quickly**
> **AND there <u>are</u> traffic problems on the highway**
> **THEN do not take the highway**

4. If you have time, the way you prefer to go is the Coast Road.

> **IF you do not need to get to work quickly**
> **THEN the Coast Road is preferred**

These 5 rules describe the logic of the decision-making process. They are in an IF/THEN form that will make them easy to input in Corvid.

Corvid Variables

All the logic in Corvid is defined using Corvid Variables. Variables are the building blocks that Corvid uses to build rules and describe the logic. When the system is run, variables used in the IF part of rules will need to be assigned a value. The value may come from:

- **Directly asking the system user to provide a value**
- **Being derived from other rules**
- **Other sources such as a database**

There are 7 types of variables, but most systems can be built using just 3 types.

1. Static List Variable

This is a simple multiple-choice list. It is the most commonly used Corvid variable and is the first choice to use whenever possible. It is made up of a statement, and 2 or more possible values. This can be a simple Yes/No:

The highway has traffic problems? YES / NO

In this case, the variable has values of "YES" and "NO". When building the rules, an IF statement will be made using one, or some, of the values. For example:

Traffic problems on highway: YES

To determine if that statement is true, the system user will be asked a question and presented with the possible values to select among. Their answer will determine if the IF statement is true or false.

A static list can also have many values. For example, the statement could be:
The state you live in is… Each of the 50 US states is a possible value.

Static list variables make it easy to write logic that covers all the possible values, and automatically creates a user interface that limits the values that the user can select.

2. Numeric Variable

This type of variable can have a range of numeric values. The value will be a specific number, but there are too many possible values to list in a static list. For example, the statement could be: **The amount of gas in the car is (in gallons)…** The rules can use numeric variables in algebraic expressions that test the value.

For example:

IF amount_of_gas < 2
THEN City Streets are the only possibility

When the system is run, the system user will be asked to input the numeric value for the amount of gas in the car.

3. Confidence Variable

These variables are usually the possible options that the system will select among. They can be assigned a "Confidence Value" that determines if they are an appropriate, or inappropriate recommendation based on the end user's input. Corvid provides many ways to work with confidence values to select the best recommendation(s) to give.

Many rules have a form similar to:

IF … THEN X is a good choice
IF … THEN X is likely to be the cause
IF … THEN X should not be used
IF … THEN X is better than Y

This is the best type of variable to use if an item is being selected, rejected, or indicated to be more likely. In these cases, X and Y should be Confidence Variables.

For simple systems, the items that the system decides among should be Confidence Variables. These are items that only appear in the THEN part of the rules. (As you get to more advanced system, this is not always true, but it is a good approach for many systems.)

Corvid provides many ways to work with Confidence variables, especially when there are various independent factors that must be combined, and where some of the factors are more important than others.

Confidence Values

Confidence Variables are given "confidence values" in the rules. The numeric value assigned indicates if that choice is "good" or "bad" based on the logic of the rule and the end user's input. A specific variable may be assigned values by many rules. These values are combined from all the rules to determine the overall confidence value for the variable - which determines if that variable is the best recommendation.

This may sound complicated at first, but it is actually pretty simple. Corvid provides many ways to combine confidence values, but the simplest is just to sum them up. One way is to think of the value as "points" added to or taken from the “score” for the item or recommendation. In a rule, the Confidence Variables are assigned a value in the THEN part. This is indicated in the rule by an " = " followed by the value to assign. For example, if a system is trying to help decide what to wear, and it has a Confidence Variable "Jacket", there could be rules:

IF it is below 50 degrees THEN Jacket =10
IF it is raining THEN Jacket=15

The first rule means that if it is below 50 degrees, assign a confidence value of 10 to the Confidence Variable "Jacket", or give "Jacket" 10 points. Likewise, the second rule would assign a value of 15, or add another 15 points. If the system used both rules, the overall confidence for "Jacket" would be 10+15 or 25 points. If there were other rules that fired, which assigned a value to "Jacket", their values would be added to the overall sum.

There could also be a rule,

IF it is below 10 degrees
THEN Jacket= -100 AND Coat=50

which would mean if it is very cold (below 10 degrees), then a jacket is not appropriate, so it is given a large negative value (-100) to greatly reduce its overall confidence value and eliminate it. This would be a deduction of 100 points. At the same time, another Confidence Variable "Coat" is given a value of 50.

At the end of a run, the Confidence Variable(s) with the highest overall point score will be displayed in the results as the "Best" option(s) based on the input. However, remember there are other ways to combine the values besides summing them up. (See the full Exsys Corvid manual for a description of the various formulas for computing Confidence Variables.)

The actual values to assign can be any number. They should be scaled appropriately relative to each other. If a rule should slightly increase the likelihood of a particular Confidence Variable, it can be given a few points. If it is a more significant factor, it could increase it by a many of points. Rules that indicate that a Confidence Variable is not appropriate should assign similar negative values to decrease its likelihood. Adding a value of hundreds will assure that a variable is in the recommendations, and large negative values (-hundreds) will eliminate a variable from the results.

This simple approach allows competing factors to combine to give an overall best recommendation, even when a particular variable may not be perfect in all respects.

Corvid provides a wide range of other mathematical approaches to combining the individual values assigned to Confidence Variables. You can even use different methods for each individual Confidence Variable. This is very useful when building more complex systems.

5 Variables in the Sample System

The sample system rules needs several items of data to make the decision:

Are the driving conditions bad?
Do you need to get to work quickly?
Are there traffic problems on the highway?
How much gas is in the car?

Corvid uses variables for each item of data. This provides a way to build the rule, and defines the questions that will be asked of the system user.

1. Driving conditions

One of the sample system rules is:

IF driving conditions are bad
THEN do not take the Coast Road

This is a multiple-choice list and easy to implement as a Static List variable. Corvid requires that each variable have a name. To make the system simple, the variable "Driving conditions" will have only 2 values: "Bad" and "Good".

2. Get to work quickly?

In the same way, the rules use information on how quickly you need to get to work. This can also be defined as a Static List variable - "Need to get to work quickly" with values: "Yes" and "No".

3. Traffic problem on highway?

The question of traffic problems on the highway can be handled the same way as a Static List variable. "Traffic problems on highway" with values: "Yes" and "No".

4. Amount of gas in the car

The system will need to know the amount of gas in the car. Since the rule tests if the amount is greater than a numeric value (2 gallons), the system will use a numeric variable. (This question could also be asked in a different way since most gas gauges do not show the actual number of gallons, but the sample system is intended to show you how to use a numeric variable.)

5. Routes

In addition to the variables that are needed in the IF parts of the rules, there are the 3 possible routes that the system will select among. Since each of these is independent of the others, and more than one may be possible, each should be a separate Confidence Variable. In the sample system, the 3 possible routes are the options to decide among, and they appear only in the THEN part of the rules. Each route will be a Confidence Variable:

Highway
City Streets
Coast Road

The Rules Using the New Variables

Restating the sample system in IF /THEN rules with the new variables:

IF amount of gas is less than 2 gallons
THEN City Streets are the only possibility

```
IF
        Amount_of_gas < 2
THEN
        City Streets = 100
        Highway = -100
        Coast Road = -100
```

The only route that has gas stations is the City Streets. Taking the other routes without enough gas is not an acceptable recommendation. This rule, assigns a high confidence to the City Streets variable making sure it will have an overall high score. Large negative values are given to the other two routes to eliminate them regardless of other values they may get from other rules.

IF there is 2 gallons or more
THEN any route can be taken

```
IF
        Amount_of_gas >= 2
THEN
        City Streets = 10
        Highway = 10
        Coast Road = 10
```

When there is 2 or more gallons of gas, any of the routes can be taken. This rule will assure that all the routes will be given the same number of initial points. The actual amount of points is kept small, since there will be other rules that will decide which route is actually taken.

IF driving conditions are bad
THEN do not take the Coast Road

```
IF
        Driving conditions: Bad
THEN
        Coast Road = -100
```

The Coast Road should be eliminated in bad conditions, so the rule assigns a large negative number. Bad conditions are not as much of a problem on the other roads, so the rule does not assign anything to those variables. It is NOT necessary to have every rule assign a value to each Confidence Variable. Only assign values when there is a logical reason to do so.

IF you need to get to work quickly
AND there are no traffic problems on the highway
THEN take the highway

```
IF
        Need to get to work quickly: YES
        Traffic problems on highway: NO
THEN
        Highway = 20
```

This rule recommends the highway for speed when there is no traffic problem, but it only assigns a value of 20. This is because it is a rule that recommends the highway, but other factors have to be considered, such as gas. If this rule fires, but there is not enough gas in the car, the system should not recommend the highway. The value of 20 will push the "Highway" variable up in the recommendations, but will not override a possible -100 from other factors. It is important to keep confidence factors scaled relative to each other.

IF you need to get to work quickly
AND there are traffic problems on the highway
THEN do not take the highway

```
IF
        Need to get to work quickly: YES
        Traffic problems on highway: YES
THEN
        Highway = -100
```

In this rule, the Highway should be eliminated so it can safely be given -100.

IF you do not need to get to work quickly
THEN the Coast Road is preferred

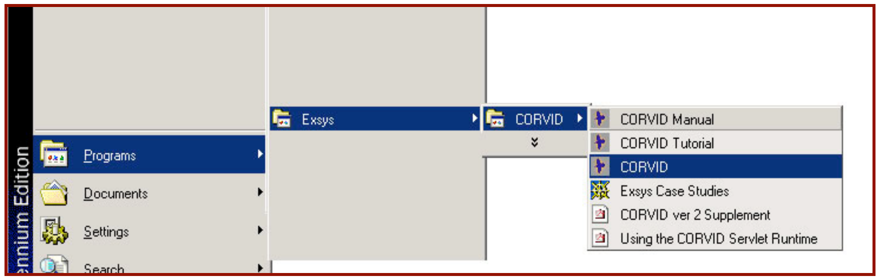
```
IF
        Need to get to work quickly: NO
THEN
        Coast Road = 20
```

This rule suggests the "Coast Road", but adds a lower value since there might be other (weather) reasons that would eliminate it.

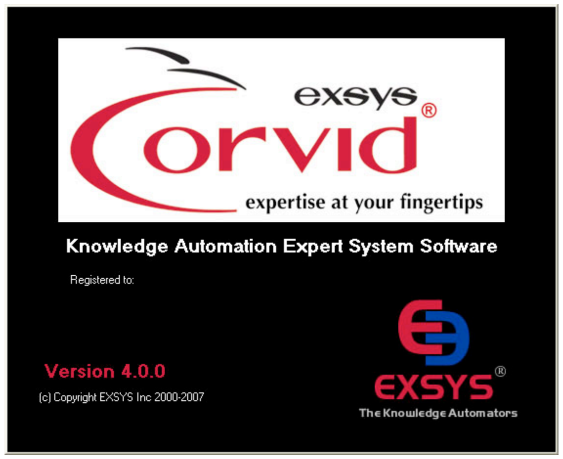
The system now has the 5 rules defined in terms of variables, and in a form that will make them easy to add into Corvid.

Starting Exsys Corvid

Once Exsys Corvid is installed, you can start it from the Start button.

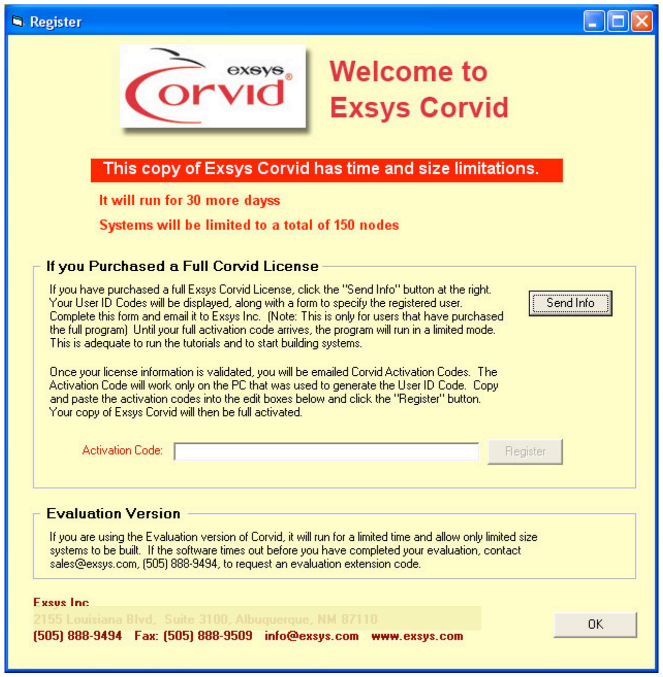


Select “Programs”, "Exsys", "CORVID", "CORVID". This will start the program. If you will be using CORVID frequently, it is easier to make a shortcut. Once Corvid starts, the splash screen will be displayed.


exsys
Corvid
expertise at your fingertips
Knowledge Automation Expert System Software
Registered to:
Version 4.0.0
(c) Copyright EXSYS Inc 2000-2007
EXSYS
The Knowledge Automators

This will be displayed for a few seconds and then the main program screen will come up. (To quickly dismiss the splash screen, click on it and you will immediately go to the main screen.)

If you are using a "Student" or "Evaluation" version of the program, or you have not yet registered your licensed copy with Exsys Inc, you will see the Unregistered screen.



Welcome to Exsys Corvid

This copy of Exsys Corvid has time and size limitations.

It will run for 30 more dayss

Systems will be limited to a total of 150 nodes

If you Purchased a Full Corvid License

If you have purchased a full Exsys Corvid License, click the "Send Info" button at the right. Your User ID Codes will be displayed, along with a form to specify the registered user. Complete this form and email it to Exsys Inc. (Note: This is only for users that have purchased the full program) Until your full activation code arrives, the program will run in a limited mode. This is adequate to run the tutorials and to start building systems.

Once your license information is validated, you will be emailed Corvid Activation Codes. The Activation Code will work only on the PC that was used to generate the User ID Code. Copy and paste the activation codes into the edit boxes below and click the "Register" button. Your copy of Exsys Corvid will then be full activated.

Evaluation Version

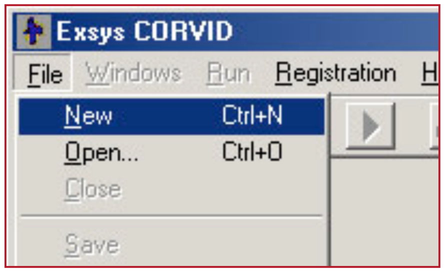
If you are using the Evaluation version of Corvid, it will run for a limited time and allow only limited size systems to be built. If the software times out before you have completed your evaluation, contact sales@exsys.com, (505) 888-9494, to request an evaluation extension code.

Exsys Inc
2155 Louisiana Blvd, Suite 3100, Albuquerque, NM 87110
(505) 888-9494 Fax: (505) 888-9509 info@exsys.com www.exsys.com

If you see this screen, your copy of Corvid has time and/or size limitations. Even if you purchased a full copy of Corvid, this screen will be displayed until it is registered and you receive an activation code.

The bottom of the screen shows how many more days the system will run for and how large a system can be built.

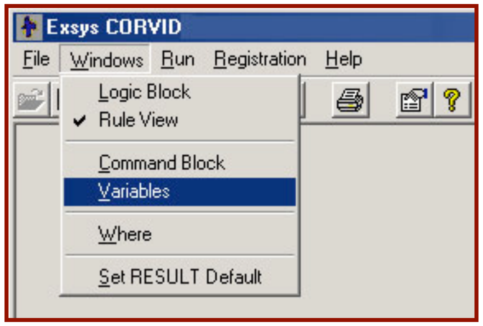
For this tutorial, start a new system. Select New under the File menu.



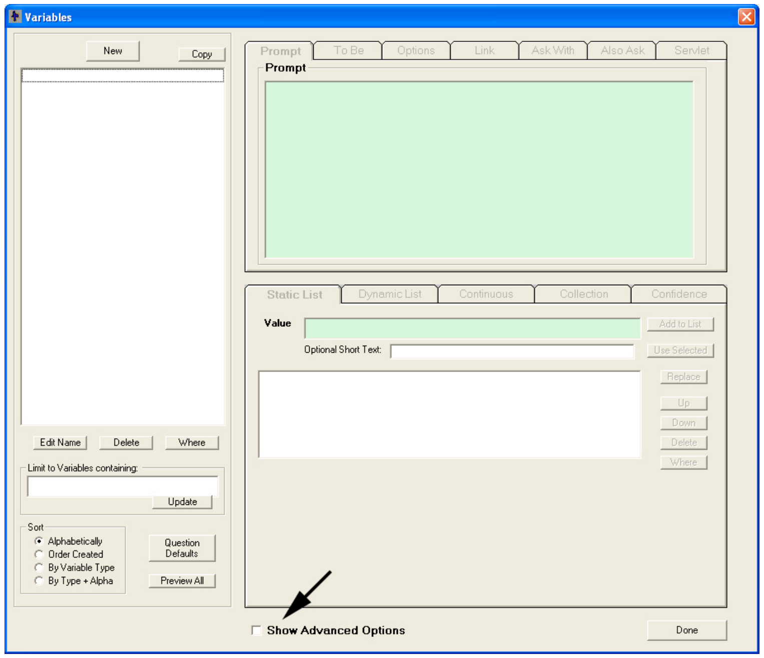
This will display the standard MS Windows Open File dialog. Select a name for your tutorial system such as “demo”. Corvid will automatically make this a .CVD file, which is the extension for Corvid knowledge base files.

The Corvid Variable Window

Variables are added, and edited, in Corvid using the Variables window. This window is displayed by selecting "Variables" under the "Windows" menu. Or you can click on the Variable Window icon at the top of the Corvid window.

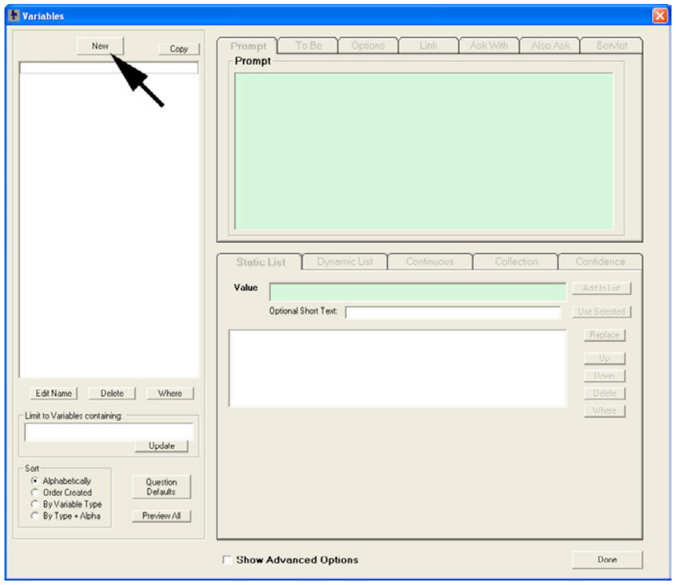


This will display the Variable window.



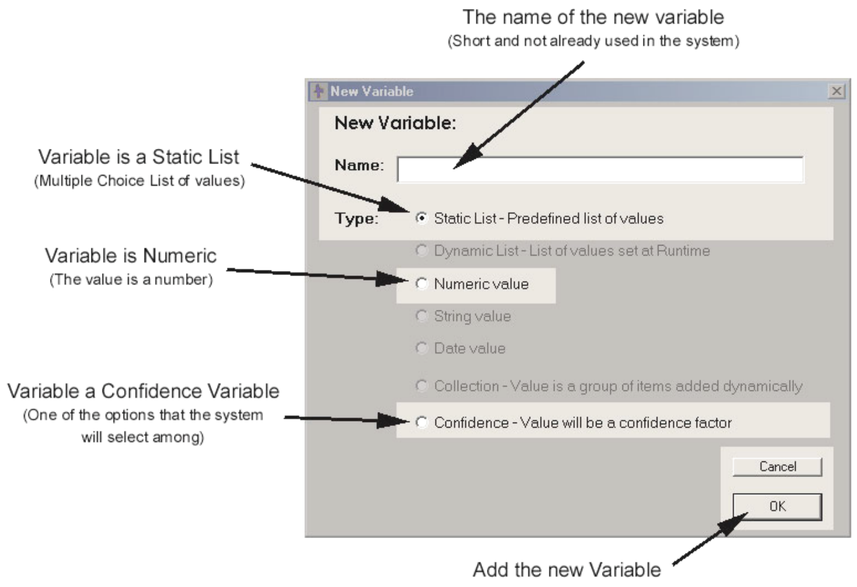
Make sure that the “Show Advanced Options” checkbox at the bottom of the window is NOT checked. Selecting this will display many other control options in the window. While these controls are used for some very powerful features in Corvid, they are not needed for most systems and are not needed for this tutorial.

Click on the "New" button to create a new Corvid variable.



This brings up the “New Variable” window.

Each new variable must have a name and a "type". The name will be used to refer to the variable throughout the system. The name must be unique - not already used as the name for another variable in the system. It should be fairly short, but adequately descriptive to be able to remember what it means. A cryptic name may make sense now, but not in 6 months when you go in to update the system.

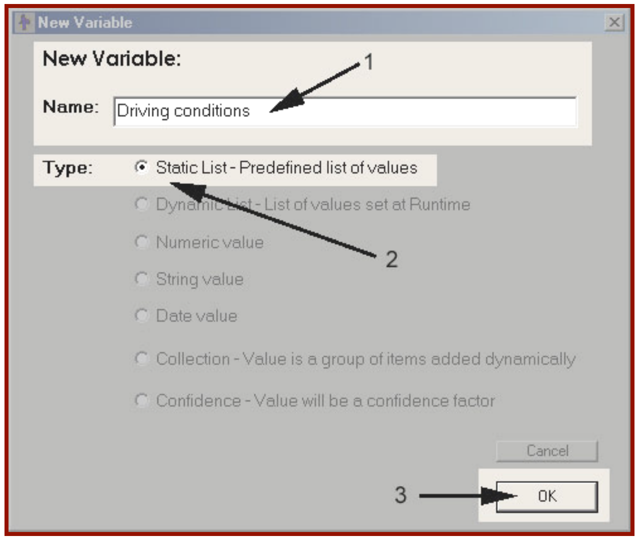

The name of the new variable
(Short and not already used in the system)
New Variable
New Variable:
Name:
Variable is a Static List
(Multiple Choice List of values)
Type:
Static List - Predefined list of values
Dynamic List - List of values set at Runtime
Variable is Numeric
(The value is a number)
Numeric value
String value
Date value
Collection - Value is a group of items added dynamically
Variable a Confidence Variable
(One of the options that the system will select among)
Confidence - Value will be a confidence factor
Cancel
OK
Add the new Variable

There are 7 types of variables in Corvid, but most systems can be built using only 3. These 3 types are highlighted in the illustration above. They are Static Lists, Numeric and Confidence. The other variable types are very powerful for certain systems but not needed as often.

Adding Static List Variables

First add the Static List variable on "Driving conditions". In the new Variable window:

1. In the Name edit box, enter "Driving conditions". This is a good name. It is short, but descriptive of the meaning of the variable.
2. Select the "Static List" radio button
3. Click the "OK" button.

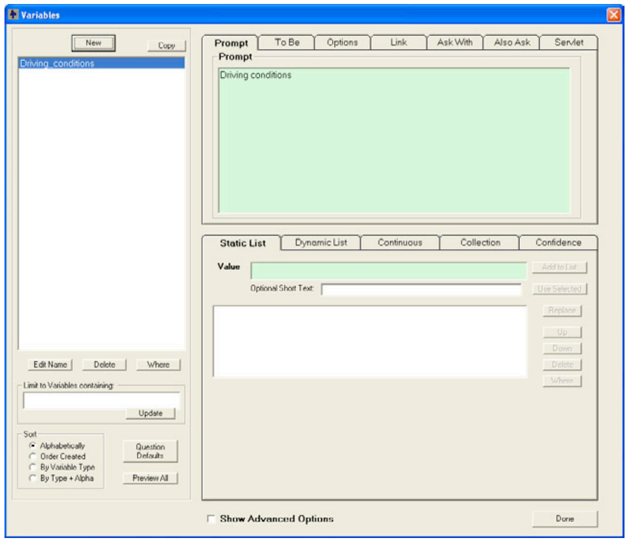


New Variable:

Name: Driving conditions

Type: [✓ Static List - Predefined list of values] Dynamic List - List of values set at Runtime / Numeric value / String value / Date value / Collection - Value is a group of items added dynamically / Confidence - Value will be a confidence factor

This will return you to the Variables Window. The variable "Driving_conditions" has been added to the list on the left. Notice that the space in the name has been converted to an underscore character. Spaces are not a legal character for variable names. Corvid will automatically convert any illegal characters to an underscore.



Driving_conditions

Prompt: Driving conditions

Value

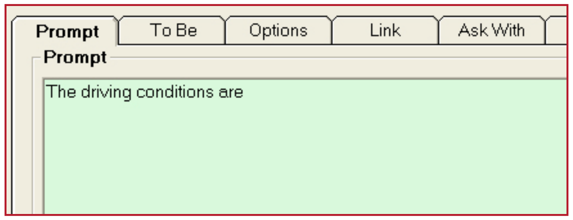
Optional Short Text

Limit to Variables containing:

Sort: [✓ Alphabetically] Order Created / By Variable Type / By Type + Alpha

Show Advanced Options

There are 2 groups of tabs on the right. The upper group sets the options for how the variable will be asked and displayed. All of these tabs apply to all types of variables, though typically only a few are actually used. For this system the only one you need to use is the "Prompt".



Prompt: The driving conditions are

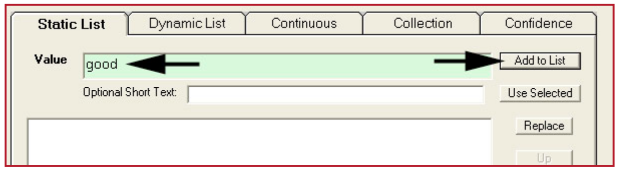
The Prompt is the text that will be displayed to ask questions of the system user and, sometimes, to display the variable in the final results. (Corvid automatically sets the Prompt text to be the same as the variable name, and in many cases this does not need to be changed.) Since this is a Static List, there will be a list of possible values - in this case "good" and "bad". The Prompt text should combine with the value text to make a readable statement. In this case, make the Prompt: **"The driving conditions are"**.

When combined with the value text, this will make the statements: "The driving conditions are good" and, "The driving conditions are bad". To change the Prompt text, just click in the edit box and type in the changes.

Since this is a Static List variable, the "Static List" tab is selected in the tabs on the lower right. This tab will be set automatically based on the type of variable you selected. It is possible to change the type by clicking on the tab for another variable type, but this should be done before the variable is used. If the type of the variable was selected correctly on the new variable window, you should not have to click on any of the other tabs on the lower right.

This variable will have 2 values: "good" and "bad". A variable can have any number of values. They are entered one at a time.

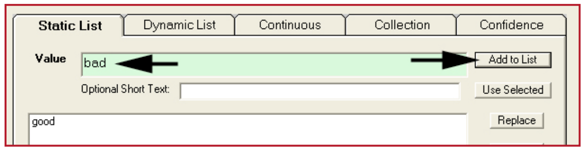
1. Enter the text of the value. This can be any text, but it should combine with the Prompt text to make a readable statement. This is entered in the "Value" edit box. In this case, enter "good".



Value: good
Optional Short Text:

2. Click the “Add to List” button.

The second value is entered the same way.



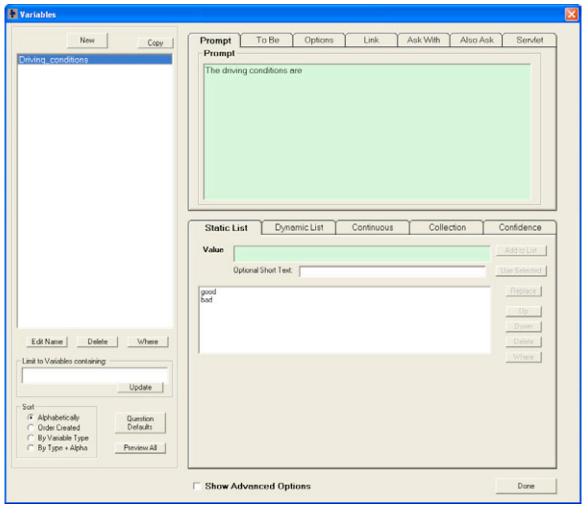
Value: bad
Optional Short Text:
good

1. In the Value edit box, enter the text of the second value: "bad".

2. Click the "Add to List".

That is all you need to do to add a variable in Corvid. There are many other controls, but for most systems, this is all that is needed to be done.

The Variable Window should now look like:



Driving_conditions

Prompt: The driving conditions are

Value:
Optional Short Text:
good
bad

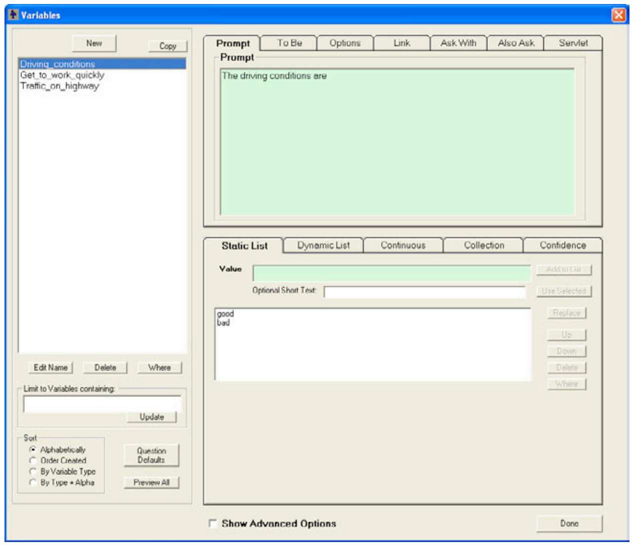
Limit to Variables containing:

Sort: (● Alphabetically) Order Created / By Variable Type / By Type + Alpha

Now add the other Static List variables that will be needed in the system exactly the same way. To start each one, click the "New" button on the Variable Window. Make each variable a Static List variable.

Variable Name	Prompt	Values
Get to work quickly	Do you need to get to work quickly?	Yes No
Traffic on highway	Are there traffic problems on the highway?	Yes No

Once these variables are added, the Variables Window should look like:



Driving_conditions
Get_to_work_quickly
Traffic_on_highway

Prompt: The driving conditions are

good
bad

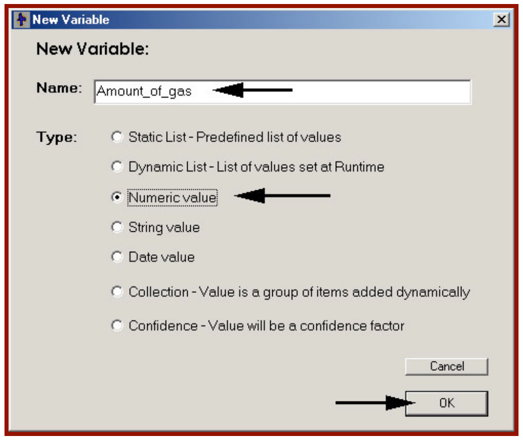
That adds all the Static List variables needed for the sample system.

Adding Numeric Variables

Numeric variables are added in a similar way to Static List variables, but they do not have a value list. Numeric variables should be used when the possible values cover a wide range of numeric values and the value will be used in algebraic expressions. (If there are only a few possible values, it is generally better to use a Static List variable).

To add a Numeric variable:

1. Click the "New" button on the Variables window.
2. In the window for adding variables, enter the name of the variable ("Amount of gas") and select the "Numeric" radio button.
3. Click the "OK" button.

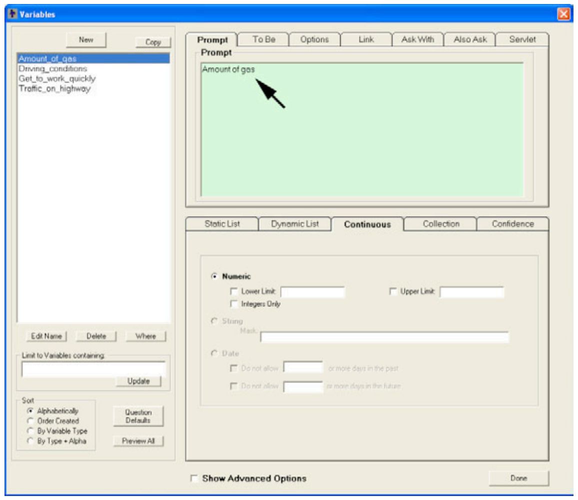


New Variable:

Name: Amount_of_gas

Type:
- Static List - Predefined list of values
- Dynamic List - List of values set at Runtime
- [✓ Numeric value]
- String value
- Date value
- Collection - Value is a group of items added dynamically
- Confidence - Value will be a confidence factor

This will add the variable to the Variables window.

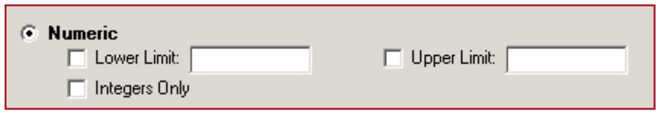


Amount_of_gas
Driving_conditions
Get_to_work_quickly
Traffic_on_highway

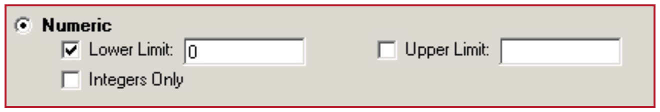
Prompt: Amount of gas

The name of the variable has been automatically copied to the Prompt edit box. This Prompt is the text that will be used to ask the system user for input. Just asking "Amount of gas" could be confusing - should they answer with a fraction of the tank ("1/4 tank"), or text ("Plenty"), or metric ("7 liters")? Questions should always be made to be as unambiguous as possible. Change the prompt to "The number of gallons of gas in the car".

That is all that is needed to add the numeric variable. However, the numeric tab has range checking options:


[✓ Numeric]
Lower Limit:
Upper Limit:
Integers Only

These can be used to restrict the user input to a range that makes sense in the system. Since there cannot be less than 0 gallons in the car, enter "0" in the "Lower Limit" edit box.

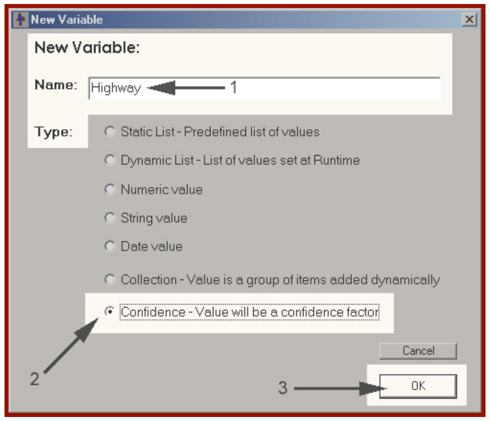

[✓ Numeric]
[✓ Lower Limit:] 0
Upper Limit:
Integers Only

Entering the value in the edit box will automatically select the checkbox next to lower limit.

These range checking options apply when the system is asking the user for input. Values that do not meet the limits are immediately rejected. You can apply any, or none, of the limit options. In some systems, using range checking on input makes the logic of the system simpler, since it does not have to handle out-of-range values in the rules.

Adding Confidence Variables

Confidence Variables are typically the options, items or recommendations that the Corvid system will select among. As with the Static List variables, these are added by clicking the "New" button on the Variables Window.



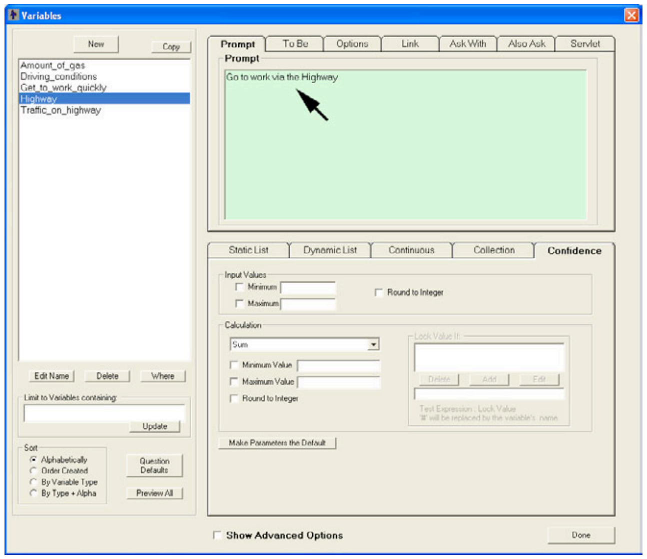
New Variable:
Name: Highway
Type:
Static List - Predefined list of values
Dynamic List - List of values set at Runtime
Numeric value
String value
Date value
Collection - Value is a group of items added dynamically
[✓ Confidence - Value will be a confidence factor]

1. In the New Variable window, enter the name of the Confidence Variable in the "Name" edit box. One of the options that the sample system will include is traveling the highway, so just enter "Highway".

2. Click on the "Confidence - Value will be a confidence factor" radio button.

3. Click the "OK" button.

The Variable window will now have the new Confidence Variable "Highway" selected. The Prompt will be set to the name of the variable - "Highway".

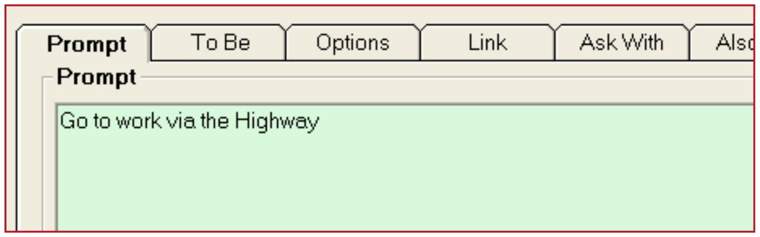


Amount_of_gas
Driving_conditions
Get_to_work_quickly
Highway
Traffic_on_highway

Prompt: Go to work via the Highway

Input Values: Minimum; Maximum; Round to Integer
Calculation: Sum; Minimum Value; Maximum Value; Round to Integer
Lock Value If:
Test Expression : Lock Value
"#" will be replaced by the variable's name
Limit to Variables containing:
Sort: [✓ Alphabetically] Order Created By Variable Type By Type + Alpha
Show Advanced Options

The only change that needs to be made is to make a more descriptive Prompt text. Change it to "Go to work via the Highway".



Prompt: Go to work via the Highway

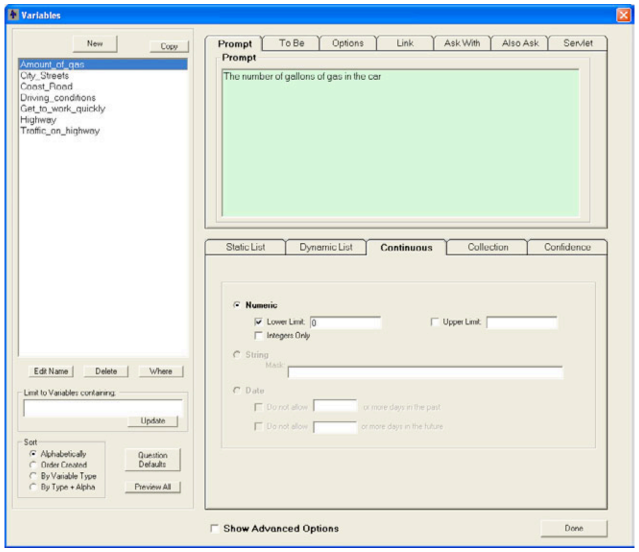
The tab group in the lower right will have the "Confidence" tab selected. There are many options on the Confidence tab that allow you to combine the values that are assigned to the variable in various ways. This can be a very powerful feature of Corvid for building advanced systems. However, the default option, and the one that is automatically selected, is to use the "Sum" method. This simply sums up the values assigned to the variable. This is the method that was discussed earlier, and the method that the sample system rules were designed to use. To use the "Sum" approach to combining Confidence values, do not make any changes on the Confidence Tab.

That is all there is to adding a Confidence Variable.

Now, in exactly the same way, add Confidence Variables for:

Name	Prompt
Coast Road	Go to work via the Coast Road
City Streets	Go to work via the City Streets

The Variable Window should now look like:



Amount_of_gas
City_Streets
Coast_Road
Driving_conditions
Get_to_work_quickly
Highway
Traffic_on_highway

Limit to Variables containing:

Sort: [✓ Alphabetically] Order Created / By Variable Type / By Type + Alpha

Prompt: The number of gallons of gas in the car

[✓ Numeric]
[✓] Lower Limit: 0 ☐ Upper Limit:
☐ Integers Only
String Mask:
Date
☐ Do not allow ___ or more days in the past
☐ Do not allow ___ or more days in the future

☐ Show Advanced Options

What is a Node?

Corvid decision-making logic is described and built using "nodes". A node can generally be thought of as a statement in the IF, or THEN part of a rule. For example, in the rule:

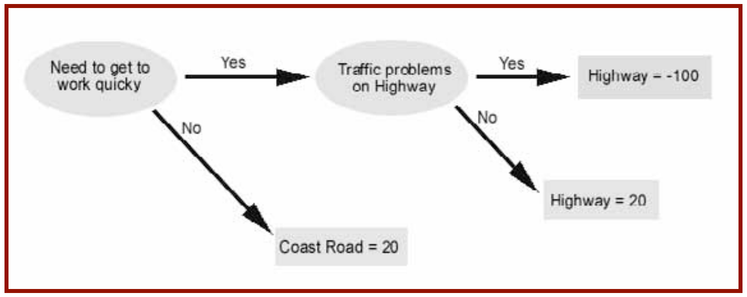
```
IF
        Need to get to work quickly: YES
        Traffic problems on highway: NO
THEN
        Highway = 20
```

Has 3 nodes, 2 IF nodes: "Need to get to work quickly: YES" and "Traffic problems on highway: NO", and a THEN node "Highway = 20". What makes nodes unique is that they can be nested to apply to all the nodes below them. This allows nodes to be used to build a logic tree diagram, which both makes it easier to view the overall structure of the logic and helps to make sure all possible cases are covered.

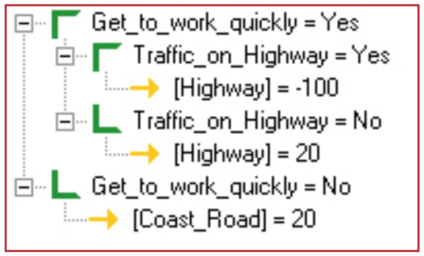
For example, the sample system has these 3 rules:

```
IF
        Need to get to work quickly: YES
        Traffic problems on highway: YES
THEN
        Highway = -100


IF
        Need to get to work quickly: YES
        Traffic problems on highway: NO
THEN
        Highway = 20


IF
        Need to get to work quickly: NO
THEN
        Coast Road = 20
```

In a tree diagram, this would look like:



The tree diagram makes it easier to see the overall structure of the rules. In Corvid the logic of a system is built in Logic Blocks, which hold the tree diagrams and other logical information. The following equivalent Logic Block tree would be created:



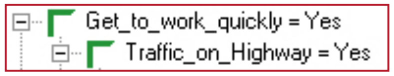
```
Get_to_work_quickly = Yes
    Traffic_on_Highway = Yes
        [Highway] = -100
    Traffic_on_Highway = No
        [Highway] = 20
Get_to_work_quickly = No
    [Coast_Road] = 20
```

Each line in the Corvid tree is a node.

Green square brackets indicate the IF nodes. The brackets mark a group of nodes that use the same Corvid variable but have different values.

THEN nodes are marked with an arrow.

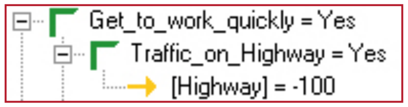
When an IF node is indented under another IF node, the nodes are combined with a logical AND - meaning both nodes must be true. The nodes:



```
Get_to_work_quickly = Yes
    Traffic_on_Highway = Yes
```

mean: If "Need to get to work quickly” is “YES" **AND** "There is traffic on the highway” is YES".

When there is a THEN node, indicated by an arrow, it is true when the IF nodes it is indented under are true. For example, these nodes:

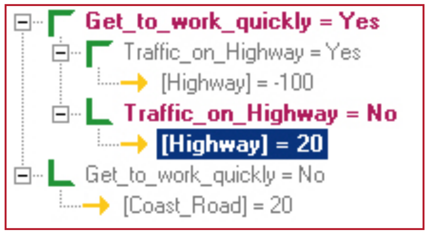


```
Get_to_work_quickly = Yes
    Traffic_on_Highway = Yes
        [Highway] = -100
```

are equivalent to the rule:

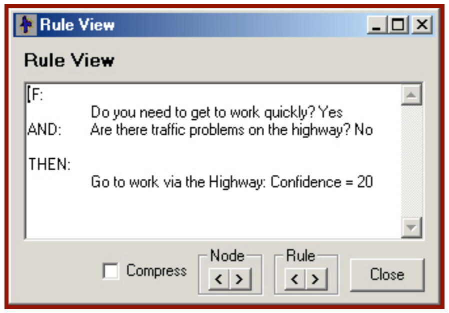
IF
    Need to get to work quickly: YES
    Traffic problems on highway: YES
THEN
    Highway = -100

It is very important to understand the way nodes are displayed in Corvid and the logical meaning of the indentation. Corvid makes it easy to see this structure in a Logic Block window by highlighting the nodes associated with a rule. In addition, a Rule View window displays the full text of a rule whenever you click on a node in the Logic Block. For example, clicking on the "Highway = 20" node would display:



```
Get_to_work_quickly = Yes
    Traffic_on_Highway = Yes
        [Highway] = -100
    Traffic_on_Highway = No
        [Highway] = 20
Get_to_work_quickly = No
    [Coast_Road] = 20
```

This displays the two IF nodes that are associated with the THEN node in bold magenta. Corvid's Rule View window, which shows the full text of the rule that is associated with the node, would display:



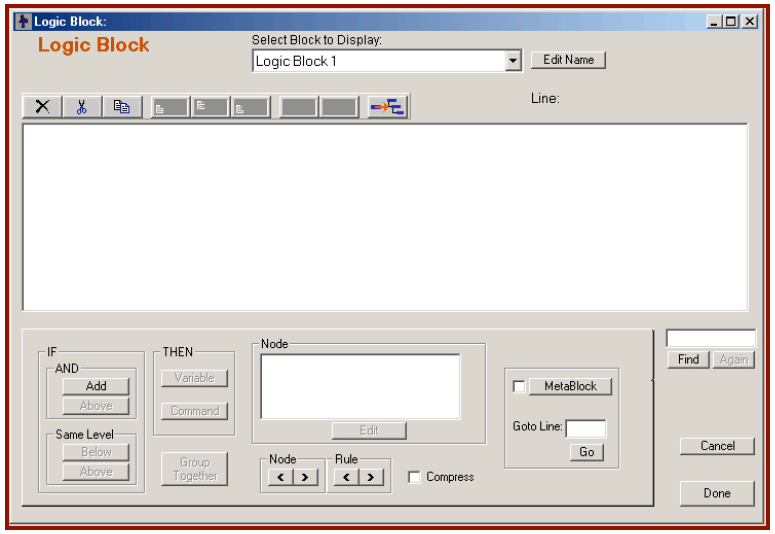
Rule View

```
IF:
        Do you need to get to work quickly? Yes
AND:    Are there traffic problems on the highway? No

THEN:
        Go to work via the Highway: Confidence = 20
```

The Logic Block Window

Rules are built using the variables to add nodes in a Logic Block window.

To open a Logic Block window, click the Logic Block icon:
This will display a new Logic Block window.



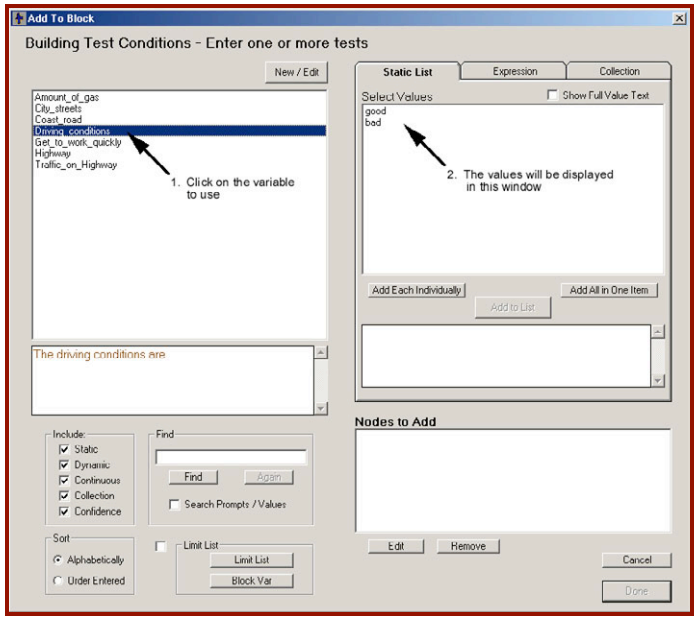
First add the rule:

IF
Driving conditions: Bad
THEN
Coast Road = -100

This rule does not have any other associated rules that use the same IF variable. There are no rules that start with "Driving conditions: Good" since in that case any of the routes can be used and there in no decision-making value. Consequently, this rule will be added as an individual rule, rather than as part of a larger tree diagram.

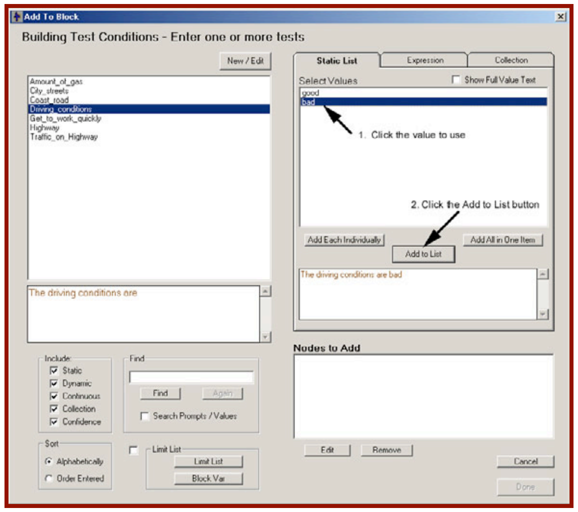
The controls for adding nodes are in the window's lower left. When a new Logic Block window is first displayed, only the "Add" button is enabled. Click the "Add" button to add the first node in the Logic Block.

The "Add To Block" window for adding nodes will be displayed.

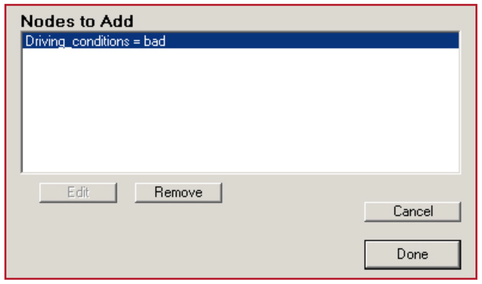


First click on the variable that is to be used to build the node. In this case, it is the "Driving_conditions" variable. Clicking on a Static List variable will display a list of that variable's values in the right hand window.

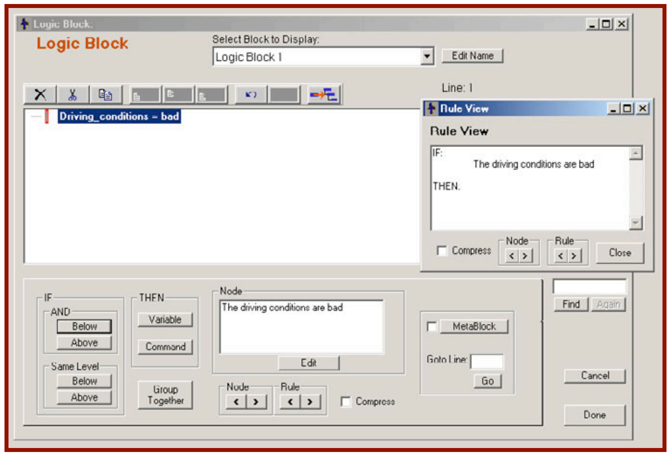
To add the simple node "Driving conditions: BAD", just click on the value "bad" and click the "Add to List" button.



This will add the node to the "Nodes to Add" list.

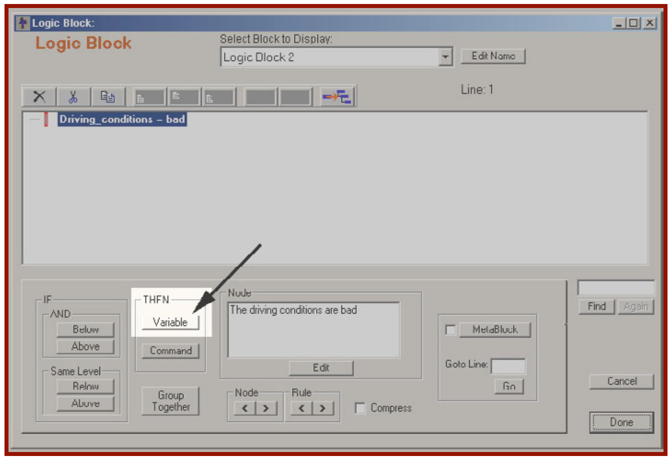


As you will see later, multiple nodes can be added at the same time, but here only a single node is needed, so just click the "Done" button. This will add the node to the Logic Block and display the full text in the Rule View window.



Adding THEN Nodes Under a Selected Node

A THEN node can be added to the Logic Block by selecting the node to add it under, and clicking the "Variable" button under "THEN".



There are 2 types of THEN nodes:

1. Assignment of a value to a Corvid variable
2. A Corvid Command

Since the THEN node will assign a value to a variable, use the button is labeled "Variable". (THEN nodes that are Corvid commands are used in some advanced systems and can be very powerful, but are not generally needed for most systems.)

When a node is clicked on, the Rule View window will display the associated rule. IF the node is a THEN node, you will see the full rule. If the node is an IF node, you see the IF part of a rule.

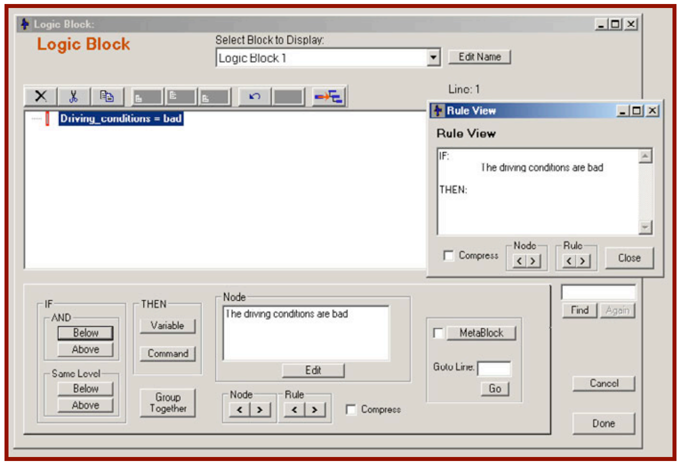
To add a THEN node to a rule:

1. Click on the last (most indented) IF node for the rule.
2. Check the Rule View window to confirm that the IF part is what you want for the rule.
3. Click the "Variable" button in the THEN button group to add the THEN node.

The rule being added is:

IF
    Driving conditions: Bad
THEN
    Coast Road = -100

In the Logic Block click on the "Driving_conditions = bad" node.



Select Block to Display: Logic Block 1

Line: 1

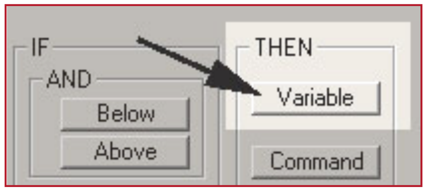
Driving_conditions = bad

Rule View

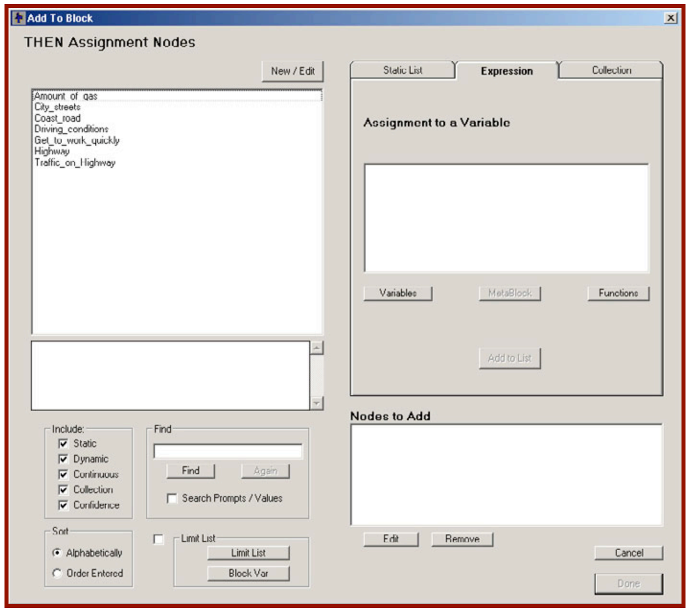
IF:
The driving conditions are bad
THEN:

Node: The driving conditions are bad

Click the "Variable" button under THEN.



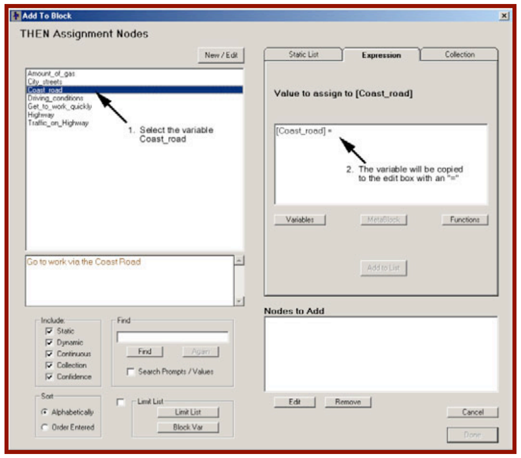
This will display the “Add To Block” window for adding nodes.



Amount_of_gas
City_streets
Coast_road
Driving_conditions
Get_to_work_quickly
Highway
Traffic_on_Highway

This window is very similar to the one used before to build an IF test node, but this time a value is being assigned to a variable to build a THEN node. The node should be "Coast Road = -100".

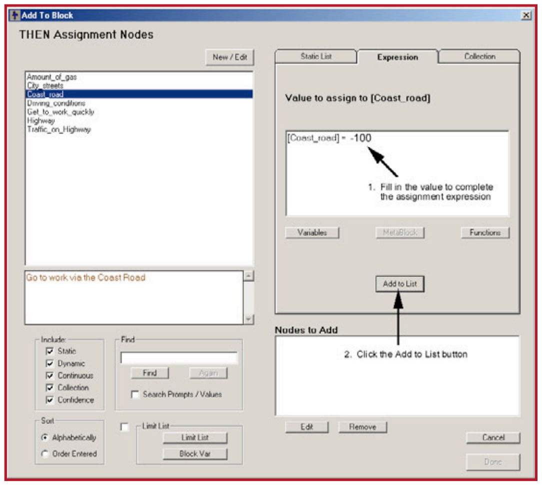
Click on the "Coast road" variable in the list. This will automatically activate the Expression tab, and enter the expression: "[Coast road] = " in the edit box.



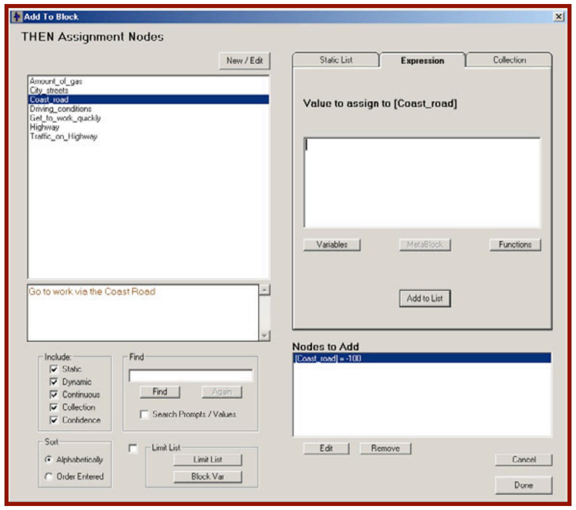
Square Brackets: In CORVID expressions, variable names are always enclosed in square brackets [ ]. The square brackets are usually optional for Static List variables, making them easier to read. However, all other variable types always use square brackets.

In the edit box, fill in the value to assign, making the assignment:

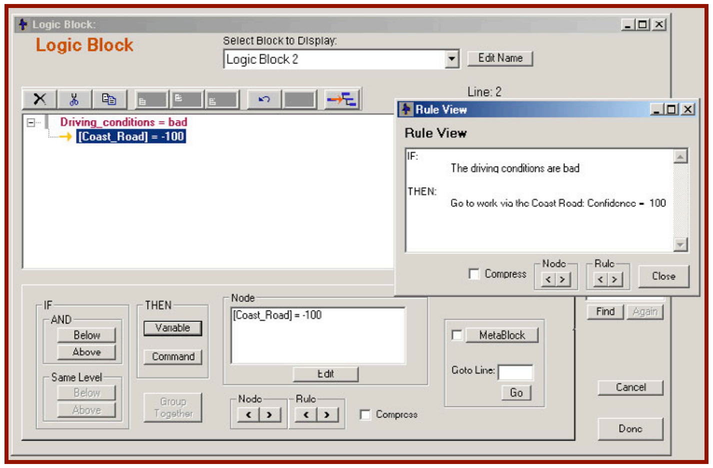
[Coast road] = -100



Click the "Add to List" button to add the node to the list of Nodes to Add. (As you will see later, you can add many THEN nodes at the same time, making it faster to build rules with multiple THEN nodes, but here there is only a single THEN node.)



Click the "Done" button to add the node to the Logic Block.

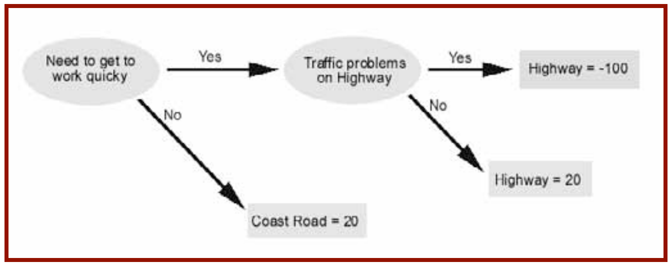


The Logic Block now shows the new node and the Rule View window displays the rule. That is all there is to adding a simple rule to Corvid.

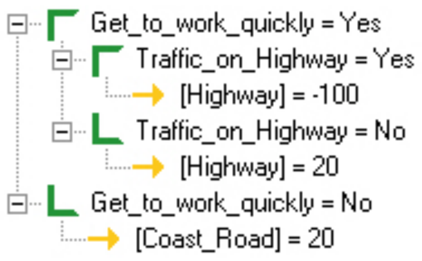
Building a Tree in a Logic Block

The first Logic Block built only a single rule. A single Logic Block can contain many rules in one or more tree diagrams. Using a tree approach to the rules makes it faster to build a system and makes it easier to validate and maintain it.

Part of the sample system was diagrammed as a tree:

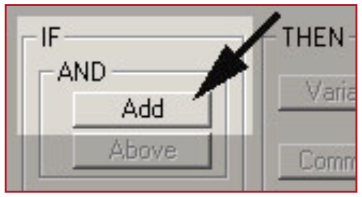


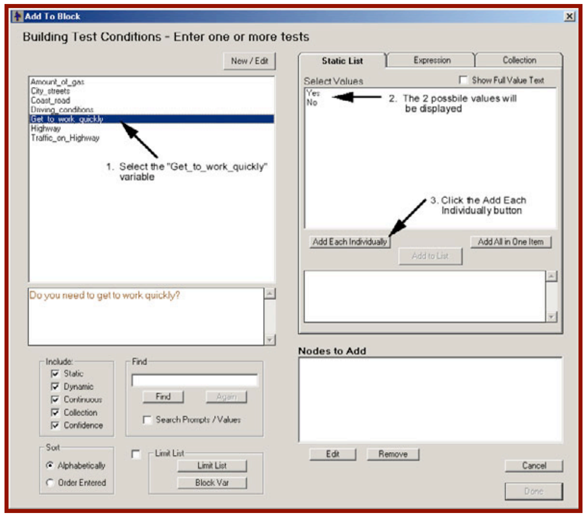
The equivalent CORVID tree is:


```
Get_to_work_quickly = Yes
    Traffic_on_Highway = Yes
        [Highway] = -100
    Traffic_on_Highway = No
        [Highway] = 20
Get_to_work_quickly = No
    [Coast_Road] = 20
```

Now you will build this in a Logic Block. Click on the Logic Block icon to start a new Logic Block.

Click the "Add" button in the IF section to add the first nodes. This will display the "Add To Block" window for adding nodes.




Add To Block

Building Test Conditions - Enter one or more tests

New / Edit

Amount_of_gas
City_streets
Coast_road
Driving_conditions
Get_to_work_quickly
Highway
Traffic_on_Highway

1. Select the "Get_to_work_quickly" variable

Do you need to get to work quickly?

Static List | Expression | Collection

Select Values Show Full Value Text

Yes
No

2. The 2 possbile values will be displayed

3. Click the Add Each Individually button

Add Each Individually | Add All in One Item | Add to List

Nodes to Add

Edit | Remove | Cancel | Done

Include: ☑ Static ☑ Dynamic ☑ Continuous ☑ Collection ☑ Confidence

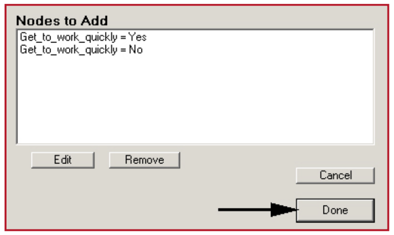
Find: Find | Again | ☐ Search Prompts / Values

Sort: ● Alphabetically ○ Order Entered

☐ Limit List: Limit List | Block Var

The top-level node in the tree is for the variable "Get_to_work_quickly", so click on that variable to select it. This will display the two possible values "Yes" and "No".

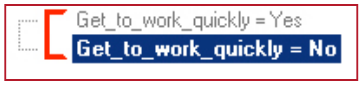
This time instead of adding a single value, the tree has branches for each of the two possible values. To add a node for each value, click the "Add Each Individually" button. This will add a node to the “Nodes to Add” list for each value.


Nodes to Add

Get_to_work_quickly = Yes
Get_to_work_quickly = No

Edit | Remove | Cancel | Done

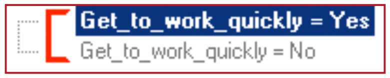
Now just click the "Done" button to add both nodes to the Logic Block.

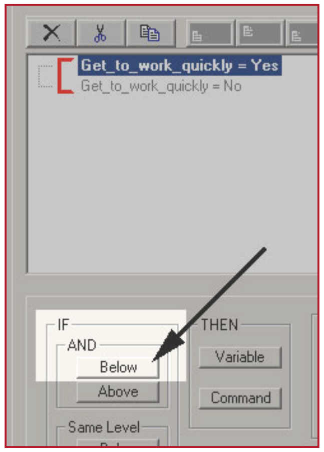

Get_to_work_quickly = Yes
Get_to_work_quickly = No

Two nodes were added to the Logic Block - one for each value. Notice, there is a red bracket connecting the two nodes. This is because they were added at the same time. The bracket shows that the nodes are related - usually various values of the same variable.

The bracket is red (warning) because these are IF nodes with no associated THEN nodes. These IF nodes do not make complete rules. When THEN nodes are added to them, the bracket color will change to green. This makes it easy to look at a Logic Block and see if any of the rules are incomplete.

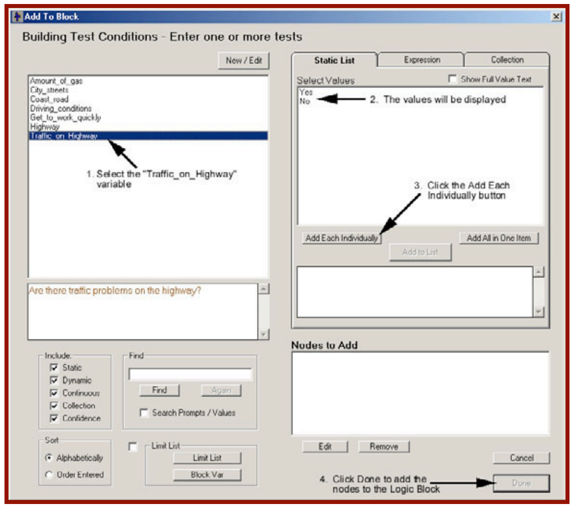
In the tree diagram, when "Get to work quickly is YES" a second factor must be considered - "Are there traffic problems on the highway". To add this to the Logic Block, first click on the "Get_to_work_quickly = Yes" node to select it. **<u>All nodes are added relative to the currently selected node.</u>**





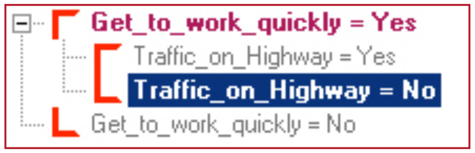
Now to add another set of nodes "ANDed" with the selected node. A logical AND means both nodes must be true. In Corvid this will be a set of IF nodes indented below the selected node. These are added with the "IF - AND - Below" button.

This will display the window for Adding Nodes that you have used before. This time, add two nodes, one for each value of the variable "Traffic_on_highway".

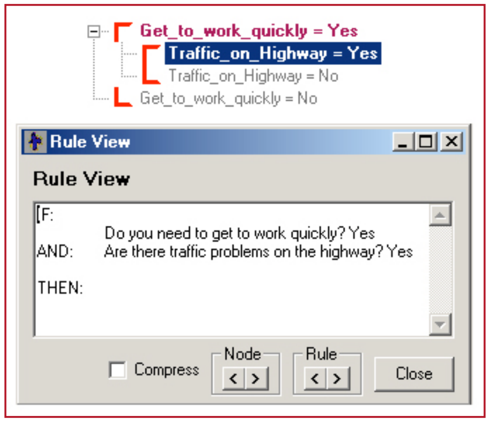


1. Click the "Traffic_on_highway" variable to select it.
2. The two possible values will be displayed
3. Since the tree has branches for each value, click the "Add Each Individually" button to build 2 nodes
4. Click the "Done" button to add the nodes to the Logic Block.

The Logic Block now looks like:



The new nodes are indented below "Get_to_work_quickly = Yes", meaning they are combined with that node using AND. If you click on the "Traffic_on_highway = Yes" node, the Rule View window will show:


Get_to_work_quickly = Yes
Traffic_on_Highway = Yes
Traffic_on_Highway = No
Get_to_work_quickly = No

Rule View

IF: Do you need to get to work quickly? Yes
AND: Are there traffic problems on the highway? Yes
THEN:

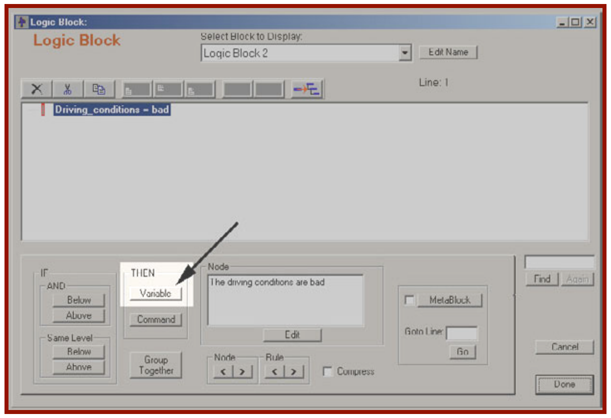
Now all that is needed is the THEN part to finish the rule:

IF
Need to get to work quickly: YES
Traffic problems on highway: YES
THEN
Highway = -100

This is done the same as in the individual rule that was added earlier.

1. If not already selected, click on the "Traffic_on_Highway = Yes" node to select it.

2. Click on the "THEN - Variable" button.


Logic Block

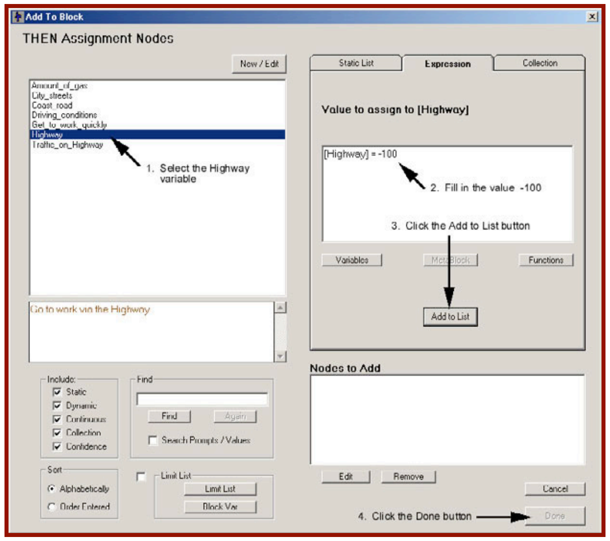
Select Block to Display: Logic Block 2

Line: 1

Driving_conditions = bad

Node: The driving conditions are bad

3. In the window for Adding Nodes, click on the "Highway" variable to select it, and fill in the value to make "[Highway] = -100".


THEN Assignment Nodes

Amount_of_gas
City_streets
Coast_road
Driving_conditions
Get_to_work_quickly
Highway
Traffic_on_Highway

1. Select the Highway variable

Go to work via the Highway

Value to assign to [Highway]

[Highway] = -100

2. Fill in the value -100

3. Click the Add to List button

Nodes to Add

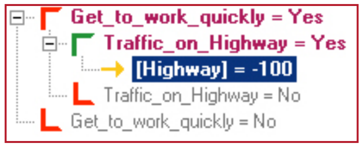
4. Click the Done button

Include: [✓ Static] [✓ Dynamic] [✓ Continuous] [✓ Collection] [✓ Confidence]

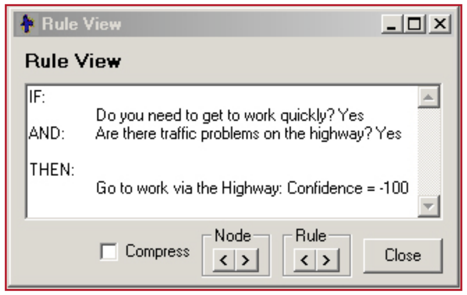
Sort: [✓ Alphabetically] Order Entered

4. Click the "Add to List" button, and then the "Done" button to add the node to the Logic Block.

The Logic Block now looks like:


Get_to_work_quickly = Yes
Traffic_on_Highway = Yes
[Highway] = -100
Traffic_on_Highway = No
Get_to_work_quickly = No

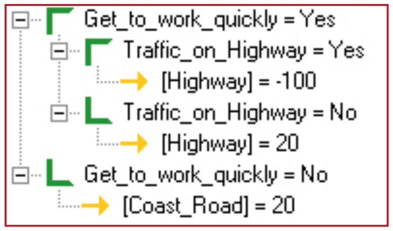
The Rule View window now shows the full rule:


Rule View

IF: Do you need to get to work quickly? Yes
AND: Are there traffic problems on the highway? Yes
THEN: Go to work via the Highway: Confidence = -100

To complete the tree, add THEN nodes to the other 2 branches in the same way. Click on the "Traffic_on_highway = No" node to select it and add a THEN node assigning the variable "Highway" a value of 20. Then click on the "Get_to_work_quickly = No" node and add a THEN node assigning the variable "Coast road" a value of 20.

The Logic Block should now look like:



```
Get_to_work_quickly = Yes
    Traffic_on_Highway = Yes
        [Highway] = -100
    Traffic_on_Highway = No
        [Highway] = 20
Get_to_work_quickly = No
    [Coast_Road] = 20
```

Click on the various THEN nodes and check the rule displayed in the Rule View window.

Adding a Numeric Rule

Now to add the last rule in the sample system.

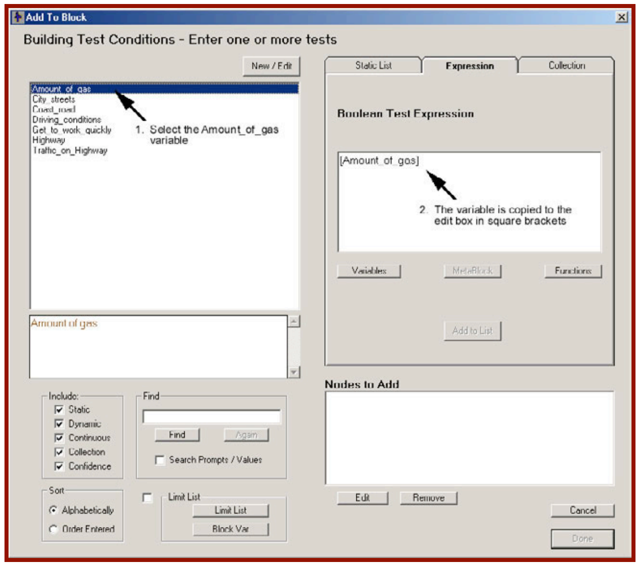
```
IF
        Amount_of_gas < 2
THEN
        City Streets = 100
        Highway = -100
        Coast Road = -100
```

The only differences here are that the IF node is an algebraic expression, and the THEN part has 3 nodes.

1. Start a new Logic Block by clicking the Logic Block icon:

2. Click the "Add" button to add the first node.

3. In the window for adding nodes, click on the variable "Amount_of_gas" to select it.



Add To Block

Building Test Conditions - Enter one or more tests

Amount_of_gas
City_streets
Coast_road
Driving_conditions
Get_to_work_quickly
Highway
Traffic_on_Highway

1. Select the Amount_of_gas variable

Boolean Test Expression

[Amount_of_gas]

2. The variable is copied to the edit box in square brackets

Amount of gas

Include: [✓ Static] [✓ Dynamic] [✓ Continuous] [✓ Collection] [✓ Confidence]

Sort: [✓ Alphabetically] Order Entered

Since the variable selected is a Numeric variable, the edit box on the right side brings up the "Expression" tab and copies the variable, putting it in square brackets. Because it is the first node in the Logic Block, the node must be an IF node. IF nodes on numeric variables are algebraic Boolean expressions that evaluate to "True" or "False". Whenever a variable is used in an expression it MUST be identified by the variable name in square brackets.

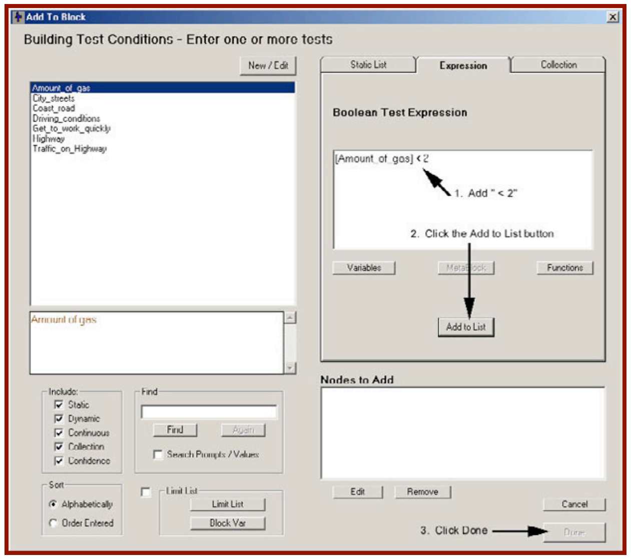
In appearance this is similar to the nodes that assigned values to the Confidence Variables, but here it is a Boolean test expression. The expression is simple. It is just a test to see if the amount of gas is less than 2 gallons:

[Amount_of_gas] < 2

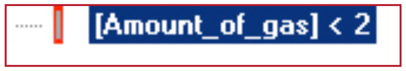
1. Just add the "< 2" to the expression.

2. Click the “Add to List” button.

3. Click the “Done” button.

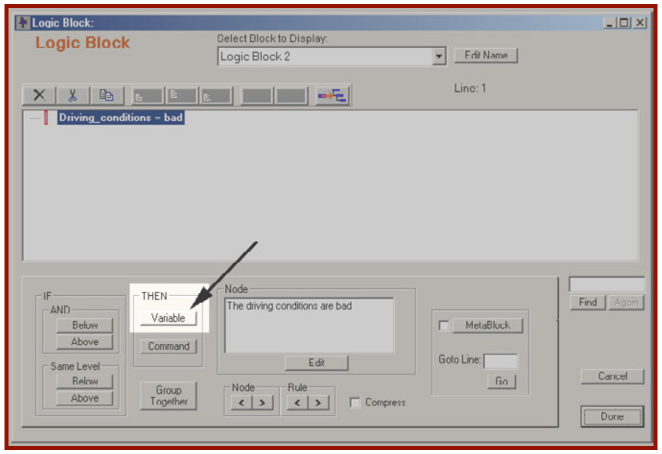


The node will be added to the Logic Block:



[Amount_of_gas] < 2

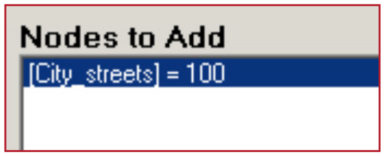
Keeping the new node selected, click the “THEN Variable” button to add the THEN nodes to the Logic Block.



This rule has 3 THEN nodes to add.

In the window for adding nodes:

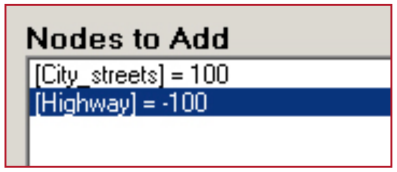
1. Click on the "City_streets" variable to select it.
2. As before, add the value to assign in the edit box. This time the value is: 100.
3. Click the “Add to List” button.



Nodes to Add

[City_streets] = 100

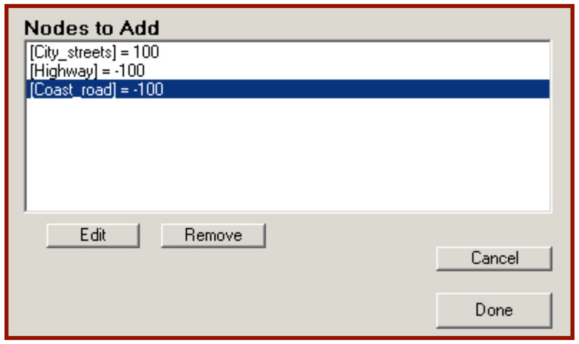
4. DO NOT click the “Done” button yet.
5. Click on the "Highway" variable to select it from the variable list.
6. Assign it a value of -100.
7. Click the Add to List button.



Nodes to Add

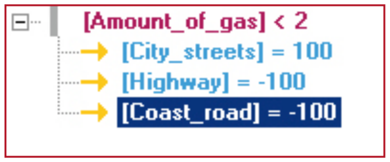
[City_streets] = 100
[Highway] = -100

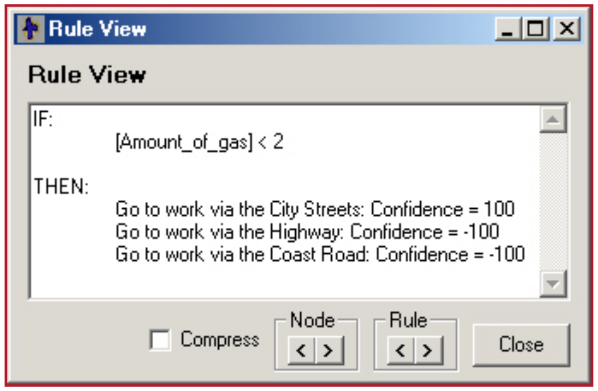
8. Click on the "Coast road" variable to select it from the variable list.
9. Assign it a value of -100.
10. Click the Add to List button.


Nodes to Add

[City_streets] = 100
[Highway] = -100
[Coast_road] = -100

11. Now Click the Done button to add all 3 nodes to the Logic Block.


```
[Amount_of_gas] < 2
    [City_streets] = 100
    [Highway] = -100
    [Coast_road] = -100
```


Rule View

```
IF:
        [Amount_of_gas] < 2

THEN:
        Go to work via the City Streets: Confidence = 100
        Go to work via the Highway: Confidence = -100
        Go to work via the Coast Road: Confidence = -100
```

Using "Same Level Below"

There is one more rule to add to the system.

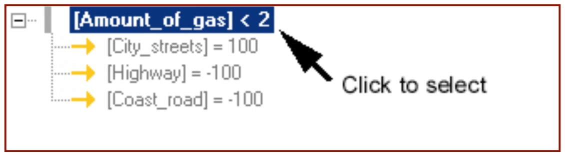
IF
    Amount_of_gas >= 2
THEN
    City Streets = 10
    Highway = 10
    Coast Road = 10

This could be added in another Logic Block, but it is related to the rule that was just added, since it is about the gas in the car. When rules are related, it is a good idea (though not a requirement) to put them in the same Logic Block.

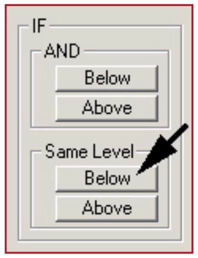
In this case, because of the way they are added, the IF node **[Amount_of_gas] >= 2** is independent of the node **[Amount_of_gas] < 2.**

Previously the nodes were added to the Logic Block indented under a node. Here the node should be at the same level as the **[Amount_of_gas] < 2** node. To do this:

1. Click on the **[Amount_of_gas] < 2** node to select it.

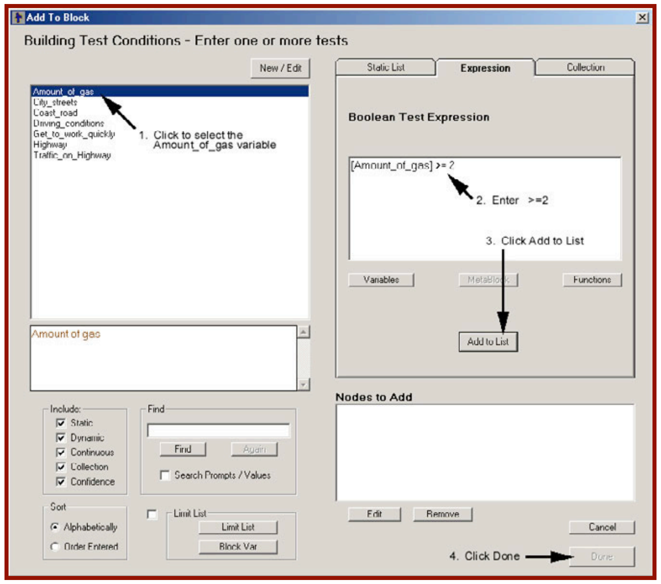

```
[Amount_of_gas] < 2
    [City_streets] = 100
    [Highway] = -100
    [Coast_road] = -100
```

2. Click the "Same Level - Below" button to add the new node under, but **NOT indented** under the selected node.



3. Now add the node **[Amount_of_gas] >=2**

- Click the Amount_of _gas variable to select it.
- In the edit box, add ">=2" to make **[Amount_of_gas] >=2**.
- Click the "Add to List" button.
- Click the "Done" button.



Amount_of_gas
City_streets
Coast_road
Driving_conditions
Get_to_work_quickly
Highway
Traffic_on_Highway

Amount of gas

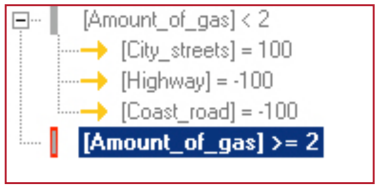
[Amount_of_gas] >= 2

Include: [✓ Static] [✓ Dynamic] [✓ Continuous] [✓ Collection] [✓ Confidence]

Search Prompts / Values

Sort: [✓ Alphabetically] Order Entered

4. This will add the new node to the Logic Block:



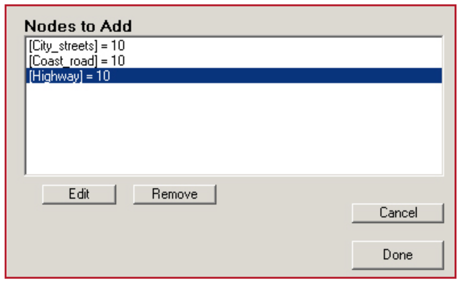
[Amount_of_gas] < 2
[City_streets] = 100
[Highway] = -100
[Coast_road] = -100
[Amount_of_gas] >= 2

Notice that the new node is at the same level of indentation as the **[Amount_of_gas] < 2** node.

5. Now just add THEN nodes under the new **[Amount_of_gas] >=2** node.

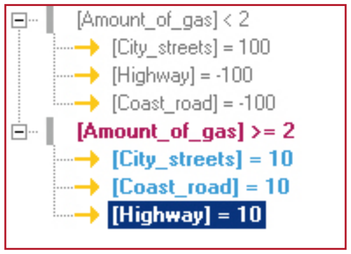
- If the new node is not selected, click on it to select it.
- Click on the THEN-Variable button.
- Build 3 nodes, just as above, but this time with values:

City Streets = 10
Highway = 10
Coast Road = 10



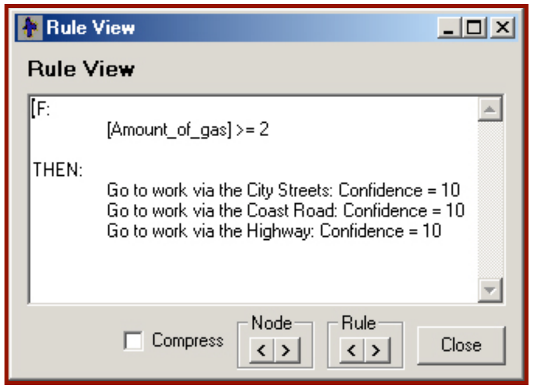
[City_streets] = 10
[Coast_road] = 10
[Highway] = 10

- Click the Done button to add the nodes to the Logic Block



[Amount_of_gas] < 2
[City_streets] = 100
[Highway] = -100
[Coast_road] = -100
[Amount_of_gas] >= 2
[City_streets] = 10
[Coast_road] = 10
[Highway] = 10

This Logic Block now has 2 related, but independent rules. Click on the **[Highway] = 10** node and look at the Rule View window.



Rule View

IF:
[Amount_of_gas] >= 2

THEN:
Go to work via the City Streets: Confidence = 10
Go to work via the Coast Road: Confidence = 10
Go to work via the Highway: Confidence = 10

That adds all the rules for the sample system.

Corvid Command Blocks

Every Corvid system MUST have a Command Block that controls the system.

The Logic and Action Blocks in a system tell it HOW to do something.

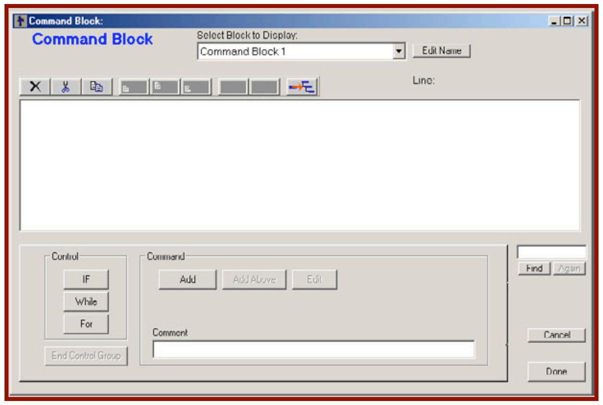
The Command Blocks tell it WHAT to do and WHEN.

Command Blocks can include very complex procedural control with looping and nested layers of control - but most systems have very simple Command Blocks with only a few commands.

Typical Command Blocks simply tell the system what variable, or variables, to derive values for, and then display the results. This can be done with 2 commands.

To start a Command Block, select "Command Block" under the "Windows" menu or click on the Command Block icon:

This will display the Command Block window:

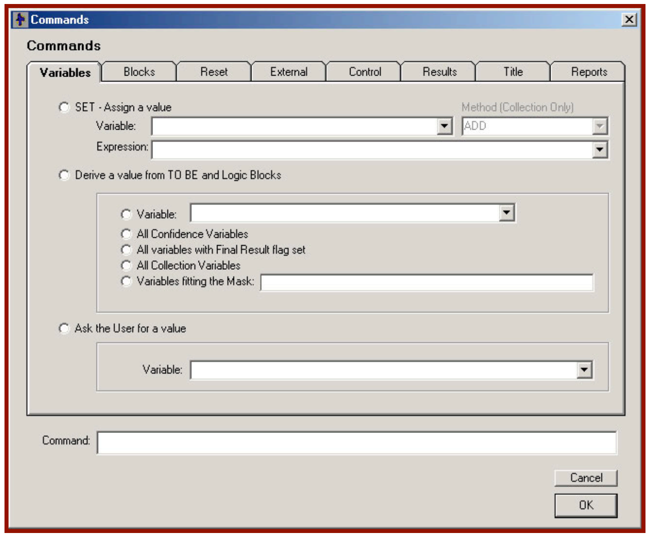


There are 2 types of commands that can be added to a Command Block - Control Commands and Operational Commands.

Control commands handle IF, WHILE and FOR loops. These are added using the Control button group on the left. These are very powerful commands for advanced systems, but are rarely needed for most systems.

The Operational Commands instruct Corvid to perform specific operations. **Click the "Add" button to add an operation command to the Command Block.**

This displays the "Commands" window for building commands:



The Command Builder window has several tabs and many options. It allows all Corvid commands to be built without having to remember command syntax.

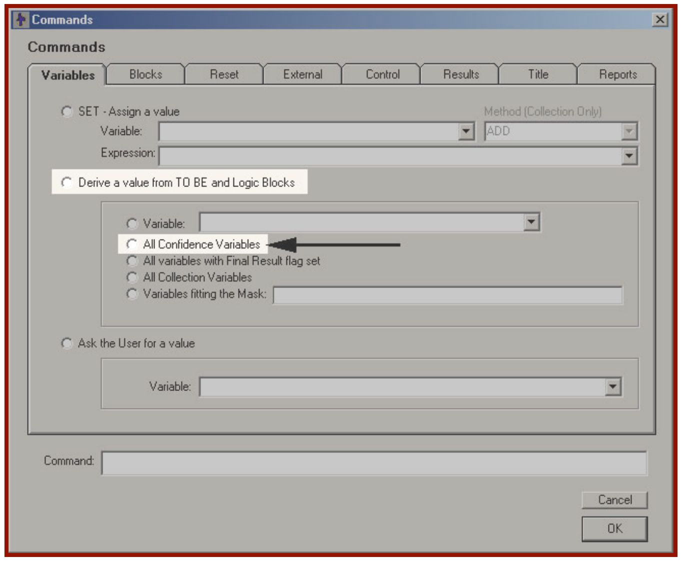
The top "Variables" tab is the most used tab. It allows you to build 3 types of commands for variables:

- SET - Assigns a value to a variable. In most cases this is better done in the Logic Blocks or in the initialization of the variable itself, so this is not usually needed in simple systems.

- DERIVE - Use the Logic Blocks, and other means, to find the value for a variable. This is the section that is most commonly used to run systems. A simple DERIVE command will instruct the Corvid Inference Engine to use all the Logic Blocks to determine the value of one or more variables using backward chaining if needed.

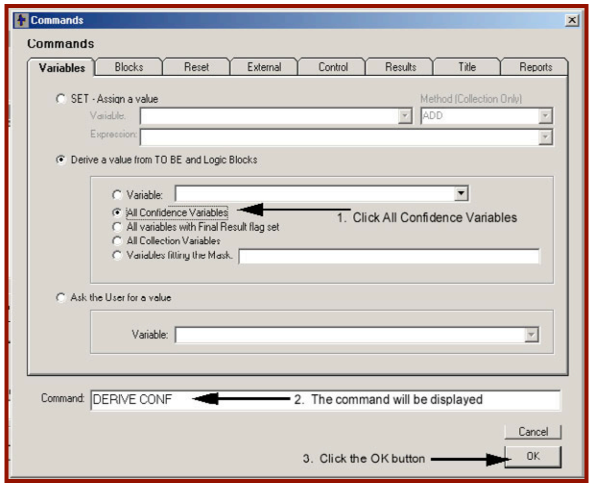
- ASK - Asks the end user for the value of a variable, regardless of the logic of the system. Corvid automatically asks the user for variables in an order determined by the logic of the system. Sometimes a developer may want to override this operation and force a variable to be asked at a particular time. This is not usually needed for simple systems where Corvid's sequence of questions is appropriate.

Most simple systems use a single DERIVE command to run the rules. In the case of the sample system, the various "routes to work" that the system will decide among are all Confidence Variables. All that is needed is to tell Corvid to use the rules in the Logic Blocks to calculate the value for each of the Confidence Variables in the system. This can be done with a single command.



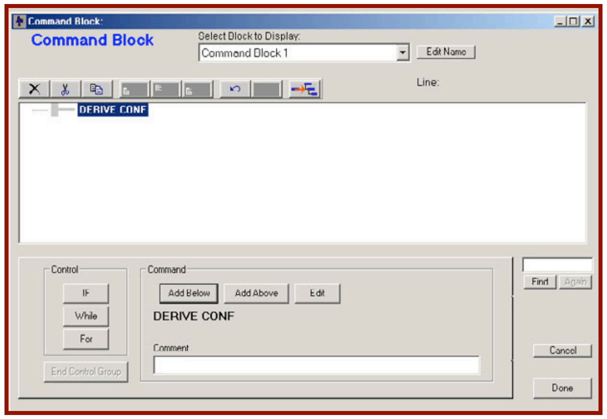
In the DERIVE section, click on the "All Confidence Variables" radio button to select it. (This will automatically also select the "Derive" radio button.)

This builds the command DERIVE CONF.



Command: DERIVE CONF

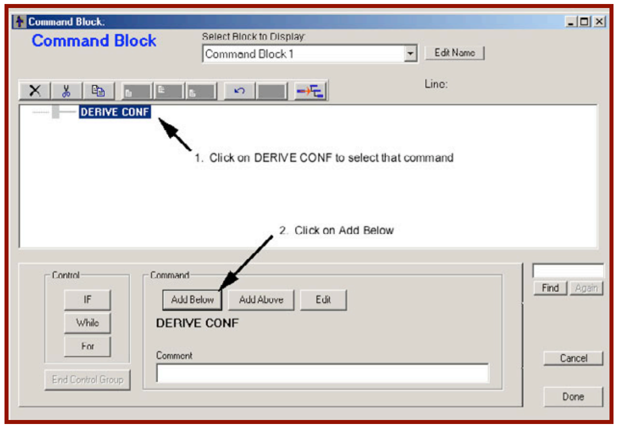
Clicking on the OK button adds the command to the Command Block.



DERIVE CONF

That is all that is needed to have the Logic Blocks derive the values for all the Confidence Variables. The rules in the Logic Blocks that set values for Confidence Variables will be tested. If the values of other variables are needed, the system user will be asked to provide that information. This will do everything except display the results.

Another command will be needed to display the results. All commands are added relative to a currently selected command. If it is not already selected, click on "DERIVE CONF" in the Command Block to select it. Then click on the "Add Below" button to add the next command below this one.



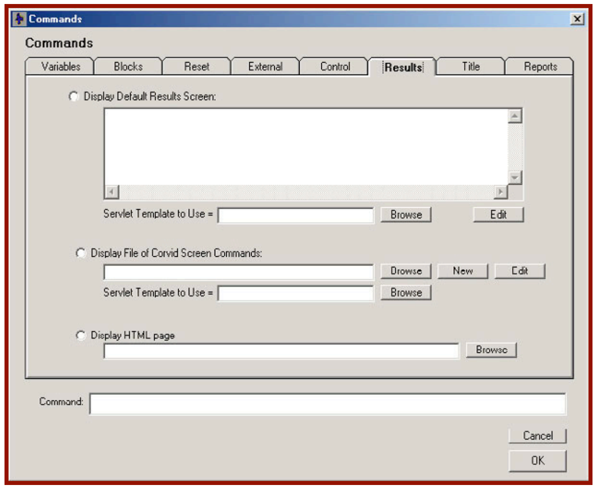
1. Click on DERIVE CONF to select that command

2. Click on Add Below

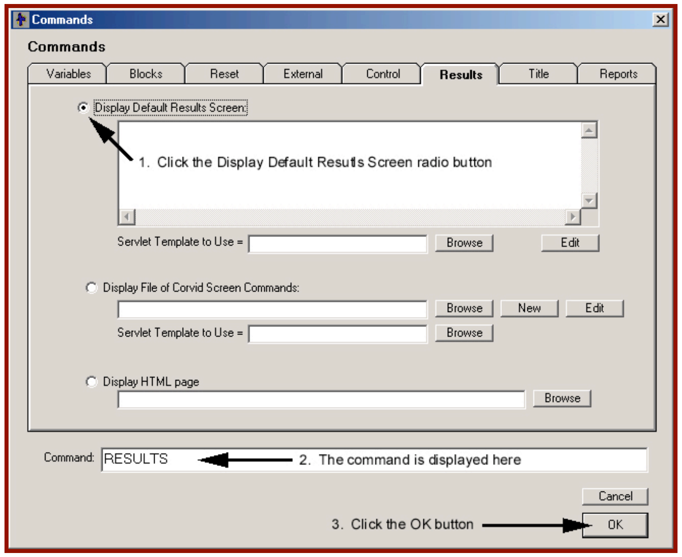
DERIVE CONF

Comment

This again displays the Command Builder window. This time click on the Results tab to select it.



As you will see later, the Results tab provides options to control and format the results in many ways. However, here all that you want to do is to see the values that are set when the system is run. This can be done very simply with the default RESULTS command.



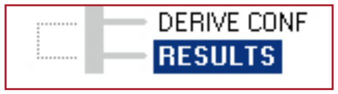
1. Click the Display Default Resutls Screen radio button

Command: RESULTS

2. The command is displayed here

3. Click the OK button

Click the "Display Default Results Screen" radio button, and then click the OK button to add the command to the Command Block.

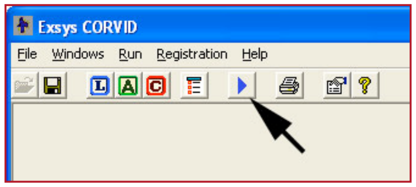


DERIVE CONF
RESULTS

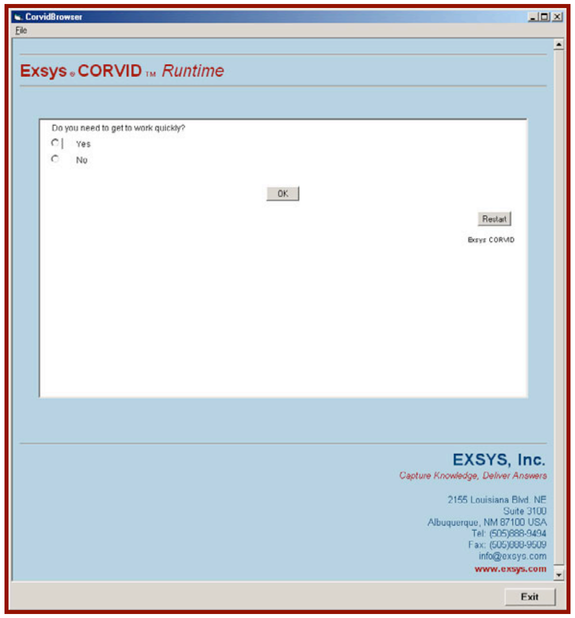
This very simple Command Block will derive the values for all Confidence Variables, using backward chaining and user input, and then display the value of all variables set in the system. For many systems that use Confidence Variables for the options that the system selects from, this is the only Command Block you will need.

Running the Sample System

The sample system now can be run. Select "Start Run" under the "Run" menu, or click on the blue triangle Run icon:



When running a system, Corvid automatically builds an HTML page, and uses the Corvid Runtime Applet, which is displayed in the Corvid Browser window. This window uses the core of Microsoft Internet Explorer but does not have the normal browser navigation controls.



Do you need to get to work quickly?

Yes

No

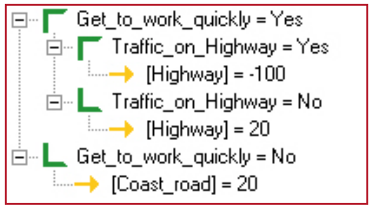
EXSYS, Inc.
Capture Knowledge, Deliver Answers

2155 Louisiana Blvd. NE
Suite 3100
Albuquerque, NM 87100 USA
Tel: (505)888-9494
Fax: (505)888-9509
info@exsys.com
www.exsys.com

The system is run in the applet window in the middle of the page. (Later, to move a system to a Web server, just copy this page, the CORVID Applet Runtime program and the knowledge base files to the server - that's all there is to it.)

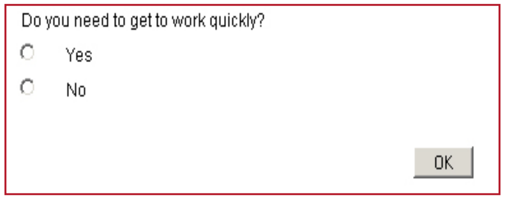
The Corvid Applet Runtime contains the Corvid Inference Engine. This is the "brain power" of Corvid. It checks the Command Block to see what action you have told the system to do, and then uses the Logic Blocks to do what it has been instructed.

In this case, the Command Block says to derive the value for all the Confidence Variables. The first Confidence Variable in the system is the "Highway" route, so that is the first one that will be derived. The first Logic Block that could set a value for this variable is:



```
Get_to_work_quickly = Yes
    Traffic_on_Highway = Yes
        [Highway] = -100
    Traffic_on_Highway = No
        [Highway] = 20
Get_to_work_quickly = No
    [Coast_road] = 20
```

In order to test this Logic Block and determine what values to assign, the system first needs to know the value of the "Get_to_work_quickly" variable. The only way this value can be obtained is by asking the system user, so Corvid displays a question for the user:



Do you need to get to work quickly?

Yes

No

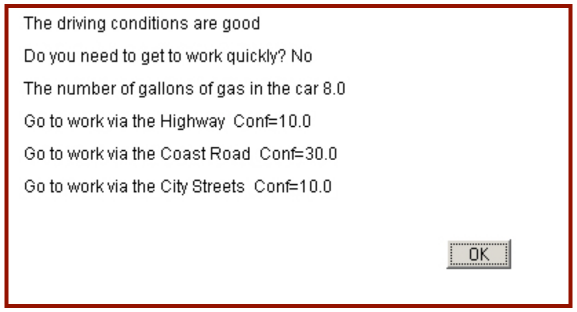
Just select the appropriate answer and click OK.

Corvid will continue asking questions until it has enough information to reach a conclusion about the best route to take.

If the questions are answered:

Need to get to work quickly: NO
Amount of Gas: 8
Driving conditions: GOOD

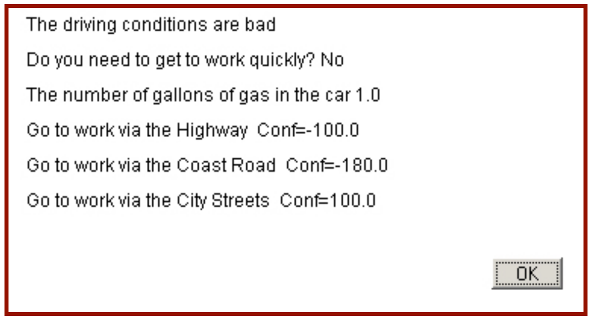
The results will look like:


```
The driving conditions are good
Do you need to get to work quickly? No
The number of gallons of gas in the car 8.0
Go to work via the Highway  Conf=10.0
Go to work via the Coast Road  Conf=30.0
Go to work via the City Streets  Conf=10.0
```

If the input is:

Need to get to work quickly: NO
Amount of Gas: 1
Driving conditions: BAD

The results will be:


```
The driving conditions are bad
Do you need to get to work quickly? No
The number of gallons of gas in the car 1.0
Go to work via the Highway  Conf=-100.0
Go to work via the Coast Road  Conf=-180.0
Go to work via the City Streets  Conf=100.0
```

In this case, the "Highway" route received a value of -100 since there would not be enough gas, and the "Coast Road" route received -180 (little gas and bad driving conditions). These large negative values indicate that they are very bad choices. The "City Street" route received a value of 100, indicating it is really the only viable alternative.

The default RESULTS command displays all of the variables in the system that received a value. This shows the input and the Confidence value assigned to each of the possible routes. The higher the "Conf" value that a route has, the more it is recommended. At the moment, the results show the Confidence Variables in the order that they were entered into the system, regardless of the value they received. Shortly you will see how to arrange them in numeric order and format the results.

There is also a “Back” button available, which will move back one question and allow you to change the answer, and a “Restart” button which will restart the system from the beginning.

Try running the system with various combinations of input to see which route is recommended. When you are finished running the system, click the Exit button at the bottom of the Corvid Browser window.

Changing the User Interface

There are many ways to change the look and feel of the system user interface with Corvid. The Corvid Applet Runtime allows simple changes such as font, color, and alignment, up to advanced highly customized changes such as using image maps to ask questions.

How Corvid Applet Runtime Graphics are Defined

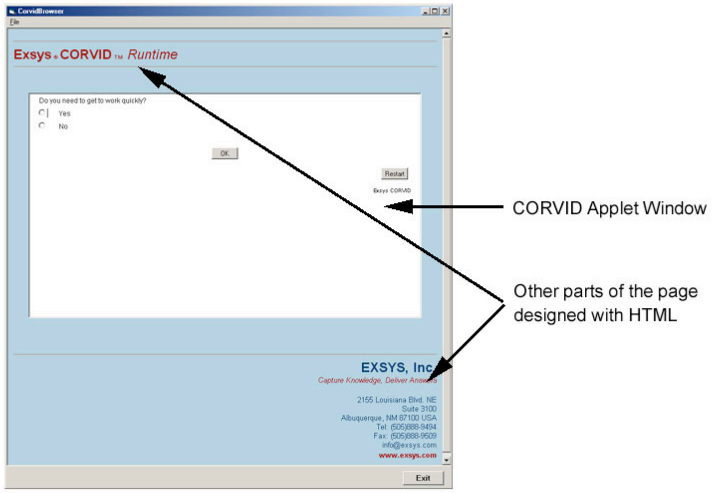
Designing screens for the Corvid Applet Runtime is simple once the underlying structure is understood.

- A screen design is made up of a series of commands.
- Each command displays an item, or group of related items, on the screen.
- Each command can have optional formatting that applies to the item(s) in that command.
- The commands are added to the screen in the order that they are listed in the screen design.

For example, a command might say to display a particular image (.jpg or .gif file), and the formatting might be to center it in the screen. Another command might say to display the value of a particular variable, and the formatting might be the color and font size. Another command might display all the Confidence Variables that received a value greater than 10, one per line. The formatting would apply to each of the variables displayed by the command.

All Corvid Applet screen designs use this same approach.

The screen design commands apply only to the Corvid applet window, which is part of an overall HTML page. You can control the size and placement of the applet window. The rest of the page can be designed using standard HTML commands, and can match the look and feel of an existing site.



CorvidBrowser
File

Exsys® CORVID™ *Runtime*

Do you need to get to work quickly?
Yes
No
OK
Restart
Exsys CORVID

EXSYS, Inc.
Capture Knowledge, Deliver Answers

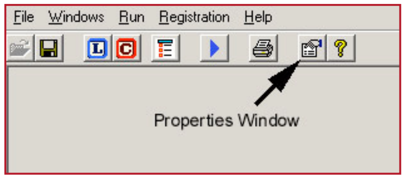
2155 Louisiana Blvd. NE
Suite 3100
Albuquerque, NM 87100 USA
Tel: (505)888-9494
Fax: (505)888-9509
info@exsys.com
www.exsys.com

Exit

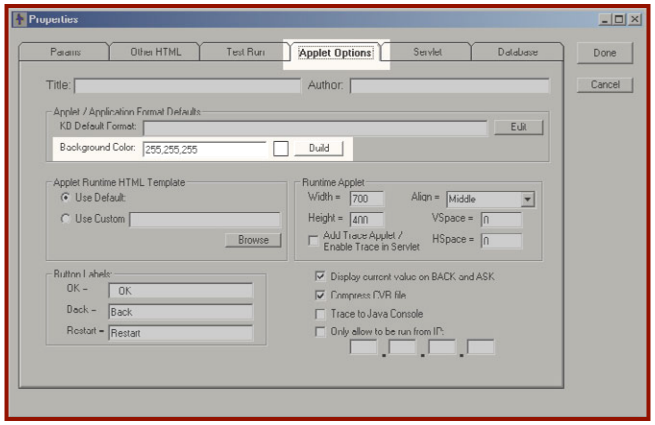
Setting the Background Color

The first step is usually to set the background color for the applet window. The default color is white, but many designs may want a different color. The color can be set individually for each screen, but it is easier to set a consistent background for all screens.

1. Click the Properties Window icon.



2. Select the Applet Options tab.



Title: Author:

Applet / Application Format Defaults
KB Default Format:
Background Color: 255,255,255

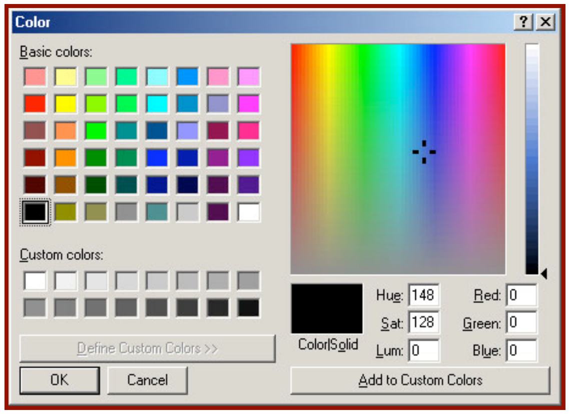
Applet Runtime HTML Template
(● Use Default) Use Custom

Runtime Applet
Width = 700 Align = Middle
Height = 400 VSpace = 0
☐ Add Trace Applet / Enable Trace in Servlet HSpace = 0

Button Labels
OK – OK
Back – Back
Restart – Restart

☑ Display current value on BACK and ASK
☑ Compress CVR file
☐ Trace to Java Console
☐ Only allow to be run from IP:

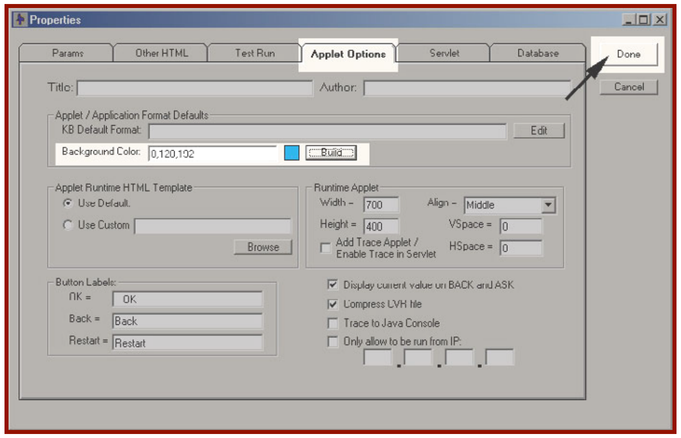
3. Click the Build button to the right of "Background Color". This will display the standard MS Windows Color chooser window.



Hue: 148 Red: 0
Sat: 128 Green: 0
Lum: 0 Blue: 0

4. Select the desired background color and click the OK button.

5. Click the Done button on the Properties window.



Title:
Author:

Applet / Application Format Defaults
KB Default Format:
Background Color: 0,120,192

Applet Runtime HTML Template
[✓ Use Default.]
Use Custom

Runtime Applet
Width = 700
Align = Middle
Height = 400
VSpace = 0
Add Trace Applet / Enable Trace in Servlet
HSpace = 0

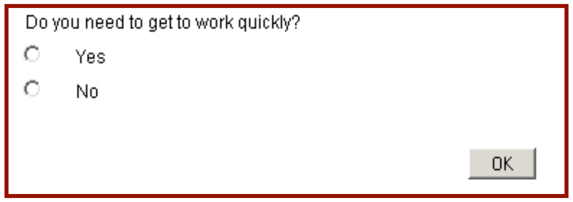
Button Labels:
OK = OK
Back = Back
Restart = Restart

[✓ Display current value on BACK and ASK]
[✓ Compress CVR file]
Trace to Java Console
Only allow to be run from IP:

Now when the system is run, the applet window will use the new background color.

Changing the Way Questions are Asked

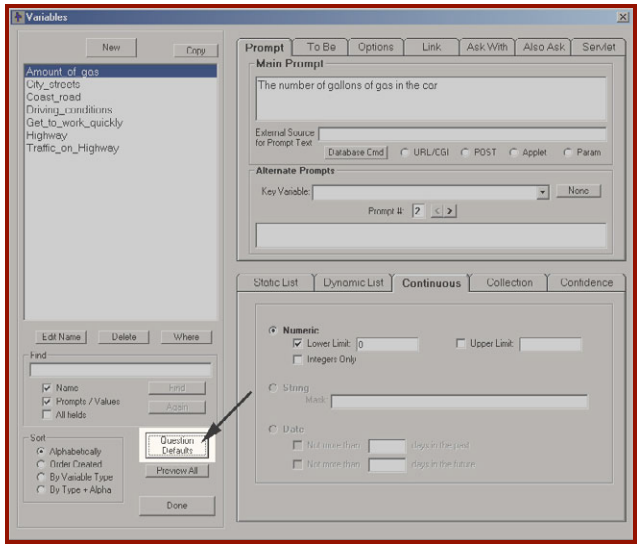
When the system is run, the first question is:



Do you need to get to work quickly?
Yes
No

This is simple and clear, but there are many ways to change it. The quickest way to change all the questions in the system is through the System Question Default formats. These apply to all questions that the system will ask, and make it easy to consistently change the overall look of the system.

To set the Question Defaults, open the Variables window and click the Question Defaults button.



Amount_of_gas
City_streets
Coast_road
Driving_conditions
Get_to_work_quickly
Highway
Traffic_on_Highway

Main Prompt
The number of gallons of gas in the car

External Source for Prompt Text
URL/CGI POST Applet Param

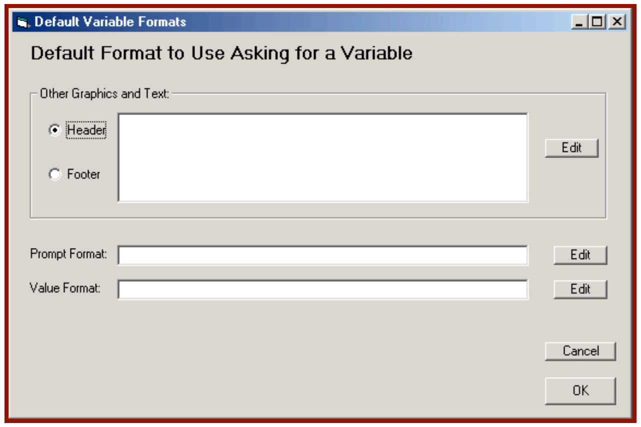
Alternate Prompts
Key Variable:
Prompt #: 2

Find
[✓ Name]
[✓ Prompts / Values]
All fields

Sort
[✓ Alphabetically]
Order Created
By Variable Type
By Type + Alpha

[✓ Numeric]
[✓ Lower Limit: 0]
Upper Limit:
Integers Only
String
Mask:
Date
Not more than days in the past
Not more than days in the future

This will display the “Default Variable Formats” window for setting the overall system properties.



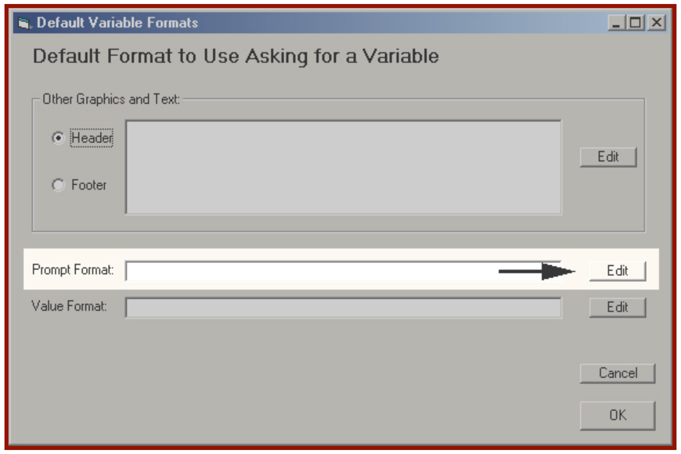
Default Format to Use Asking for a Variable

Other Graphics and Text:
[✓ Header]
Footer

Prompt Format:
Value Format:

The easiest way to control the look and feel is to change the Prompt and Value formats. Each question will be asked using the Prompt and, for Static List variables, the values that were specified when the variable was created.

To control the formatting of the Prompt of the variable, click the Edit button right of the Prompt Format edit box:


Default Variable Formats

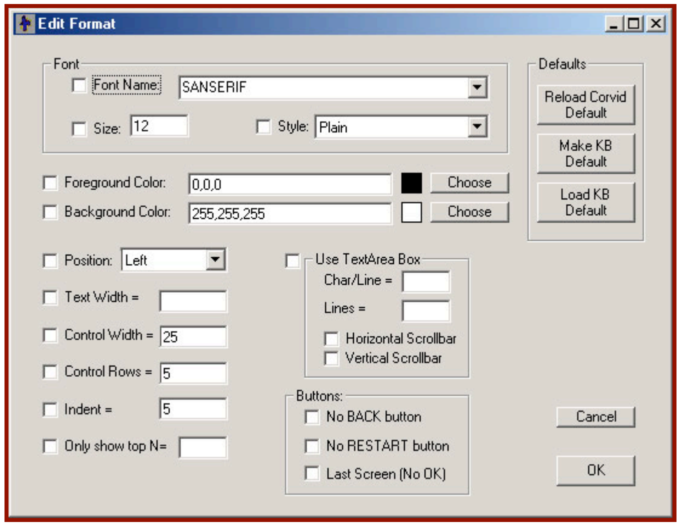
Default Format to Use Asking for a Variable

Other Graphics and Text: [✓ Header] Footer

Prompt Format:

Value Format:

This will display the “Edit Format” window.


Edit Format

Font
☐ Font Name: SANSERIF
☐ Size: 12 ☐ Style: Plain

☐ Foreground Color: 0,0,0 ■
☐ Background Color: 255,255,255 □

☐ Position: Left
☐ Text Width =
☐ Control Width = 25
☐ Control Rows = 5
☐ Indent = 5
☐ Only show top N=

☐ Use TextArea Box
Char/Line =
Lines =
☐ Horizontal Scrollbar
☐ Vertical Scrollbar

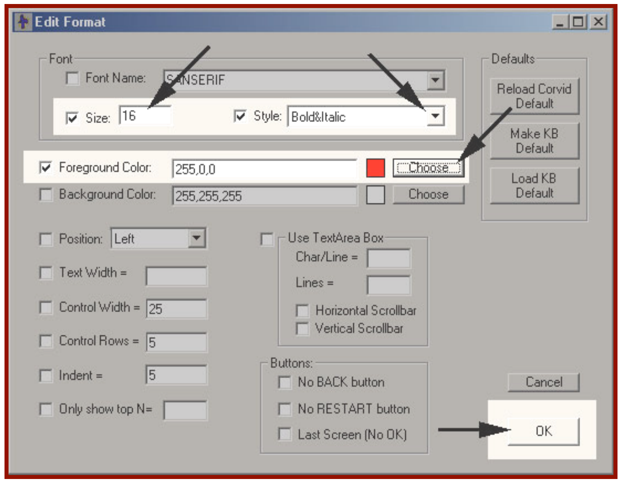
Buttons:
☐ No BACK button
☐ No RESTART button
☐ Last Screen (No OK)

There are a variety of options for how text is formatted. The top of the window sets the font size and style. Since Java is portable across many platforms the number of font styles is limited.

The foreground and background colors for the text can be set in the same way as the overall window background.

Here, make the text 16 point, bold and italic in Red. To do this:

- Enter "16" in the Size edit box.
- Click the "style" drop down list and select "Bold & Italic".
- Click the "Choose" button next to "Foreground Color" and select a bright red


Edit Format

Font
☐ Font Name: SANSERIF
☑ Size: 16 ☑ Style: Bold&Italic

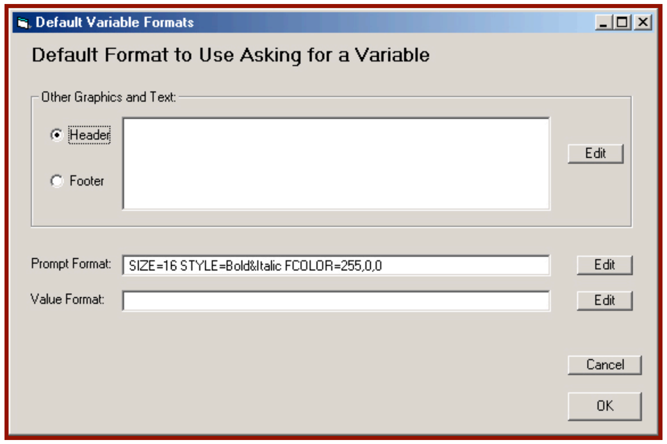
☑ Foreground Color: 255,0,0 🟥
☐ Background Color: 255,255,255 □

☐ Position: Left
☐ Text Width =
☐ Control Width = 25
☐ Control Rows = 5
☐ Indent = 5
☐ Only show top N=

☐ Use TextArea Box
Char/Line =
Lines =
☐ Horizontal Scrollbar
☐ Vertical Scrollbar

Buttons:
☐ No BACK button
☐ No RESTART button
☐ Last Screen (No OK)

Once the selections are made, click the OK button.


Default Variable Formats

Default Format to Use Asking for a Variable

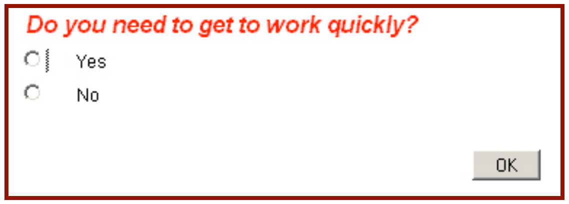
Other Graphics and Text: [✓ Header] Footer

Prompt Format: SIZE=16 STYLE=Bold&Italic FCOLOR=255,0,0

Value Format:

This shows the new Prompt Format style that will apply to all questions in the system. (You can override the default format for any specific question by adding a format for that individual variable.)

Now when the system is run, the first question appears:

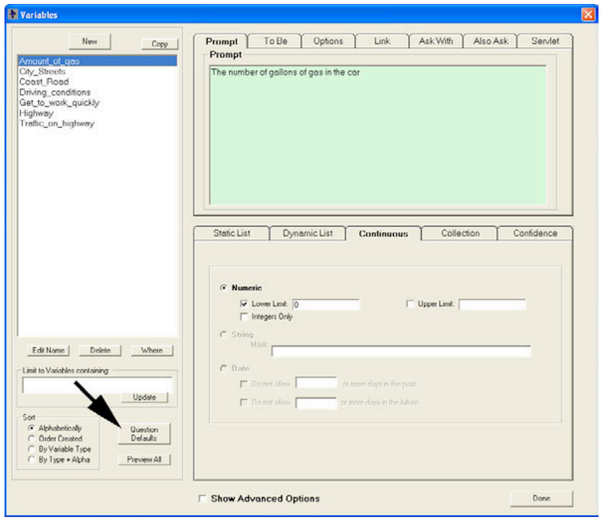


Do you need to get to work quickly?

Yes

No

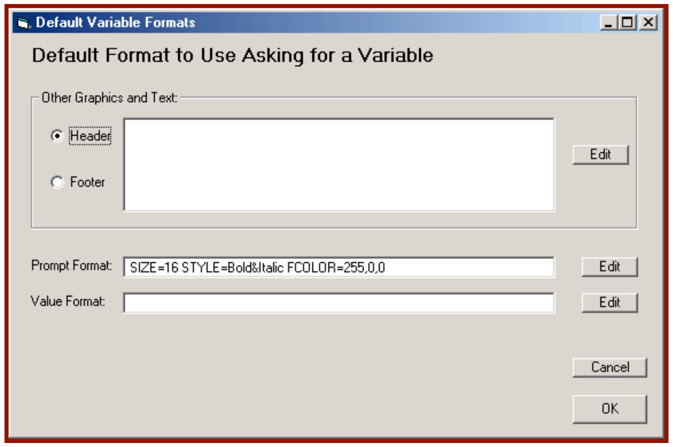
An image could also be added in the window. If you wanted this image added in each question, click the "Question Default" button on the Variables window.



Prompt: The number of gallons of gas in the car

Lower Limit: 0

This will display the "Default Variable Formats" window.



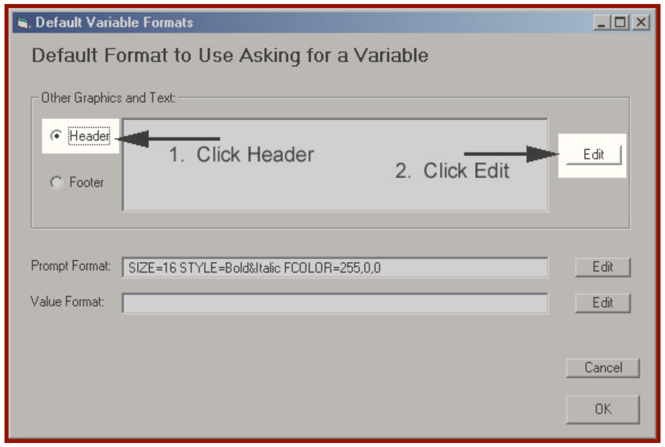
Prompt Format: SIZE=16 STYLE=Bold&Italic FCOLOR=255,0,0

Value Format:

There are 3 parts to a question window:

1. **Header** - This is an area of graphics and text that appears above the actual question.
2. **The question itself** – The variable prompt is displayed with an area for input or selecting a value from a list. The formatting of this part is controlled by the Prompt Format and Value Format settings, along with options set for the variable itself.
3. **Footer** - This is an area of graphics and text that appears below the question.

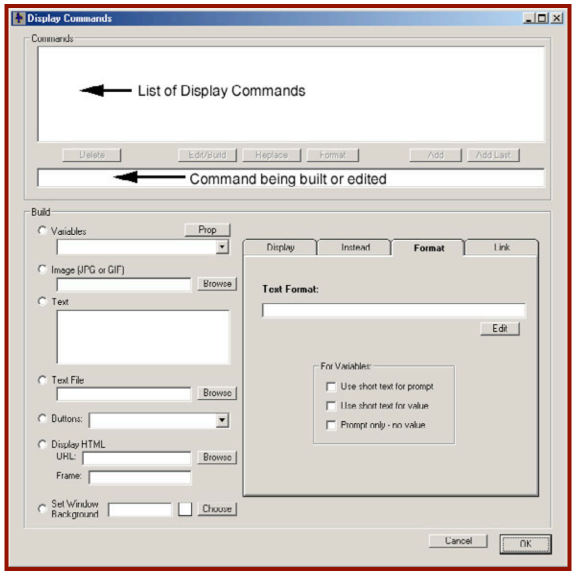
In this case, an image is to be added above each question, so it will be a Header. Click the Header radio button and then click the Edit button to the right



Prompt Format: SIZE=16 STYLE=Bold&Italic FCOLOR=255,0,0

Value Format:

This will bring up the Display Command window:



This window has 3 main areas:

1. A list of Display Commands - this is in the top edit box.
2. A window showing the command currently being built or edited.
3. A set of controls for building and formatting various display commands.

Since there are no Display Commands added yet, the list is empty.

To add a JPG image to the window, enter the image name in the Image edit box, or click the Browse button and select the image to use. The default applet window size is 700 pixels wide by 400 pixels high. (This can be changed to any value desired). The image should be sized in pixels to fit in a portion of the window. For example, in the default window, an image 300 pixels wide by 75 pixels high would fit nicely in the upper left corner.

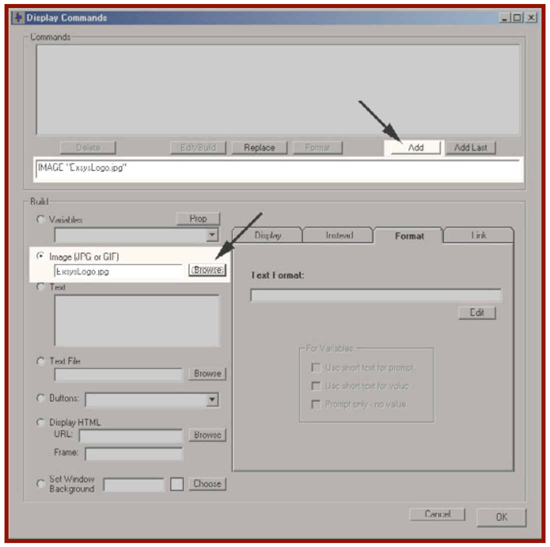


IMAGE "ExsysLogo.jpg"

ExsysLogo.jpg

Once an image is entered, or selected, the edit box for the current command will show: **IMAGE "filename"**, where filename is the name of the selected image file. To add the command to the list of Display Commands, click the Add button.

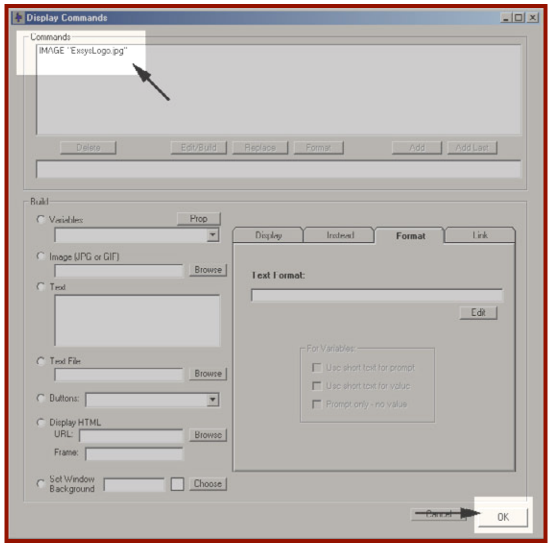
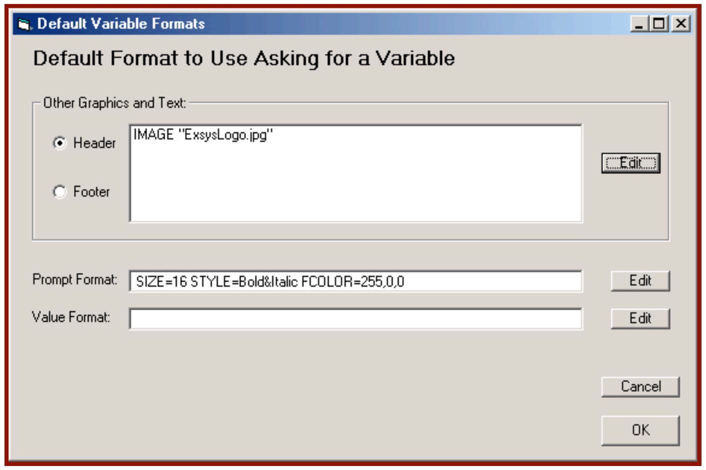


IMAGE "ExsysLogo.jpg"

Now the IMAGE command has been added to the Display Command List. That is the only command needed, so click the OK button.



Default Variable Formats

Default Format to Use Asking for a Variable

Other Graphics and Text:

[✓ Header] Footer

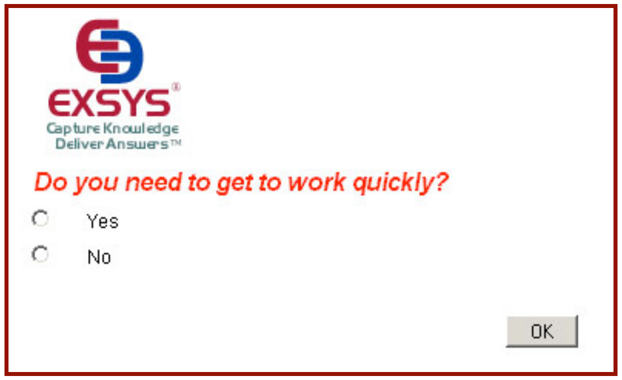
IMAGE "ExsysLogo.jpg"

Prompt Format: SIZE=16 STYLE=Bold&Italic FCOLOR=255,0,0

Value Format:

The Default Format window now shows the image for the header and the Prompt format.

Running the system now will display:



Do you need to get to work quickly?

Yes

No

Every question will have the image at the top.

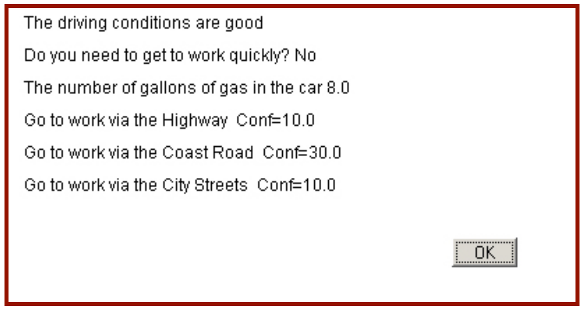
It is easy to build screen designs that are portable across browsers and operating systems.

Formatting the Results

When the system is run with input:

Need to get to work quickly: **NO**
Gallons of Gas: **8**
Driving Conditions: **GOOD**

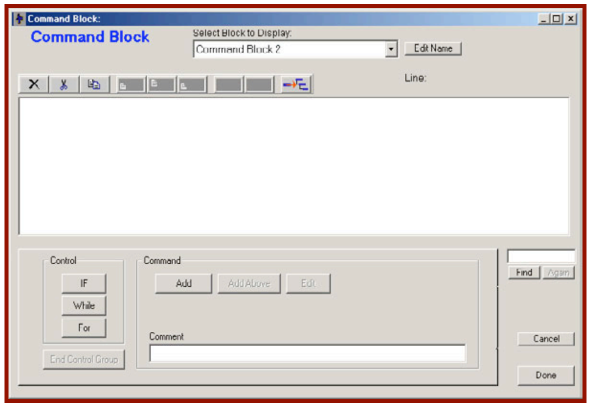
The results are:



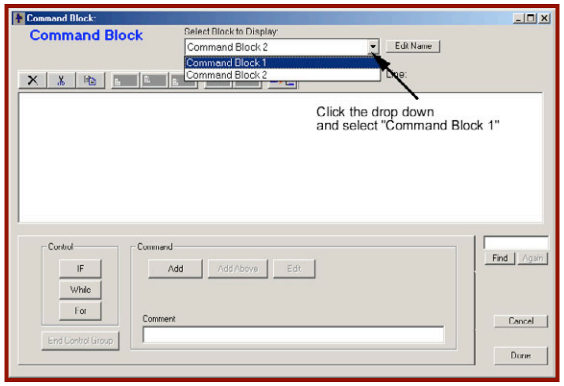
```
The driving conditions are good
Do you need to get to work quickly? No
The number of gallons of gas in the car 8.0
Go to work via the Highway  Conf=10.0
Go to work via the Coast Road  Conf=30.0
Go to work via the City Streets  Conf=10.0
```

These are the correct results based on the logic, but it would be better if the user input was not displayed and the routes recommended were arranged in order of Confidence value, rather than the order of the variables in the system.

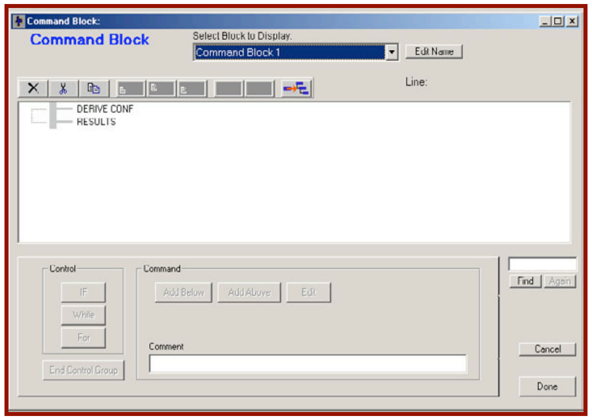
To do this requires editing the RESULTS command in the Command Block. Click the Command Block icon to display a Command Block window:



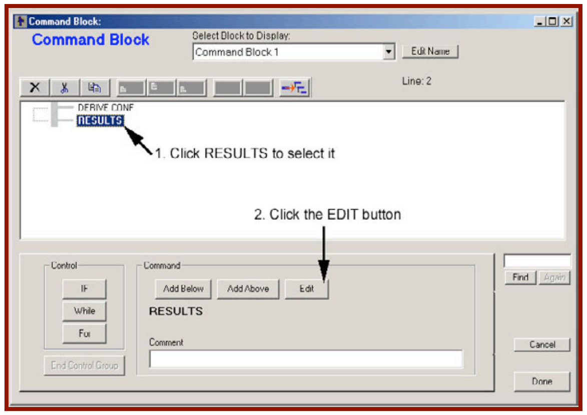
This will be a new Command Block window and is labeled "Command Block 2". The block that was added earlier was "Command Block 1". To see that window, click the drop down in the window title and select "Command Block 1".



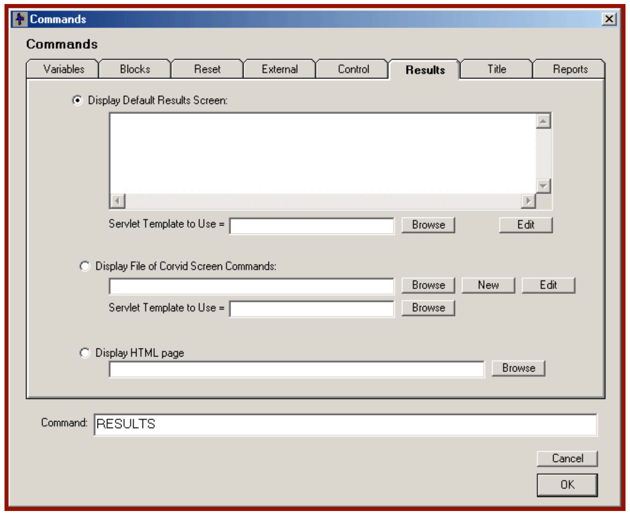
This will display the Command Block that was entered earlier.



To edit the results, click the RESULTS line to select it and then click the Edit button.

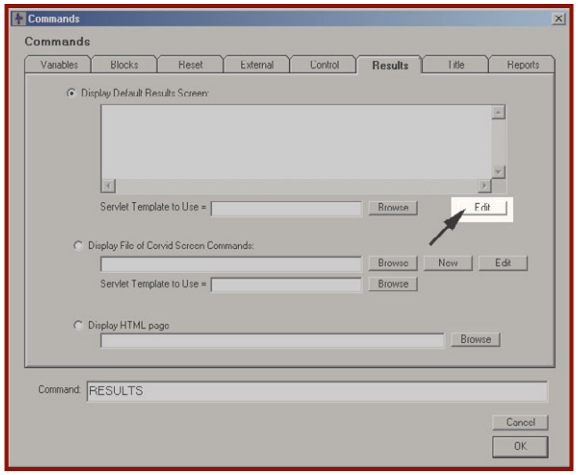


This displays the Commands building window that was used to build the RESULTS command.



The Display Results radio button is selected, but the list of commands under it is empty. When this is empty, the default is to display the value of all variables used in the system. This is changed by adding Display Commands associated with the RESULTS command.

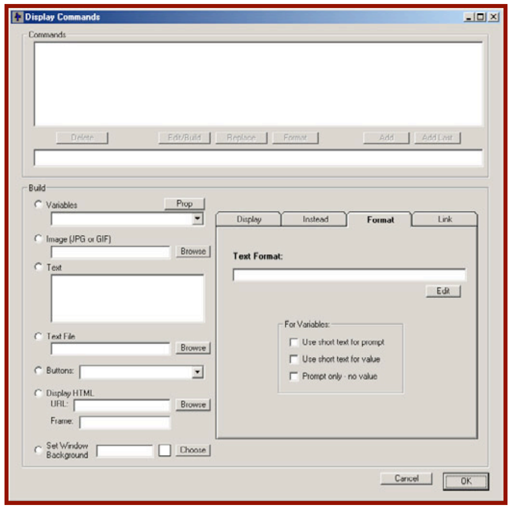
To do this, click the Edit button under the Results section.



Display Default Results Screen [✓]

Command: RESULTS

This shows the Display Command window that was used before to add a Header to the questions.



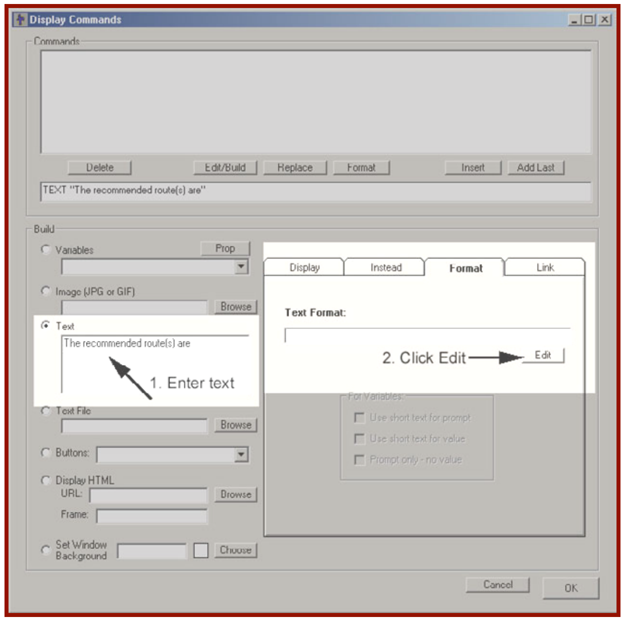
This time it will be used to build a set of commands to display the results.

The list of commands is empty. A good results screen should explain to the system user what the results mean, and display the recommendations in the most useful way.

This will require adding 2 commands:

1. Text explaining the content of the results screen.
2. A list of the recommended routes, in order of confidence value.

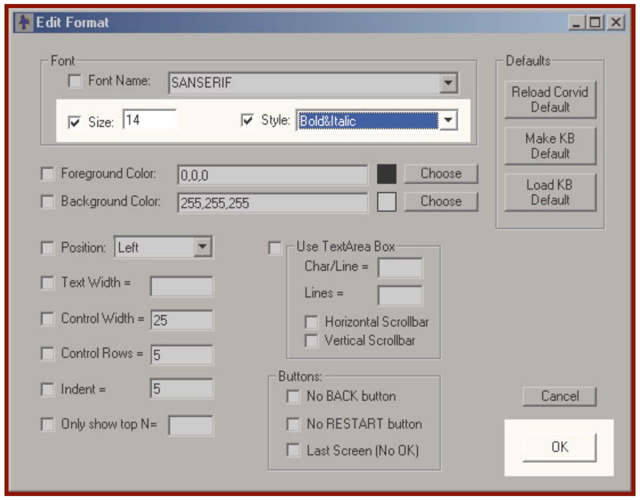
This is done in the Display Command window. First use the "Text" edit box to enter the text: "The recommended route(s) are:"



TEXT "The recommended route(s) are"

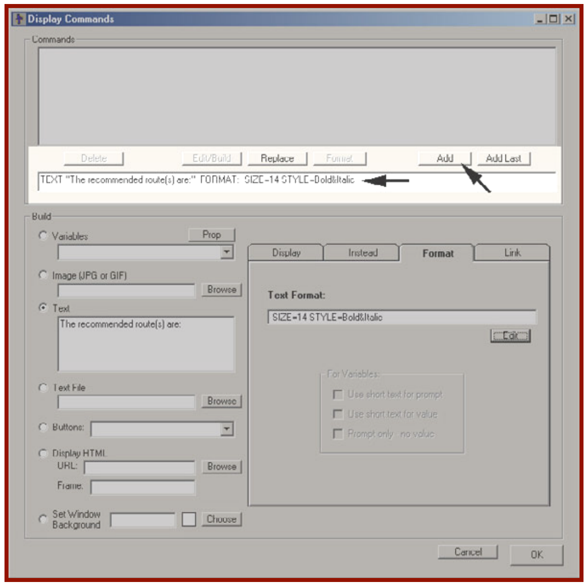
Text [✓]: The recommended route(s) are

Then click the Edit button on the Format tab. This will allow the formatting of the new text.



Font Name: SANSERIF
[✓] Size: 14
[✓] Style: Bold&Italic
Foreground Color: 0,0,0
Background Color: 255,255,255
Position: Left
Text Width =
Control Width = 25
Control Rows = 5
Indent = 5
Only show top N=
Use TextArea Box
Char/Line =
Lines =
Horizontal Scrollbar
Vertical Scrollbar
No BACK button
No RESTART button
Last Screen (No OK)

In the Format window, select a Size of "14" and Style of "Bold&Italic". Then click the OK button.

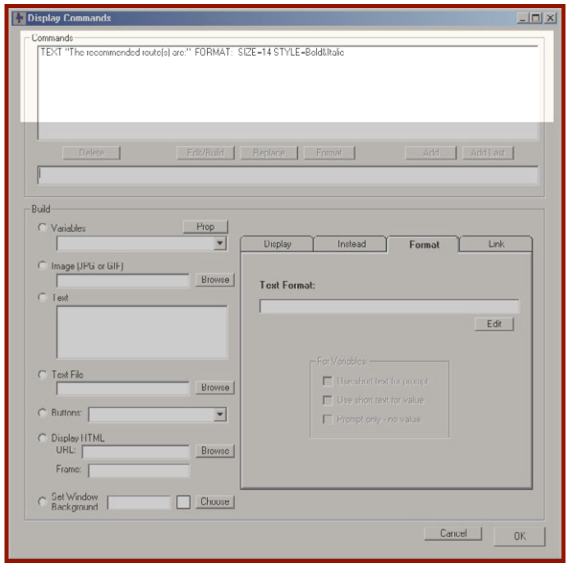


TEXT "The recommended route(s) are:" FORMAT: SIZE=14 STYLE=Bold&Italic

Text: The recommended route(s) are:

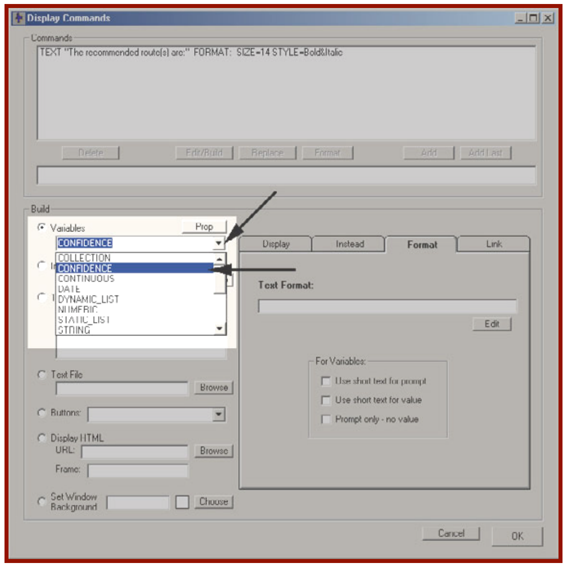
Text Format: SIZE=14 STYLE=Bold&Italic

This returns to the Display Command window. The command that was built is shown in the edit box. Since this is the command wanted, click the Add button to add it to the list of display commands.



TEXT "The recommended route(s) are:" FORMAT: SIZE=14 STYLE=Bold&Italic

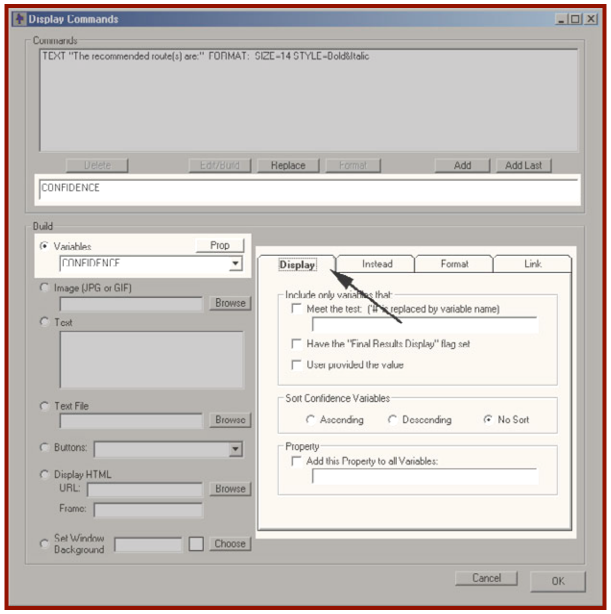
That will provide an explanatory heading. Now to add the list of selected routes. The routes are the only Confidence Variables in the system, so first you want to display all the Confidence Variables:



TEXT "The recommended route(s) are:" FORMAT: SIZE=14 STYLE=Bold&Italic

Variables: CONFIDENCE
COLLECTION
CONFIDENCE
CONTINUOUS
DATE
DYNAMIC_LIST
NUMERIC
STATIC_LIST
STRING

Click on the dropdown next to the "Variables" radio button and select "Confidence".



```
TEXT "The recommended route(s) are:" FORMAT: SIZE=14 STYLE=Bold&Italic
```

CONFIDENCE

[✓ Variables]: CONFIDENCE

Image (JPG or GIF)

Text

Text File

Buttons:

Display HTML URL: Frame:

Set Window Background

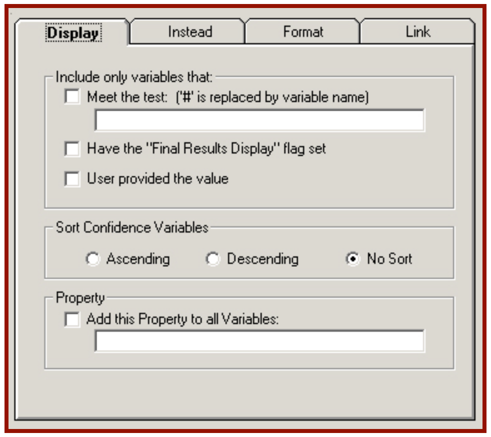
Include only variables that:
- ☐ Meet the test: ('#' is replaced by variable name)
- ☐ Have the "Final Results Display" flag set
- ☐ User provided the value

Sort Confidence Variables: Ascending Descending [✓ No Sort]

Property
- ☐ Add this Property to all Variables:

This command alone would display all the Confidence Variables, but they would not be sorted in order of confidence value. Also, some combinations of input would include routes that are given negative values - meaning they are NOT recommended, and should not be displayed.

The list of displayed Confidence values can be limited by using the options on the Display tab.

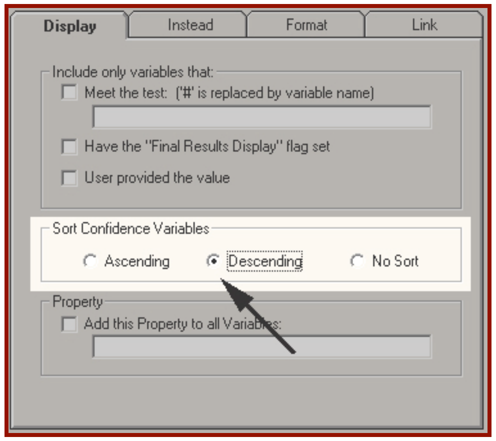


Include only variables that:
- ☐ Meet the test: ('#' is replaced by variable name)
- ☐ Have the "Final Results Display" flag set
- ☐ User provided the value

Sort Confidence Variables: Ascending Descending [✓ No Sort]

Property
- ☐ Add this Property to all Variables:

The first step is to arrange the list of displayed Confidence Variables with the highest confidence value at the top. This is done using the "Sort Confidence Variables" options.



Include only variables that:
- ☐ Meet the test: ('#' is replaced by variable name)
- ☐ Have the "Final Results Display" flag set
- ☐ User provided the value

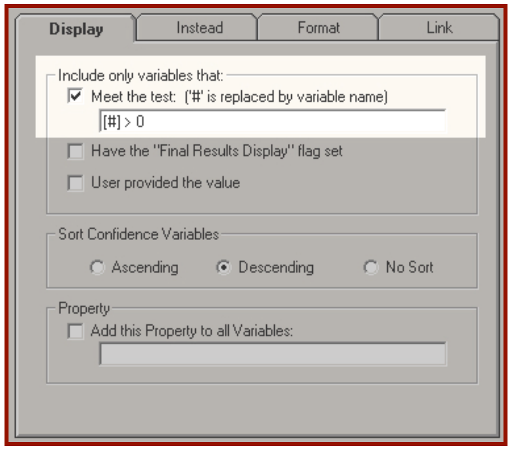
Sort Confidence Variables: Ascending [✓ Descending] No Sort

Property
- ☐ Add this Property to all Variables:

Select the "Descending" radio button. This will have the highest confidence value at the top, with the others arranged in order of confidence value.

The next step is to make sure that only Confidence Variables that have a value greater than 0 are displayed. (Ones with negative values are not recommended - in fact they are specifically rejected.)

This is done from the "Include only variables that:" section.



Include only variables that:
- ☑ Meet the test: ('#' is replaced by variable name)
  [#] > 0
- ☐ Have the "Final Results Display" flag set
- ☐ User provided the value

Sort Confidence Variables: Ascending [✓ Descending] No Sort

Property
- ☐ Add this Property to all Variables:

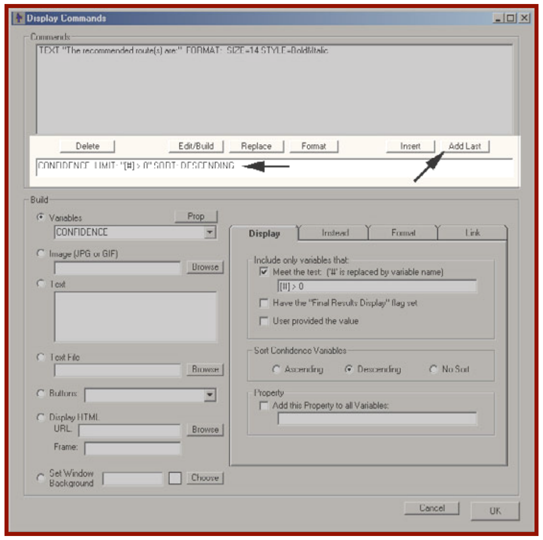
The "Meet the test" section allows entering a Boolean test expression. The value of the Confidence Variable must meet this test to be displayed. In order to make the test apply to each variable in turn, the "#" character is used to represent the NAME of the variable. This allows quite complex tests in advanced systems, but most systems only need the value. This is indicated by using the name of the variable in square brackets [#], which in the test expression will be the value of the variable. For example, if there is a variable [COST], using [#] > 0 will have the # replaced by COST making [COST]>0 as the test expression for the variable [COST]. This will be repeated for each variable allowed by the command (e.g. all confidence variables). Each will have the # replaced by the name of the variable, and the resulting test expression evaluated.

To only allow Confidence Variables that have a value greater then 0, enter the test expression

[#] > 0

The test expressions can be quite complex, but most are simple limits such as this one.

That is all that is needed to limit the display to variables that have a value greater than 0.

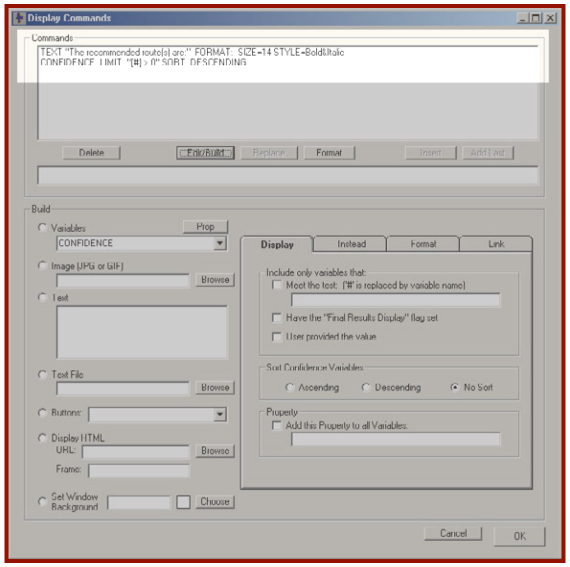

TEXT "The recommended route(s) are:" FORMAT: SIZE=14 STYLE=Bold|Italic

CONFIDENCE LIMIT: "[#] > 0" SORT: DESCENDING

Meet the test: ('#' is replaced by variable name): [#] > 0

Sort Confidence Variables: Descending

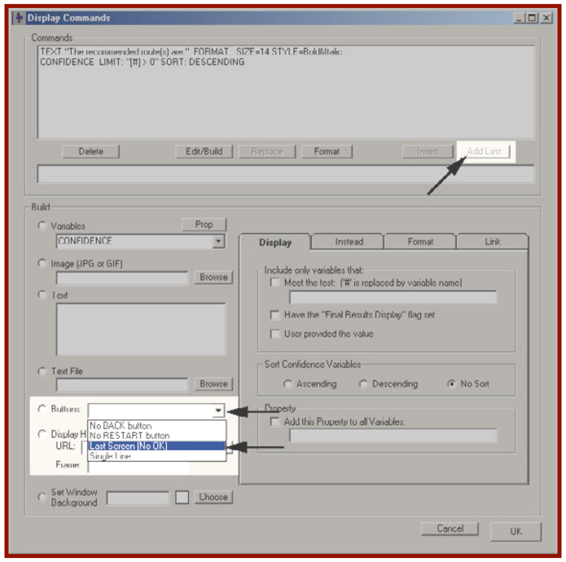
The command is displayed in the edit window. Click the Add Last button to add the new command as the last command in the Display Command list.


TEXT "The recommended route(s) are:" FORMAT: SIZE=14 STYLE=Bold|Italic

CONFIDENCE LIMIT: "[#] > 0" SORT: DESCENDING

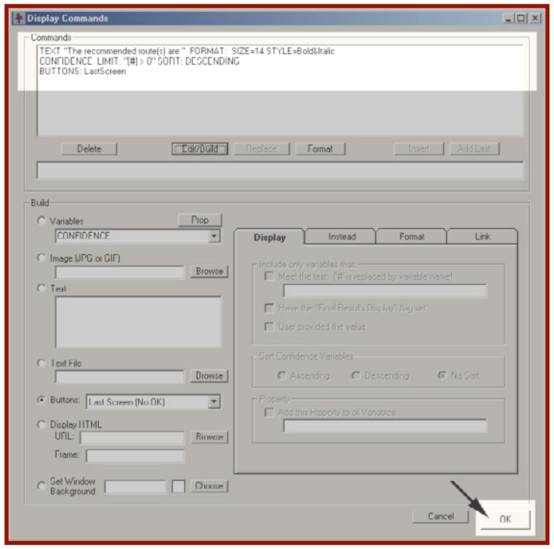
That is all that is needed to display the results.

Usually all user interface screens have an OK button to continue processing. This would also happen here, but clicking the OK button would just display a "System Done" message. This can be avoided by not including an OK button. To do this, add one more command on the Results screen.


TEXT "The recommended route(s) are:" FORMAT: SIZE=14 STYLE=Bold|Italic

CONFIDENCE LIMIT: "[#] > 0" SORT: DESCENDING

This time click the drop down list next to "Buttons" and select "Last Screen (No OK)". These buttons do not take any formatting options, so just click "Add Last" to add the command to the end of the list.

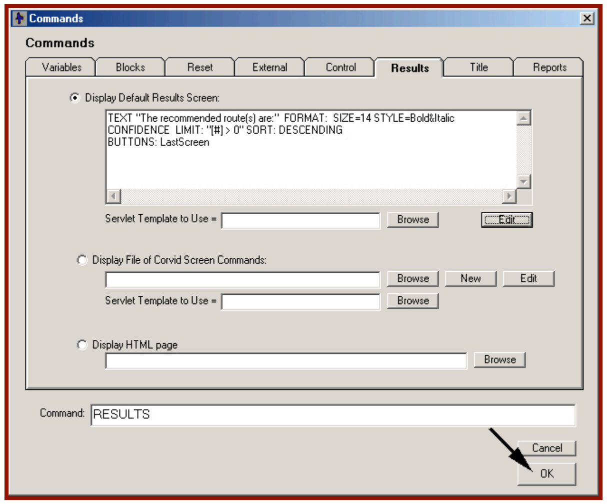


```
TEXT "The recommended route(s) are:"  FORMAT:  SIZE=14 STYLE=Bold&Italic
CONFIDENCE  LIMIT: "[#] > 0" SORT: DESCENDING
BUTTONS: LastScreen
```

Variables: CONFIDENCE
Buttons: Last Screen (No OK)
Sort Confidence Variables: Ascending / Descending / [✓ No Sort]

Those are all the display commands needed, so click the OK button.

This returns to the command builder, but now with the new Display Commands in the list under "Display Default Result Screen". These new commands that were entered will be used whenever the RESULTS command is added to the Command Block.

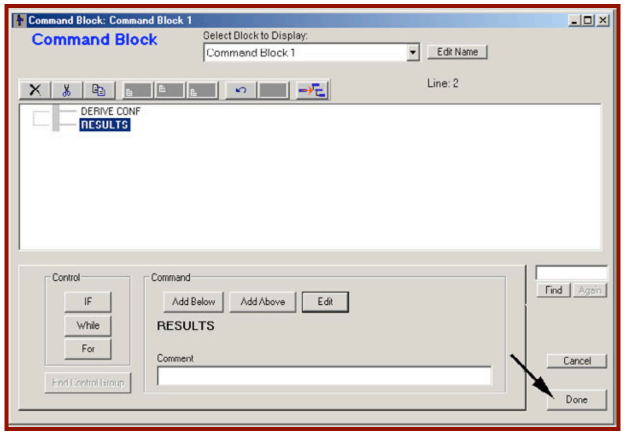


Display Default Results Screen:

```
TEXT "The recommended route(s) are:"  FORMAT:  SIZE=14 STYLE=Bold&Italic
CONFIDENCE  LIMIT: "[#] > 0" SORT: DESCENDING
BUTTONS: LastScreen
```

Command: RESULTS

Click the OK button to return to the Command Block.

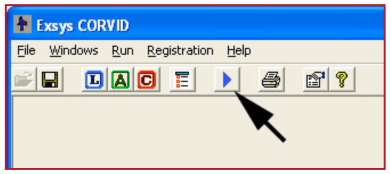


```
DERIVE CONF
RESULTS
```

Select Block to Display: Command Block 1
Command: RESULTS

Click the Done button, and you are ready to run the system with the new interface.

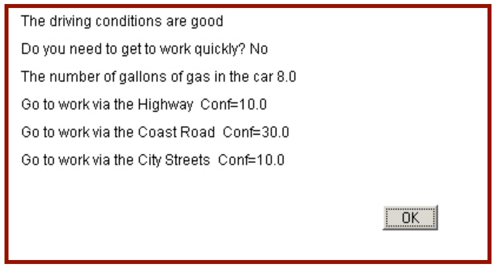
Click the Run icon:



Input the answers:

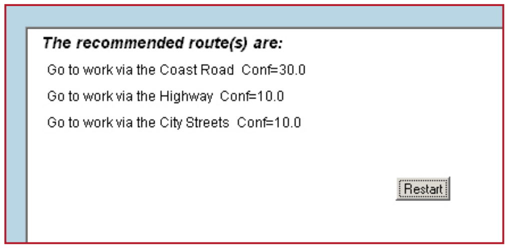
Need to get to work quickly: **NO**
Amount of Gas: **8**
Driving conditions: **GOOD**

Before the results looked like:



```
The driving conditions are good
Do you need to get to work quickly? No
The number of gallons of gas in the car 8.0
Go to work via the Highway  Conf=10.0
Go to work via the Coast Road  Conf=30.0
Go to work via the City Streets  Conf=10.0
```

This time they appear:



The recommended route(s) are:

```
Go to work via the Coast Road  Conf=30.0
Go to work via the Highway  Conf=10.0
Go to work via the City Streets  Conf=10.0
```

The heading has been added. The user input is not included. The routes are sorted in order of value, and no negative values are included. The OK button has been replaced with a Restart button.

There are many ways to format and control the results screens. Experiment with some of the other options to see their effect.

Backward Chaining in Exsys Corvid

Backward Chaining is one of the most powerful capabilities in Exsys Corvid, and a good understanding of it is fundamental to building many knowledge automation expert systems. Backward Chaining allows larger problems to be broken into small, easily defined sections, which are automatically used by the system when, and if, they are needed.

Even though Backward Chaining is very powerful, it is quite easy to use - in fact, it just happens. If the system needs to know the value of a variable, and it has a rule that allows Corvid to derive the value for that variable, it will automatically execute the rule to obtain the value. This can happen over many rules and is recursive for as many layers as needed. You simply add rules that assign values to variables that are tested in other rules. The Backward Chaining rules can be put anywhere in the system. When it needs them, Corvid will find and use the rules.

Backward Chaining is referred to as Goal Driven. When run in a backward chaining mode, the system always has a "Goal List" of the Goals it is trying to achieve. This list is fundamental to backward chaining. Each item in the list is a Corvid variable. This could be a numeric variable such as a weight or cost, a multiple-choice list or other types of variables that can be assigned a value.

The top variable on the list is the one that the Inference Engine is actively trying to find a value for. To get that value, the Inference Engine will try to find rules that could be used to derive, or set, a value for the variable. If such a rule is found, the IF conditions in the rule are tested to determine if the rule is true. If there is enough data already in the system, the IF conditions of the rule may be able to be immediately determined to be true or false. If they are true, the rule can be used to set the value for the top variable on the Goal List. If the rule is determined to be false, the rule is discarded. However, if there is not enough data to tell if the rule is true or false, the Inference Engine determines what additional information it would need to evaluate the rule. This results in the system putting a new variable on the top of the Goal List. This becomes the new active "Goal" to be evaluated. This variable is needed to evaluate the rule being tested, but for the moment, that rule is ignored and the Inference Engine concentrates on finding a value for the new top variable on the Goal List.

As variables have values assigned, they are removed from the Goal List. The next variable in the list becomes the new top variable being actively worked on. This process continues until all relevant variables have been assigned values, including the initial variable that started the process. Once that starting variable is assigned a value and removed from the list, the process is complete.

Understanding backward chaining is very important, so let's work through a simple case in datail. In Corvid the top level Goal is established by a command from a Corvid Command Block. A simple command that would start backward chaining is "Determine if recommendation X is valid". In Corvid, the recommendation would be associated with a specific Corvid variable. The top level Goal will be to find the value for that specific variable. The actual command may be to find the values for a group of variables, such as all Confidence Variables, but they will be handled one variable at a time.

The variable the system is trying to establish a value for becomes the top-level Goal. Corvid looks through all the rules to find any rules that assign a value to that variable in their THEN part. These are the rules that are relevant to that variable. They are rules that potentially could set the value of the variable, but until there is more data, they cannot actually be used.

Corvid then starts sequentially testing the relevant rules in order to see if they are true. Typically, each rule's IF part will be made up of Boolean tests using other variables. To evaluate the Boolean tests, Corvid needs to know the value of the variables used. The first variable that is needed becomes the new top-level Goal, pushing the old top-level goal down one level in the list.

Corvid now repeats the process of looking through all the rules for a rule that assigns a value to the new variable now at the top of the Goal List. If a rule is found, this rule is tested to set a value for the current top-level Goal, and the conditions in its IF part are tested. This may lead to another variable becoming the top-level Goal.

The process is repeated until there are no rules found that could be used to set a value for the current Goal. Since it cannot be derived, Corvid will then ask the user for the value of that variable. The user input sets that value for the variable, and it can be removed from the Goal List. The next goal in the list again becomes the top-level goal, but now there is additional information. The Goal variable may be able to be assigned a value, or other rules may need to be tested to set the final value.

At some points, there may be many Goals in the list. As each Goal variable is assigned a final value, it is removed from the list and the next Goal again becomes the top-level Goal. Evaluating that Goal may result in the Goal list growing again, and eventually shrinking. This process continues until the initial variable is assigned a value and all variables are removed from the Goal List.

This may seem confusing at first, but is actually quite simple. The best way to understand it is to read about an example. A backward chaining rule will later be added to the "How to get to Work" system.

Example:

This example shows how backward chaining works in Corvid. It shows how the Goals in the system change, and how Backward Chaining can be used to modularize a system.

The system is to help first-level support staff prioritize support requests. It makes sure that the best customers get priority and receive a response within 4 hours.

When building a Backward Chaining system, start with the highest-level rules and then add more detailed rules that will be called by backward chaining.

At the highest level, the system is one rule:

```
IF
   The customer should receive priority service
THEN
   Call within 4 hours
```

A Command Block would tell the system that the initial Goal is to find a value for the variable [Call_in_4_hours]. This becomes the top-level Goal in the list.

Goal List:
1. Determine if the response should be within 4 hours.

The system would look through the rules (only 1 rule so far) and find the rule with the Goal variable in the THEN part. This rule could potentially set the value for the Goal variable, so it would be tested.

To determine if the relevant rule is true, and can be used to set a value for the Goal variable, the system needs to determine if the IF conditions are true. In this rule, that requires determining if "*The customer should receive priority service".* So that becomes the new top-level Goal.

Goal List:

1. Determine if the customer is a Priority customer.
2. Determine if the response should be within 4 hours.

Remember, to the system, ONLY the top-level Goal matters at the moment. So now the Inference Engine will temporarily stop trying to set a value for the "respond in 4 hours" goal, and concentrate on the new top Goal, *"Priority customer"*.

Since there are no other rules in the system, the system has no way to derive the value from other rules. When there is no way to derive a value, all the system can do is ask the system user. Once the user answers the question, the value for "*The customer should receive priority service"* will be known, and that goal can be dropped off the Goal List. The Goal List would now return to the original goal of determining if the response should be within 4 hours. If the customer was determined to be a priority customer, the single rule in the system can be used to determine the value for that Goal, and the session would be done. If the customer was not determined to be a priority customer, there are no rules in the system that would allow setting a value for the "respond in 4 hours" variable.

In reality, asking the system user if "*The customer should receive priority service*" is probably not a reasonable question to ask since the staff may not have the background to answer it correctly and consistently. Instead, the system can be expanded to make it much more capable and less subjective. These additional rules will automatically be used by backward chaining.

First, start with the same rule as before:

```
IF
   The customer should receive priority service
THEN
   Call within 4 hours
```

There is also the same top-level goal:

Goal List:
1. Determine if the response should be within 4 hours.

Since directly asking the system user if *"The customer should receive priority service"* is not a reasonable question that they can answer, additional rules will be added to derive the value. The user should only be asked questions they can unambiguously and objectively answer. The values for questions that are subjective should be derived from the objective data. To implement this, add a set of rules that specify when a customer should receive priority service. These rules will ask the user more objective questions and derive the needed information. Backward chaining will automatically use these rules to derive the value for the "Priority customer" question.

In this case, add 3 rules that identify a priority customer:

```
IF
   The customer purchases are over $250,000 per year
THEN
   The customer should receive priority service

IF
   The customer works for a Partner company
THEN
   The customer should receive priority service

IF
   The customer's company has significant growth potential
THEN
   The customer should receive priority service
```

Now when the system runs, the same initial Goal is set by the Command Block. The same first rule is found and tested, setting the new top-level Goal to determine if the customer is a Priority customer.

Goal List:

1. Determine if the customer is a Priority customer.
2. Determine if the response should be within 4 hours.

However, now the system has rules to derive if the customer is a priority customer and these rules will automatically be used rather than directly asking the user.

Each rule will be tested in order. The first rule found is:

```
IF
   The customer purchases are over $250,000 per year
THEN
   The customer should receive priority service
```

The IF condition in that rule becomes the new top-level Goal:

Goal List:

1. Determine if the purchases are over $250,000.
2. Determine if the customer is a Priority customer.
3. Determine if the response should be within 4 hours.

The system would automatically search for any rule that would set a value for the "*Purchases over $250,000*" variable. There is no such rule in the system, so the data would have to be asked of the user, or obtained from an external source such as a database. This is a much more reasonable question to ask of the user. If they have access to a sales database, they can check to see the volume of sales. (In practice this would more likely be done automatically by the Corvid system, which can interface directly to external databases.)

Suppose the answer is that the customer's purchases are $20,000. The top-level Goal on purchase amount now has been satisfied and it is dropped off the Goal List.

Goal List:

1. Determine if the customer is a Priority customer.
2. Determine if the response should be within 4 hours.

The next Goal in the list, "Priority Customer" again becomes the top-level Goal and the rule associated again becomes the rule being tested.

```
IF
   The customer purchases are over $250,000 per year
THEN
   The customer should receive priority service
```

Based on the value of the customer's purchases of $20,000, the rule can now be determined to be false, so it will be discarded and cannot tell us anything about the top-level Goal variable.

However, there is another rule in the system that could also provide information on the top-level Goal variable:

```
IF
   The customer works for a Partner company
THEN
   The customer should receive priority service
```

So that rule is now tested. The variable used in the IF part of that rule becomes the new top-level Goal.

Goal List:

1. Determine if customer is a Partner company.
2. Determine if the customer is a Priority customer.
3. Determine if the response should be within 4 hours.

There are no other rules that allow the "*Partner company*" variable to be derived, so the user is asked if the customer works for a Partner company. This is a reasonable question to ask the user since the number of partner companies is probably reasonably small, and well known.

If the customer is from a Partner company, the value of the top "Partner Company" goal will be true, and it will be dropped from the Goal List. This put the "Priority customer" goal back on top. Since the customer is from a Partner Company, the rule being tested assigns the customer as a "Priority customer", and that goal is dropped from the Goal List. Now the list is back to the original "Respond in 4 hours" goal, and the original rule in the system. Now that rule can be determined to be true, and the "Respond in 4 hours" goal determined to be true. Since that is the last goal in the list, the system is done.

Adding One More Layer of Chaining

In this example above, this time assume that the customer is <u>not</u> from a Partner company and did not have sales over $250,000. This will eliminate the first 2 rules for deriving the "Priority customer" goal. After the Partner company rule, the next one that would be tested is:

```
IF
   The customer's company has significant growth potential
THEN
   The customer should receive priority service
```

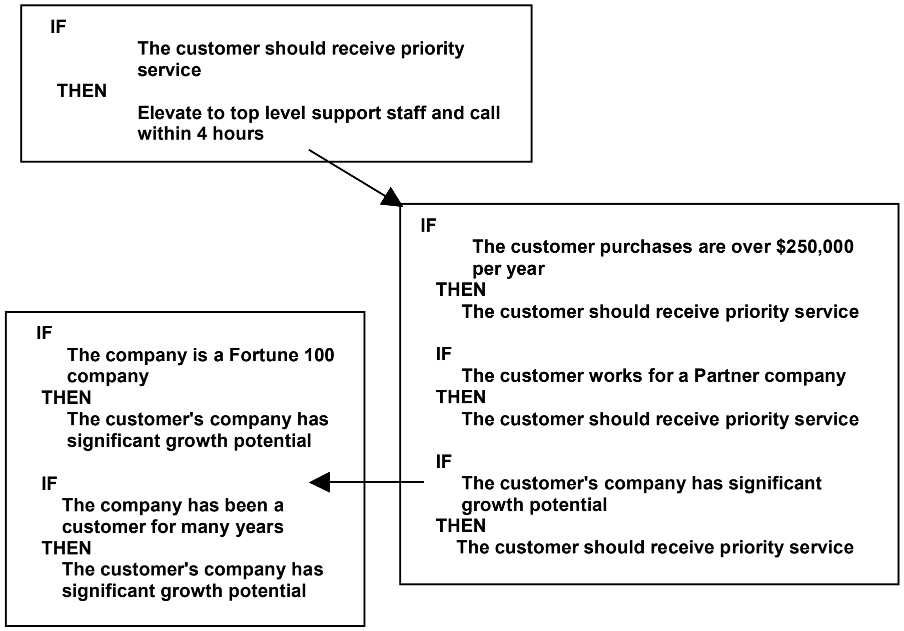
This may also be a question that the typical system user could not answer. To enhance the system, it needs a few more rules to explain which customers have "significant growth potential". One option would be to have the Sales Manager write a few rules that describe how to identify a company with significant growth potential. The following rules could be added:

```
IF
   The company is a Fortune 100 company
THEN
   The customer's company has significant growth potential

IF
   The company has been a customer for many years
THEN
   The customer's company has significant growth potential
```

If the system needed to know if the company had significant growth potential, the new rules would automatically be called and the user would be asked these more objective questions. (In a real system, this data might come from a database that kept information on customer companies.)

Backward Chaining allows complex problems to be broken into small modules. Starting with the highest-level description of the rules that solve the problem allows a working system to be produced very quickly. If the questions it asks are not at the appropriate level for the intended system user, rules could be added to derive the information from simpler questions.

This shows the rules in the system and how individual rules call other blocks of rules.



IF
The customer should receive priority service
THEN
Elevate to top level support staff and call within 4 hours

IF
The customer purchases are over $250,000 per year
THEN
The customer should receive priority service

IF
The customer works for a Partner company
THEN
The customer should receive priority service

IF
The customer's company has significant growth potential
THEN
The customer should receive priority service

IF
The company is a Fortune 100 company
THEN
The customer's company has significant growth potential

IF
The company has been a customer for many years
THEN
The customer's company has significant growth potential

A system can be started with high-level rules that describe the decision-making process and expanded to whatever level is needed by adding blocks of rules that cover specific detailed aspects of the decision.

Also, the blocks can be used in more than one area. If another part of the system needed to know if a customer should receive priority service, it would automatically invoke the same block of rules to derive the value.

If there is another criterion that determines if a customer is a "priority customer", all that has to be done is to add another rule that sets the value. That new rule will automatically be called and tested in any situation where it might be relevant.

Comparison of Inference Engines to Traditional Programming

The Exsys Corvid backward chaining Inference Engine makes system development and maintenance much easier. When first exposed to IF/THEN rule logic, it is easy to confuse it with the simple IF/THEN statements of computer languages such as C++ and BASIC. However, the Inference Engine is fundamentally VERY different and much more powerful. In a typical programming language, there can be nested IF/THEN blocks, but if a deeply nested IF/THEN relationship is needed by another section of the code, it can only be called by duplicating the code, or making a function that can be called from several places. A standard program will not simply and automatically "call" the necessary section of computer code just because it exists in the system - yet this is exactly what the Inference Engine does.

In backward chaining, if any rule assigns a value to variable X, that rule will automatically be used whenever another rule being tested needs to know the value of X. Rules can be anywhere in the system, and there does not need to be any explicit linking of rules. Having 2 rules that use the same variable is all that is needed to link them.

This rather "free-form" nature of the rules in a Corvid system makes development very simple. You just provide the IF/THEN rules needed to make a decision, tell the system what to derive and the Inference Engine does the rest. Questions are automatically asked in a focused manner and, only relevant questions that cannot be derived from other rules, are asked.

Frequently, a programmer will look at the IF/THEN rules in a simple system, such as those used to demonstrate concepts, and say "I could just program that in a few lines of Visual Basic". For very simple systems, that is true. However, if the system grows even a modest amount, the problem can rapidly become very complex to program by traditional techniques. It requires far more than just nested IF/THEN statements to handle cases where there can be multiple sources to derive a fact, multiple uses of the same rules, or many levels of derivation that may depend dynamically on user input.

Traditional code can rapidly become very complicated in order to handle all the situations. It becomes even more complex dealing with the issues of adding new rules and maintaining the system. Adding even a single new rule with traditional programming could be quite difficult, and if not added correctly, could have ripple effects that would be difficult to detect and fix. With Corvid adding a new rule is easy and it will automatically be used where it is needed.

For systems of any significant level of complexity, the only approach that works is to handle the rules as "data", rather than incorporating them in the actual code of the system. Then, writing a program to process the rules as "data". This is what the Exsys CorvidInference Engine does for you. Attempting to build a robust expert systems by traditional programming techniques will usually be much more expensive and far less likely to be successful or maintainable.

Forward Chaining in Exsys Corvid

In addition to backward chaining, Exsys Corvid also supports Forward Chaining, and several hybrids between the two methods.

Forward chaining is conceptually much simpler than backward chaining. The rules are simply tested in the order that they occur. If a variable is needed to determine if a rule is true or false, and the value of that variable is not known, the variable is asked of the user. If a rule is determined to be true, the assignments in the THEN part add data to what the system "knows", and can be used in subsequent rules. If a rule is determined to be false, it is discarded.

Forward chaining systems are "data driven" in that a set of data is simply processed by the rules with no specific defined Goal. Forward chaining systems typically execute faster than backward chaining ones since there is not the overhead of dynamically determining what to ask when. However, forward chaining asks less focused questions and is not as good an emulation of a human interaction with an expert. Also, the order of questions is very dependent on rule order. Forward chaining is well suited to monitoring systems where a set of data is available at the start of a session. The logic in the rules is applied to the data to produce results, but it does not matter in what order the system uses the data.

Backward chaining is much more useful for systems that emulate a human decision-making process. A human expert intuitively thinks: *"The cause could be X. To determine that I need to know the value of Y, but to find Y, I first need to know Z".* This is the same sort of chain that a backward chaining system would produce from rules. A backward chaining system asks questions in a focused manner and only asks questions if and when the data is actually needed.

Exsys Corvid also supports hybrid approaches where the basic system uses forward chaining, but allows backward chaining to derive needed values. This combination can provide the best of both approaches and is often very effective.

Adding Backward Chaining to the Demo System

In the system for deciding on the best route to take to work, there is a question that asks the system user if the driving conditions are "good" or "bad". This information is then used to decide on routes. This works, but the determination of when the conditions are "good" or "bad" is subjective and different users might answer in different ways.

This can easily be corrected by using backward chaining. The current system does not have to be changed, just a few more rules added. The system could ask more specific and easily answered questions about the weather and then set the more subjective value for "good" or "bad".

For example, you may consider the driving conditions to be bad if:

- It is raining or snowing
- It is foggy
- It is night
- There is snow or ice on the road

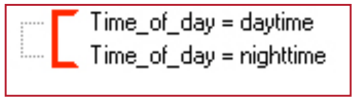
First add 3 new variables:

1.	Name:	Time_of_day
	Prompt:	The time of day is
	Values:	Daytime
		Nighttime
2.	Name:	Weather
	Prompt:	The current weather is
	Values:	Clear
		Raining
		Snowing
		Foggy
3.	Name:	Snow_on_road
	Prompt:	Is there snow or ice on the road?
	Values:	Yes
		No

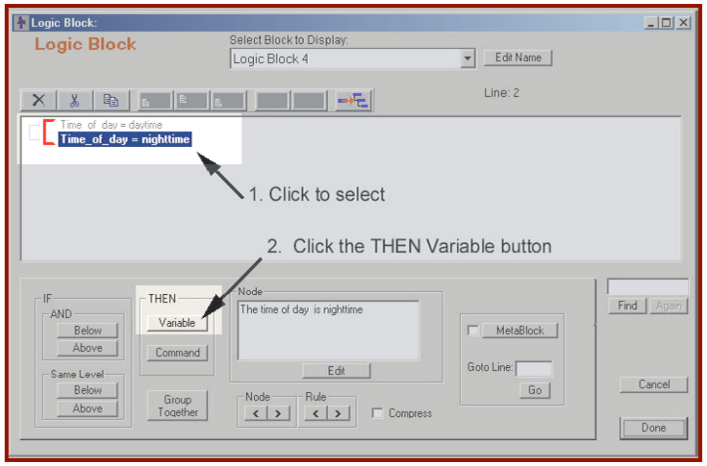
These variables can be used in a new Logic Block to set the value for the "Driving_conditions" variable. However, only a single value should be set - the system should not set both "good" and "bad" at the same time. Many answers can set "bad", but only one path will select "good".

Start a new Logic Block by clicking on the Logic Block icon.

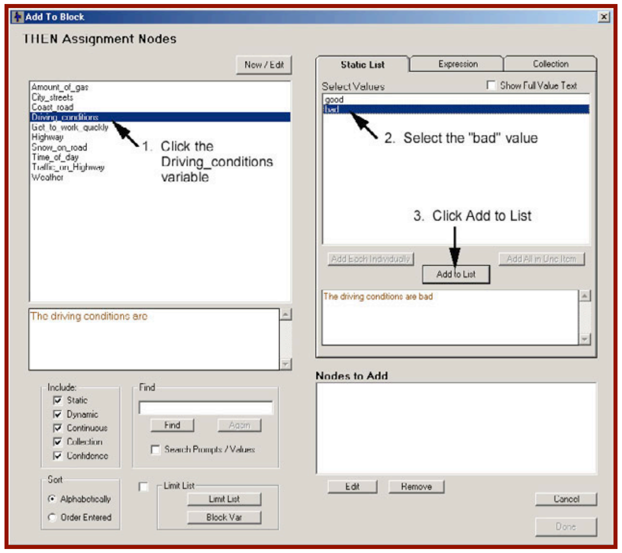
The first node to add is the Time_of_Day variable, with one node for each of the possible values.


Time_of_day = daytime
Time_of_day = nighttime

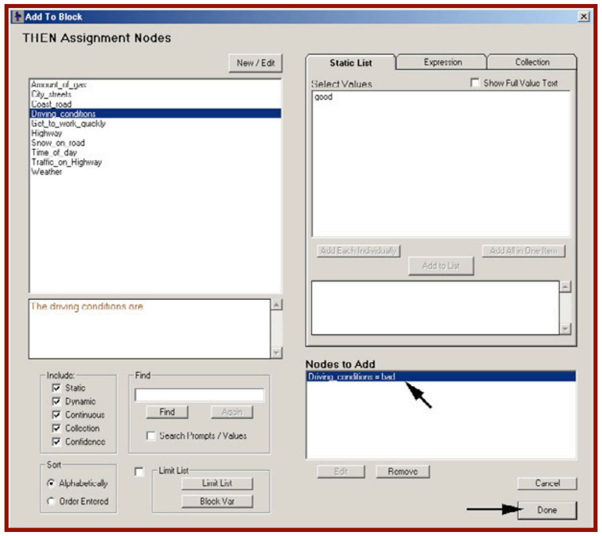
The "Time_of_day = nighttime" node is easy, it sets "Driving_conditions" to "bad" and no further questions need to be asked. Click on the "Time_of_day = nighttime" node to select it:



Click the "THEN, Variable" button. This will display the window for adding nodes. However, this time it will be used to ASSIGN a value to a Static List variable.

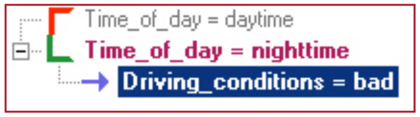


To do this, select the variable to assign a value to, in this case "Driving_conditions". The possible values will be displayed in a list on the right. Select the value to assign, in this case "bad". Then click the Add to List button.

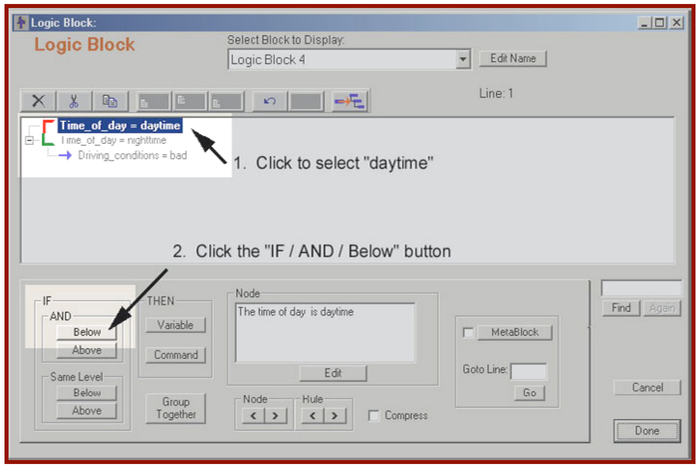


The assignment is displayed in the "Nodes to Add" section, so click the "Done" button.

This will add the assignment node to the Logic Block.

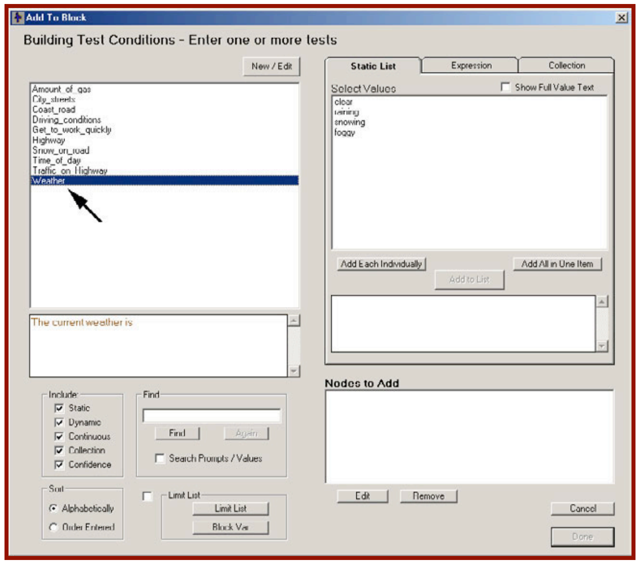

Time_of_day = daytime
Time_of_day = nighttime
Driving_conditions = bad

If the system user says it is nighttime, you know the driving conditions are bad (at least in the context of this system) and you do not need to ask any other questions. However, if it is daytime, you need more information. A new IF node group should be added under the "daytime" value. To do this, click on the "Time_of_day = daytime" node to select it and click on the "IF / Add / Below" button.



The time of day is daytime

This time IF node should be built for the variable "Weather", so click on Weather to select it.



Amount_of_gas
City_streets
Coast_road
Driving_conditions
Get_to_work_quickly
Highway
Snow_on_road
Time_of_day
Traffic_on_Highway
Weather

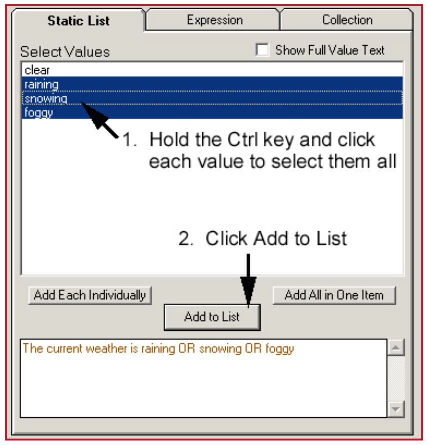
clear
raining
snowing
foggy

The current weather is

- [✓] Static
- [✓] Dynamic
- [✓] Continuous
- [✓] Collection
- [✓] Confidence

If the system user selects rain, snow or fog, the "Driving_conditions = bad" value will be set. Since there is not logical distinction between the values in this system, those 3 values should be added to the same node.

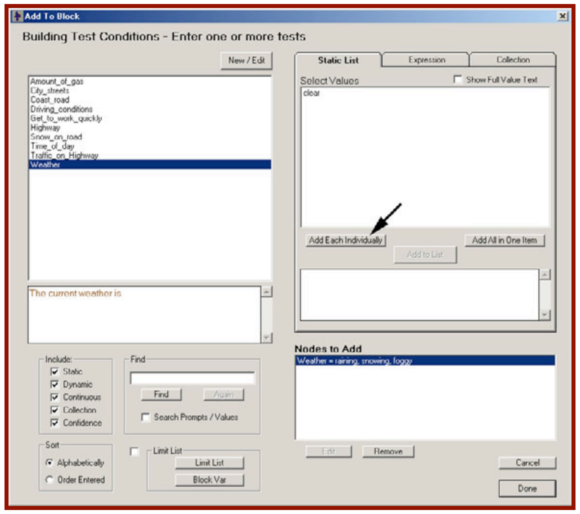
To select the 3 values in the same node, hold down the “Ctr” key and click on each of the values. When all are selected, click the Add to List button.



clear
raining
snowing
foggy

The current weather is raining OR snowing OR foggy

Then add the last value as a separate node by clicking the “Add Each Individually” button.



Building Test Conditions - Enter one or more tests

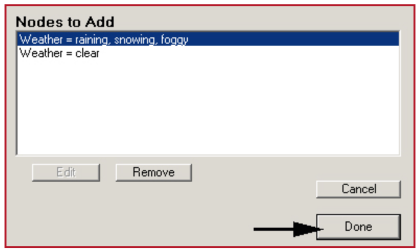
Amount_of_gas
City_streets
Coast_road
Driving_conditions
Get_to_work_quickly
Highway
Snow_on_road
Time_of_day
Traffic_on_Highway
Weather

The current weather is

Select Values
clear

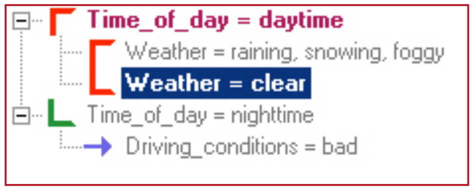
Nodes to Add
Weather = raining, snowing, foggy

This will add two nodes.



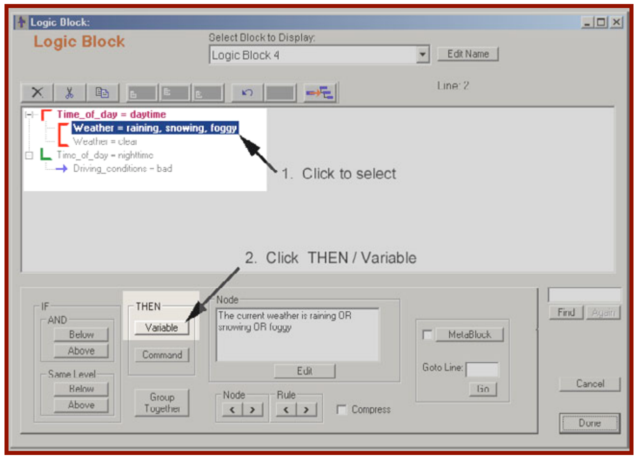
Nodes to Add
Weather = raining, snowing, foggy
Weather = clear

Click the Done button to add them to the Logic Block.



Time_of_day = daytime
Weather = raining, snowing, foggy
Weather = clear
Time_of_day = nighttime
Driving_conditions = bad

If the weather is raining, snowing or foggy, there is enough information to set the value for the variable Driving_conditons. Click the "Weather = raining, snowing, foggy" node to select it. Click the "THEN / Variable" button.

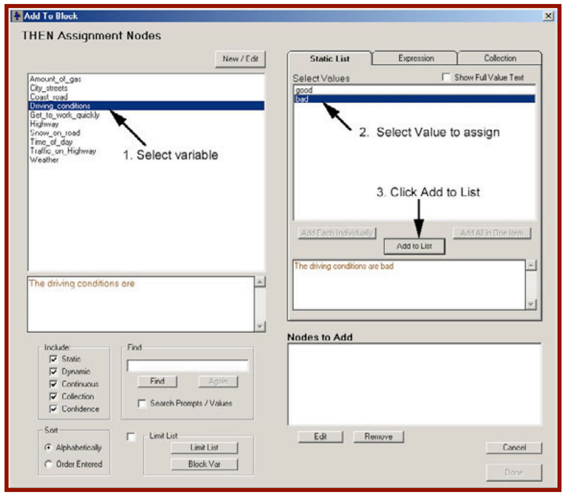


Logic Block 4

Time_of_day = daytime
Weather = raining, snowing, foggy
Weather = clear
Time_of_day = nighttime
Driving_conditions = bad

The current weather is raining OR snowing OR foggy

As before, in the Add Node window, select to set the value of the variable Driving_conditions to "bad".



THEN Assignment Nodes

Amount_of_gas
City_streets
Coast_road
Driving_conditions
Get_to_work_quickly
Highway
Snow_on_road
Time_of_day
Traffic_on_Highway
Weather

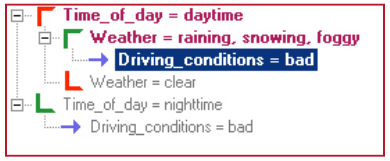
The driving conditions are

Select Values
good
bad

The driving conditions are bad

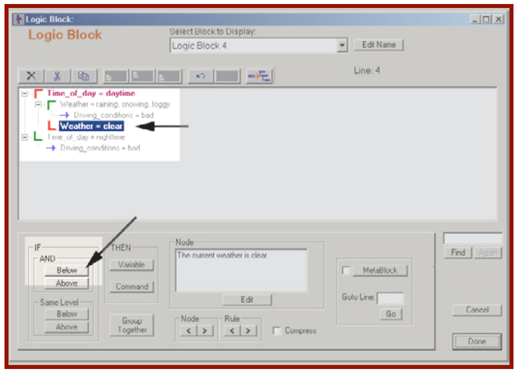
Nodes to Add

This is the only node to add, so click the “Done” button to add the node to the Logic Block.



Time_of_day = daytime
Weather = raining, snowing, foggy
Driving_conditions = bad
Weather = clear
Time_of_day = nighttime
Driving_conditions = bad

Now the Logic Block needs to include nodes for the case where it is daytime and clear, but there may still be snow or ice on the road from a previous storm. This will be another set of IF nodes. To add it the logic, click the "Weather = clear" node to select it, and click the "IF / AND / Below" button.

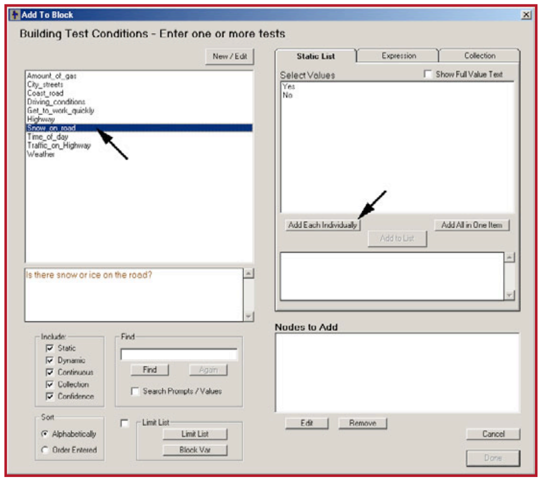


Logic Block:
Logic Block
Select Block to Display: Logic Block 4
Line: 4

Time_of_day = daytime
Weather = raining, snowing, foggy
Driving_conditions = bad
Weather = clear
Time_of_day = nighttime
Driving_conditions = bad

Node: The current weather is clear

This time the variable that is needed is the "Snow_on_road" variable. Click this variable to select it and then click the "Add Each Individually" button to add a node for each of the 2 possible values.



Add To Block
Building Test Conditions - Enter one or more tests

Amount_of_gas
City_streets
Coast_road
Driving_conditions
Get_to_work_quickly
Highway
Snow_on_road
Time_of_day
Traffic_on_Highway
Weather

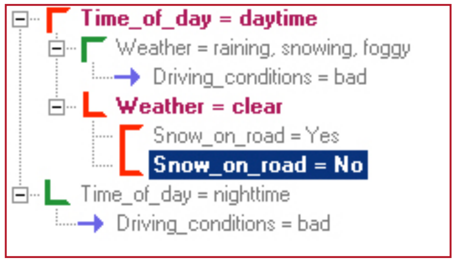
Is there snow or ice on the road?

Select Values
Yes
No

Nodes to Add

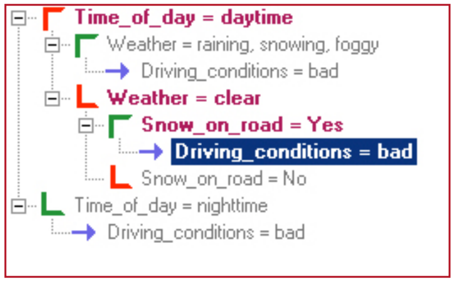
Include: ☑ Static ☑ Dynamic ☑ Continuous ☑ Collection ☑ Confidence
Sort: ● Alphabetically ○ Order Entered
☐ Search Prompts / Values
☐ Show Full Value Text

Then click the “Done” button to add the nodes to the Logic Block.



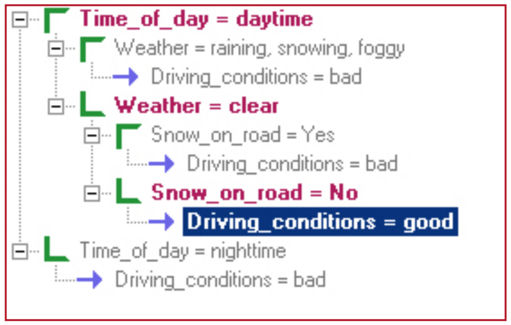
Time_of_day = daytime
Weather = raining, snowing, foggy
Driving_conditions = bad
Weather = clear
Snow_on_road = Yes
Snow_on_road = No
Time_of_day = nighttime
Driving_conditions = bad

Now values for the variable Driving_conditions can be set in the two new nodes. Click the "Snow_on_road = yes" node to select it. Then click the "THEN / Variable" button. Just as before, add a node that sets "Driving_conditions = bad".



Time_of_day = daytime
Weather = raining, snowing, foggy
Driving_conditions = bad
Weather = clear
Snow_on_road = Yes
Driving_conditions = bad
Snow_on_road = No
Time_of_day = nighttime
Driving_conditions = bad

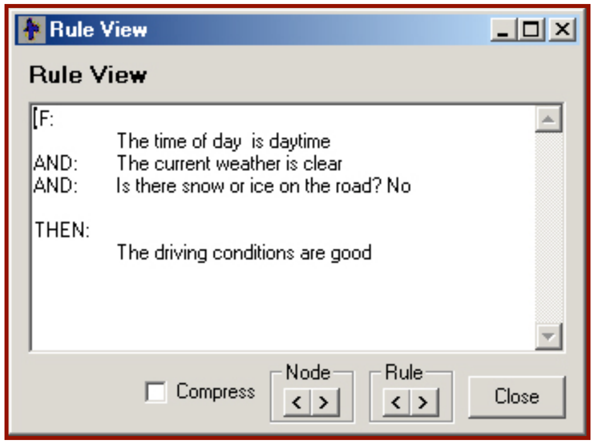
Now, click the "Snow_on_road = No" node to select it and then click the "THEN / Variable" button. This time add a node that sets "Driving_conditions = **good**". This is done the same as before, but select the value "good".



Time_of_day = daytime
Weather = raining, snowing, foggy
Driving_conditions = bad
Weather = clear
Snow_on_road = Yes
Driving_conditions = bad
Snow_on_road = No
Driving_conditions = good
Time_of_day = nighttime
Driving_conditions = bad

This now completes the Logic Block. The variable Driving_conditions will always be assigned one, and only one, value by any path through the Logic Block. This assures that the variable will always be derived by asking questions that are easier for the system user to answer.

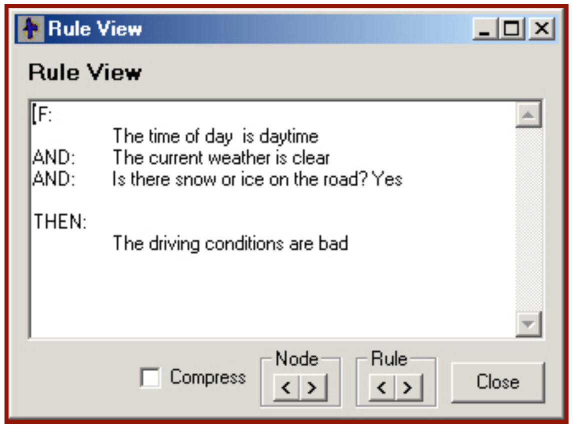
To check the logic, click on various nodes and look at the Rule View window. For example, click on the "Driving_conditions = good" node and the Rule View will show:



Rule View

```
IF:
        The time of day  is daytime
AND:    The current weather is clear
AND:    Is there snow or ice on the road? No

THEN:
        The driving conditions are good
```

This is exactly what was expected. Click on the "Driving_conditions = bad" just above it and the Rule View will show:

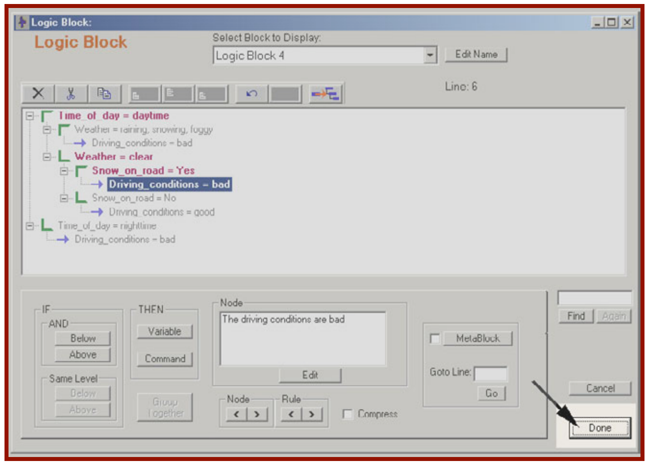


Rule View

```
IF:
        The time of day  is daytime
AND:    The current weather is clear
AND:    Is there snow or ice on the road? Yes

THEN:
        The driving conditions are bad
```

That is all that is needed to add rules for backward chaining. There is no need to explicitly link these new rules into the old rules. All that is needed is a rule, anywhere in the system, which sets the value of a variable that is needed in another rule. Since these rules set the value of the variable "Driving_conditions", they will automatically be used by the Corvid Inference Engine whenever the value for "Driving_conditions" is needed.

To see this, close the Logic Block by clicking the Done button on the block.



Logic Block 4

```
Time_of_day = daytime
    Weather = raining, snowing, foggy
        Driving_conditions = bad
    Weather = clear
        Snow_on_road = Yes
            Driving_conditions = bad
        Snow_on_road = No
            Driving_conditions = good
Time_of_day = nighttime
    Driving_conditions = bad
```

Node: The driving conditions are bad

Then run the system by clicking the blue triangle icon.

The system will start with the same 3 questions as before:

1. Do you need to get to work quickly?
2. Are there traffic problems on the highway?
3. Number of gallons of gas in the car

Previously the next question was about the driving conditions. Now instead of asking that question, the system asks about the "Time of Day". This is because it has automatically called the new rules that were just added to derive the value of the Driving_conditions variable.

The system will only ask enough questions to set the value for Driving_conditions. If you answer the first question "night", the system knows the value of Driving_conditions is "bad" and does not need to ask additional questions. If you answer "day" it will ask additional questions until a value is set. Once the value for Driving_conditions is established, the system can finish the session and display the results.

Build from the Top Down

Backward chaining is a very powerful technique. It allows complex problems to be broken into smaller parts that can be handled by a small block of logic. The Inference Engine will automatically pull all the rules together to derive values whenever possible and ask the appropriate questions.

For many systems, it is best to start the system by describing the logic at the highest level. Once that is working and giving the correct answers, consider if the questions are appropriate for the intended system user. If they are not, use backward chaining to derive the values of the variables that should not be directly asked. This allows building systems very rapidly. Since the top-level logic is created first, the system can be run and produce results as the backward chaining layers are added. This allows much easier testing of the system as it is being developed.

You're Done! Next Steps

Exsys Corvid has many more features and capabilities. However, with just the main elements covered here, you can build very effective and powerful expert systems. Of the thousands of expert systems that have been built with Exsys tools, most use only the logical structures and concepts covered in this introduction. It is recommended you build and complete your first systems using this approach. Later you can familiarize yourself with other features, integration, customization and deployment options.

The best way to fully learn Corvid quickly is to take one of the Exsys Corvid training classes – available at the Exsys Training Facility in Albuquerque, NM, onsite or via Web conferencing. These cover not only how the program works, but also many approaches for handling specific types of situations. The classes help students building a variety of systems from simple backward chaining systems to complex MetaBlock product selection systems, and cover interface issues such as databases.

To get your expert system project built and deployed as rapidly as possible, consider one of the "Pilot Project" or "Prototype" packages available from Exsys Inc. These include the Corvid tools you will need, training and, most importantly, one-on-one consultation with an Exsys Inc. knowledge engineer on your particular application(s). This allows you to get expert advice on the best way to approach a particular problem and ongoing support from Exsys knowledge engineer experts familiar with your specific system. This gets systems built quickly and with a high assurance of success.

EXSYS®
Capture Knowledge
Deliver Answers™

Exsys Inc. has an over 25-year history of helping companies implement expert systems that are easy to build, affordable and quickly providing cost and time savings results, as well as giving your Web site an interactive "smart" competitive advantage.

EXSYS, Inc.
6301 Indian School Rd. NE, Suite 700
Albuquerque, NM 87110 U.S.A.
Tel: 1.505.888.9494 Fax: 1.505.888.9509
info@exsys.com **www.exsys.com**